



GI-Edition

Lecture Notes in Informatics

Torsten Grust, Felix Naumann, Alexander Böhm, Wolfgang Lehner, Jens Teubner, Meike Klettke, Theo Härder, Erhard Rahm, Andreas Heuer, Holger Meyer (Hrsg.)

Datenbanksysteme für Business, Technologie und Web (BTW 2019)

**4.–8. März 2019
Rostock**

Proceedings



GESELLSCHAFT
FÜR INFORMATIK



Torsten Grust, Felix Naumann,
Alexander Böhm, Wolfgang Lehner,
Jens Teubner, Meike Klettke,
Theo Härder, Erhard Rahm,
Andreas Heuer, Holger Meyer (Hrsg.)

**Datenbanksysteme für
Business, Technologie und Web
(BTW 2019)**

**18. Fachtagung des GI-Fachbereichs
„Datenbanken und Informationssysteme“ (DBIS)**

**4.–8. März 2019
in Rostock, Deutschland**

Lecture Notes in Informatics (LNI) — Proceedings

Series of the Gesellschaft für Informatik (GI)

Volume P-289

ISBN 978-3-88579-683-1

ISSN 1617-5468

Volume Editors

Torsten Grust

Universität Tübingen
Lehrstuhl für Datenbanksysteme
72076 Tübingen, Germany
Email: torsten.grust@uni-tuebingen.de

Felix Naumann

Hasso-Plattner-Institut Potsdam
Information Systems
14482 Potsdam, Germany
Email: felix.naumann@hpi.de

Alexander Böhm

SAP Walldorf
69190 Walldorf, Germany

Wolfgang Lehner

TU Dresden
Lehrstuhl für Datenbanken
01062 Dresden, Deutschland
Email: wolfgang.lehner@tu-dresden.de

Theo Härder

TU Kaiserslautern
Fachbereich Informatik
67653 Kaiserslautern, Germany
Email: haerder@informatik.uni-kl.de

Erhard Rahm

Universität Leipzig
Institut für Informatik
04109 Leipzig, Germany
Email: rahm@informatik.uni-leipzig.de

Andreas Heuer

Universität Rostock
Lehrstuhl für Datenbank- und Informationssysteme
18055 Rostock, Germany
Email: ah@informatik.uni-rostock.de

Meike Klettke

Universität Rostock
Institut für Informatik
18055 Rostock, Germany
Email: meike.klettke@uni-rostock.de

Holger Meyer

Universität Rostock
Lehrstuhl für Datenbank- und Informationssysteme
18055 Rostock, Germany
Email: hm@ieee.org

Series Editorial Board

Heinrich C. Mayr, Alpen-Adria-Universität Klagenfurt, Austria
(Chairman, mayr@ifit.uni-klu.ac.at)

Torsten Brinda, Universität Duisburg-Essen, Germany

Dieter Fellner, Technische Universität Darmstadt, Germany

Ulrich Flegel, Infineon, Germany

Ulrich Frank, Universität Duisburg-Essen, Germany

Michael Goedicke, Universität Duisburg-Essen, Germany

Ralf Hofestädt, Universität Bielefeld, Germany

Wolfgang Karl, KIT Karlsruhe, Germany

Michael Koch, Universität der Bundeswehr München, Germany

Thomas Roth-Berghofer, University of West London, Great Britain

Peter Sanders, Karlsruher Institut für Technologie (KIT), Germany

Andreas Thor, HFT Leipzig, Germany

Ingo Timm, Universität Trier, Germany

Karin Vosseberg, Hochschule Bremerhaven, Germany

Maria Wimmer, Universität Koblenz-Landau, Germany

Dissertations

Steffen Hölldobler, Technische Universität Dresden, Germany

Thematics

Andreas Oberweis, Karlsruher Institut für Technologie (KIT), Germany

© Gesellschaft für Informatik, Bonn 2019

printed by Köllen Druck+Verlag GmbH, Bonn



*This book is licensed under a
Creative Commons Attribution-NonCommercial 3.0 License.*

Vorwort

Die 18. Fachtagung “Datenbanksysteme für Business, Technologie und Web” (BTW) des Fachbereichs “Datenbanken und Informationssysteme” (DBIS) der Gesellschaft für Informatik (GI) findet vom 4. bis 8. März 2019 an der Universität Rostock statt. Pünktlich zu einem Multi-Jubiläum besucht damit **die** deutsche Datenbanktagung zum ersten Mal in ihrer Geschichte die Ostseeküste: Stadt und Universität Rostock feiern ein Doppeljubiläum (800 Jahre Stadt Rostock in 2018, 600 Jahre Universität Rostock in 2019). Daneben feiert auch die Rostocker Informatik einige Jubiläen in 2019: eine Computergrafik gibt es seit 50 Jahren an der Universität, eine Informatik seit 35 Jahren, den Lehrstuhl Datenbank- und Informationssysteme seit 25 Jahren.

Auf der BTW trifft sich nun auch schon seit fast 35 Jahren im zweijährigen Rhythmus die deutschsprachige Datenbankgemeinde, um neue Fragestellungen zu erörtern und aktuelle Forschungsergebnisse zu präsentieren und zu diskutieren. Nicht nur Wissenschaftler, sondern auch Praktiker und Anwender finden sich hier zum Wissens- und Erfahrungsaustausch zusammen. Die BTW 2019 bietet ein wissenschaftliches Programm, ein Industrieprogramm, ein Demonstrationsprogramm und ein Studierendenprogramm, dazu verschiedene Workshops und Tutorien. Zum zweiten Mal wird auf der BTW ein Wettbewerb veranstaltet, die sogenannte Data Science Challenge — in diesem Jahr zum aktuellen Thema Feinstaubbelastung in Städten.

Die Datenbanktechnologie befindet sich in einem interessanten Spannungsfeld zwischen ihren etablierten Kernthemen — die (glücklicherweise) wohl niemals aus der Diskussion und den Konferenzprogrammen verschwinden werden — und neuen Fronten und Anwendungen, die wir uns mit unserem “Datenbank-Verstand” erobern. Das zeigte sich exakt so in den Einreichungen, die wir in diesem Jahr erhielten. Unter den insgesamt 37 Beiträgen, die im Scientific Track eingereicht wurden, dominieren Papiere zur Anfrageverarbeitung und Implementationstechniken. Direkt dahinter stehen jedoch Machine Learning, Wissensmanagement und die Verarbeitung von Nicht-Standarddatentypen. Fünfzehn dieser Beiträge werden im Wissenschaftlichen Programm in Rostock präsentiert.

Zwölf weitere Einreichungen wurden uns im Industrial Track zugesandt. Hier findet sich eine ganz ähnliche thematische Breite, die von “Domain Query Optimization” bis zur Bekämpfung von “Spam in Dating Apps” reicht. Es ist einfach nur spannend zu sehen, welche Fortentwicklung und welchen Einsatz Datenbanksysteme in der industriellen Praxis erfahren. Wir freuen uns auf neun Papiere, die im Industriellen Programm der BTW 2019 vorgestellt werden.

Das Gesamtprogramm der BTW ist eine Kooperation zwischen den insgesamt 231 Autoren und den 69 Mitgliedern unserer Programm-Komitees. Letztere haben sich ausnahmslos zügig, kooperativ und mit Expertise den Beiträgen gestellt. Sowohl die eigentlichen Reviews, die nachfolgende Diskussion — in diesem Jahr auch wieder auf einem der schon traditionellen PC-Meetings in Frankfurt — als auch das Mentoring von Autoren haben geholfen, ein

Programm zu formen, auf das wir stolz sein können. Die PC-Chairs bedanken sich ganz herzlich bei allen Autoren und ihren Komitees für die harte Arbeit auf beiden Seiten!

Der Workshopband bietet zu den drei Workshops “Novel Data Management Ideas on Heterogeneous (Co-)Processors (NoDMC)”, “Digitale Lehre im Fach Datenbanken” und “Big (and Small) Data in Science and Humanities” insgesamt 17 Beiträge. Daneben werden der SPP 2037: “Scalable Data Management for Future Hardware”, das Tutorienprogramm mit fünf und das Studierendenprogramm mit neun Beiträgen sowie die beteiligten Gruppen der Data Science Challenge vorgestellt. Der “BTW 2019 Workshopband” erscheint ebenfalls in der LNI-Reihe der GI als Band 290, ISBN 978-3-88579-684-8.

Für das Demo-Programm der BTW wurden aus den Einreichungen 16 Beiträge ausgewählt, die erzielte Ergebnisse der Forschung im Bereich der Datenbank- und Informationssystemtechnologien auf interaktive Weise einem breiten Publikum vorzustellen. Das breite Themenspektrum im Demo-Programm reicht von Tools zur Veranschaulichung relationaler Algebra, Analyse von Linked Data, Efficient Data Processing, Profiling von NoSQL-Datenbanken über Methods for Interactive Clustering, Graph Pattern Matching, Big Graph Analysis bis hin zu eLearning Tools für Datenbankthemen. Zahlreiche Forschungsgruppen stellen dabei ihre Arbeiten vor. Ein besonderer Schwerpunkt liegt im Demo-Programm auf Beiträgen, in denen Datenbanktechnologien angewendet werden, etwa zu zuverlässigen Verspätungsvorhersagen, Precision Oncology und Movie Recommender Systemen. Alle Beiträge des Demo-Programms sind in diesem Tagungsband enthalten.

Im Rahmen des Hauptprogramms findet auch die Panel-Diskussion zum Thema “Datenbanksysteme im Zeitalter von KI und Data Science“ (Moderator: Volker Markl) statt. Insbesondere wird hier geklärt, welche Rolle Datenbanksysteme bei KI und Data Science spielen und welche technologischen Herausforderungen zu lösen sind.

Leuchttürme des BTW-Programms an der Ostsee sind die drei eingeladenen Vorträge. Stefanie Rinderle-Ma (Universität Wien) wird mit “From LEGO to the Shopfloor: Driving Digitalization Through Process Technology” gleich zu Beginn eine (LEGO-)Brücke zwischen Datenbankforschung und -praxis bauen. Ihab Ilyas (University of Waterloo) verkörpert mit “Building Scalable Machine Learning Solutions for Data Cleaning” das oben erwähnte neue, breitere Verständnis von Datenbanktechnologie. Genau dieses unterstreichen Frank Renkes und Christian Sommer (SAP Walldorf) mit “Blockchain in the Context of Business Applications and Enterprise Databases” gleich nochmals.

Dieses Jahr wurden zudem zum zehnten Mal die BTW-Dissertationspreise des GI-Fachbereichs DBIS vergeben. Die eingereichten Dissertationen aus dem Zeitraum Oktober 2016 bis September 2018 waren durchweg von hervorragender Qualität. Auf Basis einer umfassenden vergleichenden Begutachtung wurden drei Arbeiten für die Auszeichnung ausgewählt: “Architectural Principles for Database Systems on Storage-Class Memory” von Ismail Oukid (TU Dresden), “Data Profiling — Efficient Discovery of Dependencies” von

Thorsten Papenbrock (HPI Potsdam) und “Modern Techniques for transaction-oriented Database Recovery” von Caetano Sauer (TU Kaiserslautern).

Die Informationen und Materialien zur BTW 2019 stehen über die Web-Seiten der Tagung unter <https://www.btw2019.de> zur Verfügung. Die Organisation der BTW-Tagung nebst allen angeschlossenen Veranstaltungen ist nicht ohne die Unterstützung vieler Partner möglich. Diese sind auf den folgenden Seiten aufgeführt. Zu ihnen zählen insbesondere alle Sponsoren, als Ko-Veranstalter die Universität Rostock und als Unterstützer das Steinbeis-Transferzentrum DBIS an der Universität Rostock. Organisiert wurde die BTW 2019 vom Lehrstuhl Datenbank- und Informationssysteme der Universität Rostock. Insbesondere aber gilt ein Dank der GI-Geschäftsstelle für die finanzielle Abwicklung der Tagung.

Vielen Dank an alle Beteiligten!

Rostock, im Januar 2019

Die Leiter des Programm-Komitees Wissenschaft:

Torsten Grust, Universität Tübingen

Felix Naumann, Hasso-Plattner-Institut Potsdam

Die Leiter des Programm-Komitees Industrie:

Alexander Böhm, SAP Walldorf

Wolfgang Lehner, TU Dresden

Die Leiter des Demonstrationsprogramm-Komitees:

Jens Teubner, Universität Dortmund

Meike Klettke, Universität Rostock

Die Leiter des Dissertationspreis-Komitees des GI-Fachbereichs DBIS:

Theo Härder, TU Kaiserslautern

Erhard Rahm, Universität Leipzig

Der Leiter des Organisationskomitees:

Holger Meyer, Universität Rostock

Die Tagungsleiter der BTW 2019:

Andreas Heuer, Universität Rostock

Meike Klettke, Universität Rostock

Tagungsleitung

Andreas Heuer, Universität Rostock

Meike Klettke, Universität Rostock

Organisationskomitee

Vorsitz: Holger Meyer, Universität Rostock

Tanja Auge, Universität Rostock

Hannes Grunert, Universität Rostock

Andreas Heuer, Universität Rostock

Sigrun Hoffmann, Universität Rostock

Meike Klettke, Universität Rostock

Dennis Marten, Universität Rostock

Mark Lukas Möller, Universität Rostock

Donald Reeb, Universität Rostock

Wissenschaftliches Programm

Vorsitz: Torsten Grust, Universität Tübingen

Felix Naumann, Hasso-Plattner-Institut Potsdam

Ziawasch Abedjan, TU Berlin

Carsten Binnig, TU Darmstadt

Stefan Brass, Universität Halle-Wittenberg

Stefan Conrad, Heinrich-Heine-Universität Düsseldorf

Stefan Deßloch, TU Kaiserslautern

Jens Dittrich, Universität des Saarlandes

Markus Endres, Universität Augsburg

Peter M. Fischer, Universität Augsburg

Rainer Gemulla, Universität Mannheim

Michael Gertz, Universität Heidelberg

Goetz Graefe, Google

Michael Grossniklaus, Universität Konstanz

Katja Hose, Aalborg University

Zbigniew Jerzak, SAP SE

Alfons Kemper, TU München

Georg Lausen, Universität Freiburg

Alexander Löser, Beuth Hochschule für Technik Berlin

Stefan Manegold, CWI Amsterdam

Sebastian Maneth, Universität Bremen
Norman May, SAP SE
Sebastian Michel, Universität Kaiserslautern
Bernhard Mitschang, Universität Stuttgart
Wolfgang Nejdil, L3S und Universität Hannover
Thomas Neumann, TU München
Daniela Nicklas, Universität Bamberg
Tilman Rabl, TU Berlin
Erhard Rahm, Universität Leipzig
Stefanie Rinderle-Ma, Universität Wien
Norbert Ritter, Universität Hamburg
Gunter Saake, Universität Magdeburg
Kai-Uwe Sattler, TU Ilmenau
Stefanie Scherzinger, OTH Regensburg
Ingo Schmitt, BTU Cottbus-Senftenberg
Marc H. Scholl, Universität Konstanz
Felix Martin Schuhknecht, Universität des Saarlandes
Holger Schwarz, Universität Stuttgart
Bernhard Seeger, Universität Marburg
Thomas Seidl, LMU München
Günther Specht, Universität Innsbruck
Uta Störl, Hochschule Darmstadt
Martin Theobald, Université du Luxembourg
Gottfried Vossen, ERCIS Muenster
Lena Wiese, Universität Münster

Dissertationspreise

Vorsitz: Theo Härder, TU Kaiserslautern
Erhard Rahm, Universität Leipzig

Andreas Heuer, Universität Rostock
Wolfgang Lehner, TU Dresden
Bernhard Mitschang, Universität Stuttgart
Gunter Saake, Universität Magdeburg
Kai-Uwe Sattler, TU Ilmenau
Thomas Seidl, LMU München
Gerhard Weikum, Universität des Saarlandes

Industrieprogramm

Vorsitz: Alexander Böhm, SAP SE
Wolfgang Lehner, TU Dresden

Roman Dementiev, Intel GmbH
Pit Fender, Oracle Labs
Isabelle Hang, Universität Bremen
Fisnik Kastrati, Exasol AG
Evelina Koycheva, BASF Schwarzheide GmbH
Kim-Thomas Rehmann, SAP SE
Thomas Ruf, Kynetec
Harald Schöning, Software AG

Demoprogramm

Vorsitz: Jens Teubner, TU Dortmund
Meike Klettke, Universität Rostock

Ulf Leser, Humboldt-Universität zu Berlin
Sebastian Breß, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH (DFKI)
Stefanie Scherzinger, OTH Regensburg
Ilija Petrov, Universität Reutlingen
Stefan Manegold, CWI Amsterdam
Viktor Leis, TU München
Peter Fischer, Universität Augsburg
Erich Schubert, Technische Universität Dortmund
Günther Specht, Universität Innsbruck

Externe Gutachter

Aoubakr Benabbas, Otto-Friedrich-Universität Bamberg
Michael Brendle, Universität Konstanz
Golnaz Elmamooz, Otto-Friedrich-Universität Bamberg
Michael Färber, Universität Innsbruck
Nikolaus Glombiewski, Philipps Universität Marburg
Nasr Kasrin, Otto-Friedrich-Universität Bamberg
Jens Lechtenböcker, Universität Münster
Joris Nix, Universität des Saarlandes
Ankur Sharma, Universität des Saarlandes

Inhaltsverzeichnis

Eingeladene Vorträge

Stefanie Rinderle-Ma

From LEGO to the Shopfloor: Driving Digitalization Through Process Technology 25

Ihab Ilyas

Building Scalable Machine Learning Solutions for Data Cleaning 27

Frank Renkes, Christian Sommer

Blockchain in the Context of Business Applications and Enterprise Databases 29

Wissenschaftliche Beiträge

High-Performance Queries

Johannes Pietrzyk, Annett Ungethüm, Dirk Habich, Wolfgang Lehner

Fighting the Duplicates in Hashing: Conflict Detection-aware Vectorization of Linear Probing 35

Query Processing and Optimization

Bernhard Radke, Thomas Neumann

LinDP++: Generalizing Linearized DP to Crossproducts and Non-Inner Joins 57

Nikolaus Glombiewski, Bernhard Seeger, Goetz Graefe

Waves of Misery After Index Creation 77

Stefan Klauck, Max Plauth, Sven Knebel, Marius Strobl, Douglas Santry, Lars Eggert <i>Eliminating the Bandwidth Bottleneck of Central Query Dispatching Through TCP Connection Hand-Over</i>	97
Maximilian Schüle, Linnea Passing, Alfons Kemper, Thomas Neumann <i>Ja-(zu-)SQL: Evaluation einer SQL-Skriptsprache für Hauptspeicherdatenbanksysteme</i>	107
Adrian Bartnik, Bonaventura Del Monte, Tilmann Rabl, Volker Markl <i>On-the-fly Reconfiguration of Query Plans for Stateful Stream Processing Engines</i>	127
 Text	
Cornelia Kiefer, Peter Reimann, Bernhard Mitschang <i>A Hybrid Information Extraction Approach Exploiting Structured Data Within a Text Mining Process</i>	149
Christoph Lofi, Manuel Valle Torre, Mengmeng Ye <i>Perceptual Relational Attributes: Navigating and Discovering Shared Perspectives from User-Generated Reviews</i>	169
 Graphs	
Matthias Kricke, Eric Peukert, Erhard Rahm <i>Graph Data Transformations in Gradoop</i>	193
 Similarity	
Jan Martin Keil <i>Efficient Bounded Jaro-Winkler Similarity Based Search</i>	205

Xiao Chen, Gabriel Campero Durand, Roman Zoun, David Broneske, Yang Li, Gunter Saake
The Best of Both Worlds: Combining Hand-Tuned and Word-Embedding-Based Similarity Measures for Entity Resolution 215

Michael Günther, Maik Thiele, Wolfgang Lehner
Fast Approximated Nearest Neighbor Joins For Relational Database Systems 225

Machine Learning

Maximilian Schüle, Frédéric Simonis, Thomas Heyenbrock, Alfons Kemper, Stephan Günemann, Thomas Neumann
In-Database Machine Learning: Gradient Descent and Tensor Algebra for Main Memory Database Systems 247

Matthias Boehm, Alexandre Evfimievski, Berthold Reinwald
Efficient Data-Parallel Cumulative Aggregates for Large-Scale Machine Learning 267

Challenges in Data Processing

Nadine Steinmetz, Ann-Katrin Arning, Kai-Uwe Sattler
From Natural Language Questions to SPARQL Queries: A Pattern-based Approach 289

Industriebeiträge

High-Performance Queries

Adrian Vogelsgesang, Tobias Muehlbauer, Viktor Leis, Thomas Neumann, Alfons Kemper
Domain Query Optimization: Adapting the General-Purpose Database System Hyper for Tableau Workloads 313

Text

Mark Reinke, André Kischkel, Volker Jahns, Uwe Crenze, Olga Beltcheva

Einsatz kognitiver Verfahren am Deutschen Patent- und Markenamt 357

Database Systems in the Cloud

Tim Waizenegger, Thomas Lumpp

IBM Cloud Databases: Turning Open Source Databases Into Cloud Services 359

Uwe Jugel, Juan De Dios Santos, Evelyn Trautmann, Diogo Behrens

Fighting Spam in Dating Apps 361

Graphs

David Allen, Amy Hodler, Michael Hunger, Martin Knobloch, William Lyon, Mark Needham, Hannes Voigt

Understanding Trolls with Efficient Analytics of Large Graphs in Neo4j 377

Query Processing and Optimization

Yvonne Hegenbarth, Gerald Ristow

Konzept und Implementierung eines echtzeitfähigen Model Management Systems — am Beispiel zur Überwachung von Lastprognosen für den Intraday Stromhandel 399

Machine Learning

Martin Oberhofer, Lars Bremer, Mariya Chkalova

Machine Learning Applied to the Clerical Task Management Problem in Master Data Management Systems 419

Challenges in Data Processing

Christoph Gröger, Eva Hoos

Ganzheitliches Metadatenmanagement im Data Lake: Anforderungen, IT-Werkzeuge und Herausforderungen in der Praxis 435

Knut Stolze, Felix Beier, Jens Müller

Partial Reload of Incrementally Updated Tables in Analytic Database Accelerators 453

Dissertationspreise

Thorsten Papenbrock

Data Profiling — Effiziente Entdeckung Struktureller Abhängigkeiten . . . 467

Ismail Oukid

Architectural Principles for Database Systems on Storage-Class Memory . 477

Caetano Sauer

Modern techniques for transaction-oriented database recovery 487

Demonstrationen

Johannes Kastner, Nemanja Ranitovic, Markus Endres

The Borda Social Choice Movie Recommender 499

Johannes Kessler, Michael Tschuggnall, Günther Specht

RelaX: A Webbased Execution and Learning Tool for Relational Algebra . 503

Roman Zoun, Kay Schallert, David Broneske, Wolfram Fenske, Marcus Pinnecke, Robert Heyer, Sven Brehmer, Dirk Benndorf, Gunter Saake

MSDataStream — Connecting a Bruker Mass Spectrometer to the Internet 507

Daniel O’Grady

Database-Supported Video Game Engines: Data-Driven Map Generation 511

Marcus Pinnecke, Gabriel Durand Campero, Roman Zoun, David Broneske, Gunter Saake <i>PROTOBASE: It's About Time for Backend/Database Co-Design</i>	515
Christoph Stach, Corinna Giebler, Simone Schmidt <i>Zuverlässige Verspätungsvorhersagen mithilfe von TAROT</i>	519
Wolfram Wingerath, Felix Gessert, Norbert Ritter <i>Twoogle: Searching Twitter With MongoDB Queries</i>	523
Michael Günther, Maik Thiele, Wolfgang Lehner <i>Explore FREDDY: Fast Word Embeddings in Database Systems</i>	529
Jurica Seva, Julian Goetze, Mario Lamping, Damian Tobias Rieke, Reinhold Schaefer, Ulf Leser <i>Information Retrieval for Precision Oncology</i>	533
Alexander Krause, Annett Ungethüm, Thomas Kissinger, Dirk Habich, Wolfgang Lehner <i>NeMeSys — Energy Adaptive Graph Pattern Matching on NUMA-based Multiprocessor Systems</i>	537
Thomas Lindemann, Patrick Brinkmann, Fadi Dalbah, Christian Hakert, Philipp-Jan Honysz, Daniel Matuszczyk, Nikolas Müller, Alexander Schmulbach, Stefan Petyov Todorinski, Oliver Tüselmann, Shimon Wonsak, Jens Teubner <i>MAGPIE: A Scalable Data Storage System for Efficient High Volume Data Queries</i>	543
Daniyal Kazempour, Maksim Kazakov, Peer Kröger, Thomas Seidl <i>DICE: Density-based Interactive Clustering and Exploration</i>	547
Stefan Hagedorn, Oliver Birli, Kai-Uwe Sattler <i>Processing Large Raster and Vector Data in Apache Spark</i>	551
Mark Lukas Möller, Nicolas Berton, Meike Klettke, Stefanie Scherzinger, Uta Störl <i>jHound: Large-Scale Profiling of Open JSON Data</i>	555
M. Ali Rostami, Eric Peukert, Moritz Wilke, Erhard Rahm <i>Big graph analysis by visually created workflows</i>	559

Nadine Steinmetz, Ann-Katrin Arning, Kai-Uwe Sattler

When is Harry Potters birthday? — Question Answering on Linked Data . 565

Autorenverzeichnis

Eingeladene Vorträge

From LEGO to the Shopfloor: Driving Digitalization Through Process Technology

Stefanie Rinderle-Ma¹

Abstract

Processes constitute the major vehicle to support companies to move towards digital business (models). One major application area is smart manufacturing, also referred to as Industrie 4.0. Here, processes connect and integrate machines, actors, sensors, information systems, and business partners, collecting, processing, and exchanging production-relevant data. This yields manifold benefits such as optimized processing and integrated data analysis. Designing and implementing such solutions touches and raises many research challenges, including the flexible and robust implementation of distributed process networks and the application and extension of process-oriented data analysis methods. We illustrate these challenges by our own journey from a LEGO-based Industrie 4.0 lab setting to the centurio.work manufacturing orchestration suite.

Author

Univ.-Prof. Dr. Stefanie Rinderle-Ma leads the Research Group Workflow Systems and Technology at the Faculty of Computer Science, University of Vienna, Austria. She received her PhD and habilitation degree in Computer Science from Ulm University, Germany where she also worked as research assistant at the Department of Databases and Information Systems. Stefanie's main research interests comprise distributed and flexible process technology, process and data science, as well as compliance and security in process-aware information systems.

¹ Research Group Workflow Systems and Technology, University of Vienna, Austria, stefanie.rinderle-ma@univie.ac.at

Building Scalable Machine Learning Solutions for Data Cleaning

Ihab Ilyas¹

Abstract

Machine learning tools promise to help solve data curation problems. While the principles are well understood, the engineering details in configuring and deploying ML techniques are the biggest hurdle. In this talk I discuss why leveraging data semantics and domain-specific knowledge is key in delivering the optimizations necessary for truly scalable ML curation solutions. The talk focuses on two main problems: (1) entity consolidation, which is arguably the most difficult data curation challenge because it is notoriously complex and hard to scale; and (2) using probabilistic inference to suggest data repair for identified errors and anomalies using our new system called HoloClean. Both problems have been challenging researchers and practitioners for decades due to the fundamentally combinatorial explosion in the space of solutions and the lack of ground truth. There's a large body of work on this problem by both academia and industry. Techniques have included human curation, rules-based systems, and automatic discovery of clusters using predefined thresholds on record similarity. Unfortunately, none of these techniques alone has been able to provide sufficient accuracy and scalability. The talk aims at providing deeper insight into the entity consolidation and data repair problems and discusses how machine learning, human expertise, and problem semantics collectively can deliver a scalable, high-accuracy solution.

Author

Ihab Ilyas is a professor in the Cheriton School of Computer Science and the NSERC-Thomson Reuters Research Chair on data quality at the University of Waterloo. His main research focuses on the areas of big data and database systems, with special interest in data quality and integration, managing uncertain data, rank-aware query processing, and information extraction. Ihab is also a co-founder of Tamr, a startup focusing on large-scale data integration and cleaning. He is a recipient of the Ontario Early Researcher Award (2009), a Cheriton Faculty Fellowship (2013), an NSERC

¹ Cheriton School of Computer Science, University of Waterloo, Canada, ilyas@uwaterloo.ca

Discovery Accelerator Award (2014), and a Google Faculty Award (2014), and he is an ACM Distinguished Scientist. Ihab is an elected member of the VLDB Endowment board of trustees, elected SIGMOD vice , and an associate editor of the ACM Transactions of Database Systems (TODS). He holds a PhD in computer science from Purdue University, West Lafayette.

Blockchain in the Context of Business Applications and Enterprise Databases

Frank Renkes¹, Christian Sommer¹

Abstract

Blockchain seems to be the future of all cross-company business applications. Similar to the adoption of machine learning into all novel and existing business applications and processes we can see the same trend for blockchain. Nearly every application tries to leverage blockchain technology to improve the application related process chains. Is this just a hype or is blockchain really the solution to all problems, in which applications rely on an intelligent and secure data distribution / sharing? What are the most relevant qualities of blockchain needed in modern business applications and which role can a traditional database play in this? Wouldn't be an integration of some of the qualities into traditional databases a better approach to build the so called 'distributed business applications'? What is the relationship and overlap between core blockchain and core database concepts like (redo) logging, security features like auditing and encryption, distributed (query) processing, as well as stored procedures/smart contracts?

This talk discusses how blockchain can be integrated into existing business applications and processes, what the biggest challenges are and which role a traditional database can play in this context.

Authors

Frank Renkes works as a Chief Architects in the HANA platform development. His focus is on enterprise architecture in the context of customers projects using HANA as the data management layer and on integrating novel trends and technologies into the HANA platform.

Christian Sommer works at the SAP Innovation Center Network. As a Senior Development Manager, he is responsible for the incubation of SAP Cloud Platform Blockchain and its adoption in business applications.

¹ SAP Walldorf, Germany

Wissenschaftliche Beiträge

High-Performance Queries

Fighting the Duplicates in Hashing: Conflict Detection-aware Vectorization of Linear Probing

Johannes Pietrzyk,¹ Annett Ungethüm,¹ Dirk Habich,¹ Wolfgang Lehner¹

Abstract: Hash tables are a core data structure in database systems, because they are fundamental for many database operators like hash-based join and aggregation. In recent years, the efficient vectorized implementation using SIMD (Single Instruction Multiple Data) instructions has attracted a lot of attention. Generally, all hash table implementations need to address what happens when collisions occur. In order to do that, the collisions have to be detected first. There are two types of collisions: (i) key duplicates and (ii) hash value duplicates. The second type is more complicated than the first type. In this paper, we investigate linear probing as a heavily applied hash table implementation and we present an extension of the state-of-the-art vectorized implementation with a hardware-supported duplicate or collision detection. For that, we use novel SIMD instructions which have been introduced with Intel's SIMD instruction set extension AVX-512. As we are going to show, our approach outperforms the state-of-the-art vectorized version for the key handling, but introduces novel challenges for the value handling. We conclude the paper with some ideas how to tackle that challenge.

Keywords: Hashing; Linear Probing; Vectorization; Conflict Detection

1 Introduction

The key objective of database systems is to reliably manage data, where high query throughput and low query latency are still core challenges [Ab16, BFT16, Do13, Ou17]. To satisfy these requirements, database systems constantly adapt to novel hardware features [BKM08]. In the recent past, we have seen numerous hardware advances, in particular with respect to *memory*, *processing elements*, and *interconnects* having a huge impact on the design of database systems [Le17, LUH18, OL18]. For example, with growing capacities of main memory, efficient analytical in-memory data processing becomes viable and is now state-of-the-art [BKM08, Fa17] on the one hand. On the other hand, vectorization is a common approach to improve the processing performance of CPUs by parallelizing computations over vector registers. This vectorization is done using SIMD extensions (SIMD stands for Single Instruction Multiple Data) such as Intel's SSE (Streaming SIMD Extensions) or AVX (Advanced Vector Extensions) and have been available in modern processors for several years. SIMD instructions apply one operation to multiple elements of so-called vector registers at once. Thus, the efficient *vectorized* implementation of database operations using SIMD instructions has attracted a lot of attention in recent years [La16, LB15, PRR15, ZR02].

¹ Technische Universität Dresden, Institut für Systems Architecture, Dresden Database Systems Group, Nöthnitzer Straße 46, 01187 Dresden, firstname.lastname@tu-dresden.de

In the past years, hardware vendors have regularly introduced new SIMD instruction set extensions operating on ever wider registers. For instance, Intel’s Advanced Vector Extensions (AVX) operates on vector registers of size 256 bits, while Intel’s newest extension set AVX-512 uses now 512-bit vector registers. The wider the vector registers, the more data elements can be stored and processed in one vector. For example, Intel’s SSE 128-bit vector register can store four 32-bit data elements, AVX 256-bit vector can store eight ($2x$), and AVX-512 512-bit vector can store 16 ($4x$) of such data elements. Consequently, the SIMD instructions operating on these wider vector registers can also process $2x$ respectively $4x$ the number of data elements in one instruction, which promises significant speedups. In [Ha18], we investigated the influence of the wider vector registers on data compression. As we have shown, the achieved speedups of wider vector registers are sub-optimal in most cases, since the algorithms quickly become memory-bound when computations are accelerated through wider vector registers processing more data elements at once. Thus, an open challenge in this domain is the development of appropriate approaches which exploit the capabilities of newer SIMD extensions to the maximum extent.

To tackle that challenge for lightweight data compression algorithms, we presented a novel hardware-oriented approach in [Un18]. The starting point of this novel approach was, that in addition to an increased vector width of 512-bit, AVX-512 also offers a variety of new instructions. One of the new instruction feature sets is called Conflict Detection (AVX-512CD) which allows the vectorization of loops with possible address conflicts. Some key features of AVX-512CD are (i) the generation of conflict free subsets, i.e. subsets which contain no equal elements, and (ii) the count of leading zero bits of the elements in a vector. In [Un18], we described the application of these CD instructions for run-length encoding. In particular, we have clearly shown that the CD-based implementation is up to 3.2 times faster for sequences of integers with short run lengths.

Our Contribution: Based on these experiences, we introduce our approach for the application of the Conflict Detection instruction to hashing, which is completely different from the data compression domain, in this paper. Generally, hashing is a core primitive for many database operators such as hash-based joins and aggregations [PRR15, RAD15]. The main task of hashing is to distribute entries (key/values) across an array of buckets (hash table). Given a key, the algorithm computes a bucket that suggests where the entry can be found. All hash table implementations need to address what happens when collisions occur. In order to do that, the collisions have to be detected, which sounds like a perfect match to Conflict Detection. In particular, we evaluate linear probing as a heavily applied hash table implementation [PRR15, RAD15]. Based on that, our main findings can be summarized as follows:

1. Conflict Detection can be used to speedup linear probing, whereby the specific SIMD instructions can be utilized at different positions within the hashing implementation. On the one hand, duplicate keys within one vector register can be detected to reduce unnecessary work. On the other hand, duplicate hash values are extractable within one vector register to reduce expensive Gather and Scatter operations.

2. However, the application of Conflict Detection to hashing comes at a price of difficulty and IO-costly value handling approaches. We will elaborate that aspect in our evaluation in more detail.

Outline of the Paper: The remainder of this paper is organized as follows: In Section 2, we present all essential background information starting by a short description of linear probing followed by a detailed explanation of the state-of-the-art vectorized implementation. This section closes with a short description of new and non-standard vector instructions which have been introduced with AVX-512. Based on that, we introduce our novel vectorized linear probing implementation in Section 3. Then, we present selective results of our exhaustive evaluation on two different hardware systems in Section 4. Finally, we close with related work in Section 5 and a summary including future work in Section 6.

2 Background

Basically, hash tables are a core and a heavily-used data structure in in-memory database systems, because they are required to efficiently execute join and aggregation operations [PRR15]. For example, in a hash join, a hash table of the smaller input relation is built, in which the hash table entries consist of the join attribute as key and the rest as payload [PRR15]. Once the hash table is built, the larger input relation is scanned and join partners are looked up using the hash table. The first phase in this hash join is usually called *build phase*, while the second is called *probe phase*.

In these hash tables, hash functions play an important role [PRR15, RAD15]. Specifically, a hash function is used to map keys to hash table positions allowing to quickly locate the keys in constant time. However, the domain of a hash function (the set of possible keys) is larger than its range (the number of hash table buckets), and so it will map several keys to the same bucket which could result in collisions. That means, all hash table implementations need to address what happens when collisions occur. A common collision strategy is *open addressing*, which allows keys to *leak out* from their preferred bucket and spill over into another bucket [Be18, RAD15].

Based on that, this background section is organized as follows: In Section 2.1, we briefly describe the general idea of linear probing. Then, we introduce the state-of-the-vectorized implementation of linear probing as introduced in [PRR15] in Section 2.2. We close this background section with a description of new and non-standard vector instructions which have been introduced with Intel’s latest SIMD extension AVX-512 in Section 2.3.

2.1 Linear Probing

Linear Probing (LP) is the simplest strategy for collision handling in open-addressing. Generally, the hash table structure for open addressing is an array T whose bucket $T[i]$ stores

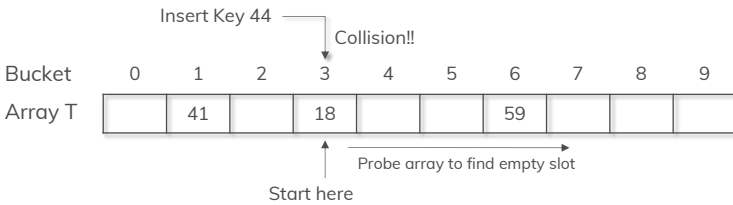


Fig. 1: Illustration of Linear Probing with Collision Example.

a single key as depicted in Fig. 1. Then, an arbitrary hash function h is used to map each key into a bucket of T where the key should be stored. A hash collision occurs when the hash function maps a key into a bucket that is already occupied by a different key. LP resolves this collision by placing the new key into the closest following empty bucket. That means, for a given key x , the buckets of T are examined, beginning with the bucket at position $h(x)$ (where h is the hash function) and continuing to the adjacent buckets $h(x)+1$, $h(x)+2$, ..., until finding either an empty bucket or a bucket whose stored key equals x . An example is given in Fig. 1. Here, key 44 shall be inserted in array bucket 3 leading to a collision. From this bucket, the next free bucket will be used to store the key 44 in linear probing. In our example, the next free bucket would be 4, which is then the storage bucket for this key. This linear scan procedure (probe) is always executed for lookup as well as insertion.

LP has two excellent advantages: (i) low code complexity based on the simplicity of the approach and (ii) very good cache efficiency due to the linear scan [RAD15]. Based on that, we decided to use LP for our case study on applying Conflict Detection to hashing.

2.2 State-of-the-Art Vectorized Implementation of Linear Probing

To speed up the overall performance of hash tables, vectorized hash tables use a SIMD register of width n to process multiple keys k_i at once. Based on that idea, Polychroniou et al. [PRR15] presented a vectorized version of LP. We will denote this SIMD LP implementation as basic or state-of-the-art variant, respectively, thereby this approach consists of several phases as illustrated in Fig. 2. The phases are repeated multiple times until all keys are finally processed. The phases are:

Load Phase: In this phase several keys k_i are loaded into a SIMD register v in each iteration. We will denote these keys as active keys. In the first iteration, n keys are transferred from memory to the vector register, whereby n is the size of the vector register. In the next iterations, keys that were successfully inserted into the hash table of the previous iterations are replaced by new keys using a selective load. This selective load exchanges vector lanes by loading contiguous keys from unaligned memory based on a k -bit mask.

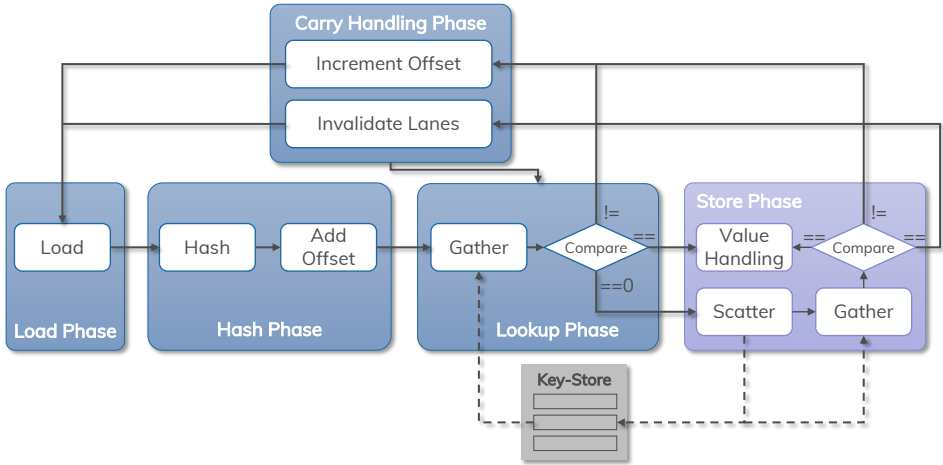


Fig. 2: Control Flow of State-of-the-Art Vectorized Implementation of Linear Probing.

Hash Phase: For each active key k_i within the SIMD register, the hash table bucket is computed using a hash function h and an offset (initial state equals zero). This requires two vector operations.

Lookup Phase: The determined buckets of the keys are used in a Gather operation to load the current content of the hash table. The loaded keys are stored in a second SIMD register and compared with the active keys, whereby three results are possible:

- (1) Bucket is empty (loaded key is zero): There is currently no key in the hash table bucket stored. Thus, the new key can be stored at that bucket (*Store Phase*) as well as invalidated (*Carry Handling Phase*) and the corresponding value has to be stored.
- (2) Bucket contains the same key (Match): The key is already in the corresponding bucket of the hash table. That means, the key can be invalidated (*Carry Handling Phase*) and only the corresponding value has to be stored.
- (3) Bucket contains different key (Mismatch): The hash table contains already a different key at that bucket. Thus, the new key can neither be stored nor invalidated and has to remain in the vector register (*Carry Handling Phase*).

Store Phase: Active keys with an identified empty bucket have to be stored using a Scatter operation. However, different keys in the vector register can have equal hash values which would lead to a conflict. If different vector lanes are stored to the same hash table bucket, the lane with the highest lane index remains at the specific bucket. To detect that, the stored active keys at the buckets are gathered again and compared with the active keys. If the gathered key equals the scattered key, the key can be invalidated and the value has to be stored (key successfully inserted into the hash table). If the gathered key is not equal to the scattered key, the key has to remain in the vector register (*Carry Handling Phase*).

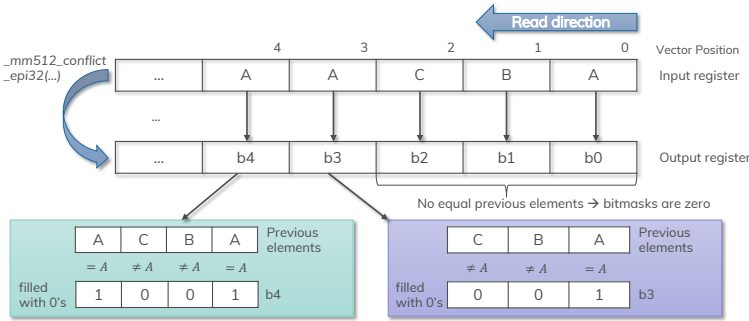


Fig. 3: Example for the `_mm512_conflict_epi32` intrinsic.

Carry Handling Phase: Successfully inserted keys which are marked as invalidated during the whole process can be exchanged in the next iteration. Additionally, the corresponding offsets are set to zero. Lanes which could not be scattered successfully remain in the vector register and corresponding offsets are incremented one by one for the next iteration.

This state-of-the-art vectorization uses standard and well-known SIMD operations like Scatter, Gather, and compare functions.

2.3 Novel SIMD Instructions

The newest version of Intel’s instruction set extension for vectorization is AVX-512. In this extension, the width of vector registers is 512-bit. That means for the state-of-the-art vectorized LP implementation, that 16 keys (each key has a width of 32-bit) can be processed at once in each iteration. Aside from an increased vector width, AVX-512 also offers a variety of new instructions. One of the new instruction feature sets is called *Conflict Detection (AVX-512CD)* allowing the vectorization of loops with possible address conflicts. This instruction feature set is currently supported by Intel Xeon Phi Knights Landing (KNL) as well as on current Xeon processors.

As already presented in [Un18], some core features of AVX-512CD are (i) the generation of conflict free subsets, i.e. subsets which contain no equal elements (no duplicates), and (ii) the count of leading zeros of the elements in a vector. For example, the intrinsic `_mm512_conflict_epi32` creates a vector register containing a conflict free subset of a given source register. An example for this is shown in Fig. 3. In other words and as illustrated in this figure, this intrinsic transforms a vector register with 16 32-bit elements (illustrated by A, B and C) into a new vector register with 16 bitmasks (each represented by 32-bit values). Each bitmask encodes the positions of equal previous elements in the vector. The bitmasks for the first three elements A, B, and C are zero in our example, because there are no equal

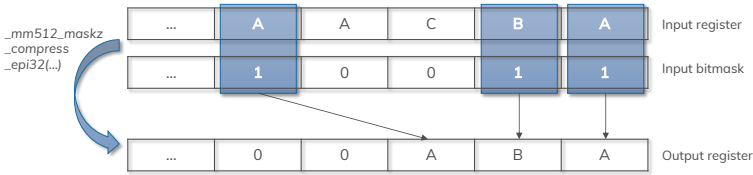


Fig. 4: Example for the `_mm512_maskz_compress_epi32` intrinsic.

previous elements. The `A` element at the third position in the input register is in conflict (equal to) with the element at position 0 in the input register. Thus, the least significant bit of the corresponding bitmask is set to 1, the rest of the bitmask is filled with zeros. The element `A` at position 4 is in conflict with the previous elements at positions 3 and 0 (equal to previous elements). Therefore, the corresponding bits in the bitmask are set to 1, all other bits are zero. A second interesting CD-feature is the intrinsic `_mm512_lzcnt_epi32`, which counts leading zeros. Given a vector of 16 values, this intrinsic counts the number of leading zeros for all values at once and writes the results in a vector register with 16 values.

Another newly introduced functionality is a set of compress instructions, e.g. `_mm512_maskz_compress_epi32`. They are part of the foundation instruction set of AVX512 (AVX-512F). The input of these compress instructions is a vector and a bitmask. Then, the elements in the input vector, which are marked by the bitmask, are stored contiguously in the output vector as depicted in Fig. 4. Using this compress instruction, the result vector contains no reserved space of the unmarked elements in the input register.

3 CD-aware Vectorized Implementation of Linear Probing

The above presented state-of-the-art vectorized implementation of linear probing is well-engineered, but the implementation has a major shortcoming. This shortcoming is related to ever increasing widths of vector registers. On the one hand, with wider vector registers, more keys can be processed simultaneously. For instance, with 128-bit wide vector registers only 4 keys, but now with 512-bit wide vector registers 16 keys are processable simultaneously. On the other hand, with more keys in parallel, the probability of collisions within one vector register increases for two reasons:

1. With more keys in parallel, the probability of duplicate keys within one vector register increases the risk of collisions at the end.
2. With more keys in parallel, the probability that different keys are mapped to a single bucket within one vector register increases the risk of collisions at the end.

Fundamentally, when more hash collisions occur in each iteration, more iterations are needed to process all input data, because more keys have to be moved to the next iteration. At the same time, more iterations also mean that more Gather and Scatter operations are

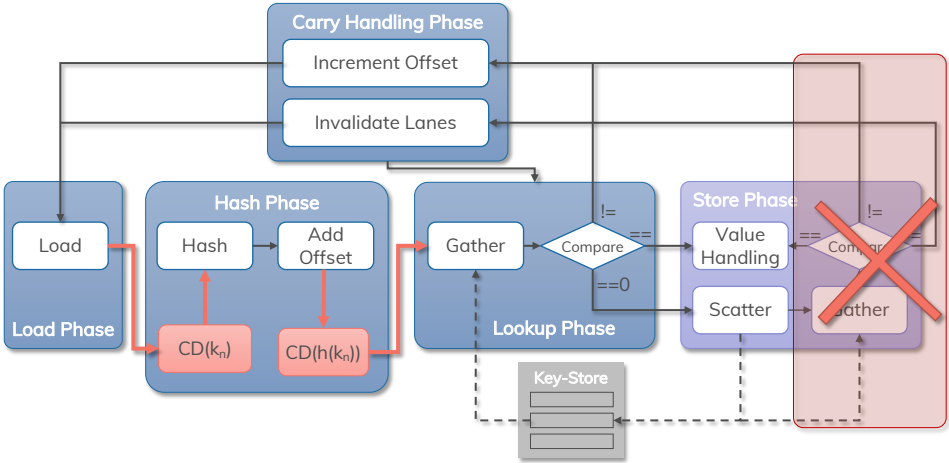


Fig. 5: Control Flow of state-of-the-art Vectorized Linear Probing using Conflict Detection.

performed leading to decreasing performance, finally. Nevertheless, this is a perfect setting for the Conflict Detection capability as described next.

3.1 CD-aware Hash Table Data Structures

Before we introduce our Conflict Detection-aware (CD-aware) vectorized implementation of linear probing in detail, we describe our underlying data structure. The hash table usually has to hold a set of key/value pairs. The keys are inserted into a so-called *key-store* as illustrated in Fig. 2 consisting of a fixed size number of buckets realized by an array. The corresponding values are stored within a separate memory location (value-store). Since a single key can exist multiple times within the given input dataset, the value-store has to hold every value for the corresponding key. Through the assumption that the total number of occurrences of a single key is unknown in advance, the value-store is realized as a fixed sized array of dynamic containers.

3.2 Handling of Bucket Duplicates

To overcome the mentioned shortcoming, we add two Conflict Detection instructions to the Hash Phase as illustrated in Fig. 5. With these instructions, the number of non-sequential memory accesses through Gather and Scatter operations in the subsequent phases are minimized.

The first Conflict Detection instruction $CD(k_n)$ is placed directly after the Load Phase as highlighted in Fig. 5. If the vector register of an iteration contains duplicate keys, we

already know that only one lane has to be used for the further steps. Lanes containing duplicate keys can be automatically invalidated. Since same keys can result in different buckets through offset addition and only the left most lane remain valid, it is feasible to use the compress intrinsic provided by AVX-512F to arrange valid lanes in a contiguous manner. The associated values of duplicate keys have to be preserved until the key can be written to memory. Because the total number of occurrences of a key within a given dataset is not known without further investigations, the values are stored within a temporal dynamic sized buffer. As a consequence, the buffer has to be resized when duplicate keys are detected. When the key is successfully stored into the key-store, the values from the corresponding buffer are appended to the corresponding value-store entry.

After this first `Conflict Detection` instruction, we are sure that the vector register contains only unique keys, whereby already some lanes could be invalidated which limits the exploitation of parallelization in this iteration. However, different keys in the vector register can result in the same buckets within the key-store after the hash phase. There are two possible reasons (i) through a pure hash collision or (ii) through the addition of the offset. This situation also has to be detected and solved. For this detection, we use the second `Conflict Detection` instruction $CD(h(k_n))$ as shown in Fig. 5. Based on that, we know the vector lanes with a conflicting position. But the conflicting lanes cannot be invalidated immediately, because each of these different keys could already be in the key-store. Thus, these keys are used in the next `Lookup Phase`. If an empty bucket is found, the key and its assigned value corresponding to the first occurrence (as a result of `Conflict Detection`) of the specific bucket is transferred into the hash table and the lane is invalidated. The remaining lanes containing conflicting buckets reside in the vector register and are treated in the `Carry Handling` phase.

Based on our procedure, the `Store Phase` can be simplified as depicted in Fig. 5. In this `Store Phase`, we do not have to load the buckets again to detect conflicts, because these conflicts are now determined during the `Hash Phase`. In conclusion, the amount of random access IO-operations can be reduced by using the conflict detection intrinsic. Also duplicate keys are substituted within the next iteration resulting in a higher degree of vector lane utilization. Still non-valid lanes are present within a single iteration step which leads to non-optimal data parallelism.

3.3 Handling of Key Duplicates

To address the non-optimal vector lane utilization through conflicting keys mentioned in section 3.2, the `Load phase` has to be adapted as depicted in Fig. 6. Instead of executing one load operation per iteration, contiguous keys are transferred to a vector register (v_0) until the vector contains only distinct elements. We call this approach `FetchD`. To avoid selective and non-cache friendly loads in our `FetchD` approach, a second full buffer vector register (v_b) is loaded at once. Then, distinct keys are identified using a conflict detection. This buffer is then used to fill up invalidated lanes from the register containing the keys for further

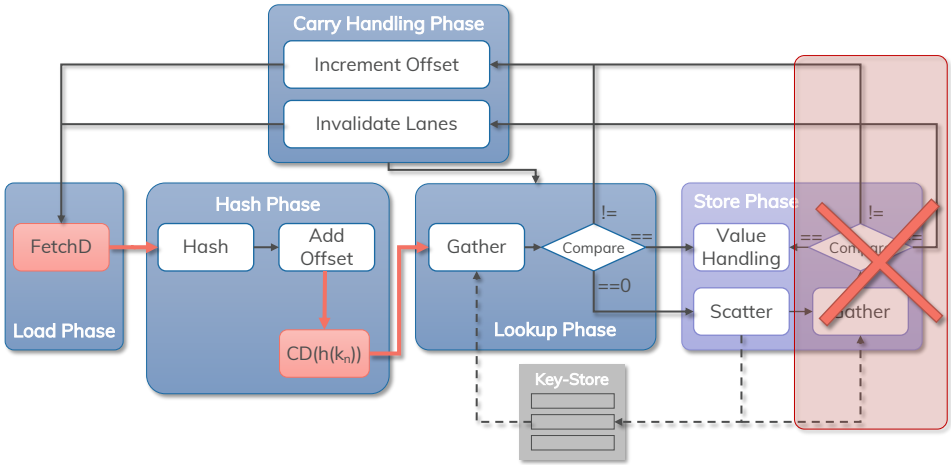


Fig. 6: Control Flow of the CD-aware Vectorized Implementation of Linear Probing.

processing. This is done using `_mm512_maskz_compress_epi32` to organize distinct keys of vector v_b in a contiguous manner. In a second step, invalidated lanes of v_0 are substituted with elements from the v_b using `_mm512_mask_expand_epi32`. These two steps are repeated until v_0 contains only distinct elements or all elements are processed. In regard to the values, a similar approach as described in the section above is used. Nevertheless, the buffer vector register needs its own temporal value buffer which is merged with the value buffer from v_0 corresponding to the replaced key lanes. This leads to additional IO-operations during the *Load Phase*.

In summary, the discussed concepts can be used to utilize vector registers with the maximum amount of parallel computation on the one hand. On the other hand, organizing the values require scalar IO-operations which can be considered to be expensive in comparison to vectorized operations.

4 Experimental Evaluation

Before we summarize the core results of our exhaustive evaluation in Sections 4.1 and 4.2, we briefly describe our overall evaluation setup. Basically, we used two different hardware platforms as depicted in Tab. 1. The first platform is a Xeon Phi™ 7250 Knights Landing (KNL), while the second is a Xeon® Gold 6130 (SKL). Both platforms provide the AVX-512 vector register extension as well as the special instruction set AVX-512CD. The cache sizes of the KNL and SKL are the same for L1 and L2. While every core of the SKL has access to a 22 MB dimensioned L3-Cache, the KNL has no L3-Cache. Instead, a high bandwidth memory which is located on the chip can be used in the KNL.

Name	Xeon Phi™	Xeon® Gold
Processor Model	7250	6130
Base Clock Frequency	1.4 GHz	2.1 GHz
Nodes x Cores x Threads	4 x 17 x 4	4 x 16 x 2
L1 Size	32 KB	32 KB
L2 Size	1 MB	1 MB
L3 Size	-	22 MB
AVX-512	F, PF, ER, CD	F, DQ, CD, BW, VL

Tab. 1: Hardware Platform Specifications.

Furthermore, all vectorized implementations of linear probing were done in C/C++ by ourselves, thereby we distinguish between the following implementations:

Basic: State-of-the-art vectorized implementation of linear probing as introduced in [PRR15] (see also Section 2.2).

CDHashProbe: This is our first proposed CD-aware vectorized implementation with two CD instructions in the *Hash Phase* and only a Scatter operation in the *Store Phase* (see Fig. 5).

FetchD: This is our second proposed CD-aware vectorized implementation with the FetchD approach in the *Load Phase* and a single CD instruction in the *Hash Phase* (see Fig. 6).

FetchD-Basic: This is an enhanced state-of-the-art vectorized implementation using our FetchD instead of a selective load in the *Load Phase* including a Scatter and Gather operation in the *Store Phase*.

In all implementations and experiments, we used the vectorized version of Mumur3 as our main hash function. Since in our work we focus on pushing the achieved data parallelism for the hash build phase through vectorization to a maximum extent, we ran all experiments single threaded. For this, we compiled all implementations with gcc (KNL: version 7.0.1, SKL: version 7.2.0) with the optimization flags `-Ofast -mavx512f -mavx512cd`. We also evaluated the novel Compress instruction of AVX-512. Unfortunately, the impact was very marginal and therefore, we do not include this aspect in our evaluation.

4.1 Evaluation Result for Hashing without Value Handling

In our first series of experiments, we investigated linear probing without value handling which can be used for aggregation, anti-join and exists operators in in-memory database systems. As clearly mentioned previously, there are two possible types of collisions: (i) bucket duplicates and (ii) key duplicates. In the following, we separately evaluate both types.

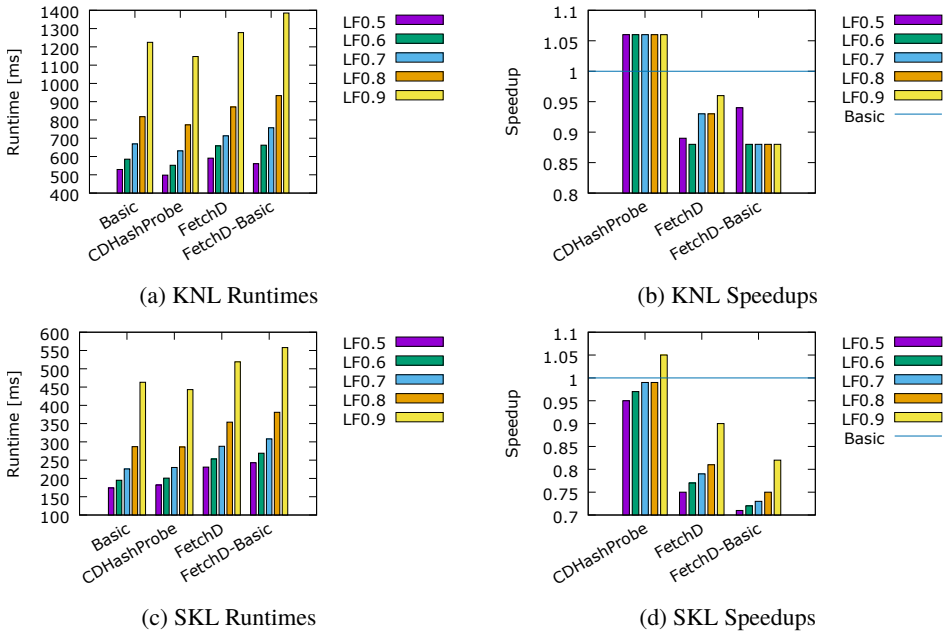


Fig. 7: Runtime Results and Speedups for a Key-Store with 1024KB Size and Input Data Consisting of Unique Keys.

4.1.1 Bucket Duplicates

To evaluate the influence of bucket duplicates on the runtime behaviour of the different linear probing implementation, we generated various data sets containing different numbers of unique keys and varied the load factor of the key store from 0.5 to 0.9 in increments of 0.1. With unique keys, we explicitly restrict ourselves to bucket duplicates and we would expect, that our first CD-aware implementation *CDHashProbe* (see Fig. 5) outperforms *Basic*, while our second CD-aware implementation *FetchD* offers too much overhead in this case. Generally, with higher load factors of the key-store (hashmap), the number of bucket duplicates increases leading to higher runtimes. Fig. 7(a) and (c) show runtimes for KNL as well as SKL on a data set size consisting of unique keys and a key-store size of 1024KB, so that the key-store fits in the L2-cache on both hardware platforms. As we can see, the runtimes for each implementation increases with increasing load factors and SKL is faster than KNL as expected. Fig. 7(b) and (d) depict the speedups of our approaches compared to the *Basic* implementation. On KNL, our *CDHashProbe* approach is slightly faster than *Basic* in all cases. In contrast to that, our *CDHashProbe* only outperforms the *Basic* approach on SKL for high load factors. The reason for that is that the CD instruction is an expensive operation and this is only beneficial when the effort is less than the additional Gather operation in *Basic*. Moreover, *FetchD* and *FetchD-Basic* are slower than the *Basic*

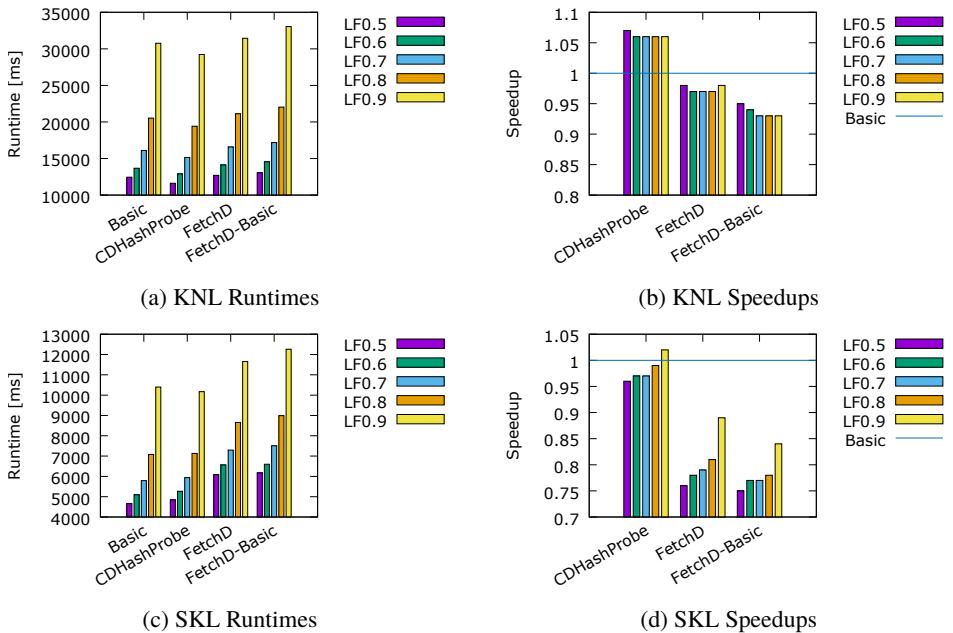


Fig. 8: Runtime Results and Speedups for a Key-Store with 16MB Size and Input Data Consisting of Unique Keys.

implementation in all cases. Of course, since we do not have key duplicates, the treatment of this introduces additional overhead.

Fig. 8 shows the results for data sets with unique keys, a key-store size of 16MB and varying load factors. As we can see, the same observations are also visible for higher amount of data as well. From this set of experiments, we can conclude that our CD-aware implementation for bucket duplicates *CDHashProbe* slightly outperforms the *Basic* approach.

4.1.2 Key Duplicates

In order to evaluate the influence of key duplicates on the runtime behaviour of the different linear probing implementation, we generated various data sets containing different numbers of repeating keys in sequence. Furthermore, we also varied the load factor of the key-store from 0.5 to 0.9 in increments of 0.1. With repeating keys, we explicitly investigate the influence of key duplicates in a best case scenario and we would expect, that our second CD-aware implementation *FetchD* (see Fig. 6) outperforms the other implementations. While Fig. 9 shows the results for KNL, Fig. 10 depicts the results for SKL. In both cases, we

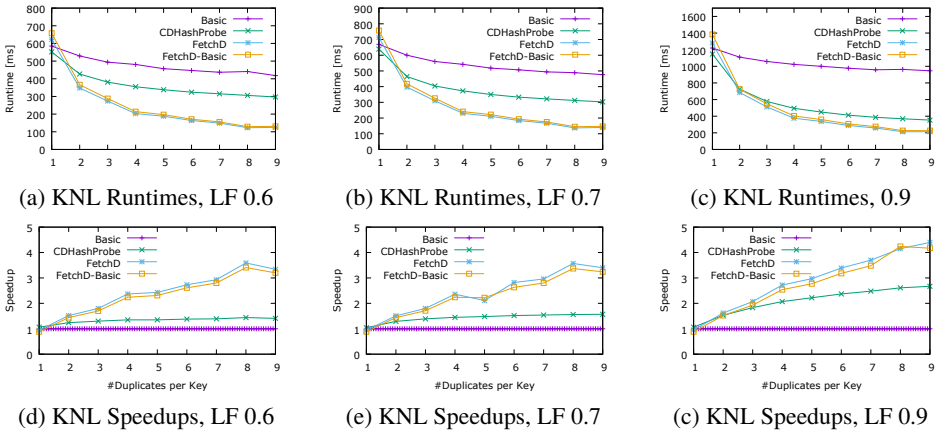


Fig. 9: KNL-Results for Key-Store of Size 1024KB and Varying Number of Repeating Keys.

illustrate the runtimes for load factors of 0.6, 0.7, and 0.9 as well as the speedups compared to the *Basic* implementation.

As we can see, the presented implementation improvements benefit from a higher amount of duplicates within the processed data, while *FetchD* has the highest impact on the performance. With increasing load factors, the speedup of our CD-aware implementations also increases compared to the *Basic* implementation. For example, we can improve the performance for data with a high number of repetitive keys up to a factor of 18 (load factor 0.9; repetition sequence length of 100) on the KNL and up to factor a factor of 10 on the SKL. Moreover, we observed better runtimes of all investigated scenarios (data size, number of duplicate key, load factors) when the underlying key-store is quite small, so it can fit into small levels of cache, but this is already well-known.

4.1.3 Intermin Conclusion

As already shown by [PRR15], the load factor should not exceed 0.6 with regard to memory consumption and total execution time. In our previous presented evaluation results, we always included this specific load factor for a key-store size of 1MB in our considerations, so that the key-store perfectly fits in the L2-Cache of our hardware platforms. From these evaluations, we are able to conclude the following two aspects:

1. If the input data only consists of unique keys, our first CD-aware implementation *CDHashProbe* performs slightly better than the *Basic* implementation.
2. However, already with a small amount of duplicate keys in the input data, our second CD-aware improvement pays more off.

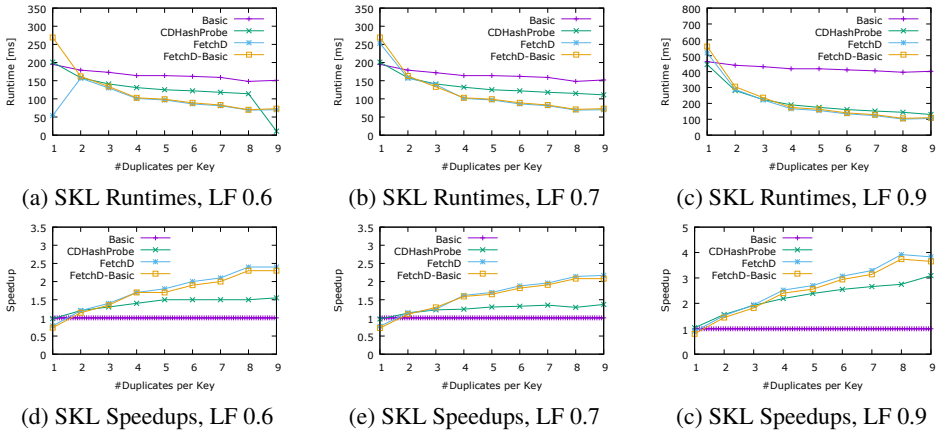


Fig. 10: SKL-Results for Key-Store of Size 1024KB and Varying Number of Repeating Keys.

Thus, our experiments show that using new instructions like `Conflict Detection` instead of random access IO operations can improve the total performance if actual conflicts occur. Furthermore, the presented approach *FetchD* which needs additional instructions and branches amortizes as soon as duplicate keys are within the range of a vector register through a cache friendly access pattern and a high degree of data parallelism.

To conclude, the presented CD-aware optimization’s for linear probing can improve the total performance if the processed data contains duplicate keys or duplicate buckets. The influence of `Conflict Detection` as well as *FetchD* grows with higher load factors and the amount of duplicates. This arises from the fact that without value handling, repeating keys within vector registers are redundant and can be discarded. However if values have to be treated, the values of repetitive keys have to be handled.

4.2 Evaluation Results for Hashing with Value Handling

To precisely evaluate our proposed value handling approach for the CD-aware implementations, we repeated all our previously introduced experiments with enabled value handling. In particular, the value handling is important in order to execute hash joins. Fig. 11 shows the results on the KNL as well as on the SKL hardware platform. In the depicted experiment, we used a key-store of size 1MB, thereby we only compare the *Basic* with the *FetchD* implementation. Through the need of dynamic temporal buffers, a growing amount of memory re-allocations and copying for all our CD-aware implementations for value handling, the performance of our *FetchD* is lower than the *Basic* one. As we can see, the negative impact on the performance gets slightly better with a growing number of duplicates and a higher load factor. Nevertheless, a more sophisticated value handling approach for *FetchD* has to be found to be competitive or even better than the *Basic* implementation.

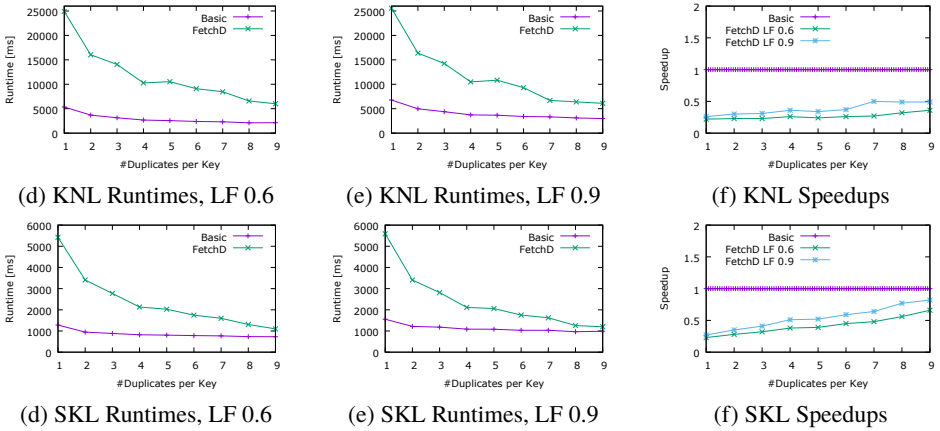


Fig. 11: Value Handling Evaluation Results with a Key-Store of Size 1MB.

5 Related Work

Fundamentally, related work in this domain is manifold, because the efficient utilization of SIMD (Single Instruction Multiple Data) instructions in database systems is a very active research field [Ha18, La16, LB15, PRR15, SWL11, Un18, ZR02].

For example, SIMD instructions are frequently applied in lightweight data compression algorithms [Da17, LB15]. In this specific domain, null suppression (NS) is the most studied lightweight compression approach, whereby the basic idea is the omission of leading zeros in the bit representation of integers [LB15, SGL10]. There are different techniques addressing the efficient implementation using SIMD instructions [Da17, LB15, SGL10]. However, most of the vectorized implementations of lightweight data compression algorithms have been developed for a fixed vector width of 128 bits (corresponding to Intel’s SIMD extension SSE). In [Ha18], we systematically investigated the impact of different SIMD instruction set extensions with vector sizes of 128-, 256-, and 512-bits on the behavior of lightweight data compression algorithms. To obtain implementations for wider vector sizes (AVX2 and AVX-512), the 128-bit implementation can be used as foundation. In a straightforward transformation, the 128-bit SIMD operations can be substituted by the corresponding operations for 256 or 512-bit vectors. As we have shown, this is possible in almost all cases, since many instructions offered by SSE are also offered by AVX2 and AVX-512 on wider vectors. Fundamentally, two effects are observable: (i) NS algorithms working on wider vector registers are more vulnerable to outliers in the data, which can affect both, the compression ratio as well as the performance negatively and (ii) the speed ups are generally sub-optimal in most cases, since the algorithms quickly become memory-bound when the computations are accelerated through wider vector registers processing more data elements at once. To overcome that, novel approaches are necessary. In [Un18], we presented a novel approach for RLE encoding using Conflict Detection. Aside from our

work, [La18] introduced efficient refill algorithms for vector registers by using the latest SIMD instruction set, AVX-512. On the other hand, SIMD instructions are also used in other database operations like scans [LP13], aggregations [ZR02], hashing [PRR15, RAD15] or joins [Ba13]. To best of our knowledge, none of these approaches uses AVX-512 CD, although the operations could benefit from CD.

From a hashing perspective, the papers [PRR15], [RAD15] and [Be18] are highly relevant. The state-of-the-art vectorized implementation of linear probing is presented in [PRR15] as described in Section 2. Richter et al. [RAD15] exhaustively studied a variety of common hash table implementations—including linear probing—in a five-dimensional requirements space: (i) data-distribution, (ii) load factor, (iii) dataset size, (iv) read/write-ratio, and (v) un/successful-ratio. As they have shown, there exists no single best-performing hash table implementation and each hash table implementation has its own application area. In [Be18], the authors translated the state-of-the-art vectorized implementation of linear probing to OpenCL with the aim to reduce code complexity and to ensure portability. For that, they realized essential primitives like Gather, Scatter, Selective Load and Selective Store in OpenCL. It would be interesting to see how the translation of the Conflict Detection would look like.

6 Conclusion and Future Work

Hash tables are a core data structure in in-memory database systems, because they are fundamental for many database operators like hash-based join and aggregation. In recent years, the efficient vectorized implementation using SIMD (Single Instruction Multiple Data) instructions has attracted a lot of attention. Generally, all hash table implementations need to address what happens when collisions occur. In order to do that, the collisions have to be detected first. There are two types of collisions: (i) key duplicates and (ii) hash value duplicates (hash collisions). The second type is more complicated than the first type. In this paper, we investigated linear probing as a heavily applied hash table implementation and we presented an extension of the state-of-the-art vectorized implementation with a hardware-supported duplicate or collision detection. For that, we use novel SIMD instructions which have been introduced with Intel’s SIMD instruction set extension AVX-512. As we have shown, our approach outperforms the state-of-the-art vectorized version for the key handling, but introduces novel challenges for the value handling.

Further research should investigate different methods of value handling. The usage of dynamic sized buffers should be replaced through a fixed sized buffer. Based on that, costly memory reallocations and copy operations can be reduced as well as handling the values could be done using SIMD scatter instructions. One opportunity for that would be to process the given dataset twice, collecting statistics for the dataset within the first run. Then, this information can be used in a further step to allocate a constant sized value store which can hold up all values having to be inserted. As a side effect of this approach, the result of the first run can be used further e.g., for database operators like aggregation.

References

- [Ab16] Abadi, Daniel; Agrawal, Rakesh; Ailamaki, Anastasia; Balazinska, Magdalena; Bernstein, Philip A.; Carey, Michael J.; Chaudhuri, Surajit; Dean, Jeffrey; Doan, AnHai; Franklin, Michael J.; Gehrke, Johannes; Haas, Laura M.; Halevy, Alon Y.; Hellerstein, Joseph M.; Ioannidis, Yannis E.; Jagadish, H. V.; Kossmann, Donald; Madden, Samuel; Mehrotra, Sharad; Milo, Tova; Naughton, Jeffrey F.; Ramakrishnan, Raghu; Markl, Volker; Olston, Christopher; Ooi, Beng Chin; Ré, Christopher; Suci, Dan; Stonebraker, Michael; Walter, Todd; Widom, Jennifer: The Beckman report on database research. *Commun. ACM*, 59(2):92–99, 2016.
- [Ba13] Balkesen, Cagri; Alonso, Gustavo; Teubner, Jens; Özsu, M. Tamer: Multi-Core, Main-Memory Joins: Sort vs. Hash Revisited. *PVLDB*, 7(1):85–96, 2013.
- [Be18] Behrens, Tobias; Rosenfeld, Viktor; Traub, Jonas; Breß, Sebastian; Markl, Volker: Efficient SIMD Vectorization for Hashing in OpenCL. In: *Proceedings of the 21th International Conference on Extending Database Technology, EDBT 2018, Vienna, Austria, March 26-29, 2018*. pp. 489–492, 2018.
- [BFT16] Breß, Sebastian; Funke, Henning; Teubner, Jens: Robust Query Processing in Co-Processor-accelerated Databases. In: *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*. pp. 1891–1906, 2016.
- [BKM08] Boncz, Peter A.; Kersten, Martin L.; Manegold, Stefan: Breaking the memory wall in MonetDB. *Commun. ACM*, 51(12):77–85, 2008.
- [Da17] Damme, Patrick; Habich, Dirk; Hildebrandt, Juliana; Lehner, Wolfgang: Lightweight Data Compression Algorithms: An Experimental Survey (Experiments and Analyses). In: *Proceedings of the 20th International Conference on Extending Database Technology, EDBT 2017, Venice, Italy, March 21-24, 2017*. pp. 72–83, 2017.
- [Do13] Do, Jaeyoung; Kee, Yang-Suk; Patel, Jignesh M.; Park, Chanik; Park, Kwanghyun; DeWitt, David J.: Query processing on smart SSDs: opportunities and challenges. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*. pp. 1221–1230, 2013.
- [Fa17] Faerber, Franz; Kemper, Alfons; Larson, Per-Åke; Levandoski, Justin J.; Neumann, Thomas; Pavlo, Andrew: Main Memory Database Systems. *Foundations and Trends in Databases*, 8(1-2):1–130, 2017.
- [Ha18] Habich, Dirk; Damme, Patrick; Ungethüm, Annett; Lehner, Wolfgang: Make Larger Vector Register Sizes New Challenges?: Lessons Learned from the Area of Vectorized Lightweight Compression Algorithms. In: *Proceedings of the 7th International Workshop on Testing Database Systems, DBTest@SIGMOD 2018, Houston, TX, USA, June 15, 2018*. pp. 8:1–8:6, 2018.
- [La16] Lang, Harald; Mühlbauer, Tobias; Funke, Florian; Boncz, Peter A.; Neumann, Thomas; Kemper, Alfons: Data Blocks: Hybrid OLTP and OLAP on Compressed Storage using both Vectorization and Compilation. In: *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*. pp. 311–326, 2016.

- [La18] Lang, Harald; Kipf, Andreas; Passing, Linnea; Boncz, Peter A.; Neumann, Thomas; Kemper, Alfons: Make the most out of your SIMD investments: counter control flow divergence in compiled query pipelines. In: Proceedings of the 14th International Workshop on Data Management on New Hardware, Houston, TX, USA, June 11, 2018. pp. 5:1–5:8, 2018.
- [LB15] Lemire, Daniel; Boytsov, Leonid: Decoding billions of integers per second through vectorization. *Softw., Pract. Exper.*, 45(1):1–29, 2015.
- [Le17] Lehner, Wolfgang: The Data Center under your Desk - How Disruptive is Modern Hardware for DB System Design? *PVLDB*, 10(12):2018–2019, 2017.
- [LP13] Li, Yinan; Patel, Jignesh M.: BitWeaving: fast scans for main memory data processing. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22–27, 2013. pp. 289–300, 2013.
- [LUH18] Lehner, Wolfgang; Ungethüm, Annett; Habich, Dirk: Diversity of Processing Units - An Attempt to Classify the Plethora of Modern Processing Units. *Datenbank-Spektrum*, 18(1):57–62, 2018.
- [OL18] Oukid, Ismail; Lersch, Lucas: On the Diversity of Memory and Storage Technologies. *Datenbank-Spektrum*, 18(2):121–127, 2018.
- [Ou17] Oukid, Ismail; Booss, Daniel; Lespinasse, Adrien; Lehner, Wolfgang; Willhalm, Thomas; Gomes, Grégoire: Memory Management Techniques for Large-Scale Persistent-Main-Memory Systems. *PVLDB*, 10(11):1166–1177, 2017.
- [PRR15] Polychroniou, Orestis; Raghavan, Arun; Ross, Kenneth A.: Rethinking SIMD Vectorization for In-Memory Databases. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015. pp. 1493–1508, 2015.
- [RAD15] Richter, Stefan; Alvarez, Victor; Dittrich, Jens: A Seven-Dimensional Analysis of Hashing Methods and its Implications on Query Processing. *PVLDB*, 9(3):96–107, 2015.
- [SGL10] Schlegel, Benjamin; Gemulla, Rainer; Lehner, Wolfgang: Fast integer compression using SIMD instructions. In: Proceedings of the Sixth International Workshop on Data Management on New Hardware, DaMoN 2010, Indianapolis, IN, USA, June 7, 2010. pp. 34–40, 2010.
- [SWL11] Schlegel, Benjamin; Willhalm, Thomas; Lehner, Wolfgang: Fast Sorted-Set Intersection using SIMD Instructions. In: International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures - ADMS 2011, Seattle, WA, USA, September 2, 2011. pp. 1–8, 2011.
- [Un18] Ungethüm, Annett; Pietrzyk, Johannes; Damme, Patrick; Habich, Dirk; Lehner, Wolfgang: Conflict Detection-Based Run-Length Encoding - AVX-512 CD Instruction Set in Action. In: 34th IEEE International Conference on Data Engineering Workshops, ICDE Workshops 2018, Paris, France, April 16–20, 2018. pp. 96–101, 2018.
- [ZR02] Zhou, Jingren; Ross, Kenneth A.: Implementing database operations using SIMD instructions. In: Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, Madison, Wisconsin, USA, June 3–6, 2002. pp. 145–156, 2002.

Query Processing and Optimization

LinDP++: Generalizing Linearized DP to Crossproducts and Non-Inner Joins

Bernhard Radke,¹ Thomas Neumann²

Abstract: Choosing the best join order is one of the main tasks of query optimization, as join ordering can easily affect query execution times by large factors. Finding the optimal join order is NP-hard in general, which means that the best known algorithms have exponential worst case complexity. As a consequence only relatively modest problems can be solved exactly, which is a problem for today's large, machine generated queries. Two developments have improved the situation: If we disallow crossproducts, graph-based DP algorithms have pushed the boundary of solvable problems to a few dozen relations. Beyond that, the linearized DP strategy, where an optimal left-deep plan is used to linearize the search space of a subsequent DP, has proven to work very well up to a hundred relations or more.

However, these strategies have limitations: Graph-based DP intentionally does not consider implicit crossproducts, which is almost always ok but sometimes undesirable, as in some cases such crossproducts are beneficial. Even more severe, linearized DP can handle neither crossproducts nor non-inner joins, which is a serious limitation. Large queries with, e. g., outer joins are quite common and having to fall back on simple greedy heuristics in this case is highly undesirable.

In this work we remove both limitations: First, we generalize the underlying linearization strategy to handle non-inner joins, which allows us to linearize the search space of arbitrary queries. And second, we explicitly recognize potential crossproduct opportunities, and expose them to the join ordering strategies by augmenting the query graph. This results in a very generic join ordering framework that can handle arbitrary queries and produces excellent results over the whole range of query sizes.

1 Introduction

One of the most important tasks of query optimization is join ordering. Due to the multiplicative nature of joins, changes in join order can easily affect query execution times by large integer factors [Le18]. Unfortunately, finding the optimal join order is NP-hard in general [IK84] and no exact algorithms with better than exponential worst case optimization time are known for the general case. This is problematic because queries tend to get larger, at least in the long tail. Today, most queries are not written by humans but by machines, and queries that join a hundred relations or more are not that uncommon [Vo18]. To put that into perspective, PostgreSQL for example switches from dynamic programming (DP) to a

¹ Technische Universität München, radke@in.tum.de

² Technische Universität München, neumann@in.tum.de

heuristic if the query contains 12 relations or more, which means that none of the larger queries will be optimized exactly.

One reason for that is their somewhat simplistic DP strategy. DP strategies that exploit the structure of the query graph for example can handle larger queries [MN08], but even there the exponential nature of the problem limits query sizes to about 30 relations, depending upon on the structure of the query graph. For even larger queries most approaches fall back to simple heuristics. An alternative to that is the relatively recent *linearized DP* strategy [NR18]. The idea is to linearize the search space by first picking a good (ideally optimal) relative order of relations, and then use a polynomial time DP step to construct the optimal bushy tree for that relative order. Of course we do not know the optimal order for the general solution, but we can use the IK/KBZ algorithm [IK84, KBZ86] to construct the optimal left-deep order in polynomial time. In practice this leads to excellent results, producing optimal or near-optimal solutions even for large queries with very low optimization time.

However, the IK/KBZ algorithm supports only inner joins, which is a problem for practical usage. Outer, semi, and anti joins are quite common: In the real-world workload presented by Vogelsgesang et al. [Vo18], about 20% of the join queries do contain at least one outer join with a maximum of 247 outer joins in a single query. Having to fall back to simple heuristics just because the query contains a single outer join is not very satisfying. Similar problems occur with complex predicates, for example predicates of the form $R_1.A + R_2.B = R_3.C + R_4.D$. These complex predicates are rare, but they can be formulated in SQL. In the query graph they form a hyperedge, connecting sets of relations with sets of relations, which is also not supported by IK/KBZ. Note that non-inner joins can be expressed by using hyperedges, too [MFE13], thus both problems are closely related from an optimizer perspective. These restrictions are very unfortunate, as now queries with simple inner joins can be optimized very efficiently, but adding just one non-inner join or one complex join predicate forces the system to switch to simple heuristics, resulting in clearly inferior plans.

Furthermore, graph-based DP as well as linearized DP ignore crossproducts. Usually this is a good idea. The search space without crossproducts is much smaller, and in most cases crossproducts are a bad idea. However, sometimes they can indeed be helpful if some input relations or intermediate results are known to be very small. Even then, crossproducts should be used prudently as mis-estimations about input cardinalities can lead to terrible execution times due to the $O(n^2)$ nature of a crossproduct. And considering crossproducts in the presence of non-inner joins is dangerous as that can lead to wrong results. Consider e. g. the query $(A \bowtie B) \bowtie_{A.x=C.y} C$ and assume $B = \emptyset$. Performing a crossproduct between B and C before evaluating the outer join would cause an empty result, whereas the original query yields the complete relation C . Nevertheless, if we make sure that a crossproduct does not bypass non-inner joins and we are certain about the input cardinalities (e. g. when a primary key is bound), crossproducts can sometimes significantly improve query performance [OL90]. Having support for crossproducts in “safe” cases is thus highly desirable.

In this paper we generalize the recently published linearized DP [NR18] by removing both limitations. Our generalized LinDP++ strategy is capable of ordering non-inner joins, which allows it to handle all kinds of join queries. We achieve this using a recursive precedence-graph decomposition at hyperedges, which allows IK/KBZ to handle hypergraphs. In addition we present a fast heuristic that explicitly enriches the search space to also consider safe crossproducts without causing the search space to grow exponentially. The combination of these two components results in a fast polynomial time heuristic for join ordering that finds very good plans, explicitly investigates relevant crossproducts, and handles non-inner joins correctly. Experimental comparisons with slow exact DP strategies show that the resulting plans are close to optimal. And the algorithm can scale to queries with a hundred relations or more, which is far beyond what normal DP algorithms can do. For practical usage this is a great improvement, as we no longer have to fall back to weaker approaches for certain classes of queries.

The rest of this paper is structured as follows: First we summarize prior work in Section 2. The extension of linearized DP to non-inner joins is described in Section 3. In Section 4 we investigate how join ordering can be extended to take beneficial crossproducts into consideration. We evaluate runtime characteristics and result quality of LinDP++ in Section 5 before we draw a conclusion and point out directions for future research in Section 6.

2 Related Work

Join ordering has first been tackled by Selinger et al. in [Se79]. They proposed a dynamic programming (DP) strategy that generates an optimal linear join tree. Optimal solutions for subproblems of increasing size are built bottom up by combining optimal solutions for smaller subproblems. Since then there has been lots of follow-up work of which we discuss the most relevant techniques in the following.

An obvious improvement over the initial DP is to consider bushy trees as well. Furthermore, for a DP algorithm to work it is not necessary to enumerate alternatives increasing in size. Other enumeration schemes work as well (e. g. integer order enumeration [VM96]), as long as optimal solutions for subproblems are generated prior to their usage in larger problems. All of these dynamic programming variants can be implemented to consider crossproducts. In this case, however, *all* possible crossproducts would implicitly be enumerated. This results in exponential complexity of the algorithms and disregards the actual structure of the query. In addition to this increase in complexity, crossproducts between arbitrary relations can produce incorrect query results in the presence of outer joins.

The most efficient DP algorithms take the query graph into account. By design, such algorithms generate only execution plans without crossproducts. With the reordering constraints induced by outer joins encoded into the query graph, they can also validly reorder across outer joins [MN08]. Besides bottom-up enumeration there also exist variants that perform the enumeration top-down, thereby enabling more aggressive pruning [FM13]. As

these algorithms strictly follow the structure of the query graph, they per se do not generate crossproducts. crossproducts can, however, explicitly be taken into account by adding edges to the query graph and updating the reordering constraints of outer joins.

Another approach to incorporate the reordering constraints imposed by non-inner joins is to add compensation operators [WC18]. These operators correct errors introduced by crossproducts across outer joins by removing or modifying spurious tuples. Materializing these spurious tuples and manipulating them, however, creates overhead at query runtime.

Hardware trends have motivated research on parallelizing dynamic programming [Ha08]. However, linearly increasing the compute power cannot compensate for the exponential growth of the search space. Doubling e. g. the compute power only allows queries with one additional join to be optimized exactly in a similar, reasonable amount of time.

Lately, the use of linear programming for join ordering has been proposed [TK17]. The mixed integer linear program (MILP) that they generate encodes relations, cardinalities, and costs. The solution of the MILP can then be interpreted as a linear join tree. As they do not fully constrain the MILP to the query graph, the solution may contain crossproducts. For queries with outer joins this can again lead to invalid execution plans.

Many commercial systems find a good join order by applying transformations onto the initial execution plan [Gr95]. These transformative approaches have the advantage that relational equivalences can directly be translated into transformation rules. Direct application of equivalences enables the algorithms to take care of reordering constraints as transformation rules can be disabled as required. However, these algorithms are considerably slower than DP style algorithms and avoiding to generate trees multiple times is non-trivial. These issues become even more prominent if additional rules to generate crossproducts are introduced.

Besides exact algorithms that give an optimal tree, a large number of heuristics have been proposed especially to handle large queries. One well known heuristic is the genetic algorithm [SMK97], a variant of which is used by PostgreSQL for queries containing more than 12 joins. The genetic meta-heuristic starts with a population of randomly generated execution plans. For a number of generations, crossover and mutation is applied and the best plans of the resulting population survive the generation.

Another interesting algorithm is Greedy Operator Ordering (GOO) [Fe98]. This heuristic builds a bushy join tree bottom up by picking the pair of relations whose join result is minimal. The picked pair is then merged into a single relation representing the join. Repeated application of this procedure finally results in a single relation representing a complete join tree. GOO is fast even for large queries and usually gives decent plans despite it's greediness.

Iterative Dynamic Programming (IDP) [KS00] is a combination of DP and GOO. It has proven to work well especially for really large queries. By using DP to refine expensive parts of a join tree generated by a greedy algorithm, plan costs can be significantly reduced.

In [NR18] we described linearized DP, a heuristic for join ordering on large queries. We avoid the exponential complexity of a full dynamic programming strategy by restricting the DP algorithm to a reduced, linearized search space. Utilizing this technique we bridged the gap between the small queries which can be optimized exactly and the really large queries, where only greedy heuristics have acceptable runtime. While this approach gives excellent results for regular queries, its inability to handle outer joins is a major drawback which we tackle in this paper.

3 Search Space Linearization

Dynamic Programming (DP) algorithms can be used to solve the join ordering problem exactly, but the exponential worst case complexity of all known algorithms limits their use to relatively small queries. The exact complexity depends upon the structure of the explored search space: A join query Q induces an undirected query graph $G = (V, E)$, where V is the set of relations, and E is the set of join possibilities between relations. In the general case, or when the query graph forms a clique, the best known algorithm has a time complexity in $O(3^n)$, which is infeasible for large n . But when the query graphs forms a linear chain and if crossproducts are not allowed, the optimal solution can be found in $O(n^3)$ [MN06], which is tractable even for large n .

This observation recently led to the concept of linearized DP [NR18]. The key idea is as follows: Assume that we would know the optimal join order. Then we could take the relative order of the relations in the optimal join tree and linearize the search space by restricting the DP algorithm to consider only sub-chains of that relative order. Given the optimal relative order as input the DP phase can construct the optimal bushy tree in $O(n^3)$ [NR18].

Of course we do not know the optimal join order, and thus we do not know the optimal relative order, either. But for a large class of queries we can construct the optimal left-deep tree in polynomial time using the IK/KBZ algorithm [IK84, KBZ86]. This gives us a relative order of relations, too, which we can then use for search space linearization. Using the IK/KBZ solutions as seed for search space linearization is a heuristic, as we can cut the optimal solution from the search space, but 1) the resulting plan is never worse than the optimal left-deep plan, and 2) it is usually close to the true optimal solution in practice, with much lower optimization time [NR18]. Note that while IK/KBZ requires the cost function to have Adjacent Sequence Interchange (ASI) property [MS79], the subsequent DP phase can use any cost function that adheres to the bellman principle.

The main limitation of this technique is that IK/KBZ cannot handle arbitrary queries. First, it requires an acyclic query graph. We can avoid that problem by first constructing a minimum spanning tree before executing IK/KBZ. The intuition behind this is that less selective joins are less likely to be part of the optimal join tree, thus dropping edges that correspond to such joins to break cycles is usually safe. Note that the DP phase of the algorithm can again operate on the original, complete query graph potentially including

Algorithm 1 The IKKBZ algorithm [IK84, KBZ86]

```

IKKBZ( $Q = (V, E)$ )
// construct an optimal left-deep tree for the acyclic query graph  $Q$ 
   $b = \emptyset$ 
  for each  $s \in V$ 
     $P_v = \text{IKKBZ-precedence}(Q, s, \emptyset)$ 
    while  $P_v$  is not a chain
      pick  $v'$  in  $P_v$  whose children are chains
      IKKBZ-normalize each child chain of  $v'$ 
      merge the child chains by rank
    if  $b = \emptyset \vee C(P_v) < C(b)$ 
       $b = P_v$ 
  return  $b$ 

IKKBZ-precedence( $Q(V, E), v, X$ )
// build a precedence tree by directing edges away from a node  $v \in V$ 
   $P_v = v$ 
  for each  $e \in E : (e = (v, u) \vee e = (u, v)) \wedge u \notin X$ 
    add IKKBZ-precedence( $Q, u, X + v$ ) as child of  $P_v$ 
  return  $P_v$ 

IKKBZ-normalize( $c$ )
//normalize a chain of relations
  while  $\exists i: \text{rank}(c[i]) > \text{rank}(c[i + 1])$ 
    merge  $c[i]$  and  $c[i + 1]$  into a compound relation

```

cycles. More severely, IK/KBZ cannot handle hyperedges in the query graph, which would be necessary to support non-inner joins and complex join predicates. For example the join query $(A \bowtie B) \bowtie (C \bowtie D)$ will have the regular edges (A, B) , (C, D) , and the hyperedge $(\{A, B\}, \{C, D\})$, which captures the reordering constraints of inner joins and outer joins. Note that this is a fundamental problem: The algorithm constructs left-deep trees, but that query graph has no valid left-deep solution, all solutions must be bushy. In order to apply the idea of search space linearization to queries with non-inner joins we must therefore extend IK/KBZ to handle hyperedges.

In the following we first briefly repeat how regular search-space linearization works, and then show an extension to handle hyperedges.

3.1 Regular Search Space Linearization

Before discussing the linearization for hypergraphs, let us briefly reiterate, how the linearization of regular graphs works. The IK/KBZ algorithm [IK84, KBZ86] constructs an optimal left-deep join tree, which is then used as relative relation order in the linearized DP.

As input the algorithm gets an acyclic query graph. For cyclic query graphs we construct a minimum spanning tree first.

The pseudo-code for the IK/KBZ algorithm is shown in Algorithm 1. It chooses each relation s as start node once, and then runs the complete construction algorithm given that start node. For each s , it first constructs the directed precedence graph P_v rooted in s by directing all join edges away from s . That precedence graph indicates which relations have to be joined first before other joins become feasible. That is, all valid join orders adhere to the partial order induced by the precedence graph. Then the algorithm tries to sort all relations by the cost/benefit ratio of performing the join with a relation. This ratio is called rank [IK84]. If we get conflicts between the rank order and the order imposed by the precedence graph, i. e., if we would like to join with R_1 before R_2 , but the precedence graph requires that R_2 is joined before R_1 , the `IKKBZ-normalize` function takes both relations and combines them into a compound relation, because we know that both must occur next to each other in the optimal solution [IK84]. The remaining relations are ordered by rank until we get a total order.

The traditional IK/KBZ algorithm returns the cheapest of these n total orders, which is guaranteed to be an optimal left-deep execution plan. For the linearized DP we take the total order and use it to restrict the search space considered by the DP phase [NR18]. Note that we get better results by running the DP phase not only on the order in the cheapest plan, but on all P_v orders. The reason for that is that the optimal bushy order can be different from the optimal left-deep order. The different P_v orders are the optimal left-deep orders given a certain start node; by considering all of them we give the DP algorithm a chance to recognize orders that are more expensive left-deep but cheaper in bushy form.

3.2 Precedence for Hypergraphs

The IK/KBZ algorithm is only capable of producing linearizations for regular, acyclic graphs. When generalizing it to handle hypergraphs we first have to construct a precedence graph, too, which is a bit non-intuitive for hyperedges. The hyperedges have to be directed away from the start node, but note that a hyperedge connects a set of relations with a set of relations. To express that, a directed hyperedge is defined similar to the definition used by Gallo et al. [GLP93]:

Definition 1. *In a directed hypergraph $H = (V, E)$, a directed edge e from $T \subseteq V$ to $H \subseteq V$ is an ordered pair $e = (T, H)$, where T is said to be the tail and H the head of the edge.*

We differentiate two types of edges during precedence graph construction: backward hyperedges $b = (T, H) : |T| > 1$ and forward hyperedges $f = (T, H) : |H| > 1$. Note that an edge can be both a backward and a forward edge.

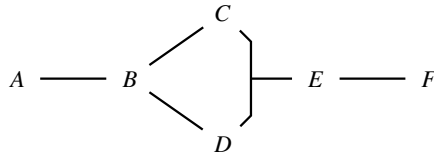


Fig. 1: A query graph with a hyperedge $(\{C, D\}, \{E\})$

For backward hyperedges all relations in the tail set T have to be available before any relation of the head H can be joined. Such edges, thus, have to be postponed until all tail relations are covered by the precedence graph. If all relations in T lie on a single path from the start relation s to the last visited relation of T we can simply append the backward edge to that relation, because we know that all other relations must have been joined before. If that is not the case, i. e., if some relations lie on different paths from the start relation we still attach the edge to the last visited relation of T , but we mark it as *partial*. We cannot statically guarantee that all relations in T will be available when considering the join and must re-check that when merging child chains.

Consider for example the query graph shown in Fig. 1. When building the precedence graph rooted in B , the backward hyperedge $(\{C, D\}, \{E\})$ has to be handled. If w.l.o.g D is visited after C during construction, then E is added as a child of D . Note that E is partial here, as it additionally requires C , which is not part of the path from B to D . All other edges are regular and handled as in IK/KBZ which results in the precedence graph given in Fig. 2.

For forward hyperedges, all relations in the head set H must be available on the right-hand side of the join. In particular, there exists no left-deep solution, the final solution must be bushy and contain a join with a super-set of H on the right hand side. The key insight here is that the query graph has to be acyclic anyways to apply IK/KBZ. Thus, if we cut the graph at the forward hyperedge we get exactly two disconnected sub-graphs, which can be optimized independently. We call the head of such a forward edge a *group* and optimize it recursively when encountering a forward edge during precedence graph construction. Note that the solution of a group is independent of the currently investigated start relation and can therefore be reused across start relations. When integrating the recursive solution into the precedence graph we could add all relations in the sub-graph as one compound relation, but that would be overly restrictive. Instead, only the minimal subchain that covers the groups relations is added as compound relation and the rest is kept as individual relations. Due to the recursive nature of this scheme, individual relations here can be compound relations from other hyperedges, of course.

An example of a precedence graph dealing with a forward hyperedge is shown in Fig. 3. This is again a precedence graph for the query in Fig. 1, this time rooted in E . When the forward hyperedge $(\{E\}, \{C, D\})$ is encountered, the group $\{C, D\}$ has to be solved. The solution of the group, which covers at least the relations B, C and D , forms a compound

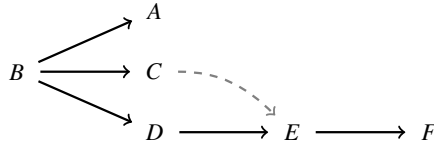


Fig. 2: Generalized precedence hypergraph of the graph shown in Fig. 1 rooted in B . E is partial and additionally requires C .

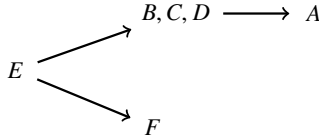


Fig. 3: Generalized precedence hypergraph of the graph shown in Fig. 1 rooted in E . $\{B, C, D\}$, the compound solution to the group of the forward hyperedge $(\{E\}, \{C, D\})$ is appended to E

node and is appended as child of E . Note that the solution to $\{C, D\}$ would additionally include A if $\text{rank}(A) < \max(\text{rank}(B), \text{rank}(C), \text{rank}(D))$.

If an edge is both, a backward and a forward hyperedge, both strategies are combined: Application of the edge has to be postponed until the complete tail is available and the solution for the head group must be inserted when the tail is completely included.

3.3 Linearization using Precedence Hypergraphs

Using the generalized precedence graph we can now run a modified IK/KBZ algorithm to find a linearization. Similar to the original IK/KBZ algorithm, this is achieved by merging the nodes in subchains ascending in their rank. One difference is that nodes might already be compound relations (if forward-edges are encountered), but that does not require code changes. Backward edges are more difficult, as they have to be recognized during normalization: A sequence AB must not be normalized if B is partial, as this would prevent interleaving other nodes that are required by B between A and B . Instead, the nodes are kept separate in the precedence graph. The rank in the subchain of B is no longer monotonic here, which requires some care during implementation, but in practice B is merged as soon as possible after A .

The modifications to IK/KBZ making it hypergraph aware are given in Algorithm 2, where IKKBZ-precedence is called as $\text{IKKBZ-precedence}(Q, \emptyset, \{s\}, \emptyset)$ for each start node s .

As an example, let us now linearize the search space of the query depicted in Fig. 1 for start relation E (cardinalities and selectivities are given in Tab. 1). The algorithm starts

Algorithm 2 IKKBZ procedures generalized to hypergraphsIKKBZ-solve-group($Q(V, E), I, G, X$)

```

// solve a group G
if  $|G| = 1$  return  $G$ 
if memoized( $G$ ) return memoized( $G$ )
 $b = \emptyset$ 
for each  $s \in G$ 
   $P_v = \text{IKKBZ-precedence}(Q, \emptyset, s, X \cup I + s)$ 
  while  $P_v$  is not a chain
    pick  $v'$  in  $P_v$  whose children are chains
    IKKBZ-normalize each child chain of  $v'$ 
    merge the child chains by rank
  if  $b = \emptyset \vee C(P_v) < C(b)$ 
     $b = P_v$ 
 $r =$  smallest subsequence of  $b$  that covers all  $g \in G$ 
memoize  $r$  as solution for  $G$ 
return  $r$ 

```

IKKBZ-precedence($Q(V, E), I, G, X$)

```

// build a precedence tree by directing edges away from a node representing the group G
 $P_v = \text{IKKBZ-solve-group}((V - (G \cup X), E), I, G, X \cup I)$ 
mark all nodes in  $P_v$ 
for each  $e \in E : (e = (U, W) \vee e = (W, U)) \wedge U \subseteq P_v \wedge \forall w \in W : w \notin X$ 
  if  $\exists !u \in U : \neg \text{isMarked}(u)$  add IKKBZ-precedence( $Q, U, W, X + U$ ) as child of  $P_v$ 
  else postpone  $e$ 
for each postponed edge  $(e = (U, W) \vee e = (W, U)) \wedge \forall w \in W : w \notin X \wedge \forall u \in U : \text{isMarked}(u)$ 
  add IKKBZ-precedence( $Q, U, W, X + U$ ) as child of  $P_v$ 
return  $P_v$ 

```

IKKBZ-normalize(c)

```

//normalize a chain of relations
while  $\exists i : \text{rank}(c[i]) > \text{rank}(c[i + 1]) \wedge \neg \text{isPartial}(c[i + 1])$ 
  merge  $c[i]$  and  $c[i + 1]$  into a compound relation

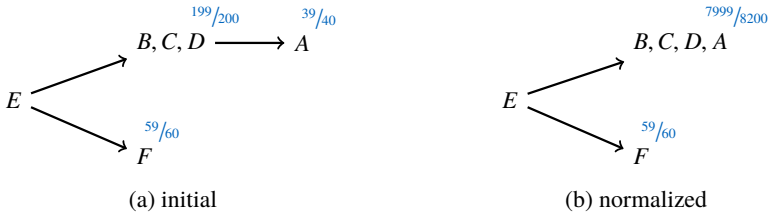
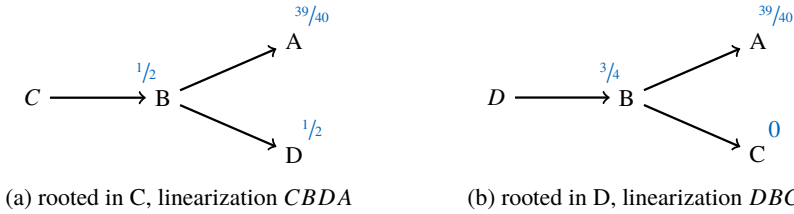
```

building the precedence graph at the start relation E and directs all edges away from E . This is immediately possible for the edge (E, F) . The forward hyperedge $(\{E\}, \{C, D\})$ however, requires to solve the group $\{C, D\}$ first. This is done by recursively linearizing the precedence graphs for the subgraph covering $\{A, B, C, D\}$ rooted in C respectively D . Those precedence graphs, annotated with ranks and their respective linearizations are depicted in Fig 5. After cost comparison, $CBDA$ is selected as the best solution for the group, from which the algorithm picks the subchain CBD . The intermediate result of CBD has a cardinality of 200 which gives a rank of $199/200$ for the group solution. Finally, the edge (B, A) is again regular and A is simply added as child of the compound node B, C, D . This completes the construction of the precedence graph which is depicted with annotated ranks in Fig. 4a.

The second phase of the algorithm then builds a total order of relations based on this

Relations	Cardinality	Join edge	Selectivity
A	100	(A,B)	0.4
B	100	(B,C)	0.02
C	50	(B,D)	0.04
D	50	({C,D},E)	0.01
E	100	(E,F)	0.5
F	120		

Tab. 1: Cardinalities and selectivities for the example query (Fig. 1)


 Fig. 4: The initial global precedence graph rooted in E for the example query in Fig. 1 and its normalization. Ranks are annotated in blue.

 Fig. 5: Sub-precedence graphs when solving group $\{C, D\}$ rooted in C respectively D . Ranks are annotated in blue.

precedence graph. The algorithm descends into the tree until it encounters a node whose children are chains, which is immediately the case for the root node E . Before merging the child chains into a total order, any contradictory sequences UV within the chains are normalized if V is not partial. In the example, there is a contradictory sequence between the compound relation BCD and A . These two nodes are normalized into a compound node and the new rank computed accordingly (see Fig. 4b). After this normalization step all contradictions are resolved and the algorithm continues with merging the children of E . This results in the linearization $ECBDAF$. Based on this linearized search space, the polynomial time DP algorithm finally constructs the logical execution plan depicted in Fig. 6. Overall, the algorithm would build execution plans based on all linearizations for the different start relations and select the one with the lowest cost.

Using this modified IK/KBZ algorithm we can now linearize the search space of arbitrary

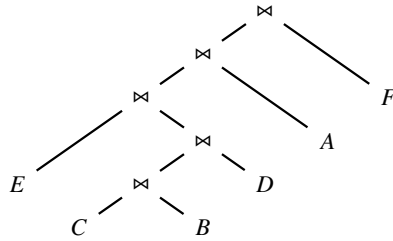


Fig. 6: Logical execution plan based on the linearization *ECBDAF*

queries, including queries with non-inner joins and complex join predicates. If the query becomes too large for the DP step (for example more than 100 or 150 relations, depending on the available hardware) we can fall back to an iterative dynamic programming strategy using LinDP++ as inner algorithm, as discussed in [NR18].

4 Considering Potentially Beneficial Crossproducts

Having introduced a technique to handle non-inner joins in the previous section we now turn our attention onto the usefulness of crossproducts. Usually, when a join ordering algorithm does consider crossproducts, it considers all of them. Unfortunately this increases the search space dramatically, and increases the optimization time to $\mathcal{O}(3^n)$, regardless of the structure of the query graph. And most of these considered crossproducts will be useless: A crossproduct $L \times R$ is inherently a $\mathcal{O}(|L||R|)$ operation, while a hash join can be executed ideally in linear time. Which means that crossproducts are only attractive if at least one of its inputs is reasonably small. On the other hand crossproducts can sometimes be used to avoid repeated joins with large relations (by building the crossproduct of small relations first). We would like to capture this (rare, but useful) use case, without paying the exponential costs of considering all crossproducts. In this section we therefore introduce a cheap heuristic to detect potentially beneficial crossproducts. We use that information to make them explicit in the query graph: If a crossproduct between R_1 and R_2 is considered beneficial we add an artificial crossproduct edge with selectivity 1 between R_1 and R_2 to the query graph. The DP phase of LinDP++ will consider this edge during plan construction and will utilize the crossproduct if beneficial. Note that the heuristic itself is not tied to LinDP++, it could be used by any query graph based optimization algorithm, like, for example, DPhyp.

When finding crossproduct candidates we have two problems: First, finding good candidates has to be reasonably cheap, and second, we must be careful in the presence of non-inner joins, as adding crossproducts there can lead to wrong results. For example in the query $R_1 \bowtie (R_2 \bowtie R_3 \bowtie R_4)$ we must not introduce a crossproduct between R_1 and R_4 , but we could introduce a crossproduct between R_2 and R_4 . We solve that problem by analyzing the paths between two relations: We only consider crossproducts between two relations R_i

and R_j if there exists a path of regular (i. e., inner join) edges between them. This avoids bypassing non-inner joins with crossproducts.

Intuitively, a crossproduct is potentially beneficial if it allows to cheaply bypass a sequence of expensive join operations. Analyzing all paths between all pairs of relations is computationally expensive and not feasible in practice. However, restricting the analysis to paths of length two gives polynomial optimization time and still catches many cases where a crossproduct could result in a better plan: For a query Q we investigate all pairs of neighboring edges $e_1 = (u, v) \in Q$ and $e_2 = (v, w) \in Q$. We augment Q with an artificial crossproduct edge (v, w) of selectivity 1 if the cardinality of the crossproduct $|u \times w|$ is less than the result sizes of both of the joins:

$$|u \times w| < |u \bowtie v| \wedge |u \times w| < |v \bowtie w|$$

For the C_{out} cost function [CM95] this criterion gives all potentially beneficial crossproducts to bypass paths of length two while still keeping optimization complexity reasonable. Of course there could be even longer paths where bypassing joins via crossproducts would result in cheaper plans. Our experimental evaluation (Section 5), however suggests, that investigating paths of length two covers most of the important crossproducts, and has negligible overhead. If one wants to be more aggressive with crossproducts we could consider even longer paths if the relations are particular small (for example a single tuple). But this leads to diminishing return compared to the optimization time, which is why we used only paths of length two in our experiments.

Note that crossproducts should be introduced very conservatively, especially if cardinality estimates are inaccurate. A crossproduct is inherently quadratic in nature, and if an input relation has estimated cardinality of 1 and a real cardinality of 10,000 (which can easily happen in practice), the performance impact will be disastrous. On the other hand the cardinality is sometimes known exactly, for example if the primary key is bound or if the input is the result of a group-by query with known group count, which makes the introduction safe. For base tables the available statistics can sometimes provide reasonably tight upper bounds for the input size, which also makes the computation safe if the upper bound is used in the formula above. This essential prudence reduces the number of cases where we will consider crossproducts, but nevertheless there remain queries where crossproducts are attractive and safe, and we can and should consider them during join tree construction.

Consider for example the query graph with cardinalities and selectivities given in Fig. 7a. The optimal execution plan without crossproducts for this query has costs of 1.84M and is depicted in Fig. 8a. When applying the crossproduct heuristic, the query graph is augmented with two additional edges (A, C) and (D, F) as shown in Fig. 7b. While these two additional edges only marginally enlarge the search space, costs are cut by almost 50% to 0.94M using the plan shown in Fig. 8b. Note that even considering all possible crossproducts, although a much larger search space is explored, does not uncover a cheaper plan. Further note that LinDP++ generates the same plan, despite exploring a reduced, linearized search space.

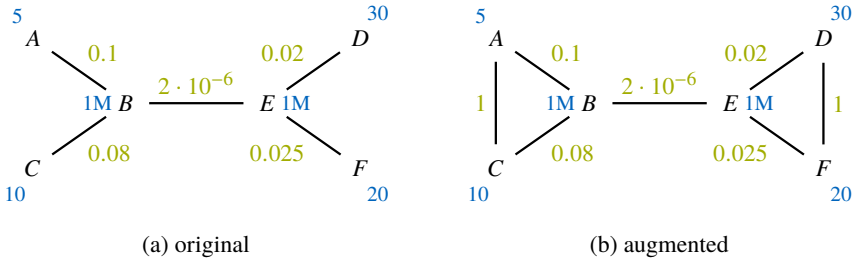


Fig. 7: Query graph where crossproducts enable cheaper execution plans. Cardinalities are annotated in blue, join selectivities in green.

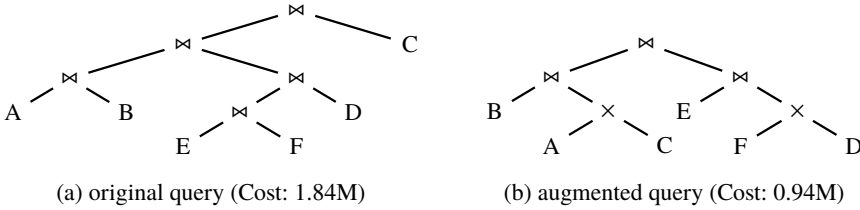


Fig. 8: Optimal execution plans for the query graphs depicted in Fig. 7

5 Evaluation

In this section we present the results of an extensive experimental analysis of the techniques described in this paper. We compare LinDP++ against a multitude of different join ordering algorithms including DPhyp [MN08], Greedy Operator Ordering (GOO) [Fe98], Iterative Dynamic Programming [KS00] using DPhyp as inner algorithm, Quickpick [WP00], genetic algorithms [SMK97], query simplification [Ne09], minsel [Sw89], and linearized DP [NR18]. The algorithms were used to optimize the queries of the following well known standard benchmarks using the C_{out} [CM95] cost function: TPC-H [Tra17b] and TPC-DS [Tra17a], LDBC BI [An14], the Join Order Benchmark (JOB) [Le18], and the SQLite test suite [Hi15].

The query graphs of the standard benchmark queries unfortunately contain only a small number of hyperedges, most of them do not contain any hyperedges at all. Furthermore, the queries are fairly small and can all easily be optimized by a full hypergraph based DP algorithm [NR18]. Nevertheless, large queries with outer joins are a reality we have to deal with [Vo18]. To thoroughly assess our approach also for larger queries with non-inner joins we thus additionally evaluate LinDP++ on a synthetic workload of large randomly generated queries. We use the same set of tree queries used in [NR18] which was generated using the procedure described in [Ne09]. The set contains 100 different random queries per size class. Sizes range from 10 to 100 relations per query, which gives a total of 1,000 queries. Hyperedges are introduced to the queries by randomly adding artificial reordering

Algorithm	TPC-H	TPC-DS	LDBC	JOB	SQLite
DPhyp	0.4	90	1.2	227	2.2K
GOO	0.8	9.5	2.2	13.7	61
linearized DP	1.4	18.7	4.4	33.4	4.7K
LinDP++	1.6	19.9	4.4	36.2	4.7K

Tab. 2: Total optimization time (ms) for standard benchmarks

constraints between neighboring joins. All algorithms were ran single-threaded with a timeout of 60 seconds on a 4 socket Intel Xeon E7-4870 v2 at a clock rate of 2.3 GHz with 1 TB of main memory attached. When comparing the quality of the plans for a query generated by different algorithms we report *normalized costs*, i. e. the factor by which a plan is more expensive than the best plan found by any of the algorithms.

5.1 Hypergraph Handling

The algorithm described in this paper targets query sizes far beyond what exact algorithms with exponential runtime can solve in a reasonable amount of time. Thus we start by analyzing the runtime characteristics of LinDP++.

For completeness reasons we first report numbers for all considered standard benchmarks even though their queries are all rather small and a full graph based DP would be the algorithm of choice here. In Tab. 2 we summarize optimization time of DPhyp, GOO, linearizedDP, and LinDP++ for all considered benchmarks. Most of the queries are optimized almost instantly and optimization times of LinDP++ are comparable to those of linearized DP. The only benchmark where optimization time becomes noticeable is the SQLite test suite, which contains more than 700 queries on up to 64 relations. But even the largest query of the SQLite test suite is optimized by LinDP++ in 27ms. Regarding the quality of the plans generated by LinDP++: 93% of the plans are indeed optimal, 6% of them are suboptimal by at most a factor of 2 and only 10 of the 1159 execution plans are worse than that. Note: we compare plans to the best plan found when considering all valid crossproducts here.

Once queries become more complex and exact optimization becomes infeasible, search space linearization helps to keep optimization times reasonable. With LinDP++ we are now able to linearize the search space of queries with non-inner joins, a class of queries that could not be handled by linearized DP. To see whether this ability to linearize hypergraph queries comes at the expense of optimization performance we compare linearized DP on regular queries with LinDP++ on hypergraph queries with the same number of relations. Figure 9 shows median, minimum and maximum optimization time per size class for linearized DP and LinDP++. The overhead incurred by hypergraph handling is negligible and LinDP++ is just as well able to optimize queries on 100 relations within 100ms on average.

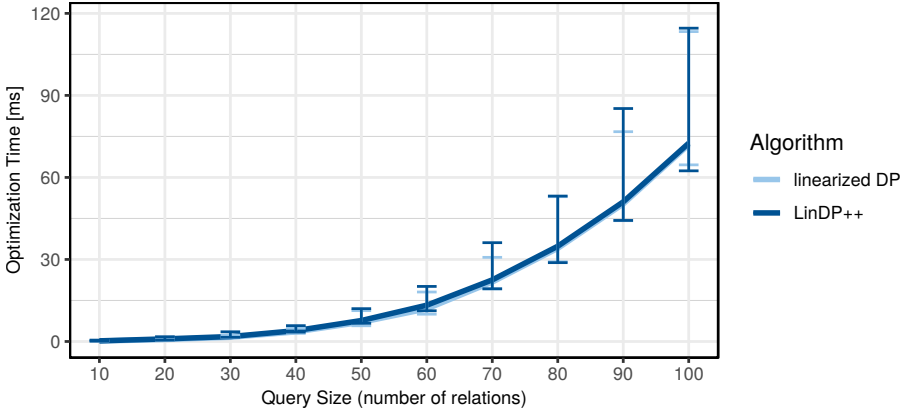


Fig. 9: Median optimization times for LinDP++ on hypergraph queries compared to linearized DP on regular graph queries for queries on up to 100 relations. The error bars indicate minimum and maximum optimization time per size class.

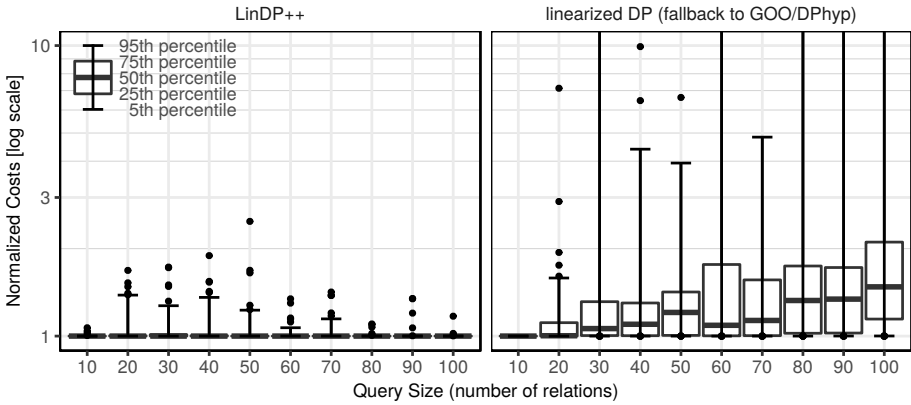


Fig. 10: Distribution of normalized costs of LinDP++ plans compared to the plans the hypergraph aware greedy GOO/DPhyp fallback of linearized DP generates for queries on up to 100 relations.

The original adaptive optimization framework [NR18] had to fall back to the greedy iterative DP with DPhyp as hypergraph aware inner algorithm (GOO/DPhyp) for large queries with non-inner joins. Depending on the structure of the query graph, this could already be the case for hypergraph queries touching as few as 14 relations. Using the generalized LinDP++ technique we can avoid the greediness for these queries and generate significantly better plans. Figure 10 compares the normalized costs of the plans generated by LinDP++ with the ones GOO/DPhyp generates for the synthetic workload with hyperedges. On average, plan costs are within 2% of the best plan and 814 plans are indeed the best known plans for their respective query and normalized costs of 123 plans are within 10% of

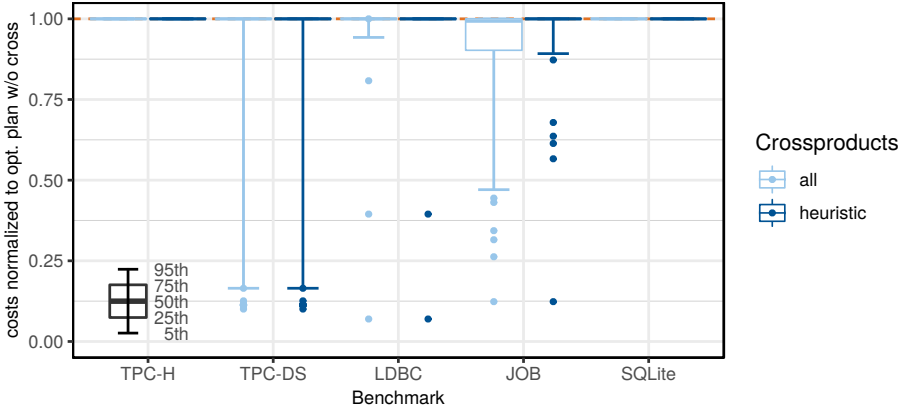


Fig. 11: Normalized costs of plans when either all valid crossproducts or some explicit crossproduct edges as suggested by the heuristic are considered. Costs are normalized to the optimal plan without crossproducts.

the best. The remaining execution plans have normalized costs below 2 with the exception of one query, for which the plan is 2.4 times as expensive as the best known plan. In contrast to that, 139 plans generated by GOO/DPhyp already have normalized cost worse than 2 and costs of 54 plans are worse than the best plan by a factor of 10 or more.

5.2 Crossproduct Benefits

Investigating the effectiveness of the crossproduct heuristic described in Section 4 on the queries of the standard benchmarks shows that crossproducts can indeed improve execution plans. Figure 11 summarizes the costs of plans normalized to the optimal plan without crossproducts. We compare these normalized costs when considering all crossproducts with those when considering only the crossproducts suggested by our heuristic. On average, introducing crossproduct edges improves plan cost by up to 18%, depending on the benchmark. Nevertheless, 90% of the execution plans remain the same even when considering all valid crossproducts. This confirms our statement, that the vast majority of possible crossproducts is irrelevant and should not be considered. However, while plan improvements are minimal for most queries, a maximum cost reduction of a factor of 14.4 reconfirms the claim of Ono and Lohman [OL90], that some crossproducts can significantly improve plan quality. The figure also shows, that our simple heuristic indeed already covers many of the relevant crossproducts. Only the Join Order Benchmark would benefit from additional crossproducts that bypass larger chains of joins (mostly of length 3). Extensively investigating all crossproduct possibilities during join ordering is thus neither required to get good plans nor feasible in terms of optimization complexity.

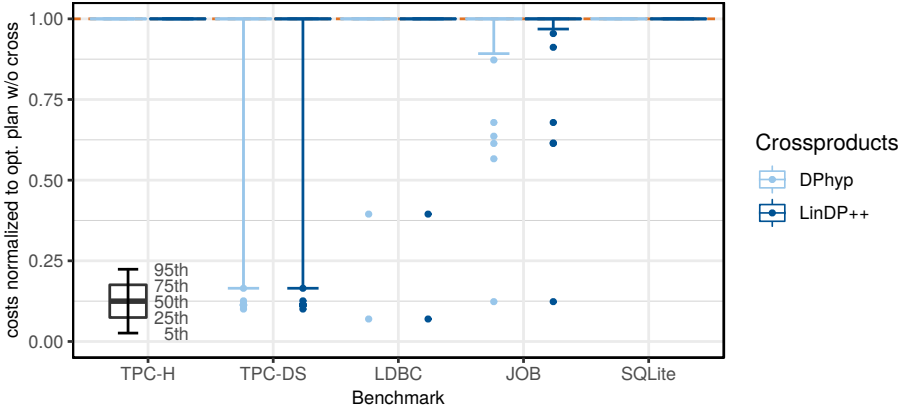


Fig. 12: Normalized costs of plans generated by LinDP++ compared to the optimal plans with some crossproducts as suggested by the heuristic. Costs are normalized to the optimal plan without crossproducts.

crossproducts are never considered during the linearization phase of LinDP++, as they are eliminated when removing cycles from the query graph. However, even though they are ignored by the first phase, the second phase does consider them and plan costs are reduced almost as much as with a full DP algorithm. Figure 12 shows the differences in cost improvements comparing LinDP++ against full DPhyp, both operating on the augmented query graph. Despite the reduced search space, which gives much better optimization times, most of the beneficial crossproducts are discovered and plan costs are within 1% of the DPhyp solutions on average.


6 Conclusion

In this paper we eliminate a severe limitation of the recently proposed adaptive join ordering framework [NR18]. While generating high quality execution plans for many large queries using search space linearization, the framework had to fall back to a greedy heuristic as soon as a large query contained a single outer join. The generalized algorithm LinDP++ described in this paper enables linearization of the search space of arbitrary queries, including those with non-inner joins. We experimentally show that the join orders generated by LinDP++ are clearly superior to those generated by the greedy fallback of the original framework.

LinDP++ in addition is equipped with a fast heuristic to detect promising opportunities to perform a crossproduct. Despite considering some crossproducts, LinDP++ deliberately avoids looking at all crossproducts which would result in exponential search space size. Furthermore, the heuristic ensures that any considered crossproduct obeys all reordering constraints induced by non-inner joins. We demonstrate the effectiveness of this heuristic on the queries of major database benchmarks. The heuristic detects most relevant crossproduct

opportunities while keeping the search space small and thus optimization time reasonable. Our experiments further verify that considering all valid crossproducts is not worth the dramatically increased optimization time, as the additionally considered crossproducts rarely lead to an additional cost reduction.

Even a polynomial time heuristic like LinDP++ becomes too expensive at some point. At that scale, iterative dynamic programming has proven to be an effective technique, as it allows to gracefully tune down plan quality in favor of acceptable optimization times. According to our experiments however, the chosen greedy algorithm – Greedy Operator Ordering (GOO) – seems to also have quality issues with non-inner joins. To ensure high quality execution plans even at that scale we thus would like to investigate alternatives or improve GOO in this setting.

This project has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No 725286). 

References

- [An14] Angles, Renzo; Boncz, Peter A.; Larriba-Pey, Josep-Lluis; Fundulaki, Irini; Neumann, Thomas; Erling, Orri; Neubauer, Peter; Martínez-Bazan, Norbert; Kotsev, Venelin; Toma, Ioan: The linked data benchmark council: a graph and RDF industry benchmarking effort. *SIGMOD Record*, 43(1):27–31, 2014.
- [CM95] Cluet, Sophie; Moerkotte, Guido: On the Complexity of Generating Optimal Left-Deep Processing Trees with Cross Products. In: *Proceedings of ICDT ’95*. pp. 54–67, 1995.
- [Fe98] Fegaras, Leonidas: A New Heuristic for Optimizing Large Queries. In: *Proceedings of DEXA ’98*. pp. 726–735, 1998.
- [FM13] Fender, Pit; Moerkotte, Guido: Top down plan generation: From theory to practice. In: *Proceedings of ICDE 2013*. pp. 1105–1116, 2013.
- [GLP93] Gallo, Giorgio; Longo, Giustino; Pallottino, Stefano: Directed Hypergraphs and Applications. *Discrete Applied Mathematics*, 42(2):177–201, 1993.
- [Gr95] Graefe, Goetz: The Cascades Framework for Query Optimization. *IEEE Data Eng. Bull.*, 18(3):19–29, 1995.
- [Ha08] Han, Wook-Shin; Kwak, Wooseong; Lee, Jinsoo; Lohman, Guy M.; Markl, Volker: Parallelizing query optimization. *PVLDB*, 1(1):188–200, 2008.
- [Hi15] Hipp, R. et al.: SQLite (Version 3.8.10.2). SQLite Development Team. Available from <https://www.sqlite.org/download.html>, 2015.
- [IK84] Ibaraki, Toshihide; Kameda, Tiko: On the Optimal Nesting Order for Computing N-Relational Joins. *ACM Trans. Database Syst.*, 9(3):482–502, 1984.
- [KBZ86] Krishnamurthy, Ravi; Boral, Haran; Zaniolo, Carlo: Optimization of Nonrecursive Queries. In: *Proceedings of VLDB ’86*. pp. 128–137, 1986.

- [KS00] Kossmann, Donald; Stocker, Konrad: Iterative dynamic programming: a new class of query optimization algorithms. *ACM Trans. Database Syst.*, 25(1):43–82, 2000.
- [Le18] Leis, Viktor; Radke, Bernhard; Gubichev, Andrey; Mirchev, Atanas; Boncz, Peter A.; Kemper, Alfons; Neumann, Thomas: Query optimization through the looking glass, and what we found running the Join Order Benchmark. *VLDB J.*, 27(5):643–668, 2018.
- [MFE13] Moerkotte, Guido; Fender, Pit; Eich, Marius: On the correct and complete enumeration of the core search space. In: *Proceedings of SIGMOD 2013*. pp. 493–504, 2013.
- [MN06] Moerkotte, Guido; Neumann, Thomas: Analysis of Two Existing and One New Dynamic Programming Algorithm for the Generation of Optimal Bushy Join Trees without Cross Products. In: *Proceedings VLDB 2006*. pp. 930–941, 2006.
- [MN08] Moerkotte, Guido; Neumann, Thomas: Dynamic programming strikes back. In: *Proceedings of SIGMOD 2008*. pp. 539–552, 2008.
- [MS79] Monma, Clyde L.; Sidney, Jeffrey B.: Sequencing with Series-Parallel Precedence Constraints. *Math. Oper. Res.*, 4(3):215–224, 1979.
- [Ne09] Neumann, Thomas: Query simplification: graceful degradation for join-order optimization. In: *Proceedings of SIGMOD 2009*. pp. 403–414, 2009.
- [NR18] Neumann, Thomas; Radke, Bernhard: Adaptive Optimization of Very Large Join Queries. In: *Proceedings of SIGMOD 2018*. pp. 677–692, 2018.
- [OL90] Ono, Kiyoshi; Lohman, Guy M.: Measuring the Complexity of Join Enumeration in Query Optimization. In: *Proceedings of VLDB 1990*. pp. 314–325, 1990.
- [Se79] Selinger, Patricia G.; Astrahan, Morton M.; Chamberlin, Donald D.; Lorie, Raymond A.; Price, Thomas G.: Access Path Selection in a Relational Database Management System. In: *Proceedings of SIGMOD 1979*. pp. 23–34, 1979.
- [SMK97] Steinbrunn, Michael; Moerkotte, Guido; Kemper, Alfons: Heuristic and Randomized Optimization for the Join Ordering Problem. *VLDB J.*, 6(3):191–208, 1997.
- [Sw89] Swami, Arun N.: Optimization of Large Join Queries: Combining Heuristic and Combinatorial Techniques. In: *Proceedings of SIGMOD 1989*. pp. 367–376, 1989.
- [TK17] Trummer, Immanuel; Koch, Christoph: Solving the Join Ordering Problem via Mixed Integer Linear Programming. In: *Proceedings of SIGMOD 2017*. pp. 1025–1040, 2017.
- [Tra17a] Transaction Processing Performance Council. TPC Benchmark DS, 2017.
- [Tra17b] Transaction Processing Performance Council. TPC Benchmark H, 2017.
- [VM96] Vance, Bennet; Maier, David: Rapid Bushy Join-order Optimization with Cartesian Products. In: *Proceedings of SIGMOD 1996*. pp. 35–46, 1996.
- [Vo18] Vogelsong, Adrian; Haubenschild, Michael; Finis, Jan; Kemper, Alfons; Leis, Viktor; Mühlbauer, Tobias; Neumann, Thomas; Then, Manuel: Get Real: How Benchmarks Fail to Represent the Real World. In: *Proceedings of DBTest@SIGMOD 2018*. pp. 1:1–1:6, 2018.
- [WC18] Wang, TaiNing; Chan, Chee-Yong: Improving Join Reorderability with Compensation Operators. In: *Proceedings of SIGMOD 2018*. pp. 693–708, 2018.
- [WP00] Waas, Florian; Pellenkoft, Arjan: Join Order Selection - Good Enough Is Easy. In: *Proceedings of the 17th BNCOD*. pp. 51–67, 2000.

Waves of misery after index creation

Nikolaus Glombiewski¹, Bernhard Seeger², Goetz Graefe³

Abstract: After creation of a new b-tree, the ordinary course of database updates and index maintenance causes waves of node splits. Thus, a new index may at first speed up database query processing but then the first “wave of misery” requires effort for frequent node splits and imposes spikes of buffer pool contention and of I/O. Waves of misery continue over multiple instances although eventually the waves widen, flatten, and spread further apart. Free space in each node left during index creation fails to prevent the problem; it merely delays the onset of the first wave. We have found a theoretically sound way to avoiding these waves of misery as well as some simple and practical means to reduce their amplitude to negligible levels. Experiments demonstrate that these techniques are also effective. Waves of misery occur in databases and in key-value stores, in primary and in secondary b-tree indexes, after load operations, and after b-tree reorganization or rebuild. The same remedies apply with equal effect.

Keywords: Indexing, Bulk Loading, B-tree

1 Introduction

The purpose of adding an index to a database table is to improve the performance of query processing. Unfortunately, after an initial “honey moon” of using a new index and of enjoying fast queries, the index may grow and require many node splits – thus creating a spike in buffer pool contention and I/O activity. In other words, the index may actually reduce query performance or at least fail to achieve the expected performance. Consider the following example. First, a new secondary b-tree index enables fast look-ups and fast ordered scans. In order to absorb updates without node splits, each node might start with 10% free space – often a parameter of index creation. However, once the table and the secondary index approach 10% growth, many of the b-tree leaf nodes require splitting. Thus, there is a wave of split activity with contention for the data structures for free space management, for the buffer pool, and for the I/O devices. Once most of the original index leaves are split, this activity subsides until the table and index grow above two times the original contents, whereupon another wave of splits happens. During each wave of splits, both update and query performance suffer, as do buffer pool contention and space utilization

¹ Department of Mathematics and Computer Science, University of Marburg, 35032 Marburg, Germany, glombien@mathematik.uni-marburg.de

² Department of Mathematics and Computer Science, University of Marburg, 35032 Marburg, Germany, seeger@mathematik.uni-marburg.de

³ Google Inc., Madison WI, USA, goetzgraefe@gmail.com

in memory and on storage. Figure 1 illustrates the effect in a b-tree with 8 million initial index entries and insertion batches of 10,000 entries, with leaf splits per insertion batch varying from 0 to over 600. The details of the experiment are given in the experimental section. In order to reduce these wave of splits, alternative strategies for the initial bulk

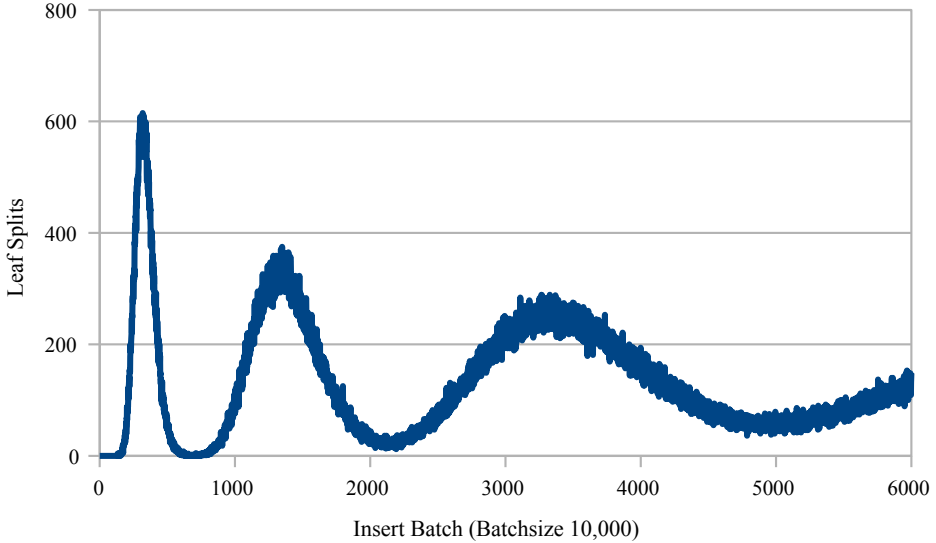


Fig. 1: Waves of Misery

loading of b-trees are needed. The main contributions of this paper are a theoretical analysis and solution for reducing splits as well as a variety of practical solutions that can easily be integrated into existing bulk-loading techniques.

In the remainder of this paper, the next section reviews related prior work. After a section illuminating the extent of the problem, two sections suggest both theoretically sound remedies and simple practical remedies or approaches to the problem. An evaluation section is then followed by a summary and suggestions for future research.

2 Related prior work

This section reviews relevant aspects of the well-known b-tree index structure. Knowledgeable readers may feel free to skip ahead.

2.1 B-trees

B-tree indexes are ubiquitous in databases, key-value stores, file systems, and information retrieval. There are many variants and optimizations [Gr11]; we review only those most

relevant to the new techniques proposed below. Suffice it to say that b-tree keys can be multiple columns (compound indexes), hash values (ordered hash indexes enable efficient creation, effective interpolation search, and easy phantom protection), space-filling curves (spatial indexes for multi-dimensional data and query predicates), or heterogeneous keys (merged indexes or master-detail clustering); and that both keys and values can be compressed, e.g., as bit vector filters instead of a set of row identifiers. B-tree creation benefits from sorting but sorting can also benefit from b-trees: runs in external merge sort in the form of b-trees, or even all runs within a single partitioned b-tree [Gr03] or a linear partitioned b-tree, enable effective read-ahead during a merge as well as query processing while a merge step is still incomplete. This is the foundation of log-structured merge-trees, discussed later. Sorted data permit efficient binary search; b-trees cache in their root node the first few keys inspected in any binary search (or reasonably good representatives for those key values), in the root's immediate children the next few keys inspected in a binary search, etc. In order to maximize the caching effect by minimizing key sizes, Bayer and Unterauer suggested suffix truncation when splitting a leaf node: rather than split rigidly into halves, i.e., at the 50% key, choose the shortest possible key value separating, say, the 40% and 60% keys [BU77]. Similar split techniques are also known from UB-trees [Ra00] and bulk-loading of R-trees [ASW12].

In order to avoid node splits immediately after index creation, e.g., during the first insertion into a new index, many implementations leave free space within each b-tree node. For a pretty typical example, Microsoft SQL Server supports two free space parameters during creation and reorganization of a b-tree index. The first one controls how much free space is left within each b-tree node, both leaf nodes and branch nodes (except along the right edge of the b-tree). The second parameter controls the number of empty pages, or more specifically the percentage of empty pages in the sequence of b-tree pages. These empty pages will be allocated during node splits. Leaving empty pages in this way ensures fast scans on traditional hard disk drives even after many insertions and node splits. SQL Server does not support free space fractions different for leaf and branch pages or free space fractions per key range.

In addition to the 'fill factor' option, the 'create index' statement in Microsoft SQL Server includes the option 'pad index,' which specifies whether to apply the fill factor not only to leaf nodes but also to branch nodes. Sybase supports the 'fill factor' option and a related option 'max rows per page.' The Oracle database and IBM DB2 support an index option 'percent free,' which is the complement to the fill factor in SQL Server. The Oracle database also supports many related options regarding the number of pages reserved for future index growth. Many other database systems support the Oracle or IBM syntax for free space within indexes. The aspect important in the present context is that all products support a fixed amount of free space in all index leaves, which causes waves of node splits.

2.2 Uneven page utilization and fringe analysis

Instead of using a fixed amount of free space, more flexible rules for the initial assignment of records to pages are required to mitigate or even avoid the waves of misery. A theoretically sound rule presented in this paper dates back to an analytical framework called fringe analysis [Ya78] that is originally used to prove the expected storage utilization of b-trees and other search trees. The framework of fringe analysis has been elaborated in [Ei82] and later applied to the analysis of various versions of b-trees [BL89]. However, none of the previous studies addressed the problem of bulk loading and the waves of misery identified in our work. Closely related to fringe analysis is spiral hashing [Ma79] where the utilization of the pages follows a logarithmic pattern. This avoids the undesirable oscillating search performance of linear hashing [La88] because the page with the highest expected load is always split next. However, the goal of spiral hashing is to guarantee a constant expected search performance while our approach addresses the problems of constant insertion performance and buffer contention.

2.3 Write-optimized b-trees

Traditional b-trees incur a high write penalty: modifying a single byte, field, or record forces a random page write to persistent storage. Write-optimized b-trees [Gr04] attempt to reduce this cost by turning the random write into a sequential write. More specifically, they employ append-only storage at page granularity, tracking new page storage locations within the b-tree structure, i.e., each page movement requires a pointer update in a parent page. Thus, write-optimized b-trees encompass the most crucial function of a flash translation layer (FTL). They do not permit neighbor pointers among sibling nodes; nonetheless, they permit comprehensive consistency checks (online/continuous or offline/utility) by fence keys in each page, i.e., copies of branch keys in ancestor pages. Write-optimized b-trees can suffer from poor scan performance on storage device with high access latency (e.g., seek and rotation delays). A possible optimization divides the key range into segments, stores each segment in continuous storage, and recycles replaced pages within a segment. The key range per segment may be dynamic, just as the key range per leaf node is dynamic in a traditional b-tree. The resulting design combines elements and advantages of write-optimized b-trees and of O'Neil's SB-trees [ON92]. On flash storage, segments may coincide with an erasure block.

2.4 Log-structured merge forests

Write-optimized b-trees optimize I/O patterns but still employ update-in-place within b-tree nodes (database pages). Log-structured merge-trees [ON96] employ random access only within memory but write to persistent storage only in append-only sequential patterns, each

page containing only new records. Thus, it seems that log-structured merge-trees solve the problem of write amplification. Log-structured merge indexes turn updates into insertions of replacement records and deletions into insertions of “tombstone” or “anti-matter” records. With only insertions remaining, the principal algorithm is that of an external merge sort, with runs in b-tree format and with the merge pattern optimized for both sorting and querying. On one hand, a high fan-in and few merge levels permit efficient sorting; on the other hand, as queries must search each one in the set (forest) of b-trees, efficient query processing demands eager merging and thus a low fan-in in each merge step. Bit vector filtering may reduce the number of b-trees a specific query needs to search, but merging in log-structured merge trees is generally less efficient than in external merge sort. In fact, small and sub-optimal merge fan-insertion can multiply the number of merge levels and thus the amount of I/O compared to an optimal merge sort. A second principal weakness of existing designs and implementations is that log-structured merge forests induce bursts of system activity in the form of merging. In other words, they don’t avoid “waves of misery” but merely replace one kind with another [Gr19].

2.5 Summary of related prior work

Among traditional b-trees, write-optimized b-trees, and log-structured merge-trees, traditional b-trees and write-optimized b-trees offer the best query latency (single partition search), traditional b-trees and log-structured merge-trees the best storage utilization, write-optimized b-trees and log-structured merge-trees the highest write bandwidth (sequential writes only), and log-structured merge-trees the highest insertion bandwidth. Traditional b-trees and write-optimized b-trees suffer from the waves of misery addressed here; log-structured merge-trees have different waves of misery.

3 Problem assessment

While the introduction describes the problem in general terms and illustrates that the problem indeed exists, the present section illustrates when the problem occurs, when it does not, what characteristics or key value distributions cause the problem, etc. The following sections offer a deeper understanding as well as one particular solution; the following section offers more practical approaches and solutions. For b-tree leaf splits to occur in substantial frequency, the index (and its underlying table or data collection) must grow. A b-tree with little update activity and with little growth do not exhibit the issues discussed here. Little or moderate update activity (without overall growth) can be absorbed without splits if the initial index creation left free space in each node, as discussed in the preceding section. On the other hand, most databases have at least some tables and indexes that capture continuous business activity, whether those are banking transactions, web activity, sensor readings from internet-of-things devices, or other types of events. For b-tree leaf splits to occur in discernable waves, the key value distributions in the initial b-tree contents and in the

set of insertions must match. The higher their correlation, the sharper the waves. It is not required that this distribution be uniform or any other particular form. Figure 2 illustrates these points. If both initial key value distribution and insertions follow a normal or uniform distribution, there are fairly sharp waves of node splits that even overlap in Figure 2. If, on the other hand, their distributions differ, the waves are much less distinct. In a b-tree on

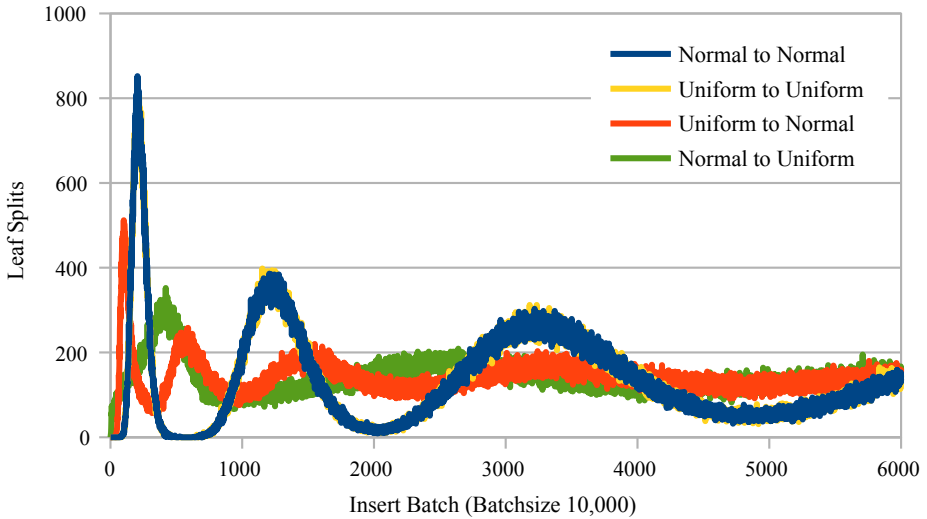


Fig. 2: Development of leaf splits for varying key distributions

hash values, a high correlation between these distributions is extremely likely. B-trees on hash values offer several advantages (over other traditional hash indexes), e.g., efficient creation after sorting future index entries, simple implementation of phantom protection (serializable concurrency control) by locking gaps between existing index entries, efficient merge joins after index scans, and more. Note that b-tree nodes with near-uniform key values permit interpolation search (instead of binary search) and that a b-tree root and its immediate descendent nodes can be cached as a single super-large node in memory. Thus, b-trees on hash values offer advantages but suffer from waves of leaf splits just as much as other b-trees, perhaps even more so because hash functions are specifically designed to produce uniform distributions and thus equal distributions in initial contents and insertions.

While free space left in each new b-tree node during index creation can absorb moderate update activity, it cannot absorb long-term growth. Initial free space merely delays the first wave of node splits, and it perhaps widens and weakens the first and subsequent waves, but it does not prevent them. Figure 3 highlights this point: even with as little as 50% initial space utilization (and thus also 50% free space on each index leaf), distinct waves occur. Thus, the facilities found in commercial relational database products and their commands for index creation do not prevent the waves of node splits addressed in the following sections.

The diagram in Figure 4 illustrates that index size matters. In a very small index (e.g., 1,000

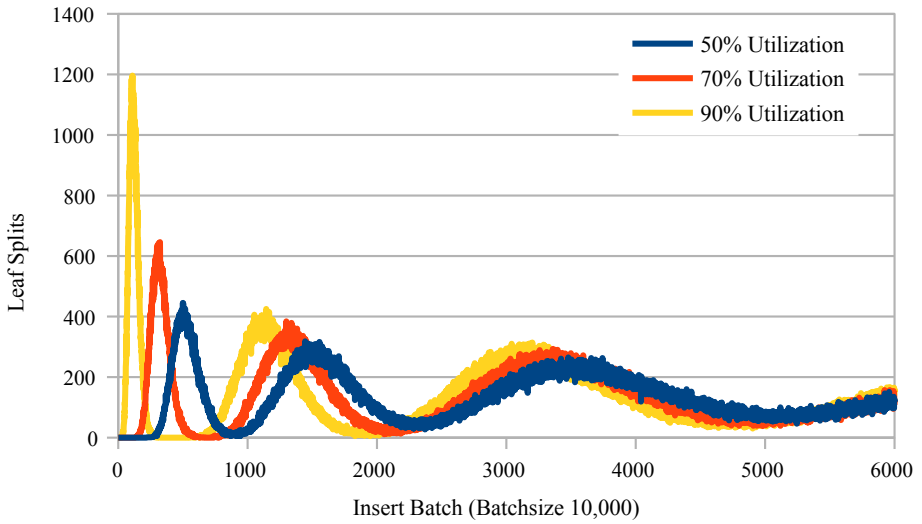


Fig. 3: Development of leaf splits for varying initial page utilizations

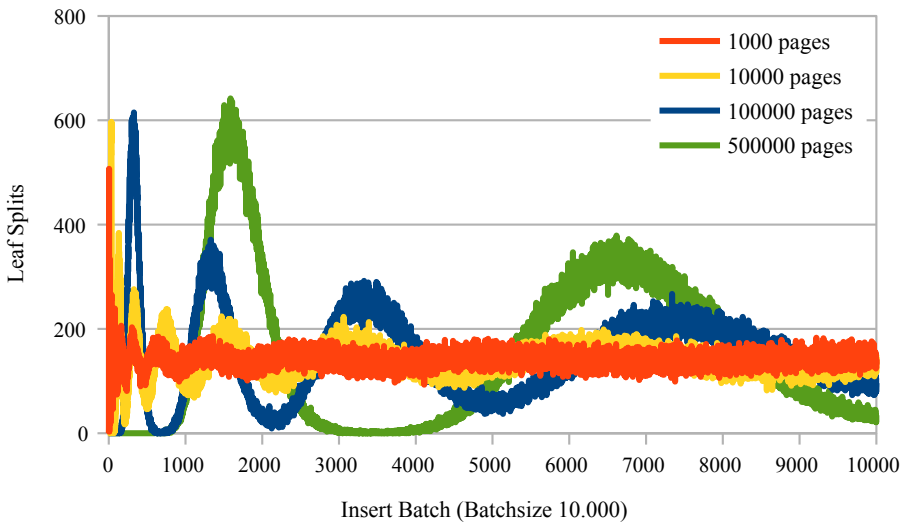


Fig. 4: Development of leaf splits for varying initial dataset sizes

leaf nodes), batches of 10,000 insertions force waves of node splits in rapid succession. They widen and weaken after a few waves. In contrast, in very large index (e.g., 500,000 leaf nodes), many batches are required to fill the initial free space in each node, but then there are distinct, high, and wide waves of node splits impacting database components such as the buffer pool for a longer period of time.

4 Sound remedies

The goal of this section is to provide a theoretically sound solution for loading b-trees for which succeeding insertions will trigger a split with a constant probability. Thus, there are no waves of misery. We limit our discussion first to the leaf level and later extend it to the upper levels of the tree. We assume records of constant size such that every leaf has a capacity of B records. For the sake of readability (and because of space limitations) let B being odd. Thus, a split of an overflowing page results into two pages each with $(B+1)/2$ records. A record has a unique comparable key that provides the ordering required for the b-tree.

The basic idea of the following approach is to keep the filling degree of the leaf pages in balance. Given a b-tree with n records, let $fd_i(n)$ denote the number of pages with i records, $(B + 1)/2 \leq i \leq B$. Then, the probability that the next insertion triggers a split is $B * fd_n(B)/(n + 1)$. As already shown in the introduction, this probability provides an oscillating behavior, our waves of misery, in case of extreme differences among $fd_i(n)$. It is also easy to see that the other extreme of a uniform fill degree with $fd_i(n) = fd_j(n)$, $(B + 1)/2 \leq i, j \leq B$ and $i \neq j$, is not the theoretically sound solution because there are too many full pages and too less pages with a low utilization. The sound solution is somewhere between these two extremes such that the number of less populated pages is higher than the one of more populated ones.

A key idea of our algorithm is to employ the steady-state probability p_j that a page with j records occurs in the sound solution, $(B + 1)/2 \leq j \leq B$. In an iterative manner, the algorithm randomly determines j with probability p_j , $(B + 1)/2 \leq j \leq B$, and assigns the next j records from the input to a new page. The iteration stops when the number of remaining records is less than $(3B + 1)/2$. Then the algorithm performs in the following way. If the remaining number of records is at most B , these records are stored in one leaf. Otherwise, two pages are created over which the records are equally distributed. Overall, this algorithm guarantees that all pages are at least half full as required for a traditional b-tree.

In order to analyze the split occurrence of the b-tree after loading, we assume equal distributions such that the distribution of insertions mirrors the key value distributions of the records in the b-tree. More formally, a b-tree with n records partitions the key domain into $n + 1$ empty intervals and the probability an insertion hitting such an empty interval is

$1/(n + 1)$. Note that this model is entirely different from the restrictive uniform distribution assumption.

Theorem 1 *Let us consider a b-tree with N records being created by our loading algorithm such that the probability q_j of the next insertion hitting a page with j records is $\frac{1}{\alpha(j+1)}$. Here, $\alpha = H_B - H_{(B+3)/2}$ denotes a weight parameter, where $H_j = \sum_{1 \leq i \leq j} 1/i$ is the partial harmonic series of size j . Furthermore, records are continuously inserted into the initial b-tree assuming equal distributions (i.e., insertion distribution mirrors the initial loading distribution). Then, the following two statements are fulfilled:*

- *First, the probability that a record insertion triggers a split of a leaf is constant, i.e., there are no waves of misery.*
- *Second, the probability p_j that a leaf consists of j records is given by $p_j = \frac{B \cdot \ln(2)}{j(j+1)}$.*

Sketch of the proof: The proof follows the basic ideas of fringe analysis [BL89; Ei82; Ya78]. In the following, we provide a rough summary and refer the interested reader to the original literature. For the following fringe analysis, we restrict our discussion to the leaf pages of the b-tree. The number of records partition the leaves into classes C_i , $(B + 1)/2 \leq j \leq B$. Class C_i consists of all the pages with i records. For each class C_i , $f_i(n)$ denotes a counter for the corresponding number of leaf pages in a b-tree with n records. Let $q_i(n)$ denote the probability that an insertion will be in a page of class C_i . It follows that $q_i(n) = i \cdot f_i(n)/(n + 1)$ since there are a total of $i \cdot f_i(n)$ records in pages of class C_i and every record is a left boundary of an empty interval where the next insertion may occur with probability $1/(n + 1)$. Note that we make a negligible simplification because the first interval has not a record as its left boundary. All these probabilities are collected in a vector $\vec{q}(n)$. The basic idea of the fringe analysis is to keep track of this vector over a sequence of insertions. By using a quadratic transition matrix T with $(B+1)/2$ columns and rows it is possible to compute $\vec{q}(n + 1)$ from $\vec{q}(n)$ in a recursive way ([Ei82]):

$$\vec{q}(n + 1) = \left(I + \frac{1}{n + 1} T \right) \times \vec{q}(n) \quad (1)$$

Here I denotes the identity matrix. The steady-state $\vec{q} = (q_{(B+1)/2}, \dots, q_B)$ of this recursive equation is then given by the solution of $T \times \vec{q} = \vec{0}$. In [Ei82], it is shown that this results in

$$q_j = 1/(j + 1) \quad (2)$$

for $j = (B + 1)/2, \dots, B$. Because the leaf pages are filled in our initially loaded b-tree with N records such that the probability $\vec{q}(N + 1) = \vec{q}$, it follows that $\vec{q}(N + k) = \vec{q}$ for $k = 1, 2, \dots$. In particular, the probability $q_B(N + k) = q_B$ remains constant for $k = 1, 2, \dots$ and so does the probability of a split. Thus, the first statement of Theorem 1 is fulfilled.

For the proof of the second statement, we make use of $q_j(n) \cdot (n + 1)/j$ being the expected

number of buckets with j records [Ei82]. Moreover, $(n + 1)/(B \cdot \ln 2)$ is the expected number of pages in a b-tree with n records because the expected storage utilization of a b-tree is $\ln 2$. The corresponding quotient of the two expected values is then $B \cdot \ln(2)/j(j + 1)$. In fact, this quotient is asymptotically equal to p_j , the probability of a page comprising j records [BL89]. Thus, the second statement is also fulfilled, and our iterative loading algorithm is equipped with a theoretically sound rule for determining the filling degree of the next page.

So far, the discussion is limited to the leaf level only. Waves of misery still may occur for the index levels when a threshold-based loading approach is used. However, it is easy to see that our algorithm is also directly applicable to every index level. Also note that the algorithm runs online on an input stream, and therefore, the entire tree, including all index levels, is built from left to right as it is known from the standard loading algorithm of b-trees.

The probability vector introduced in Equation 2 is also known as the steady-state solution of the recursive equation. It is the foundation to show that the expected storage utilization is $\ln 2$ [Ya78]. Thus, the loading algorithm also creates b-trees with an expected storage utilization of $\ln 2$. This result can be guaranteed by changing the algorithm to a deterministic one where the ratio of pages with i records, $(B + 1)/2 \leq i \leq B$ exactly matches the probabilities. Then, the loaded b-tree guarantees a storage utilization of $\ln 2$. Unfortunately, there is no steady-state solution with a higher storage utilization for the loaded b-tree. In order to obtain a higher storage utilization and a b-tree without waves of miseries, it is required to change the split policy of the b-tree by using partial expansions or elastic pages [BL89]. This is a direct extension of our approach and is not further elaborated in the paper.

The analysis in this section assumes that the key value distribution in the set of insertions mirrors that of initial b-tree contents. Moreover, it assumes that an overall space utilization of $\ln(2) \approx 69\%$ is sufficient for the application. Therefore, the following section offers some practical approaches that do not require these assumptions. The subsequent section offers an evaluation of both the techniques above and those below.

5 Practical remedies

With waves of nodes splits occurring in practice, although often not recognized, practical remedies are required that fit into scalable data center operations. For example, limiting initial or steady-state storage utilization to 69% is unacceptable. Therefore, many b-tree implementations attempt load balancing before splitting two nodes into three; and many database administrators frequently reorganize (rebuild) their indexes to reset free space per node to, say, 10%. Note that splitting 2-to-3 or even k to $k+1$ does not avoid waves of misery.

Even if the goal is to achieve, say, 10% free space in each node, it is not required to achieve 10% free space rigidly and precisely. Instead, the same overall objective is accomplished by leaving a different amount of free space in each node, chosen from a uniform random distribution from 0% to 20%, from 5% to 15%, or anything similar.

Another possible approach is to make this very systematic: 1st node 0% free space, 2nd node 1%, etc. to 21st node 20%; then restarting with 0% free space, etc. This approach favors some range queries and disfavors others, so an alternative assigns 0% free space to the 1st node, 20% to the 2nd node, 1% to the 3rd node, 19% to the 4th node, etc.

An entirely different approach assigns free space not numerically but by following the logic of suffix truncation (suffix compression) [BU77]. Originally conceived for splitting nodes in the middle, it can be adapted to index creation and reorganization when dealing with non-numeric keys. For example, instead of leaving precisely 10% free space, the key values at 80% and at 100% are compared, the shortest possible key value for the b-tree branch node determined, and the key values distributed to the current and next node according to this branch key. If there is no correlation between the shortest key value and its position, this approach promises a distribution of node utilization similar to the random approach above, with the compression effect added.

Finally, multiple of these techniques may be combined. For example, suffix truncation for the 1st node may look at the key values at the 80% and 100% positions, for the 2nd node at 70% and 90%, for the 3rd node at 79% and 99%, for the 4th node at 71% and 91%, etc. Other combinations may also be possible.

6 Evaluation

All experiments were conducted on a workstation equipped with an AMD Ryzen7 2700X CPU (8 cores, 16 threads) and 16GB of memory, running an Ubuntu Linux (18.04, kernel version 4.16). All of strategies were implemented using the Java indexing library XXL [Se01]. If not stated otherwise the b-tree was initially loaded using 100,000 pages of 8KB. The tree holds records consisting out of 21 integer values (totaling 84 bytes) of which 20 values are uniformly random. The last value is the key and is being randomly sampled according to a normal probability distribution. The loading phase is succeeded with 60 million record insertions. The results are shown based on the statistics of a batch of 10,000 insert operations. Having batches of 10,000 records is not uncommon and allows for a reduction of statistical noise. In the following, the practicality of the sound remedies, which are ideal for 69% utilization, are evaluated first. Afterwards, the various practical remedies also suitable for higher utilization requirements are analyzed.

6.1 Sound Remedies

To get a better understanding of the meaning of the sound remedies, the distribution of free space among all pages for an ideal solution (i.e., a b-tree in a steady state) and the proposed algorithm of the sound remedies section need to be evaluated. Naturally, in the ideal case, the overall space utilization amounts to $\ln(2) \approx 69\%$. Figure 5 depicts the results based on

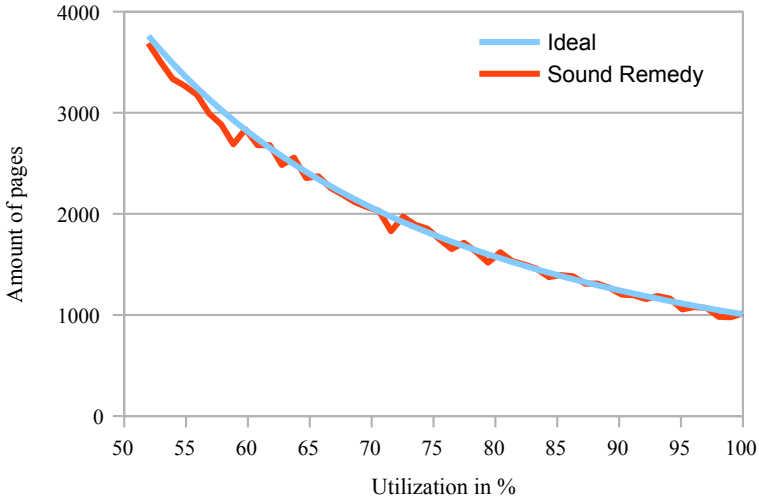


Fig. 5: Initial Page Distributions

the different utilization (x-axis) and the amount of pages having them (y-axis). The ideal solution shows the basic notion of the steady state: There are a few pages ready to be split right away (100% utilization, no free space), a lot pages not even near their split point (50%) and having all other utilization values in between them represented in the tree on a gradual slope. The sound remedy has the same features, but features some slight spikes in the slope. This is due to the asymptotic nature of the algorithm and its inherent randomness - utilization is picked at random based on the described probability function. The latter is an important feature of the algorithm due to range queries: If the slope is perfect, a certain range of pages will have more free space than other pages. Thus, those ranges require more pages to be read.

As shown in Figure 6, the sound remedy also performs well in practice, if its assumptions are met. The x-axis represents the progression of insert batches while the y-axis shows the amount of leaf splits measured during each batch. While the constant strategy of having each page having the same utilization shows clear waves of splits, the sound remedy begins in the steady state and never leaves it.

6.2 Practical Remedies

Based on Section 5, three loading strategies were implemented that can easily be adapted into most systems in order to reduce the waves of misery. The most important parameters are the amount of data items to load and a target number of pages along with the overall space utilization. The analysis for each of the strategies is presented in turn. First, results for the *linear strategy* that systematically assigns free space to each page are shown. Second,

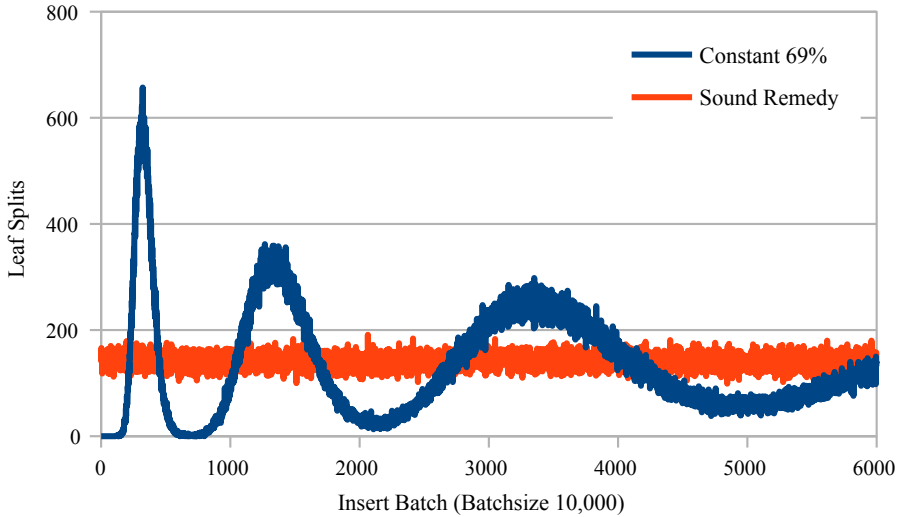


Fig. 6: Leaf Splits over time for ideal theoretical conditions (69% page utilization)

results for the *random strategy* that randomly assigns free space are presented. Third, a combination of both those strategies (*hybrid strategy*) is evaluated. Afterwards, the impact waves of misery can have on the buffer pool are discussed. The final subsection features the results on range query processing. As with the sound remedies, the constant strategy of uniformly loading each page with the same space utilization serves as a baseline for each evaluation.

6.2.1 Linear Strategy

The linear strategy assigns pages according to a linear function starting at 100% page utilization. The gradient and lower end of the function is computed based on the desired utilization and the amount of pages. In order to support range queries, values from the higher and lower end of the function are picked in an alternating fashion until both ends meet. At this point, all data has been successfully loaded into the tree. Figure 7 compares the impact of insertion of various initial utilization (70%, 80% and 90%) for the linear strategy. The y-axis shows the leaf splits while the x-axis indicates the total amount of batches inserted into the b-tree. Even at a high utilization of 90%, the linear strategy showcases slightly better leaf split performance than the constant strategy at 70%. Naturally, this means that achieving a similar split behavior requires less overall storage space for the same amount of data. Furthermore, the lower the utilization, the more the impact of waves is reduced. Even at 80%, the behaviour is somewhat close to the steady state and there are only short bursts of splits present.

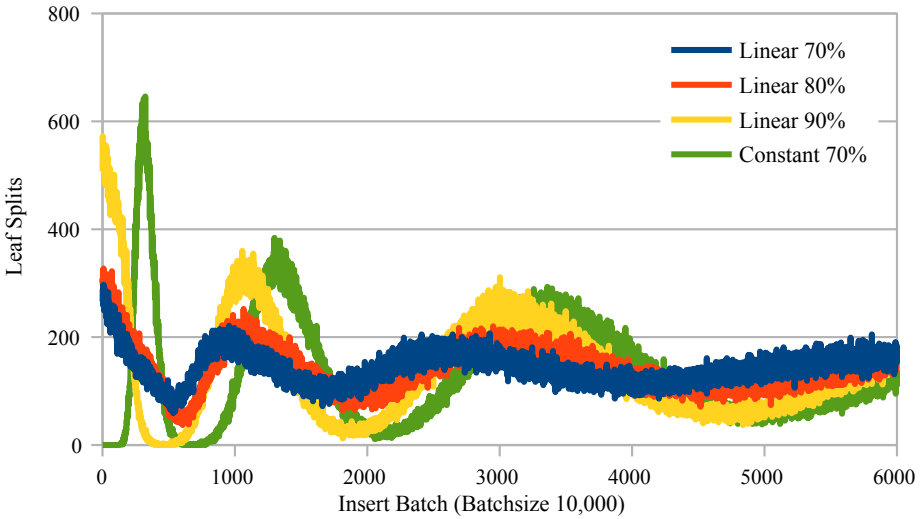


Fig. 7: Leaf Splits over time for linear strategies

6.2.2 Random Strategy

The random strategy chooses the page utilization of each successively loaded page randomly around the range of the given overall desired storage utilization. Due to randomness, the final page count may differ from the expected page count given the desired utilization, but those differences are expected to be negligible. Results presented in Figure 8 are based on a desired utilization of 80% with the same axis labeling as in Figure 7. The different lines present a varying range from which the random function chooses, i.e., *Random 5% Range* picks values from about 75% to 85% utilization while *Random 20% Range* picks values from 100% to 60%. A small range of 5% shows surprisingly little benefit compared to the constant strategy. However, wider ranges improve the desired effect significantly and temper the waves of misery.

6.2.3 Hybrid Strategy

Both the linear strategy and the random strategy can significantly reduce the waves of misery and thus offer a good solution to implement in most systems. However, the strategies still show some slight waves when compared to the sound remedies at 69%. This is due to their initial load distribution. Similar to Figure 8, the results in Figure 9 showcases the page utilization distributions for the random and linear strategies in comparison to the ideal at 69% overall utilization. The utilization is on the x-axis and the amount of pages having it on the y-axis. The best result for the random strategy requires a random range of 31%, resulting in pages with less than 50% utilization. On the flip side, a lower range of 10%

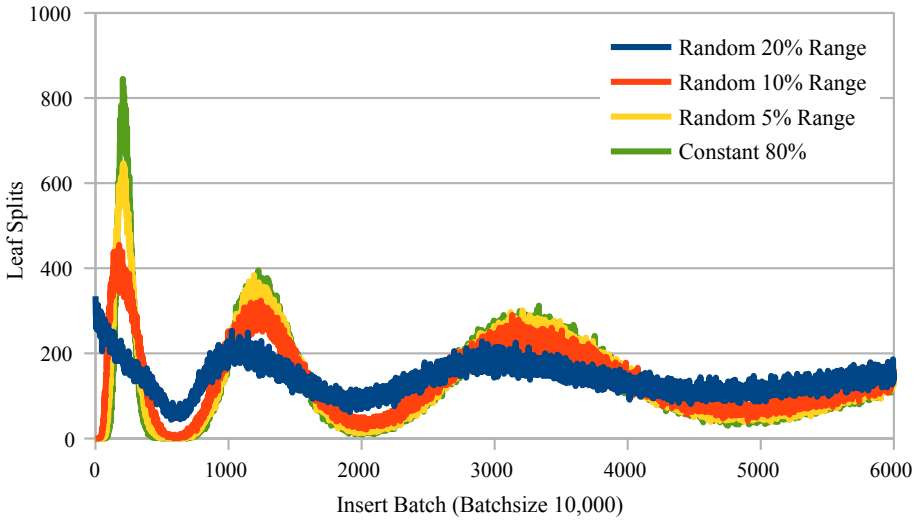


Fig. 8: Leaf Splits over time for random strategies

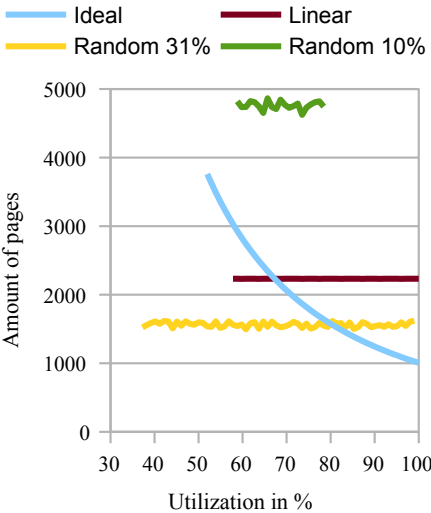


Fig. 9: Comparing the ideal initial page distribution with linear and random strategies

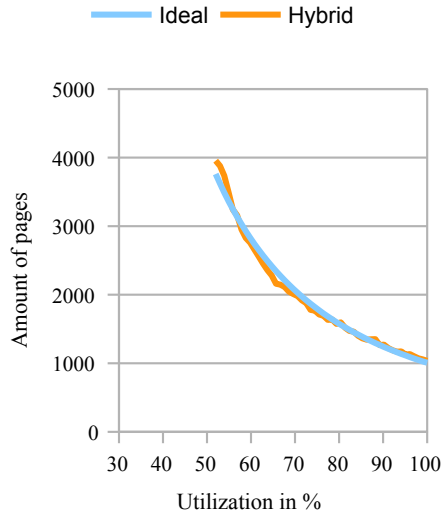


Fig. 10: Comparing the ideal initial page distribution with the hybrid strategy

does not lead to underful pages, but does not feature the overall range of different page utilization required for steady split rates. While the linear strategy features the right range of page utilization, their distribution is constant and thus not ideal.

To mitigate these deficiencies, a hybrid strategy which combines both ideas from linear and random was implemented. In order to reduce the rigid constant distribution of the linear strategy while keeping its overall range, only half of the pages are loaded according to this strategy. For the other half, randomness is applied as follows: Utilization is first randomly chosen between 50% and 100%. Afterwards, this range is narrowed linearly (i.e., lesser filled pages become more likely) until the bulk loading is complete. The resulting initial page utilization distribution is depicted in Figure 10. Clearly, combining those multiple simple strategies leads to a good approximation of the ideal solution.

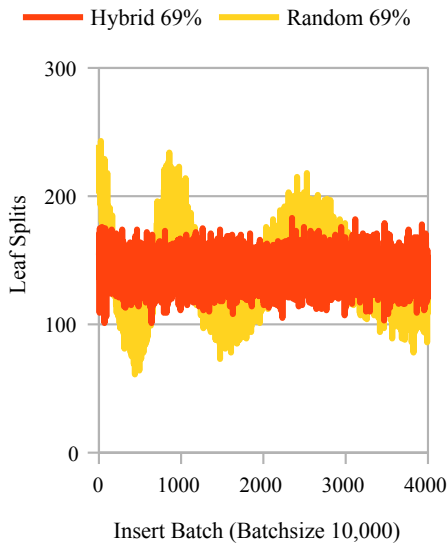


Fig. 11: Leaf Splits over time for hybrid strategy at 69% page utilization

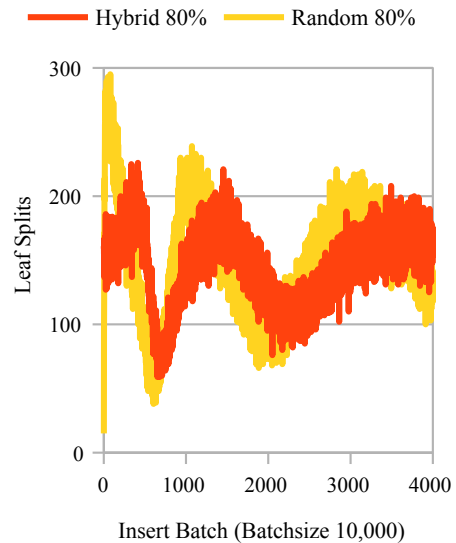


Fig. 12: Leaf Splits over time for hybrid strategy at 80% page utilization

Figure 11 verifies this statement by showcasing results for the insertion experiment for the first 4,000 insert batches at an overall target page utilization of 69%. The random strategy using a range of 31%, the best candidate for its class in terms of splits, is used as a point of comparison. The hybrid strategy outperforms it and is in the steady state from the get-go. The same experiment was repeated for 80% utilization (Figure 12). Since the b-tree does not have the theoretically ideal target utilization, the hybrid strategy does suffer from slight waves. However, it is still able to outperform the random strategy and overall has significantly reduced the impact of splits. Furthermore, while this experiment focuses on numeric keys, the hybrid strategy can be adjusted for non-numeric keys by using suffix truncation instead of randomly choosing keys within the ranges.

6.2.4 Buffer Pool Utilization

Up until now, the evaluation focused on split frequencies, which are a natural indicator of I/Os and buffer pool contentions. To showcase the effects on the actual buffer, the experiments were verified by deploying an LRU buffer and measuring its statistics. To analyze leaf splits separately, separate buffers for inner and leaf nodes were used.

The first experiment utilizes a buffer with 10,000 pages and compares the sound remedy strategy with a constant strategy with a target utilization of 69%. Figure 13 showcases

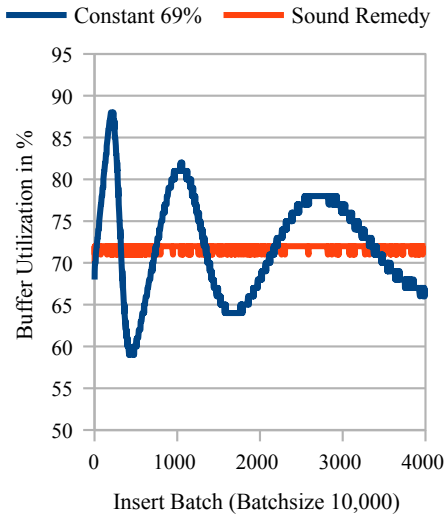


Fig. 13: Buffer pool utilization over time for 69% page utilization

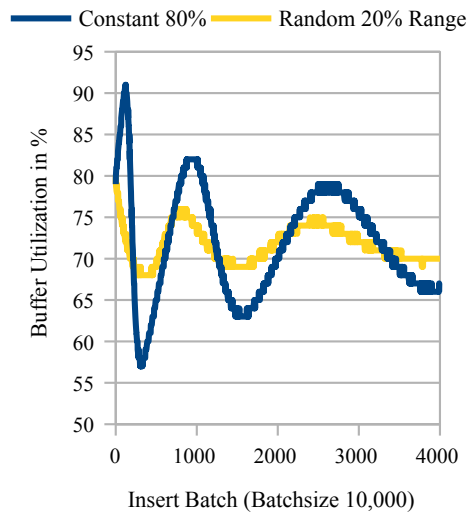


Fig. 14: Buffer pool utilization over time for 80% page utilization

the results. Here, the y-axis shows the buffer utilization, i.e. the quotient of used buffer pool bytes to totally available raw bytes. Clearly, the waves of misery also appear in the buffer utilization for the constant strategy while the algorithm based on the sound remedies eliminates them. The experiment was repeated for a higher overall utilization of 80%. This time, the constant strategy is compared to a random strategy featuring a range of 20%. As Figure 14 shows, waves for the constant strategy are even higher than at 69%. Furthermore, the random strategy reduces them just as it reduced the amount of leaf splits.

Finally, both constant and random strategies at 80% overall space utilization were compared for different buffer sizes based on buffer evictions, i.e. leaf pages being written out from the buffer onto disk due to other reads or writes requesting another page. Figure 15 depicts the total amount of evictions (y-axis) per batch (x-axis) while having varying buffer sizes (10,000, 20,000 and 100,000 pages) for each strategy. At 10,000 and 20,000 maximum buffer capacity, the waves of misery in form of increased spike of write operations are clearly visible for the constant strategy. Meanwhile, the random strategy almost eliminates the effects. For a larger buffer size of 100,000 pages, the additional writes are obviously

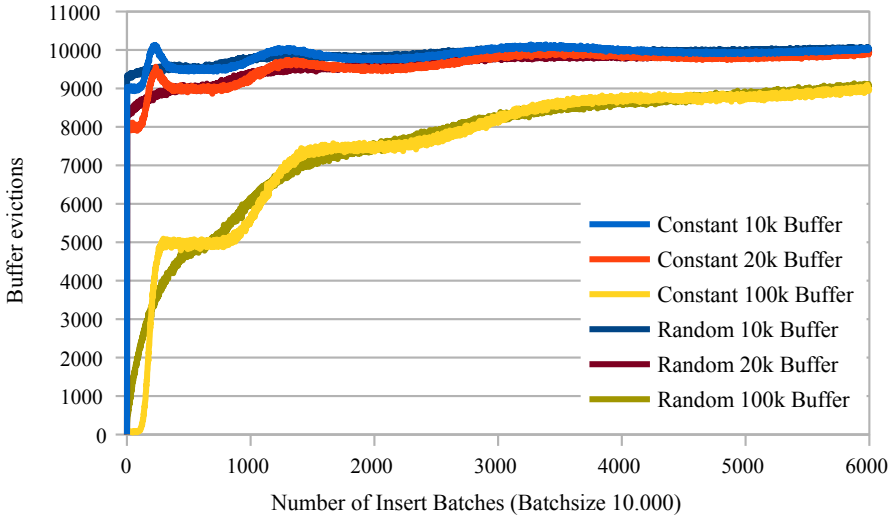


Fig. 15: Buffer evictions (writes) over time for varying buffer sizes

absorbed into buffer pool, causing only buffer contention and additional split operations, but no spikes in I/Os. Nevertheless, since the b-tree increases in size, more pages have to be constantly written out. The random strategy allows for a more gradual increase. Meanwhile, the constant strategy shows sharper *steps* at the points of the waves of misery since there is a sudden increase in new pages.

6.2.5 Range Queries

Since the proposed strategies do not load pages with a uniform free space, this obviously has some impact on range queries. For example, in the simple case of uniform key distribution, there is a chance that the same range span may hit varying degrees of full pages for different key ranges in the key space, resulting in more page reads and a negative performance impact.

To analyze this impact, the whole key space of a tree was queried using a succession of jumping range queries. First, a tree was loaded with 500,000 pages at a targeted utilization of 80% using an uniform key distribution ranging from the integer key 0 to the integer key of 100 million. Afterwards the range is successively queried with key ranges spanning 100,000 values starting from the first key, then querying the next 100,000 values, etc. until the last key is reached. Figure 16 features the results of this experiment for variations of the linear strategy. The starting value of the range query is depicted on the x-axis and the number of (leaf) pages accessed in the query on the y-axis. The *strict linear* strategy refers to a linear distribution which does not alternate between page sizes. It performs best for the first couple of ranges, because those ranges have the most full pages. On the flip side, it

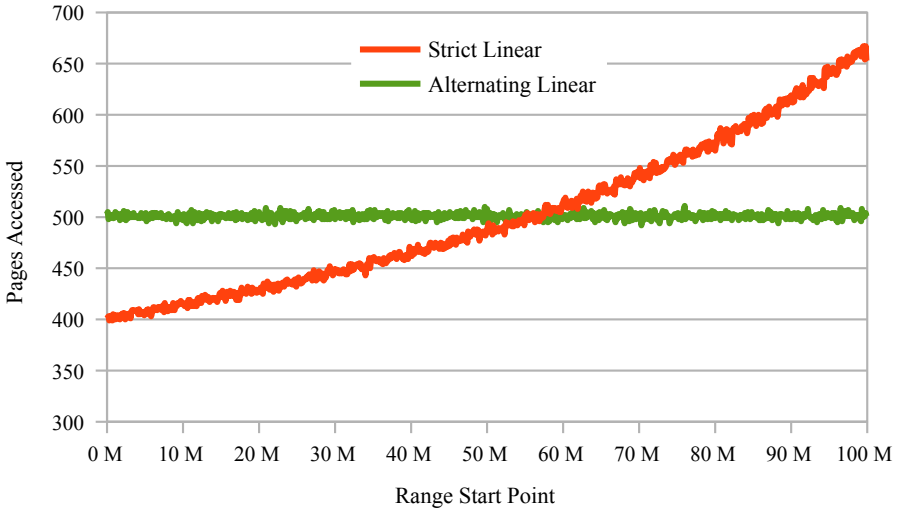


Fig. 16: B-tree range queries (range 100k) with varying starting points that cover the whole tree range

deteriorates for last ranges, since those ranges feature less occupied pages. The *alternating linear* strategy reduces this variation in performance and constantly accesses 500 pages for each read operations. We also repeated this experiment for the constant strategy, which performs about the same as alternating linear strategy. Furthermore, the random strategy also performs in the same ballpark, but, as expected, has slightly more spikes due to its randomness.

7 Summary and suggestions for future work

In summary, research in the past has overlooked the waves of node splits starting soon after b-tree creation, load, or reorganization. There are multiple means for avoiding or reducing them; many of them are both simple and effective. Among them, a free space target plus suffix truncation during leaf splits (as recommended decades ago in the context of prefix b-trees) offer the best combination of multiple advantages. Put differently, while suffix truncation during leaf splits has always been a good idea for minimizing the number of branch nodes and for speeding up root-to-leaf traversals, it is now clear that it is also a good idea and an effective means for avoiding multiple waves of misery after each index creation, load operation, and reorganization. Ongoing research focuses on “waves of misery” in log-structured b-tree forests [Gr19]. Future work will further research “waves of misery” on b-tree branch nodes, waves of node splits for specific key calculations such as space-filling curves, their forms and impacts in index and storage structures other than b-trees, specific forms of waves after specific load sequences such as sorting before loading R-trees, and what remedies may apply in the context of those index and storage structures.

Acknowledgments

This work has been partially supported by the German Research Foundation (DFG) under grant no. SE 553/9-1.

The idea for this work was made possible through discussions held at the Dagstuhl seminars 17222 (“Robust performance in database query”) and 18251 (“Database architectures for modern hardware”).

References

- [ASW12] Achakeev, D.; Seeger, B.; Widmayer, P.: Sort-based query-adaptive loading of R-trees. In: CIKM’12. Pp. 2080–2084, 2012.
- [BL89] Baeza-Yates, R. A.; Larson, P.: Performance of B+-Trees with Partial Expansions. IEEE Trans. on Knowledge and Data Engineering 1/2, pp. 248–257, 1989.
- [BU77] Bayer, R.; Unterauer, K.: Prefix B-Trees. ACM Trans. on Database Systems (TODS) 2/1, pp. 11–26, 1977.
- [Ei82] Eisenbarth, B.; Ziviani, N.; Gonnet, G. H.; Mehlhorn, K.; Wood, D.: The Theory of Fringe Analysis and Its Application to 2-3 Trees and B-Trees. Information and Control 55/1-3, pp. 125–174, 1982.
- [Gr03] Graefe, G.: Sorting And Indexing With Partitioned B-Trees. In: CIDR. 2003.
- [Gr04] Graefe, G.: Write-Optimized B-Trees. In: VLDB 2004. Pp. 672–683, 2004.
- [Gr11] Graefe, G.: Modern B-Tree Techniques. Foundations and Trends in Databases 3/4, pp. 203–402, 2011.
- [Gr19] Graefe, G.: Waves of misery in b-tree forests, in preparation, 2019.
- [La88] Larson, P.: Dynamic Hash Tables. Communications of the ACM 31/4, pp. 446–457, 1988.
- [Ma79] Martin, G.: Spiral storage: Incrementally augmentable hash addressed storage, Theory of Computation, tech. rep., University of Warwick, 1979.
- [ON92] O’Neil, P. E.: The SB-Tree: An Index-Sequential Structure for High-Performance Sequential Access. Acta Informatica 29/3, pp. 241–265, 1992.
- [ON96] O’Neil, P. E.; Cheng, E.; Gawlick, D.; O’Neil, E. J.: The Log-Structured Merge-Tree (LSM-Tree). Acta Informatica 33/4, pp. 351–385, 1996.
- [Ra00] Ramsak, F.; Markl, V.; Fenk, R.; Zirkel, M.; Elhardt, K.; Bayer, R.: Integrating the UB-Tree into a Database System Kernel. In: VLDB 2000. Pp. 263–272, 2000.
- [Se01] Seeger, B.: eXtensible and fleXible Library (XXL) for Java, 2001, URL: <https://github.com/umr-dbs/xxl>, visited on: 09/30/2018.
- [Ya78] Yao, A. C.: On Random 2-3 Trees. Acta Informatica 9/2, pp. 159–170, 1978.

Eliminating the Bandwidth Bottleneck of Central Query Dispatching Through TCP Connection Hand-Over

Stefan Klauck¹, Max Plauth¹, Sven Knebel¹, Marius Strobl², Douglas Santry², Lars Eggert²

Abstract: In scale-out database architectures, client queries must be routed to individual backend database servers for processing. In dynamic database systems, where backend servers join and leave a cluster or data partitions move between servers, clients do not know which server to send queries to. Using a central dispatcher, all queries and responses are routed via a single node. In a system with many high-performance backends, such a central node can become the system bottleneck. This paper compares three different approaches for query dispatching in terms of scaling network throughput and processing flexibility. Off-the-shelf TCP/HTTP load-balancers cannot dispatch individual queries arriving over a single connection to different backend servers, unless they are extended to understand the database wire protocol. For small response sizes up to 4 KB, a purpose-built query dispatcher delivers the highest throughput. For larger responses (i.e., BLOBs or data sets for external analysis), a novel approach for network proxying that transparently maps TCP connections between backend servers performs best. We propose hybrid query dispatching that performs a TCP connection hand-over on demand when returning large database results.

Keywords: Scale-Out Database Systems, Query Dispatching, Load-Balancing

1 Query Dispatching

In scale-out database systems, queries must be routed to individual backend servers. Clients may send queries directly to individual backends (see Figure 1a). In this case, they have to select a suitable server with respect to load-balancing and data distribution. An alternative approach uses one or multiple dedicated controller nodes (“dispatchers”), which act as central load-balancers (see Figure 1b).

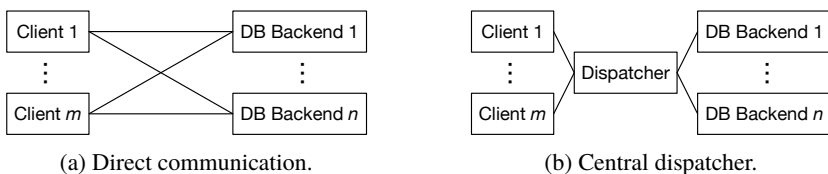


Fig. 1: Query dispatching architectures.

¹ Hasso Plattner Institute, University of Potsdam, Germany

² NetApp

TCP/IP is the most common communication protocol to exchange information in shared-nothing architectures. Likewise, database systems use wire protocols over TCP/IP. Without dispatchers, clients communicate directly with backend servers, using one or more TCP connections for each. With central dispatchers, the dispatchers mediate client communication with backend servers, often terminating client TCP connections and maintaining a separate set of (persistent) TCP connections with the backend servers. *Dispatchers introduce overhead, but also provide several advantages.* They allow for simpler clients that remain unaware of individual backend servers, their load levels, and the deployed physical design, i.e., partitioning and replication. This is advantageous for elastic and dynamic systems, because servers can transparently join or leave a cluster and data can be repartitioned on the fly. Such flexible database backends have increased in importance [Cu10, Se16, RJ17].

Dispatchers can be used independently of particular database systems, which may have different communication characteristics. Workloads for transactional systems and key-value stores are characterized by simple read and write queries (or get and put operations). Message sizes comprise usually up to a few kilo bytes, but larger data object comprising mega bytes of data may occur from time to time. The in-memory database Silo can process almost 700K TPC-C transactions per second [Tu13]. Analytical workloads are characterized by more complex queries with varying response sizes for which the query execution time dominates communication costs most of the time. However, Raasveldt et al. claim that transferring large amounts of data from databases to clients is common for complex statistical analyses or machine learning [RM17] in the application.

The existence of a database bandwidth bottleneck depends on the query characteristics, especially the ratio of database processing and result set sizes. Comparing the speed of a single 10 Gb network interface controller (NIC) and 8 CPU cores accessing main memory with 10GB/s, we informally claim that the network is 64 times slower in this scenario. Resulting, a reasonably well-programmed result-set serialization can produce messages that are an order of magnitude larger than those which can be send. (Note that serialization can be parallelized and carried out asynchronously to messages transfer.) In systems with multiple backend servers, a central query dispatcher can further constrain the data throughput. Prism [Ha17] eliminates the dispatcher bottleneck by redirecting client connections to individual backend servers on a query level. Redirecting TCP connections adds overhead for packet transformations but pays off for large data transfers.

Contributions: In this paper, we investigate database query dispatching for large messages. We integrated Prism into the in-memory database Hyrise [Gr10] and compare the performance of Prism against two approaches with the state-of-the-art architecture for central dispatchers, i.e., using two separate TCP connections, one between client and dispatcher, the other between dispatcher and backend. The first approach is the purpose-built Hyrise query dispatcher [Sc15]; the second is the off-the-shelf TCP/HTTP load-balancer HAProxy [Ta]. We investigate for which message sizes the Prism architecture outperforms the classical dispatcher architecture. In case message sizes are known in advance, Prism's connection redirection can be carried out on demand.

Outline: The remainder of this paper proceeds with an introduction of the compared dispatcher implementation in Section 2. In Section 3, we evaluate the performance of implementations. Section 4 discusses the results and most important aspects of query-based load-balancing. Section 5 describes related work, and Section 6 concludes the paper.

2 Dispatcher Implementations

To evaluate the different dispatcher implementations, we use a cluster of Hyrise [Gr10] database instances that implement lazy master replication. Load-balancing can be implemented at connection, transaction, or query level [CCA08]. While query processing at backends is independent of load-balancing, a dispatcher has to understand the message format in order to perform transaction- and query-level load-balancing. Hyrise uses JSON-encoded query plans encapsulated in HTTP request bodies. Other databases use different message-based protocols [RM17]. The Hyrise dispatcher and Prism can be extended to implement these wire protocols, because their underlying architectures are database-agnostic.

2.1 Hyrise Dispatcher

The Hyrise dispatcher is a purpose-built query load-balancer for Hyrise database clusters [Sc15]. Using purpose-built dispatchers for different database systems in common, because they have to understand the database wire protocol. To the best of our knowledge, all central query dispatchers send client queries and database responses over two separate TCP connections successively: the first between the client and the dispatcher; the second between the dispatcher and a database backend.

The Hyrise dispatcher uses the Berkeley socket API with a buffering mechanism that avoids copying data inside the user space. Client requests are first read into a buffer, after which the dispatcher parses the requests and forwards queries to a suitable backend. The Hyrise dispatcher uses the `nodejs/http-parser`³ for parsing client requests and extracting queries. Writes go to the master, while reads can go to any node, and tables loads are replicated across all database instances. Query responses are returned to clients also via the Hyrise dispatcher. The current implementation of the Hyrise dispatcher uses one thread per client connection. This implementation was sufficient for the experimental analysis, which uses a small number of persistent client connections. However, we observed some thread interference, especially when not pinning threads to CPU cores.

2.2 HAProxy

HAProxy [Ta] is a popular and general-purpose TCP/HTTP load-balancer. It has a similar design as the Hyrise dispatcher with separate TCP connection between clients and dispatcher

³ <https://github.com/nodejs/http-parser>

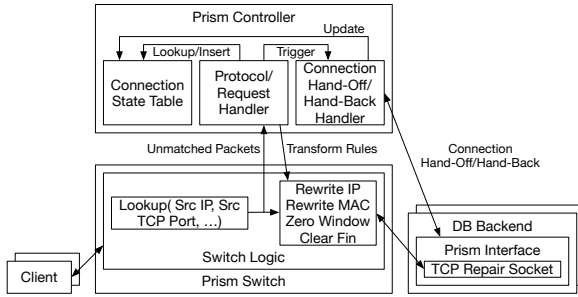


Fig. 2: Prism software architecture, based on [Ha17].

as well as dispatcher and backends. Including an off-the-shelf solution in the evaluation is interesting, because it is consistently updated for performance and support of new features. However, it requires some customization in order to adapt it to the specific scenario.

HAProxy can perform routing and processing decisions based on aspects of an HTTP request, such as the request URI. Following, HAProxy can be used for Hyrise clusters as long as the distinction between reads (to be distributed between replicas) and writes (to be delivered to the master) can be made without a deep inspection of HTTP request bodies. When it comes to load-balancing of application-level protocols other than HTTP (as for other databases), HAProxy is limited to TCP connection load-balancing. Resulting, HAProxy and possibly other off-the-shelf load-balancers cannot be used with database clusters that partition data or replicate only subsets of the data.

2.3 Prism

In analogy to an optical prism, Prism [Ha17] transparently splits a single TCP connection, breaking out different application-level requests and forwarding them to different backend servers by means of reprogrammable software-defined networking (SDN) switches, such as P4 [Bo14] hardware switches or software switches, such as mSwitch [Ho15]. By default, the mSwitch kernel software switch operates in learning bridge mode, but allows for dynamic packet forwarding decisions by ancillary modules, such as the kernel component of Prism.

Figure 2 shows Prism’s software architecture. Prism logically consists of a controller, which handles TCP handshaking and parses client request headers, and a programmable switch, which the controller reprograms to route TCP packets containing request and response payloads directly between a backend server and the client. After a backend server has handled a request, the controller resumes handing off the client’s TCP connection. Compared to traditional proxy approaches, Prism significantly reduces load on the controller, because the majority of packets is exchanged directly between clients and backends. Eliminating the controller as a central bottleneck for most of the communication also means that connections can take better advantage of typical multi-connected datacenter fabrics, significantly

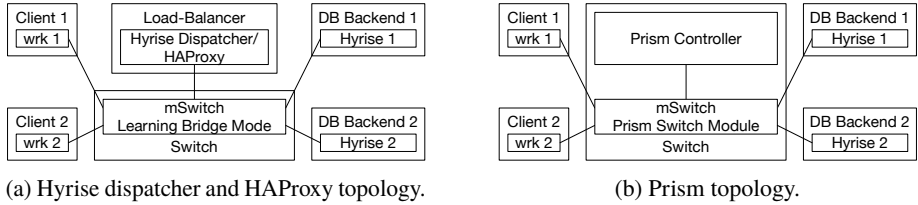


Fig. 3: Topologies for the evaluations, based on [Ha17].

improving performance. Dispatching at the switch level does incur some overheads, due to the need to reprogram switches and performing connection hand-offs, making Prism primarily suitable for larger messages. In most scenarios, the Prism connection hand-off only becomes possible after the complete query is received at the controller. Although the bandwidth for queries to the database is not increased in this case, database results can be sent with higher overall bandwidth. Details of Prism, e.g., the connection hand-over and TLS support, are described in [Ha17].

3 Evaluation

We integrated Prism into Hyrise. This integration is transparent to clients, but required changes to the database server, which has to be able to hand-over TCP connections with the Prism controller (see Figure 2). In this section, we compare the query dispatching performance of Prism, the Hyrise dispatcher, and HAProxy. We describe the experimental setup in Section 3.1. The results are presented in Section 3.2.

3.1 Experimental Setup

Our experimental setup is similar to the one employed for evaluating Prism [Ha17], but extends the experiments with 40G Ethernet. It is comprised of six nodes: two clients, two backends and two “dispatcher” machines acting as either software switches or Hyrise/HAProxy dispatchers, depending on whether measurements using 10 or 40G networking are conducted. Figure 3a depicts the logical topology used for the Hyrise dispatcher and HAProxy tests, Figure 3b likewise for Prism. Using only two clients and backend servers is no architectural limitation, but a result of limited own network hardware resources and the impossibility of running the experiments in public clouds due to insufficient hardware control.

The client machines are equipped with one Intel Xeon E5-2680 v2 CPU clocked at 2.8 GHz, 64 GB of 1333 MHz DRAM and Linux 4.11 kernels with Ubuntu 17.04. Backends have two Intel Xeon E5-2650 packages clocked at 2.0 GHz and 128 GB of 1333 MHz memory, running Linux 4.8 kernels with Ubuntu 16.10. The remaining “dispatcher” machines have one Intel Xeon E5-2680 v2 CPU clocked at 2.8 GHz, 64 GB of 1600 MHz DRAM, running Linux 4.8 with Ubuntu 16.10.

For the 10G experiments, all machines use Intel 82599ES dual-port NICs, connected through an Arista 7050QX-32-F switch. For the 40G experiments, the “dispatcher” nodes have Intel XL710-QDA2 dual-port NICs, while backends and clients use Mellanox ConnectX-3 MCX354A-FCB NICs. All 40G cards are directly connected, because of insufficient 40G ports on the Arista switch. The Ethernet MTU is set to the default of 1500 B in all cases.

Depending on which dispatcher approach is being measured, one “dispatcher” node is configured to run mSwitch, either in Prism mode or as a learning bridge. This allows for a fair comparison between the Prism and non-Prism scenarios, by using a software switch in all cases. A baseline netperf⁴ TCP network benchmark results in 19.5 Gb/s maximum throughput for 128 MB of bulk data over the 40G setup with the Mellanox adapters, i.e., the bottleneck is not the software switch. Full wire speed is achieved with 10G.

The backends are running Hyrise on all 16 cores, one acts as master with the other as replica. In order to make sure that Hyrise itself is not the bottleneck, a minimal stored procedure returns a database result with the requested number of bytes. The single-threaded Prism controller executes bound to a single core on the same “dispatcher” machine as mSwitch. For experiments not involving Prism, the other “dispatcher” node is either running the Hyrise dispatcher with its client threads bound to four cores or HAProxy 1.7.9 in a multiprocess configuration with both its backend and frontend processes also bound to four cores.

Measurement data is obtained on the clients by running one single-threaded instance of the wrk⁵ HTTP benchmark tool. Client transactions result in response payload sizes between 1 B and 128 MB, which are sampled for 20 s using one persistent connection each. The results of the two clients then are aggregated.

3.2 Experimental Results

Figure 4 shows the throughput results for the different dispatcher approaches for the 10 and 40G experiments. A subset of the results is summarized in Table 1. Throughput is provided in terms of HTTP payload in all cases, i.e., from the client/user perspective. The main result is that, for payload sizes over 1 MB, Prism outperforms the other two dispatchers by up to 2× (the number of client/backend nodes). This is because for the Hyrise dispatcher and HAProxy, each payload byte needs to traverse the dispatcher (see Figures 1b and 3a, respectively), which in a typical datacenter fabric is connected via a single 10 or 40G link to the top-of-rack switch. This limits both cases to at most 10 Gb/s and 40 Gb/s, respectively.

In the 10G experiment, the Hyrise dispatcher and HAProxy achieve almost the physical limit of 10 Gb/s for payload sizes over 16 MB. However, the maximum measured throughput in the 40G experiment is only 18.9 Gb/s for the Hyrise dispatcher and 19.4 Gb/s for HAProxy, demonstrating the overheads that limit performance over today’s faster network fabrics.

⁴ <http://www.netperf.org/>

⁵ <https://github.com/wg/wrk>

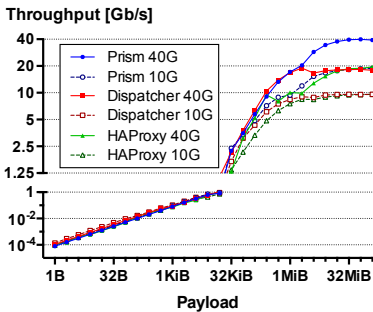


Fig. 4: Dispatcher throughputs for varying payloads.

Payload	Result Unit	Prism		Dispatcher		HAProxy	
		10G	40G	10G	40G	10G	40G
1 B	Mb/s	0.10	0.08	0.14	0.11	0.09	0.11
8 B	Mb/s	0.79	0.64	1.2	0.88	0.74	0.60
64 B	Mb/s	6.3	5.1	9.3	7.0	5.8	4.8
512 B	Mb/s	50	41	63	55	42	44
4 KB	Mb/s	378	321	396	324	303	248
32 KB	Gb/s	2.4	2.2	1.7	2.1	1.4	1.3
256 KB	Gb/s	7.2	9.1	6.1	10.4	4.8	9.5
1 MB	Gb/s	9.39	17.2	8.4	16.9	7.5	10.1
2 MB	Gb/s	11.9	20.3	9.0	18.9	8.4	9.9
16 MB	Gb/s	17.7	37.8	9.5	18.1	9.3	17.7
128 MB	Gb/s	18.8	39.1	9.6	17.9	9.6	19.4

Tab. 1: Selected throughput measurements from Figure 4.

Prism does not share this limitation, because clients and backend servers are communicating directly after a TCP connection has been redirected, i.e., responses are directly sent to the clients, taking advantage of the inherent fan-out in the fabric. Consequently, Prism throughput is only limited by the speed of the network fabric (10G or 40G) or the processing capacity of the backends, whichever is lower. In the 10G case, Prism achieves 18.8 Gb/s, which is close to the physical limit of 2×10 Gb/s and twice as high as for the Hyrise dispatcher and HAProxy. In the 40G case, Prism reaches 39.1 Gb/s, again about twice as high as the other two dispatcher approaches.

For payload sizes below 4 KB, the Hyrise dispatcher outperforms Prism’s and HAProxy’s throughput by up to 1.5 \times . For small responses, Prism cannot amortize the connection hand-off overheads. We observed that HAProxy’s performance for small payload sizes is reduced by interprocess communication overheads (see Section 4 for more details), which appear to be higher than for the Hyrise dispatcher.

No Dispatcher		Prism		Dispatcher		HAProxy	
10G	40G	10G	40G	10G	40G	10G	40G
59 μ s	70 μ s	161 μ s	214 μ s	104 μ s	144 μ s	167 μ s	151 μ s

Tab. 2: Average user latencies for querying 1B payload.

To evaluate the overhead of query dispatching in comparison to direct client-backend communication, Table 2 shows the user perceived query latencies for 1 B payloads. Latency is measured from sending the request until the reception of the response, including network transmission and database processing times. Without an intermediate dispatcher, the latency is about 59 μ s in the 10G and 70 μ s in the 40G experiment. Adding a dispatcher, increases the latency by about 45 μ s to 144 μ s depending on the dispatching approach and NIC type. Among the dispatching approaches, the latency of the Hyrise dispatcher is the lowest.

4 Discussion

In this section, we discuss several aspects of query-based load-balancing. First, we summarize our research findings (see Section 4.1). Based on these, we propose a hybrid query dispatching approach in Section 4.2. In the following, we discuss the scalability of central query dispatching in Section 4.3.

4.1 Research Findings

The experiments show that no single dispatcher approach is best across all payload sizes. The Hyrise dispatcher performs best for small payloads, because the overhead of Prism's connection hand-over does not amortize if the payload fits into a few network packets. Prism outperforms the Hyrise dispatcher for larger payloads, because the majority of payload bytes can be directly handled by the switch.

HAProxy was not able to compete with either of the two, despite a lot of effort on fine-tuning its configuration. Although a single-process HAProxy configuration delivered best throughputs for small payload sizes below 0.5 KB compared to multiprocess setups, it peaked at only 10 Gb/s for payloads above 2 MB in the 40G case. Further, we did not observe any performance benefits from HAProxy taking advantage of socket splicing [MB99] compared to the Hyrise dispatcher, which has to copy all data to and from user space. These findings indicate that although an off-the-shelf load-balancer can be used, it has severe performance limitations, besides its lack of flexibility (see Section 2.2).

4.2 Hybrid Approach

Based on our measurements, we propose a hybrid load-balancing approach, which uses Prism for queries returning large results and the Hyrise dispatcher for all other requests. This requires knowledge about response sizes of individual queries, which can be calculated (or estimated) during the result set serialization. Hence, the connection hand-off has to be postponed until the response size is determined and must be initiated by the backend.

4.3 Scalability

Additional throughput can be achieved with additional and/or faster hardware. First, we can scale-up the dispatcher with additional and/or faster NICs. However, this requires efficient software to actually take advantage of the hardware. Due to its improved usage of hardware and preexisting software, Prism is expected to provide the best price-performance ratio in this regard.

Further, we can hide multiple dispatchers behind a virtual IP, using network-level load-balancing as the first and query-level load-balancing as the second step. Network-level load-balancers can additionally aggregate multiple requests to avoid the network overhead for many connections sending small messages. Especially transactional databases, which are able to process millions of small queries per second, require query batching [Tu13].

5 Related Work

Efficient query dispatching is a research field combining database and network aspects. On the database side, clustered databases with a single entry point require query dispatching. Cecchet et al. summarize different approaches for replicated databases and discuss load-balancing alternatives [CCA08]. Database load-balancing research focuses on equal distribution of load, such as the Tashkent+ load-balancer [EDZ07], rather than efficient load-balancing from a networking perspective. Consequently, there is a lack of focus on throughput and latency measurements. Our work investigates scenarios where query dispatching and the relaying of database responses can become the bottleneck. These scenarios range from thousands of small transactions [Tu13] as one extreme to large data transfers from the database [RM17] as the other extreme. Besides networking as database client interface, high performance networks are a relevant topic for distributed query processing [Bi16, Rö15]. Associated research includes microbenchmarks for single connection RDMA and Ethernet, but no switching measurements that are relevant for query load-balancing.

On the networking side, the focus in recent years has been on large-scale network-level load-balancing, especially on distributing the ingress load of hyperscalar datacenters onto backend servers at connection-open time. Proxy approaches, in which an intermediary remains in the communication path and can therefore provide finer-grained operations than simple load-balancing, have somewhat fallen out of favor, due to their perceived higher overheads. Hayakawa et al. present related work in this space [Ha17].

6 Conclusion

Many scale-out database systems use a central query load-balancer that decides where to send individual client queries. For clusters serving many thousands of requests and returning large objects or data sets, the query dispatcher and the network fabric connecting it to the clients and backend servers can become a bandwidth bottleneck. Prism redirects TCP connections and allows database backends to directly respond to clients without changing the clients. We integrated Prism into an in-memory database and compared its performance against the common dispatching architecture in which each query and result is routed via a single node. The traditional dispatcher architecture provided the highest throughput for small database responses, whereas the Prism approach significantly outperformed it for larger payloads. We propose to use a hybrid load-balancing approach that uses Prism's connection hand-over on demand for large payloads.

References

- [Bi16] Binnig, Carsten; Crotty, Andrew; Galakatos, Alex; Kraska, Tim; Zamanian, Erfan: The End of Slow Networks: It's Time for a Redesign. *PVLDB*, 9(7):528–539, 2016.
- [Bo14] Bosshart, Pat et al.: P4: programming protocol-independent packet processors. *Computer Communication Review*, 44(3):87–95, 2014.
- [CCA08] Cecchet, Emmanuel; Candea, George; Ailamaki, Anastasia: Middleware-based database replication: the gaps between theory and practice. In: *SIGMOD*. pp. 739–752, 2008.
- [Cu10] Curino, Carlo; Zhang, Yang; Jones, Evan P. C.; Madden, Samuel: Schism: a Workload-Driven Approach to Database Replication and Partitioning. *PVLDB*, 3(1):48–57, 2010.
- [EDZ07] Elnikety, Sameh; Dropsho, Steven G.; Zwaenepoel, Willy: Tashkent+: memory-aware load balancing and update filtering in replicated databases. In: *EuroSys*. pp. 399–412, 2007.
- [Gr10] Grund, Martin; Krüger, Jens; Plattner, Hasso; Zeier, Alexander; Cudré-Mauroux, Philippe; Madden, Samuel: HYRISE - A Main Memory Hybrid Storage Engine. *PVLDB*, 4(2):105–116, 2010.
- [Ha17] Hayakawa, Yutaro; Eggert, Lars; Honda, Michio; Santry, Douglas: Prism: a proxy architecture for datacenter networks. In: *SoCC*. pp. 181–188, 2017.
- [Ho15] Honda, Michio; Huici, Felipe; Lettieri, Giuseppe; Rizzo, Luigi: mSwitch: a highly-scalable, modular software switch. In: *SOSR*. pp. 1:1–1:13, 2015.
- [MB99] Maltz, David A.; Bhagwat, Pravin: TCP Splice for application layer proxy performance. *J. High Speed Networks*, 8(3):225–240, 1999.
- [RJ17] Rabl, Tilmann; Jacobsen, Hans-Arno: Query Centric Partitioning and Allocation for Partially Replicated Database Systems. In: *SIGMOD*. pp. 315–330, 2017.
- [RM17] Raasveldt, Mark; Mühleisen, Hannes: Don't Hold My Data Hostage - A Case For Client Protocol Redesign. *PVLDB*, 10(10):1022–1033, 2017.
- [Rö15] Rödiger, Wolf; Mühlbauer, Tobias; Kemper, Alfons; Neumann, Thomas: High-Speed Query Processing over High-Speed Networks. *PVLDB*, 9(4):228–239, 2015.
- [Sc15] Schwalb, David; Kossmann, Jan; Faust, Martin; Klauck, Stefan; Uflacker, Matthias; Plattner, Hasso: Hyrise-R: Scale-out and Hot-Standby through Lazy Master Replication for Enterprise Applications. In: *IMDM@VLDB*. pp. 7:1–7:7, 2015.
- [Se16] Serafini, Marco; Taft, Rebecca; Elmore, Aaron J.; Pavlo, Andrew; Aboulnaga, Ashraf; Stonebraker, Michael: Clay: Fine-Grained Adaptive Partitioning for General Database Schemas. *PVLDB*, 10(4):445–456, 2016.
- [Ta] Tarreau, Willy: , The Reliable, High Performance TCP/HTTP Load Balancer. <https://www.haproxy.org>.
- [Tu13] Tu, Stephen; Zheng, Wenting; Kohler, Eddie; Liskov, Barbara; Madden, Samuel: Speedy transactions in multicore in-memory databases. In: *SOSP*. pp. 18–32, 2013.

Ja-(zu-)SQL: Evaluation einer SQL-Skriptsprache für Hauptspeicherdatenbanksysteme

Maximilian Schüle,¹ Linnea Passing,² Alfons Kemper,³ Thomas Neumann⁴

Abstract: Große Datenmengen in Wirtschaft und Wissenschaft werden klassischerweise in Datenbanksystemen verwaltet. Um die Daten für maschinelles Lernen sowie für die Datenanalyse zu nutzen, ist ein zeitintensiver zyklischer Transformations- und Verschiebeprozess nötig, da die Daten hierfür in anderen Formaten oder schlicht in anderen Plattformen vorliegen müssen. Forschungsarbeiten der letzten Jahre widmen sich der Aufgabe, Datenverwaltung und Datenanalyse in *einem* System zu integrieren, um teure Datentransfers zu vermeiden.

Diese Arbeit untersucht die Leistungsfähigkeit gespeicherter Prozeduren (*stored procedures*) zur Implementierung von Data-Mining-Algorithmen in Datenbanksystemen. Grundlage hierfür bildet *HyPerScript*, die PL/SQL-ähnliche Skriptsprache des Hauptspeicherdatenbanksystems HyPer. Insbesondere evaluieren wir die prototypische Implementierung von fünf Algorithmen, die ganz ohne separates Datenanalysesystem auskommt.

1 Einleitung

Mit wachsenden Leistungsanforderungen bei der Datenverarbeitung gewinnen sowohl Hauptspeicher-Datenbanksysteme für die Datenhaltung als auch die automatisierte Verarbeitung großer Datenmengen an Bedeutung. Wenn beide Domänen kombiniert werden, entfällt das zyklische Extrahieren, Transformieren und Laden von Daten (sogenannter ETL-Prozess). Somit können die Daten in Echtzeit statt um die Prozessdauer verspätet analysiert werden. Datenbanksysteme bieten prozedurale Sprachen wie PL/SQL bis hin zu in C geschriebene Erweiterungen an, um innerhalb des Datenbanksystems Algorithmen zu spezifizieren.

Neben der Möglichkeit, Algorithmen in beliebigen prozeduralen Sprachen zu implementieren, integrieren verschiedene Datenbanksysteme Data-Mining-Algorithmen direkt. Für auf *PostgreSQL* basierende Datenbanksysteme stellt *MADlib* eine Bibliothek an Funktionen zur Datenanalyse bereit. Das Projektziel ist, eine Bibliothek an Erweiterungen bereitzustellen, die an die Pakete aus *GNU R* angelehnt ist. Für eine Echtzeitdatenanalyse in Hauptspeicherdatenbanksystemen stellt *HyPer* Operatoren für die Assoziationsanalyse mit Apriori, Clusteranalyse mit k-Means und DBSCAN und Graphanalyse mit PageRank bereit. Diese

¹ TU München, Lehrstuhl für Datenbanksysteme, Boltzmannstraße 3, 85748 Garching, m.schuele@tum.de

² TU München, Lehrstuhl für Datenbanksysteme, Boltzmannstraße 3, 85748 Garching, passing@in.tum.de

³ TU München, Lehrstuhl für Datenbanksysteme, Boltzmannstraße 3, 85748 Garching, kemper@in.tum.de

⁴ TU München, Lehrstuhl für Datenbanksysteme, Boltzmannstraße 3, 85748 Garching, neumann@in.tum.de

Operatoren sind Teil der Algebra und werden von *HyPer* in die ganzheitliche Anfrageoptimierung eingebunden. Vordefinierte Operatoren kommen dem Anspruch, Datenbanksysteme um beliebige Anwendungslogik zu erweitern, nur begrenzt nach.

Um Datenbanksysteme für beliebige Algorithmen und maschinelles Lernen zu erweitern ohne jeweils einen Operator zu implementieren, ist eine domänenspezifische Sprache nötig, die prozedurale Konstrukte mit eingebettetem SQL bereitstellt. Eine prozedurale Skriptsprache kombiniert mit der deklarativen Anfragesprache SQL ermöglicht nutzerseitig definierte Funktionen (*user defined function, UDF*) in kompakterer Darstellung als rein imperative Sprachen.

Weitere Vorteile einer Skriptsprache, bei der die Anwendungslogik in das Datenbanksystem hinein verlagert wird, sind die Integration in die Anfrageoptimierung des Datenbanksystems und ein geringerer Transfer von Daten. Als gespeicherte Prozeduren sind *UDFs* Teil des Anfrageplans und somit Teil der ganzheitlichen Anfrageoptimierung. Zudem erlaubt die Leistungssteigerung der Datenbankservers durch Verwendung moderner Hardware, diese für mehr als nur reine Datenverwaltungsaufgaben zu verwenden. Damit Datenbankserver mehr Anwendungslogik übernehmen können, muss es für Datenbanknutzer möglich sein, Algorithmen für die Zielsprache universell zu entwickeln. Um den Funktionsumfang einer solchen Skriptsprache identifizieren zu können, ist relevant, wie sich Algorithmen abstrahieren lassen und welche Bausteine dafür unabdingbar sind.

In dieser Arbeit identifizieren wir notwendige Bausteine, indem wir ausgewählte Data-Mining/Machine-Learning-Algorithmen (Apriori, DBSCAN, k-Means, PageRank, Lineare Regression) in *HyPerScript*, der Skriptsprache des Hauptspeicherdatenbanksystems *HyPer*, implementieren. Anschließend vergleichen wir die Laufzeit und Skalierbarkeit der Algorithmen im Vergleich zu bereits implementierten Operatoren [Pa17], die das Datenbanksystem bereitstellt. Die Arbeit liefert folgende Beiträge:

- Die Vorstellung von *HyPerScript* als exemplarische Erweiterung eines Hauptspeicherdatenbanksystems, die es Nutzern erlaubt, Funktionen in um prozedurale Ausdrücke erweitertem SQL zu formulieren.
- Die beispielhafte Beschreibung einer betriebswirtschaftlichen Anwendung in der Form einer TPC-C-Anfrage in *HyPerScript*. Dabei handelt es sich um den primären Anwendungsfall der Skriptsprache.
- Die Erweiterung von SQL um einen Tensor-Datentyp, um bestimmte Datenanalyse-Algorithmen zu ermöglichen.
- Die Analyse des Potentials einer Skriptsprache in Datenbanksystemen, um beliebige Algorithmen zu formulieren; demonstriert an den Algorithmen PageRank, Apriori, lineare Regression, k-Means und DBSCAN, die bereits als feststehende Operatoren in *HyPer* existieren.

- Eine umfangreiche Evaluierung von *HyPerScript*-Funktionen als *stored procedures* im Vergleich zu implementierten Operatoren.

Eine umfassende und performante domänenspezifische Skriptsprache bildet die Grundlage für die Entwicklung einer deklarativen Sprache. Eine auf SQL basierende, prozedural-deklarative Sprache ermöglicht Datenanalysten, unabhängig vom darunterliegenden System zu programmieren. Die Plattformunabhängigkeit und die Wiederverwendbarkeit von SQL erhöht den Anreiz, komplexe Algorithmen bereits im Datenbanksystem auszuführen.

Die Arbeit ist wie folgt gegliedert: Nach einem Überblick über verwandte Arbeiten und einer Einführung in *HyPerScript* erfolgt eine Vorstellung der implementierten Algorithmen. Die Evaluation beurteilt die Leistungsfähigkeit der Skriptsprache im Vergleich zu den in *HyPer* integrierten Operatoren. Die Zusammenfassung diskutiert die Ergebnisse in Hinblick spezifischer Sprachen für die Datenanalyse.

2 Verwandte Arbeiten

Im Folgenden erläutern wir aktuell in der Praxis relevante Systeme und den Stand der Forschung, Algorithmen zur Wissensgewinnung in Datenbanksysteme zu integrieren.

Die code-kompilierenden, SQL-basierten Hauptspeicherdatenbanksysteme *EmptyHeaded* [Ab17] und *HyPer* [KN11] unterstützen bereits einige Data-Mining-Algorithmen. Dazu stellt *EmptyHeaded* eine Schnittstelle für SQL-Abfragen und andere Algorithmen bereit, die in einer übergeordneten Sprache wie Python adressiert werden. Das dieser Arbeit zugrundeliegende Hauptspeicherdatenbanksystem *HyPer* ist ein hybrides OLTP und OLAP Datenbanksystem, welches parallel Echtzeit-Transaktionen verarbeiten sowie mehrfache analysierende Anfragen ausführen kann. Es war bahnbrechend für die Kompilierung anstelle von Interpretation von SQL-Anfragen [Ne11], bietet einige Data-Mining-Algorithmen als feststehende Operatoren als Teil der Algebra an und integriert sie somit in die Anfrageoptimierung [Pa17; Th15].

Die für statistische Analysen verbreitete Open-Source-Anwendung *GNU R* ⁷ bietet Module wie für die hier behandelten Algorithmen an, die sich über eine eigene Kommandozeile steuern lassen. Diese Module sind vergleichbar mit den in *HyPer* angebotenen Operatoren, denn beide (sowohl die Module als auch die Operatoren) repräsentieren Bausteine mit einer hart kodierten Funktionalität. Wenn man neue Bausteine hinzufügen oder die bestehenden verändern möchte, so müssen in C++ neue Pakete (*GNU R*) bzw. neue Operatoren (*HyPer*) entwickelt werden. Die Alternative zur Neuentwicklung stellen in R definierbare Funktionen dar, für die prozedurale Sprachkonstrukte wie Schleifen und weitere Kontrollstrukturen bereitstehen. Das Gegenstück, um Funktionen in Datenbanksystemen zu definieren, sind die in den nachfolgenden Kapitel vorgestellten *stored procedures* mit *HyPerScript*.

⁷ R <http://www.r-project.org/>

Neben *GNU R* existieren weitere Programme, wie das davon für performante Datenanalysen abgeleitete *Julia*⁸, das, wie unser *HyPerScript*, LLVM-Code generiert. Für maschinelles Lernen, etwa, um neuronale Netze zu trainieren, sind *Python* mit *SciPy*⁹ oder *TensorFlow* [Ab16] ausgelegt.

Ein vergleichbarer Ansatz zu mit *HyPerScript* kompilierten Funktionen benennen Crotty et. al. [Cr15]. Ihr Ansatz kompiliert, kombiniert und optimiert beliebige Funktionen (*User Defined Functions*) zu LLVM-Code, um Vektorinstruktionen und Pipelining auf Datensätzen zu nutzen. Eine Gemeinsamkeit ist die Kompilierung der nutzerseitig definierten Funktionen zu LLVM-Code (obwohl das Suffix „Script“ in *HyPerScript* fälschlicherweise zur Annahme verleitet, die Funktionen würden interpretiert). Um eine bessere Laufzeit von PostgreSQL zu erhalten, kompiliert der Ansatz in [BG16; BG17] SQL-Anfragen zu LLVM-Code. Eine weitere Modifikation von PostgreSQL ist die Erweiterung von PL/pgSQL um Funktionen höherer Ordnung [GSU13; GU13]. Die Integration algebraischer Datentypen in relationale Datenbanksysteme [Gi13] stellt eine weitere Herausforderung dar.

Eine zu *HyPerScript* vergleichbare Programmiersprache bietet *SAP HANA* mit *SQLScript* [BMM13]. *SQLScript* erweitert SQL um prozedurale und funktionale Prozeduren, erlaubt *MapReduce* auf analytischen Anfragen und erweitert Rekursion für die Graphanalyse. Alternativ zu prozeduralen Erweiterungen erweitert *FunSQL* [Bi12] SQL um funktionale Konstrukte. Die Erweiterung intendiert, ähnlich wie diese Arbeit, die Verlagerung von Applikationslogik auf Datenbankserver.

Um Applikationslogik auf Datenbankserver verlagern zu können, hilft eine eigene Sprache, die zu SQL übersetzt. Ein Beispiel hierfür ist *Ferry* [Sc10], eine Metasprache mit den Konstrukten *for*, *where*, *group by*, *order by* und *return*. *Ferry* wird zuerst aus gängigen Skript- oder Programmiersprachen erzeugt und anschließend in SQL übersetzt.

3 HyPerScript

HyPerScript ist die prozedurale Sprache zu *HyPer*, analog zu PL/SQL in Oracle [Lo08], zu PL/pgSQL¹⁰ in *PostgreSQL* oder zu *SQLScript* in *SAP HANA* [BMM13]. Für transaktionale Anwendungen in *HyPer* entwickelt, erlaubt *HyPerScript* es, TPC-C-, TPC-E- und TPC-H-Benchmarks auszuführen. Außerdem können mit der um prozedurale Ausdrücke erweiterten SQL-Syntax beliebige Algorithmen formuliert werden.

Die in *HyPerScript* definierten Prozeduren werden während der Kompilierungsphase in LLVM-Code übersetzt. List. 1 zeigt die Sprachspezifikation in EBNF. *HyPerScript* verwendet die im SQL:2003-Standard [Ei04] spezifizierten Sprachkonstrukte inklusive Fensterfunktionen (*window functions*). Die SQL-Befehle lassen sich in prozedurale Konstrukte wie Schleifen (`while<Bedingung>{...}`) und Iterationen (`select index from`

⁸ Julia <http://julialang.org/>

⁹ SciPy <http://www.scipy.org/>

¹⁰ PL/pgSQL <https://www.postgresql.org/docs/10/static/plpgsql.html>

```

Statement      = ( ( VarDeclaration | VarDefinition | TableDeclaration
                  | SelectStatement | EmbeddedSQL | ReturnStatement
                  | IfStatement | WhileStatement | ControlStatement | FunctionCall ) ";" ) * ;

VarDeclaration = "var" NAME Type "=" Expression ;
VarDefinition  = NAME "=" Expression ;
TableDeclaration = "table" NAME "(" NAME Type ("," NAME Type)* ")";
SelectStatement = SQLSelect ("{" Statement "}") ("else" "{" Statement "}" ) ;
EmbeddedSQL    = SQLInsert | SQLUpdate | SQLDelete | CSVCopy ;
ReturnStatement = "return" NAME | "rollback" ;
IfStatement    = "if" "(" Expression ")" "{" Statement "}" "else" "{" Statement "}" ;
ControlStatement = "break" | "continue" ;
WhileStatement = "while" "(" Expression ")" "{" Statement "}" ;
FunctionCall   = NAME "(" Expression ("," Expression)* ")";

```

List. 1: Sprachdefinition von *HyPerScript*: SQL{Select, Insert, Update, Delete} entsprechen den SQL:2003-Befehlen, Type den SQL-Datentypen, Expression den SQL-Ausdrücken. Neben klassischen Ausdrücken prozeduraler Sprachen ermöglicht EmbeddedSQL auf einzelnen Tupeln der SQL-Anfrage zu operieren.

```

create or replace function insert_until(anzahl integer not null) as $$
  select index as i from sequence(0,anzahl){
    var rand = random(100); if (rand = 13) { continue }; insert into sample(rand);
  }
  $$ language 'hyperscript' strict;

```

List. 2: Beispielhafte User-Defined-Funktion mit *HyPerScript*: Hier werden *anzahl* viele Zufallszahlen in eine Relation namens *sample* eingefügt, randomisiert zwischen 0 und 100, eine Zahl 13 soll übersprungen werden.

`sequence(1,10){...}` mit den Steuerungsbefehlen `break` und `continue` und Bedingungen (`if(<Bedingung>){...}`) einbinden. Zusätzlich lassen sich temporäre Tabellen erstellen, auf Basis der SQL-Datentypen Variablen deklarieren und definieren (`var foo int[]={'}'`) und ebenfalls als Parameter beim Funktionsaufruf mit übergeben oder zurückgeben.

List. 2 demonstriert eine beispielhafte Anwendung von *HyPerScript*. Mittels deklarativem SQL wird ein Intervall definiert, das hier als Iteration ähnlich einer For-Schleife genutzt wird. Auf die einzelnen Ergebnistupel kann innerhalb des Geltungsbereiches zugegriffen und diese auch wieder in SQL-Befehlen verwendet werden. In diesem Fall wird eine Relation namens *sample* mit Zufallswerten befüllt.

3.1 HyPerScript mit Tensoroperationen

Zusätzlich zu prozeduralen Kontrollbefehlen erachten wir Tensoroperationen als elementar für die Eignung von Datenbanksystemen für maschinelles Lernen, um zum Beispiel lineare Regression numerisch zu lösen. Daher verwendet *HyPer* einen zu Tensoren erweiterten Array-Datentyp, der als Attribut in Datenbankschemata Tensoren von beispielsweise Ganzzahlen oder Gleitkommazahlen erlaubt. Die Array-Operationen erfolgen auf den Attributen von Relationen als Teil der Projektion im SELECT-Teil einer SQL-Anfrage. Das umschließt die folgenden Anwendungen auf Tensoren:

1. **Algebra:** Dazu gehören Tensorprodukt, -addition, -subtraktion und das Skalarprodukt, das Invertieren, Transponieren und Potenzieren von Tensoren.
2. **Erzeugen:** Neben dem Initialisieren mit `ARRAY[<WERTE>]` bzw. `{<WERTE>}`, benötigen wir die Identitätsmatrix `array_id(<Dimension>)` und einen Tensor gefüllt mit einem angegebenen Wert (`array_fill(<WERT>, <Dimensionen>)`).
3. **Zugriff:** Zugriff erfolgt über `array_access(<ARRAY>, <Indizes>)` bzw. `ARRAY[<Index>]`. Die Funktion zum Schneiden `array_slice(<ARRAY>, <Indizes>)` erlaubt eine Teilmenge eines Arrays zu extrahieren, um Daten zum Beispiel in Test- und Trainingsmenge zu teilen.
4. **Verändern:** Nach Erzeugung der Tensoren müssen einzelne Elemente effizient verändert werden, dazu dient `array_set(<ARRAY>, <WERT>, <Indizes>)`. Das Gegenstück zum oben erwähnten Schneiden ist das Konkatenieren von Arrays `<ARRAY> || <ARRAY>`.
5. **Mengenoperationen:** Um Teilmengen von Arrays zu identifizieren, braucht ein Datenbanksystem Mengenoperationen wie $A \subseteq B$ als `<ARRAY> <@ <ARRAY>`. Um Elemente in einer Teilmenge zu finden, steht $a \in A$ als `a ANY= A` zur Verfügung.
6. **Normalisieren:** `unnest(<ARRAY>)` als Gegenstück zum Initialisieren von Arrays extrahiert die Elemente in einzelne Tupel.

Die eingeführten Tensoroperationen erlauben die Verarbeitungen von Datensätzen aggregiert zu Tensoren. Als Beispielanwendungen mit Tensoren sind in List. 3 die Kreuzvalidierung auf einem Datensatz und in List. 4 die binäre Exponentiation in *HyPerScript* angeführt. Die binäre Exponentiation benötigt die Identitätsmatrix und die Multiplikation und steht stellvertretend für die C++-Implementierung im Datenbanksystem. Die aufgeführte einfache Kreuzvalidierung schneidet aus dem Datensatz jedes Tupel einmal heraus, um dieses als Testmenge und die restlichen als Trainingsmenge zu verwenden. Darauf wird lineare Regression (in einer eigenen Funktion) angewendet, um die optimalen Gewichte zu berechnen. Mit diesen wird der Vorhersagefehler auf dem herausgeschnitten Datensatz bestimmt.

```

create or replace function linearregression(x float[], y float[]) returns float[] as $$
  return (array_transpose(x)*x)^-1*(array_transpose(x)*y);
$$ language 'hyperscript' strict;
create or replace function cross_validate(x float[], y float[]) returns float as $$
  var error=0; var n=array_length(x,2); var m=array_length(x,1);
  select index i from sequence(2,n-1){
    var weights_o=linearregression(array_slice(x,1,i-1,1,m)||array_slice(x,i+1,n,1,m),
      array_slice(y,1,i-1,1,n)||array_slice(y,i+1,n,1,1));
    error=error+((array_slice(x,i,i,1,m)*weights_o)[1][1]-y[i][1])^2;
  }
  error=error/(n-2); return error;
$$ language 'hyperscript' strict;

```

List. 3: Einfache Kreuzvalidierung in *HyPerScript*: Mittels `array_slice()` und der Konkatenation wird schrittweise eine Zeile aus dem Datensatz herausgeschnitten und als Testdatensatz verwendet. `array_slice()` erwartet als Argument den Tensor und für jede Dimension die Start- sowie Endposition.

```

create or replace function pow(a_in float[][], e_in int) returns float[] as $$
  var a=a_in; var e=e_in; if( e<0 ) { e=e*-1; a=array_transpose(a); }
  var mask = 1<<63; var result = array_identity(array_ndims(a));
  while(mask>0){ result=result*result; if( e&mask>0 ){ result=result*a; } mask=mask>>1; }
  return result;
$$ language 'hyperscript' strict;

```

List. 4: Binäre Exponentiation `pow()` von Tensoren implementiert in *HyperScript*: als Eingabe wird ein Tensor und ein Exponent erwartet, eine Schleife mit Multiplikationen berechnet die Exponentiation.

3.2 HyPerScript für betriebswirtschaftlich transaktionale Anwendungen

HyperScript erlaubt über die SQL-Schnittstelle betriebswirtschaftlich transaktionale Anwendungen auszuführen. Beispiele hierfür sind die TPC-C-, TPC-E- und TPC-H-Benchmarks des Transaction Processing Performance Councils¹¹. So modelliert der TPC-C-Benchmark ein Handelsunternehmen mit eingehenden Aufträgen (New-Order) von Kunden und misst, wie viele schreibende Transaktionen ein Datenbanksystem pro Zeiteinheit verarbeiten kann.

```

create type newOrderPosition as (line_number int not null,supware int not null,itemid int not null, qty
int not null);
create function newOrder (w_id int not null, d_id smallint not null, c_id int not null, positions setof
newOrderPosition not null, datetime timestamp not null) as $$
  select w_tax from warehouse w where w.w_id=w_id;
  select c_discount from customer c where c.w_id=w_id and c.d_id=d_id and c.c_id=c_id;
  select d_next_o_id as o_id,d_tax from district d where d.w_id=w_id and d.d_id=d_id;
  update district set d_next_o_id=o_id+1 where d.w_id=w_id and district.d_id=d_id;
  select count(*) as cnt from positions;
  select case when count(*)=0 then 1 else 0 end as all_local from positions where supware<>w_id;
  insert into "order" values (o_id,d_id,w_id,c_id,datetime,null,cnt,all_local);
  insert into neworder values (o_id,d_id,w_id);
  update stock
  set s_quantity=case when s_quantity>=qty+10 then s_quantity-qty else s_quantity+91-qty end,
  s_remote_cnt=s_remote_cnt+case when supware<>w_id then 1 else 0 end,
  s_order_cnt=s_order_cnt+1, s_ytd=s_ytd+qty
  from positions where s_w_id=supware and s_i_id=itemid;
  insert into orderline
    select o_id,d_id,w_id,line_number,itemid,supware,null,qty,
    qty*i_price*(1.0+w_tax+d_tax)*(1.0-c_discount),
    case d_id when 1 then s_dist_01 when 2 then s_dist_02 when 3 then s_dist_03 when 4 then
    s_dist_04 when 5 then s_dist_05 when 6 then s_dist_06 when 7 then s_dist_07 when
    8 then s_dist_08 when 9 then s_dist_09 when 10 then s_dist_10 end
    from positions, item, stock
    where itemid=i_id
    and s_w_id=supware and s_i_id=itemid
  returning count(*) as inserted;
  if (inserted<cnt) rollback;
$$ language 'hyperscript' strict;

```

List. 5: *HyperScript*-Funktion `newOrder()` des TPC-C-Benchmarks: Die Funktion nimmt eine Menge von Bestellpositionen `positions` für ein Warenhaus `w_id` eines Distrikts `d_id` von einem Kunden `c_id` entgegen.

List. 5 zeigt die zugehörige Prozedur in *HyperScript*. Die Prozedur nimmt einen neuen Auftrag aus mehreren Bestellpositionen (`setof`) für ein Warenhaus (`w_id`) von einem

¹¹ <http://www.tpc.org>

Kunden (`c_id`) in einem Distrikt (`d_id`) entgegen. Zunächst wird dieser Auftrag in der Datenbank angelegt. Anschließend aktualisiert die Prozedur die Lagerbestände in `stock` und fügt die Bestellposition zusammen mit dem berechneten Preis in `orderline` ein.

HyPerScript ermöglicht Attributwerte von SQL-Anfragen in Variablen zwischenspeichern, um sie so wiederverwenden zu können. So werden beispielsweise der Steuersatz des Warenhauses und der Kundenrabatt in den Variablen `w_tay` und `c_discount` zwischengespeichert und beim Einfügen in die Relation `orderline` wiederverwendet.

Das Beispiel prüft die Atomarität und Konsistenz für Transaktionen, die implizit durch die Verwendung eines Datenbanksystems gegeben ist. So überprüft die Prozedur am Ende, ob die Bestellungen erfolgreich in `orderline` eingefügt worden sind (die Zahl der eingefügten Bestellungen valide ist) und setzt andernfalls die Transaktion zurück (`rollback`). Außerdem verdeutlicht das Beispiel die kompakte Schreibweise: Die *HyPerScript*-Prozedur kommt mit weniger als 40 Zeilen aus, während ein C++-Programm für dieselbe Funktion deutlich mehr Zeilen beansprucht¹².

4 HyPerScript für Datenanalysealgorithmen

Neben den klassischen transaktionalen Anwendungen ermöglicht *HyPerScript* auch die Ausführung beliebiger Algorithmen im Datenbanksystem. In diesem Kapitel stellen wir dazu beispielhaft Formulierungen einiger Algorithmen zur Datenanalyse dar. Um möglichst verschiedene Bereiche der Datenanalyse abzudecken, wählen wir grundlegende Algorithmen aus Assoziationsanalyse (Apriori-Algorithmus), Clustering (k-Means und DBSCAN), Graphmetriken (PageRank) und Regression (lineare Regression). Modernes SQL – mit Fensterfunktionen (*window functions*) und rekursiven temporären Tabellen (*recursive common table expressions*) – ist bereits Turing-vollständig. Mit *HyPerScript* lassen sich Algorithmen jedoch teils deutlich kompakter und verständlicher formulieren.

Dabei folgen die *HyPerScript*-Formulierungen jeweils folgendem Aufbau: Für jeden Algorithmus wird ein Schema als eigener Namensbereich definiert. Bei Aufruf der UDF `<Schemaname>.run()` wird der Algorithmus ausgeführt. Das Ergebnis des Algorithmus wird in eine Relation desselben Namensbereiches materialisiert. Die Logik der Algorithmen wird großteils in SQL-Funktionen ausgedrückt. Die UDF `<Schemaname>.run()` stellt diese Funktionen zum eigentlichen Algorithmus zusammen, beispielsweise durch iterativen Aufruf bis ein Abbruchkriterium erreicht wird.

Alle hier untersuchten Algorithmen sind in *HyPer* bereits als Operatoren vorhanden [Pa17; Th15]. Das heißt, sie wurden, ähnlich wie relationale Operatoren (etwa Hash-Join oder Aggregation), im Datenbankkern implementiert und lassen sich kombinieren, um SQL-Anfragen abzubilden. Die folgenden Abschnitte erläutern jeweils kurz den gewählten Algorithmus, stellen die Formulierung in *HyPerScript* dar und beschreiben die Vergleichsoperatoren. Im

¹² vgl.: <https://github.com/evanj/tpccbench/blob/master/tpcctables.cc>

anschließenden Evaluationskapitel werden Operatoren und *HyPerScript*-Formulierungen miteinander verglichen.

4.1 Assoziationsanalyse

Der 1993 eingeführte Apriori-Algorithmus [AIS93] ist der bekannteste Vertreter im Bereich der Assoziationsanalyse. Er basiert auf Warenkorbdaten, die als Tupel aus Transaktionsnummer (*tid*) und Waren abgelegt werden (sales: $\{\{tid, item\}\}$). Zuerst zählt er häufig vorkommende Item-Mengen, die mit einer minimalen relativen Häufigkeit (*Support*) von mindestens s_0 in allen Warenkörben vorkommen und bildet anschließend Assoziationsregeln daraus. Die Item-Mengen wachsen mit jeder Iteration um ein Element beginnend mit der einelementigen Menge. Dabei ist die Anzahl der Iterationen und zu überprüfender Mengen durch das *Apriori-Prinzip* begrenzt. Das Prinzip besagt, dass Mengen an Items, deren Teilmengen nicht häufig auftreten, selbst nicht häufig auftretend sein können. List. 6 zeigt den Aufruf des integrierten Operators wie der Skript-Funktion mit einem minimalen Support von 10 %, deren genaue Funktionsweisen kurz erklärt werden.

```
-- nativer Operator:
select * from apriori((table aprioriscript.sales),0.1,1);
-- HyPerScript
select aprioriscript.findFI(10); select * from aprioriscript.frequentitemsets;
```

List. 6: Aufruf des Apriori-Algorithmus als Operator wie als gespeicherte Prozedur.

Die Implementierung in *HyPerScript* (siehe List. 7) basiert auf zu Warenkörben aggregierten, häufig auftretenden Item-Mengen, die mit rekursivem SQL iterativ erweitert werden. Hier werden die Arrays als Mengen verwendet und in jeder Iteration um ein Element erweitert. Anschließend wird die Häufigkeit der Tupel gezählt. Dazu wird jedes Itemset mit jedem Warenkorb anhand des Mengenoperators `Tupel <@ Warenkorb` verglichen.

```
create or replace function aprioriscript.findFrequentItemsets(minsupp integer not null) as $$
with recursive -- Erzeugung von Warenkörben
transactions (tid, bucket) as (select tid, array_agg(item) from aprioriscript.sales group by tid),
-- häufig auftretende 1-Item-Mengen
sales_supp as (select item from aprioriscript.sales group by item having count(*) >= minsupp),
frequentitemsets as ( -- Items mit support >= minsupp
(select distinct array[p.item)::integer[] as items from sales_supp p) -- 1-Item-Mengen  $L_1 \in \mathcal{L}_1$ 
union all ( -- erweitere Item-Mengen um ein Element:  $L_{k-1}$  erweitert
select distinct array_append(t.items,p.item::integer)::integer[] --  $L_k$ 
from frequentitemsets t, sales_supp p
where minsupp <= ( -- zähle Support
select count(*) from transactions t2 --  $C_k \in \mathcal{C}_k$ . support( $C_k$ ) >= minsupp  $\Rightarrow C \in \mathcal{L}_k$ 
where array_append(t.items,p.item::integer)::integer[] <@ ( t2.bucket )
) and t.items[(select count(*) from unnest(t.items))]<p.item -- nur sortierte Arrays
))
insert into aprioriscript.frequentitemsets (select * from frequentitemsets);
$$ language 'hyperscript' strict;
```

List. 7: Ermittlung der häufig auftretenden Item-Mengen für den Apriori-Algorithmus: Die Funktion erhält den minimalen Support s_0 als Parameter übergeben und berechnet eine rekursiv wachsende Relation mit häufigen Itemsets, beginnend bei den einelementigen, also den Waren als einelementige Arrays.

Der in *HyPer* implementierte Operator basiert auf der Speicherung der Elemente in einem mit jeder Iteration wachsenden Präfixbaum. Besonderheiten der Implementierung in *HyPer* sind die Parallelität pro Iterationsschritt sowie die Behandlung von Duplikaten. So berücksichtigen die Assoziationsregeln die Häufigkeit gleicher Elemente in Warenkörben.

4.2 Clustering

Ein weiterer wichtiger Bereich der Datenanalyse ist Clustering, also die Gruppierung ähnlicher Datenpunkte. Mit k-Means [L182] und DBSCAN [Sc17] formulieren wir zwei klassische Clustering-Algorithmen in *HyPerScript*, die über ganz unterschiedliche Iterationsmuster verfügen. k-Means weist in jeder Iteration jeden Datenpunkt dem nächstgelegenen Cluster zu. Durch die Zuweisung verändert sich der Mittelpunkt der Cluster, wodurch sich in der nächsten Iteration wiederum Zuordnungen ändern können. DBSCAN hingegen prüft für jeden Datenpunkt, ob in der Nähe ausreichend viele andere Datenpunkte liegen. Falls ja und falls diese Datenpunkte bereits einem Cluster zugeordnet wurden, wird auch der aktuell betrachtete Datenpunkt dem Cluster hinzugefügt. Andernfalls wird ein neuer Cluster begründet. Falls sich nicht genug andere Datenpunkte in der Nähe befinden, wird der aktuell betrachtete Datenpunkt als *Noise* deklariert. Anschließend wird der nächste Datenpunkt geprüft. List. 8 zeigt den Aufruf der beiden Clustering-Algorithmen sowohl als Operator wie als Skript-Funktion.

```
-- nativer Operator:
select * from kmeans((select x,y from kmeansscript.points),
                    (select x,y from kmeansscript.points LIMIT 5));
select * from dbscan((select x,y from dbscanscript.points),20,2);
-- HyPerScript:
select kmeansscript.run(5);
select dbscanscript.run(20,2);
```

List. 8: Aufruf der Clustering-Algorithmen in *HyPer*: Eingabedaten des k-Means-Operators sind die Datenpunkte und die initialen Zentren (im dargestellten Fall $k = 5$ zufällig gewählte Datenpunkte). Eingabe des DBSCAN-Operators sind ebenfalls die Datenpunkte, sowie als Parameter der Suchradius für nahegelegene Datenpunkte ϵ und die minimale Anzahl an Punkten pro Cluster minPts . Parameter für die *HyPerScript*-Funktionsaufrufe sind ebenfalls k für k-Means und ϵ sowie minPts für DBSCAN.

Die *HyPerScript*-Implementierung für k-Means (s. List. 9) basiert auf der in *HyPer* besonders effizient implementierten Fensterfunktion (*window function*) [Le15], die pro Datenpunkt eine Rangfolge der nächstgelegenen Clusterzentren berechnet. Aus den Mittelwerten der zugeordneten Punkte werden dann die neuen Clusterzentren bestimmt. Diese zwei Schritte werden wiederholt, bis die Zuordnung der Datenpunkte zu Clustern stabil ist.

Die DBSCAN-Implementierung [Hu17] (s. List. 10) basiert auf der rekursiven Erweiterung von Clustern: Zuerst bildet jeder Punkt einen eigenen Cluster. Anschließend werden Cluster, die weniger als ϵ weit voneinander entfernt liegen, vereinigt. Sowohl k-Means als auch DBSCAN liegen als in *HyPer* implementierte Operatoren vor. Der k-Means-Operator [Pa17] erhält als Eingabe die Datenpunkte sowie k initiale Clusterzentren. Im Normalfall wird

```

create or replace function kmeansscript.run(centers integer not null) as $$
truncate kmeansscript.clusters;
insert into kmeansscript.clusters(select id,x,y,0 from (select *,rank() over (order by id) from
kmeansscript.points) where rank <= centers);
while (true){
select kmeansscript.computeCenters() as c_count;
if(c_count = 0){ break; }
}
$$ language 'hyperscript' strict;
create or replace function kmeansscript.computeCenters() returns integer not null as $$
table clusters_tmp(cid int, x float, y float, count int);
insert into clusters_tmp (select cid, avg(px), avg(py), count(*) from (
select cid, p.x as px, p.y as py, rank() OVER ( partition by p.id
order by (p.x-c.x)*(p.x-c.x)+(p.y-c.y)*(p.y-c.y) asc, (c.x*c.x+c.y*c.y) asc)
from kmeansscript.points p, kmeansscript.clusters c ) x where x.rank=1 group by cid);
select count(*) as c_count from (select * from kmeansscript.clusters except select * from
clusters_tmp);
truncate kmeansscript.clusters; -- lösche alte Daten
insert into kmeansscript.clusters (select * from clusters_tmp);
return c_count; -- gebe Anzahl der geänderten Zentren zurück
$$ language 'hyperscript' strict;

```

List. 9: Fixpunktiteration für k-Means: Die SQL-Funktion `computeCenters()` berechnet die Clusterzentren – im Beispiel als `avg(px)`, `avg(py)` – und nutzt die Fensterfunktion `rank()`, um alle Datenpunkte dem jeweils nächstgelegenen Clusterzentrum zuzuordnen. Die Rahmenfunktion `run()` ruft `computeCenters()` so oft auf, bis in einer Iteration kein Datenpunkt mehr einem anderen Clusterzentrum zugeordnet wird.

```

create or replace function dbscanscript.run(eps float, minPoints int) as $$
select count(*) as maxiter from dbscanscript.points;
select index from sequence (1,maxiter){
select dbscanscript.insertCluster(index,eps,minPoints,maxiter) as cond;
if(cond=0){ break; }
}
$$ language 'hyperscript' strict;
create or replace function dbscanscript.insertCluster(clusterid int not null, eps float not null,
minPoints int not null, maxiter int not null) returns int not null as $$
table pointstmp (id int, x float, y float);
insert into pointstmp(select * from dbscanscript.points except (select id,x,y from dbscanscript.
pointsclustered) LIMIT 1);
select count(*) as ret from dbscanscript.pointstmp;
select index from sequence(1,maxiter){
select count(*) as newc from pointstmp c, dbscanscript.points p
where (p.x-c.x)^2+(p.y-c.y)^2<eps^2 and c.id<>p.id;
insert into pointstmp(
select * from (select * from dbscanscript.points except select * from pointstmp) c where exists
(select * from pointstmp p where (p.x-c.x)^2+(p.y-c.y)^2<eps^2 and c.id<>p.id)
);
if(newc=0){ break; }
ret=ret+newc;
}
if(ret < minPoints) { clusterid=NULL; }
insert into dbscanscript.pointsclustered(select *,clusterid, false from dbscanscript.pointstmp);
return ret;
$$ language 'hyperscript' strict;

```

List. 10: DBSCAN: Die Rahmenfunktion `run()` ruft `insertCluster()` auf: Pro Iteration entsteht ein neuer Cluster mit der als ersten Parameter angegebenen ID. Die Funktion `insertCluster()` nimmt pro Aufruf einen noch keinem Cluster zugewiesenen Punkt als neuen Cluster und erweitert diesen, bis er sich nicht mehr verändert. Die Funktion gibt die Anzahl der sich im Cluster befindlichen Punkte zurück. Die Anzahl der Iterationen ist limitiert durch die Anzahl aller Punkte.

der Operator parallel aufgerufen, das heißt die Datenpunkte liegen auf mehrere Threads verteilt vor. Die initialen Clusterzentren werden dann zunächst in jeden Thread repliziert, anschließend ordnet jeder Thread lokal seine Datenpunkte den nächstgelegenen Clusterzentren zu. Zur Ermittlung der neuen Clusterzentren müssen die Ergebnisse aller Threads zusammengefügt werden, jedoch werden die Ergebnisse bereits threadlokal aggregiert, um die Menge der zu übertragenen Daten sowie die Menge synchronisierter Berechnungen zu minimieren. Die neuen Clusterzentren werden nun wieder in alle Threads repliziert und die nächste Iteration beginnt, solange sich noch Clusterzuordnungen ändern. Der Operator generiert LLVM-Code, sodass der Algorithmus nahtlos in SQL-Anfragen hineinkompiliert werden kann und teure Funktionsaufrufe vermieden werden.

Der DBSCAN-Operator erhält als Eingabe die Datenpunkte sowie die Parameter ϵ und minPts . Für jeden unbesuchten Datenpunkt wird geprüft, ob genügend (minPts) andere Datenpunkte in der ϵ -Umgebung des Datenpunktes liegen, und welchem Cluster diese zugeordnet sind. Je nach Ergebnis dieser Prüfung wird der Datenpunkt einem Cluster zugeordnet, als *Noise* deklariert oder ein neuer Cluster wird begründet. Auch dieser Operator generiert LLVM-Code, ist im Gegensatz zu *k-Means* in *HyPer* jedoch nicht parallel implementiert.

4.3 Lineare Regression

Lineare Regression ist ein Optimierungsproblem um Labels vorherzusagen, dem ein lineares Modell $m(x) = w * x + w_0$ auf Daten x mit Gewichten w zugrunde liegt. Lineare Regression lässt sich als Optimierungsproblem mittels Gradientenabstiegsverfahren lösen. Der in *HyPer* implementierte Operator nutzt diesen Ansatz. Wir haben das Gradientenabstiegsverfahren mit festem Gradienten bei linearer Regression ebenfalls in *HyPerScript* implementiert. Als Alternative zu einem Optimierungsverfahren ist lineare Regression numerisch mit dem Gleichungssystem $\vec{w} = (X^T X)^{-1} X^T \vec{y}$ lösbar. List. 11 zeigt den Aufruf der verschiedenen Varianten.

```
-- nativer Operator:
select * from linearregression((select x_1, x_2, y from linregscript.sample));
-- HyPerScript:
select linregscript.run();
-- mit Matrixoperationen:
select (array_transpose(x)*x)^-1*(array_transpose(x)*y)
from (select array_agg(x) x from (select ARRAY[1,x_1,x_2] as x from linregscript.sample) sx) tx,
      (select array_agg(y) y from (select ARRAY[y] y from linregscript.sample) sy) ty;
```

List. 11: Aufruf linearer Regression in *HyPer*: als Operator wie als gespeicherte Prozedur und komplett mit Matrixoperationen.

List. 12 zeigt den Gradientenabstieg programmiert in *HyPerScript*. Dazu haben wir den Gradienten hart kodiert, den wir in jeder Iteration für jedes Tupel berechnen. Als Besonderheit dieser Implementierung hängen die Rückgabewerte nur von den Eingabeparametern ab

(sind *immutable*, bzw. *stable* bei Lesezugriff auf die Datenbasis). Das ist möglich, da eine temporäre Tabelle (table) als Funktionsparameter mit übergeben werden kann (setof).

```

create or replace function linregscript.iterate(tuples float[] not null, y float not null, weights float
[] not null, learningrate float not null) returns float[] not null as $$
var gradient=array_access(weights,1);
select index as i from sequence(1,array_length(tuples,1)){
  gradient = gradient + tuples[i]*weights[i+1];
}
gradient = gradient-y;
var ret=array_set(weights,array_access(weights,1)-learningrate*gradient*2,1);
select index as i from sequence(1,array_length(tuples,1)){
  ret=array_set(ret,weights[i+1]-learningrate*gradient*2*tuples[i],i+1);
}
return ret;
$$ language 'hyperscript' strict immutable;

create type features as (a float, b float, c float);
create or replace function linregscript.gradientdescent(tuples setof features, learningrate float not
null, maxIter int not null) returns float[] not null as $$
var weights='{1,1,1}':float[];
select index as i from sequence(1,maxIter){
  select a,b,c from tuples{
    weights=linregscript.iterate(ARRAY[a,b],c,weights,learningrate);
  }
}
return weights;
$$ language 'hyperscript' strict stable;

create or replace function linregscript.run(learningrate float not null, maxIter int not null) returns
float[] not null as $$
table sample(a float, b float, c float);
insert into sample (select x_1,x_2,y from linregscript.sample);
var weights=linregscript.run(sample,learningrate,maxIter);
$$ language 'hyperscript' strict stable;

```

List. 12: Stochastischer Gradientenabstieg mit linearer Regression in *HyPerScript*: Eine Funktion `iterate()` berechnet den Fehler pro Tupel mit den angegebenen Gewichten und gibt die optimierten Gewichte zurück. Eine Rahmenfunktion iteriert über alle Tupel, die die Lernrate und die Anzahl der Iterationen als Argumente erwartet.

Der *HyPer*-Operator unterstützt Parallelität, indem er wahlweise durch ein Flag parallelisiert zuerst lokale Gewichte berechnet oder die optimalen Gewichte pro eingehendem Tupel vorberechnet. Der Implementierung des Gradientenabstiegs in C++ gleicht der *HyPerScript*-Prozedur.

4.4 PageRank

PageRank ist ein Algorithmus für gerichtete Graphen, der ursprünglich zur Bestimmung der Wichtigkeit von Internetseiten gedient hat [BP98]. Eine Internetseite (Knoten) gilt dabei als wichtig, wenn wichtige Internetseiten auf sie verweisen (gerichtete Kanten). In jeder Iteration verteilt jeder Knoten einen Teil $(1 - \alpha)$ seines PageRank-Wertes gleichwertig über die ausgehenden Kanten auf die nächsten Knoten, und erhält entsprechend die Werte der auf ihn

verweisenden Knoten. Die Summe der eingehenden Werte ist der neue PageRank-Wert des Knotens. List. 13 zeigt den Aufruf von PageRank mittels Operators wie als Skript-Funktion, jeweils ohne Dämpfung.

```
-- nativer Operator:
select * from pagerank((table pagerankscript.edges),0);
-- HyPerScript:
select pagerankscript.run(0); select * from pagerankscript.pagerank;
```

List. 13: Aufruf von PageRank in *HyPer*: Sowohl der Operator als auch die *HyPerScript*-Formulierung benötigen als Eingabe die Kanten sowie den Dämpfungsfaktor $\alpha = 0$.

Die Berechnung der neuen PageRank-Werte lässt sich in SQL durch einen Join mit den Vorgängerknoten sowie eine Summen-Aggregation abbilden. Zur einfachen Formulierung der Iteration benötigen wir die Schleifen aus *HyPerScript* (s. List. 14).

```
create or replace function pagerankscript.computePR(alpha float not null) returns int not null as $$
  table pr_tmp(node int, edges int);
  insert into pr_tmp (
    select VTo, alpha*(cast((select count(*) from pagerankscript.pagerank) as float))
      +(1-alpha)*sum(Beitrag)
    from (select e.VTo, p.Pagerank/(select count (*) from pagerankscript.edges x where x.VFrom=e.VFrom)
      as Beitrag from pagerankscript.edges e, pagerankscript.pagerank p where e.VFrom=p.Node) i
    group by VTo);
  select count(*) as c_count from (select * from pagerankscript.pagerank except select * from pr_tmp);
  truncate pagerankscript.pagerank;
  insert into pagerankscript.pagerank (select * from pr_tmp);
  return c_count; -- gebe Anzahl der geänderten Werte zurück
$$ language 'hyperscript' strict;

create or replace function pagerankscript.run(alpha float not null, maxIterations int not null) as $$
  select index as i from sequence(0,maxIterations) {
    select pagerankscript.computePR(alpha) as c_count;
    if(c_count = 0){ break; }
  }
$$ language 'hyperscript' strict;
```

List. 14: Approximation für PageRank: Die Rahmenfunktion `run()` ruft die Berechnung der PageRank-Werte `computePR()` solange auf, bis die Werte stabil sind. Die SQL-Funktion `computePR()` berechnet die PageRank-Werte aller Knoten mittels Aggregation. Dabei behält jeder Knoten den Anteil α seines alten PageRank-Wertes, und verteilt den Rest auf die durch ausgehende Kanten direkt mit ihm verbundenen Knoten.

Auch PageRank liegt als in *HyPer* implementierter Operator vor [Th15]. Die Besonderheit dieses Operators ist die Erzeugung eines temporären Indexes, der schnellen Zugriff auf die Vorgängerknoten erlaubt. Während der Ausführung des Algorithmus muss dann nicht mehr auf die Eingabedaten zugegriffen werden, sondern lediglich der Index verwendet und die PageRank-Werte aktualisiert werden. Dies kann von mehreren Threads parallel durchgeführt werden. Lediglich ein Synchronisationspunkt am Ende jeder Iteration ist nötig, um den Threads zu signalisieren, dass von nun an die neuen PageRank-Werte genutzt werden. Im folgenden Kapitel werden die Laufzeiten der Operatoren mit denen der vorgestellten *HyPerScript*-Implementierungen verglichen.

5 Evaluation

Die Evaluation vergleicht die Leistungsfähigkeit von mit *HyPerScript* erstellten Prozeduren und von Operatoren im Datenbanksystem. Alle Experimente wurden auf einem Rechner mit Intel Xeon E5-2660 v2 CPU Prozessor (20 Kerne mit jeweils 2,20 GHz) und 256 GB DDR4 RAM gemessen, als Betriebssystem diente Ubuntu 17.04. Für den Apriori-Algorithmus wurden 100 verschiedenen Waren in 1000 Warenkörben synthetisch hergestellt. Die Anzahl der Elemente pro Warenkorb variierte gleichverteilt im Intervall $[0, 10]$. Für Clustering erzeugten wir 10^6 Punkte, deren x- wie y-Koordinaten gleichverteilt im Intervall $[0, 10^6]$ lagen. PageRank arbeitete auf 10^5 Knoten mit genauso vielen Kanten, für lineare Regressionen erzeugten wir 10^6 Tupel. Alle Experimente wurden dreimal wiederholt und für die Messungen der Median verwendet.

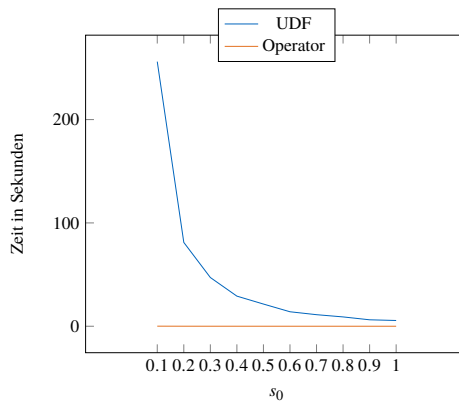
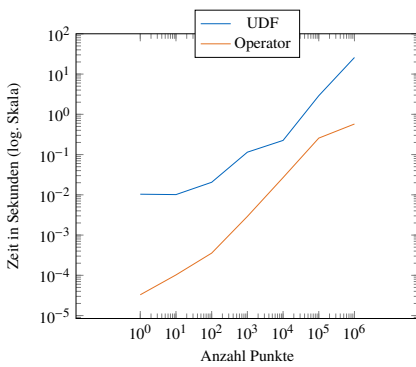


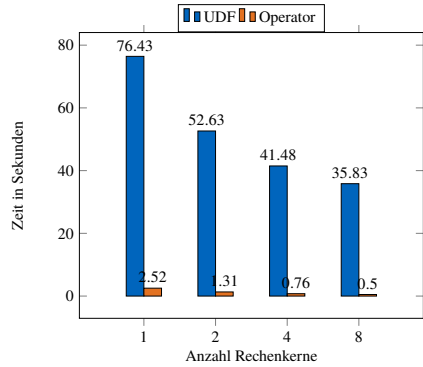
Abb. 1: Laufzeit für Assoziationsanalyse mit Apriori bei konstant 20 Rechenkernen: Abhängigkeit vom minimalen Support s_0 als Parameter von Apriori. Je größer s_0 , desto strenger die Auswahl assoziierter Produkte, da weniger Kandidaten.

Für den Apriori-Algorithmus veränderten wir den minimalen Support (je größer, desto weniger häufige Item-Mengen existieren, s. Abb. 1). Mit wachsendem minimalen Support gleichen sich die Laufzeiten an, da die Zahl häufiger Item-Mengen abnimmt.

Die Laufzeiten der Clustering-Algorithmen wachsen linear mit der Eingabegröße (s. Abb. 2a, 3). Der k-Means-Algorithmus berechnet um 30 % schneller mit Hinzunahme eines weiteren Kernes (s. Abb. 2b). Die Parallelität des darunterliegenden Datenbanksystems gewährleistet die implizite parallele Ausführung des als Prozedur implementierten k-Means-Algorithmus; der k-Means-Operator ist explizit parallelisiert angelegt. Weder DBSCAN als Operator, noch als gespeicherte Prozedur unterstützen Skalierung. Während der Operator keine parallele Berechnung unterstützt, skalieren die verwendeten SQL-Anfragen in der *HyPerScript*-Prozedur schlecht, da pro (nicht parallelisierter) Iteration nur ein Tupel als neuer Cluster initialisiert wird.



(a) k-Means: Eingabegröße.



(b) k-Means: Skalierung.

Abb. 2: Laufzeit für Clustering-Algorithmus k-Means mit fünf Zentren: (a) Laufzeit in Abhängigkeit von der Eingabegröße bei konstant 20 Rechenkernen sowie (b) in Abhängigkeit von der zur Verfügung stehenden Rechenkerne.

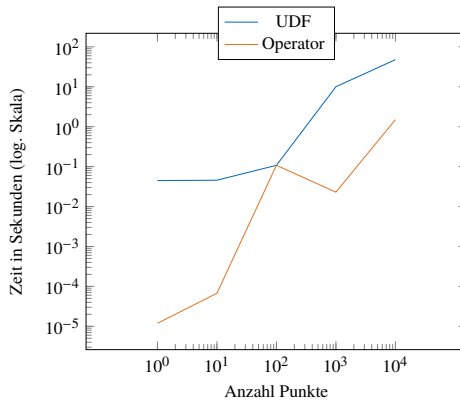


Abb. 3: Laufzeit für Clustering-Algorithmus DBSCAN mit $\epsilon = 20$ und $\text{minPts} = 2$: Laufzeit in Abhängigkeit von der Eingabegröße bei konstant 20 Rechenkernen.

Lineare Regression mit Gradientenabstieg in *HyPerScript* ist für bis etwa 100 Tupel schneller als der implementierte Operator (s. Abb. 4a). Beide Versionen sind gleich strukturiert, beide materialisieren zuerst die Tupel lokal und berechnen anschließend die optimalen Gewichte mit festem Gradienten. Allerdings kompiliert *HyPerScript* beide Schritte zu LLVM-Code basierend auf den Datenbanktypen, während der Operator C++-Funktionen auf den Basistypen aufruft. Bei wenigen Tupeln überwiegt der Aufwand durch Funktionsaufrufe, bei vielen Tupeln der Aufwand durch die Nutzung komplexer Datentypen.

Die Laufzeit des in *HyPerScript* implementierten PageRank-Algorithmus ist bei wenigen

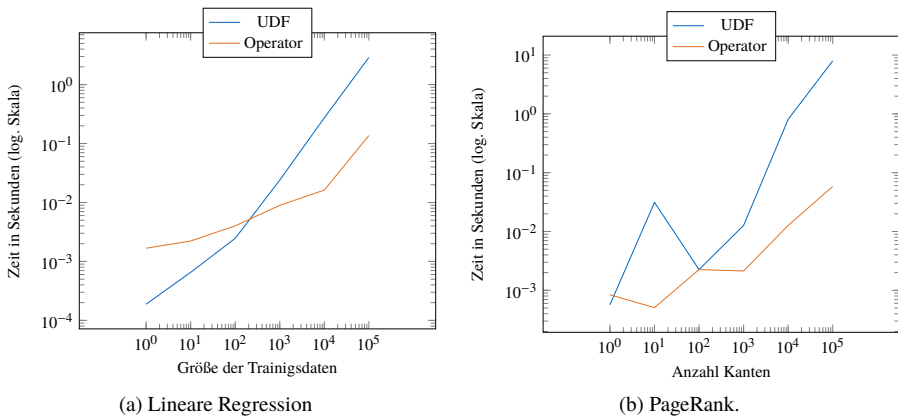


Abb. 4: Laufzeit bei jeweils konstant 20 Rechenkernen von (a) linearer Regression bei einer Lernrate von 0,85 und 45 Iterationen in Abhängigkeit der Größe zu trainierender Daten, sowie (b) zur Berechnung des PageRank-Wertes bei Veränderung der Anzahl der Kanten.

Tupeln vergleichbar mit der des PageRank-Operators (s. Abb. 4b). Bei wenigen Kanten ist die Stored-Procedure-Version des PageRank-Algorithmus sogar schneller als der optimierte Operator von *HyPer*, verliert allerdings mit zunehmender Anzahl an Kanten. Bei wenigen Kanten zeigt sich die Mehraufwand des integrierten Operators, da dieser ein Wörterbuch für die Knoten anlegt und Kanten in einer dünnbesetzte Matrix als Compressed-Sparse-Row (CSR) speichert. Mit zunehmender Kantenanzahl amortisieren sich der Mehraufwand für das Wörterbuch und die CSR-Datenstruktur, sodass der Operator schneller als die Skriptfunktion den PageRank-Wert berechnet.

Zusammenfassend zeigt die Evaluation, dass *HyPerScript* es ermöglicht, Prototypen zu entwickeln, ohne die relationale Algebra des Datenbanksystems zu erweitern. Die Laufzeiten der Prozeduren sind bei wenigen Tupeln vergleichbar zu derer der Operatoren der relationalen Algebra, erlauben allerdings nur Parallelität, wenn die zugrundeliegenden SQL-Anfragen entsprechend skalieren, wie es bei k-Means-Clustering zu sehen ist.

6 Zusammenfassung und Fazit



Skriptsprachen erlauben betriebswirtschaftlich transaktionale Anwendungen im Kern von Datenbanksystemen auszuführen. Typische Anwendungsfälle von SQL-Skriptsprachen sind die Implementierung von Datenbanktriggern und die serverseitige Ausführung von Teilen einer Applikation. Letzteres wird beispielsweise bei der Ausführung von Benchmarks benötigt, weshalb die große Mehrheit der Datenbanksysteme über solche prozeduralen SQL-Erweiterungen verfügt. Auch in *HyPer*, dem dieser Arbeit zugrundeliegenden Hauptspeicher-

datenbanksystem, wurde das prozedurale *HyPerScript* zunächst für die Leistungsbewertung verwendet.

Ziel der vorliegenden Arbeit war, das Potenzial von SQL-Skriptsprachen für andere Anwendungsfälle, insbesondere für Algorithmen zur Datenanalyse, auszuloten. Am Beispiel von *HyPerScript* zeigten wir, dass nutzerseitig definierte Funktionen in Kombination mit prozeduralen Ausdrücken eine kompakte Schreibweise von Algorithmen zur Datenanalyse erlauben. So zeigten wir anhand einer Beispielimplementierung, dass eine TPC-C-Implementierung in *HyPerScript* nur einen Bruchteil an Codezeilen einer Implementierung in einer rein prozeduralen Sprache benötigt. Neben Schleifen und Konditionen halfen Tensoroperationen, welche wir im Rahmen dieser Arbeit in *HyPer* integrierten, bei der numerischen Berechnung linearer Regression wie bei der Assoziationsanalyse.

Durch die Ausführung von Datenanalysealgorithmen im Datenbanksystem statt auf einer separaten Datenanalyseplattform werden teure ETL-Prozesse eingespart. Unsere Evaluation zeigt, dass die in *HyPerScript* formulierten Algorithmen nicht mit der Schnelligkeit hartkodierter Datenbankoperatoren mithalten können. Durch die kompakte Formulierung und die hohe Flexibilität im Vergleich zu hartkodierten Operatoren ist *HyPerScript* jedoch dennoch sinnvoll einsetzbar, um neue Algorithmen auf Datensätzen zu studieren ohne den Datenbankkern zu verändern. Zwar ist eine korrekte Ausführung der Algorithmen sichergestellt, deren Leistungsfähigkeit als Prozedur ist jedoch nachrangig. Eine SQL-nahe Skriptsprache wie *HyPerScript* spricht in erster Linie Nutzer aus dem Datenbankbereich an. Insofern ist *HyPerScript* erst als Beginn der Entwicklung einer universellen Sprache zu sehen, die Datenwissenschaftler direkt auf Datenbanksystemen nutzen können, um Daten aufzubereiten und zu analysieren.

Danksagung

Diese Arbeit ist im Rahmen des Projekts TUM Living Lab Connected Mobility (TUMLLCM) entstanden, das vom Bayerischen Staatsministerium für Wirtschaft, Energie und Technologie (StMWi) durch das Zentrum Digitalisierung.Bayern, einer Initiative der Bayerischen Staatsregierung, finanziert wird. Das diesem Bericht zugrundeliegende Vorhaben wurde mit Mitteln des Bundesministeriums für Bildung und Forschung unter dem Förderkennzeichen TUM: 01IS17049 gefördert, sowie vom Europäischen Forschungsrat (ERC) im Rahmen des Forschungs- und Innovationsprogramms der Europäischen Union (Horizont 2020) (Finanzhilfvereinbarung Nr. 725286).   Die Verantwortung für den Inhalt dieser Veröffentlichung liegt bei den Autoren.

Literatur

- [Ab16] Abadi, M. et al.: TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. CoRR abs/1603.04467/, 2016, arXiv: 1603.04467, URL: <http://arxiv.org/abs/1603.04467>.

- [Ab17] Aberger, C. R.; Lamb, A.; Olukotun, K.; Ré, C.: Mind the Gap: Bridging Multi-Domain Query Workloads with EmptyHeaded. PVLDB 10/12, S. 1849–1852, 2017, URL: <http://www.vldb.org/pvldb/vol10/p1849-aberger.pdf>.
- [AIS93] Agrawal, R.; Imielinski, T.; Swami, A. N.: Mining Association Rules between Sets of Items in Large Databases. In: ACM SIGMOD, Washington, DC, USA, May 26-28, 1993. S. 207–216, 1993, URL: <http://doi.acm.org/10.1145/170035.170072>.
- [BG16] Butterstein, D.; Grust, T.: Precision Performance Surgery for PostgreSQL: LLVM-based Expression Compilation, Just in Time. PVLDB 9/13, S. 1517–1520, 2016, URL: <http://www.vldb.org/pvldb/vol9/p1517-butterstein.pdf>.
- [BG17] Butterstein, D.; Grust, T.: Invest Once, Save a Million Times - LLVM-based Expression Compilation in PostgreSQL. In: BTW (DBIS), 6.-10. März 2017, Stuttgart, Germany, Proceedings. S. 623–624, 2017, URL: <https://dl.gi.de/20.500.12116/672>.
- [Bi12] Binnig, C.; Rehrmann, R.; Faerber, F.; Riewe, R.: FunSQL: it is time to make SQL functional. In: Proceedings of the 2012 Joint EDBT/ICDT Workshops, Berlin, Germany, March 30, 2012. S. 41–46, 2012, URL: <https://doi.org/10.1145/2320765.2320786>.
- [BMM13] Binnig, C.; May, N.; Mindnich, T.: SQLScript: Efficiently Analyzing Big Enterprise Data in SAP HANA. In: BTW (DBIS), 11.-15.3.2013 in Magdeburg, Germany. Proceedings. S. 363–382, 2013, URL: <https://dl.gi.de/20.500.12116/17332>.
- [BP98] Brin, S.; Page, L.: The Anatomy of a Large-Scale Hypertextual Web Search Engine. Computer Networks 30/1-7, S. 107–117, 1998, URL: [https://doi.org/10.1016/S0169-7552\(98\)00110-X](https://doi.org/10.1016/S0169-7552(98)00110-X).
- [Cr15] Crotty, A.; Galakatos, A.; Dursun, K.; Kraska, T.; Binnig, C.; Çetintemel, U.; Zdonik, S.: An Architecture for Compiling UDF-centric Workflows. PVLDB 8/12, S. 1466–1477, 2015, URL: <http://www.vldb.org/pvldb/vol8/p1466-crotty.pdf>.
- [Ei04] Eisenberg, A.; Melton, J.; Kulkarni, K.; Michels, J.-E.; Zemke, F.: SQL:2003 Has Been Published. SIGMOD Conference 2004 33/1, S. 119–126, März 2004, ISSN: 0163-5808, URL: <http://doi.acm.org/10.1145/974121.974142>.
- [Gi13] Giorgidze, G.; Grust, T.; Ulrich, A.; Weijers, J.: Algebraic data types for language-integrated queries. In: DDFP 2013, Rome, Italy, January 22, 2013. S. 5–10, 2013, URL: <https://doi.org/10.1145/2429376.2429379>.
- [GSU13] Grust, T.; Schweinsberg, N.; Ulrich, A.: Functions Are Data Too (Defunctionalization for PL/SQL). PVLDB 6/12, S. 1214–1217, 2013, URL: <http://www.vldb.org/pvldb/vol6/p1214-grust.pdf>.
- [GU13] Grust, T.; Ulrich, A.: First-Class Functions for First-Order Database Engines. In: (DBPL 2013), August 30, 2013, Riva del Garda, Trento, Italy. 2013, URL: <http://arxiv.org/abs/1308.0158>.
- [Hu17] Hubig, N.; Passing, L.; Schüle, M. E.; Vorona, D.; Kemper, A.; Neumann, T.: HyPerInsight: Data Exploration Deep Inside HyPer. In: CIKM 2017, Singapore, November 06–10, 2017. S. 2467–2470, 2017, URL: <https://doi.org/10.1145/3132847.3133167>.
- [KN11] Kemper, A.; Neumann, T.: HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots. In: ICDE 2011, April 11-16, 2011, Hannover, Germany. S. 195–206, 2011, URL: <https://doi.org/10.1109/ICDE.2011.5767867>.
- [Le15] Leis, V.; Kundhikanjana, K.; Kemper, A.; Neumann, T.: Efficient Processing of Window Functions in Analytical SQL Queries. PVLDB 8/10, S. 1058–1069, 2015, URL: <http://www.vldb.org/pvldb/vol8/p1058-leis.pdf>.

- [Ll82] Lloyd, S. P.: Least squares quantization in PCM. *IEEE Trans. Information Theory* 28/2, S. 129–136, 1982, URL: <https://doi.org/10.1109/TIT.1982.1056489>.
- [Lo08] Loney, K.: *Oracle Database 11g The Complete Reference*. McGraw-Hill, Inc., 2008.
- [Ne11] Neumann, T.: Efficiently Compiling Efficient Query Plans for Modern Hardware. *PVLDB* 4/9, S. 539–550, 2011, URL: <http://www.vldb.org/pvldb/vol14/p539-neumann.pdf>.
- [Pa17] Passing, L.; Then, M.; Hubig, N.; Lang, H.; Schreier, M.; Günemann, S.; Kemper, A.; Neumann, T.: SQL- and Operator-centric Data Analytics in Relational Main-Memory Databases. In: *EDBT 2017, Venice, Italy, March 21–24, 2017*. S. 84–95, 2017.
- [Sc10] Schreiber, T.; Bonetti, S.; Grust, T.; Mayr, M.; Rittinger, J.: Thirteen New Players in the Team: A Ferry-based LINQ to SQL Provider. *PVLDB* 3/2, S. 1549–1552, 2010, URL: <http://www.comp.nus.edu.sg/%5C%7Evldb2010/proceedings/files/papers/D09.pdf>.
- [Sc17] Schubert, E.; Sander, J.; Ester, M.; Kriegel, H.; Xu, X.: DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN. *ACM Trans. Database Syst.* 42/3, 2017, URL: <http://doi.acm.org/10.1145/3068335>.
- [Th15] Then, M.; Passing, L.; Hubig, N.; Günemann, S.; Kemper, A.; Neumann, T.: Effiziente Integration von Data- und Graph-Mining-Algorithmen in relationale Datenbanksysteme. In: *Proceedings of the LWA 2015 Workshops: KDML, FGWM, IR, and FGDB, Trier, Germany, October 7–9, 2015*. S. 45–49, 2015.

On-the-fly Reconfiguration of Query Plans for Stateful Stream Processing Engines

Adrian Bartnik¹, Bonaventura Del Monte², Tilmann Rabl³, Volker Markl⁴

Abstract: Stream Processing Engines (SPEs) must tolerate the dynamic nature of unbounded data streams and provide means to quickly adapt to fluctuations in the data rate. Many major SPEs however provide very little functionality to adjust the execution of a potentially infinite streaming query at runtime. Each modification requires a complete query restart, which involves an expensive redistribution of the state of a query and may require external systems in order to guarantee correct processing semantics. This results in significant downtime, which increase the operational cost of those SPEs. We present a modification protocol that enables modifying specific operators as well as the data flow of a running query while ensuring exactly-once processing semantics. We provide an implementation for Apache Flink, which enables stateful operator migration across machines, the introduction of new operators into a running query, and changes to a specific operator based on external triggers. Our results on two benchmarks show that migrating operators for queries with small state is as fast as using the savepoint mechanism of Flink. Migrating operators in the presence of large state even outperforms the savepoint mechanism by a factor of more than 2.3. Introducing and replacing operators at runtime is performed in less than 10 s. Our modification protocol demonstrates the general feasibility of runtime modifications and opens the door for many other modification use cases, such as online algorithm tweaking and up- or downscaling operator instances.

Keywords: Data Stream Processing, Resource Elasticity, Query Plan Maintenance, Fault Tolerance

1 Introduction

Stream processing engines (SPEs) have become an essential component of many business use cases and need to reliably ensure correct processing of the incoming, high-speed workload. However, the characteristics of a streaming workload may vary over time, since it includes foreseeable and predictable changes within a time frame (e.g., the day and night usage patterns for social media posts) but also bursty spikes in the case of irregular events such as sport or weather events. The ability of an SPE to adapt to workload changes at runtime is one aspect of *elasticity*. Currently, many SPEs (e.g., Apache Flink [Ca17], Apache Storm [To14], and Apache Spark [Za13]) provide limited functionality to support changes in their running configuration, if any at all. These systems allow modifications of a running streaming query only by restarting its execution with a new configuration. However,

¹ Technische Universität Berlin, bartnik@campus.tu-berlin.de

² DFKI GmbH, bonaventura.delmonte@dfki.de

³ Technische Universität Berlin - DFKI GmbH, rabl@tu-berlin.de, tilmann.rabl@dfki.de

⁴ Technische Universität Berlin - DFKI GmbH, volker.markl@tu-berlin.de, volker.markl@dfki.de

this may violate SLAs when other, critical systems rely on the output of the SPEs and cannot tolerate any downtime. As a result, end-users of those SPEs face potentially ever-running streaming queries that they cannot alter at runtime. Motivated by this technical challenge, our goal is to enable modifications of a running query at runtime, without stopping the SPE. We analyze diverse modifications and we derive a set of generic protocols that alter the physical plan of a running query. Our protocols enable introducing, tuning, and removing operators in a running query. This opens the door for a new class of online optimizations. Through our protocols, the SPE can tune the behavior of an operator, e.g., increasing its number of network buffers to sustain higher incoming load. Our protocols also enables the SPEs to migrate operators among different nodes, e.g., when a node is scheduled for maintenance. This is particularly challenging for jobs with large state, which have long recovery times. Finally, our protocols potentially allow in- or decreasing the degree of parallelism of an operator, which is beneficial for the system to adapt to changes of the incoming workload. Our contribution are as follows:

- Migration of stateless and stateful stream operators at runtime
- Introduction of new operators into a running streaming query
- Changing user defined functions of operators in running query
- Implementation of our protocols in Apache Flink
- Evaluation of our solution on Nexmark and a custom benchmark

This paper is structured as follows: we first provide background concepts in Section 2 and then we describe the design of our protocols in Section 3. After that, we present the architecture of an SPE integrating our protocols in Section 4 and its new features in Section 5. In Section 6, we show the experimental evaluation of our system. In Section 7, we conclude by summarizing our contributions and providing insight about future works that our current work enables.

2 Background

In the following, we present the background concepts that lay the foundation to our work. In particular, we provide an overview of data stream processing and a description of Apache Flink with its checkpoint mechanism, which we use as building block for our techniques.

2.1 Data Stream Processing

Data stream processing enables continuous analysis of real-time, unbounded data. Although SPEs are part of the data processing stack for more than a decade [ScZ05], only recently the need for real-time big data processing has fostered the development of a new generation of SPEs. Those new SPEs target massively parallel cloud infrastructures by extending the batch-oriented MapReduce paradigm [DG04]. The new engines improve on MapReduce as they consider low latency an important constraint. New SPEs adopt one of two processing models: micro-batching and tuple-at-a-time processing.

The micro-batching model discretizes the processing of an unbounded stream in a series of

finite chunks of data. While this approach ensures higher throughput, the size of each chunk drastically impacts the latency of running queries [Za13, Ku15]. The tuple-at-a-time model allows for a more fine-grained processing of incoming records, thus achieving lower latency. This model still uses a batching mechanism at the physical level to ensure high throughput, i.e., operators pack tuples into buffers [Ca17].

Regardless of the processing model, a natural way of modeling data flows in SPEs is by means of a *Directed Acyclic Graph* (DAG), which contains source, processing, and sink nodes. *Source nodes* continuously emit a stream of data elements, which are also commonly referred to as *tuples* or *records*. A stream is a potentially unbounded sequence of tuples generated continuously over time. *Processing nodes* consume, process, and emit new data elements. A *Sink* consumes but does not forward any new elements, e.g., it writes its output to disk. Source, processing, and sink nodes are connected through edges, which represent the communication channels for the data exchange among operators.

Processing nodes are either *stateful* or *state-less*. For state-less nodes, the output only depends on each incoming data element. In contrast, the output for stateful nodes depends on the incoming data elements as well as on some internally managed state.

2.2 Apache Flink

Apache Flink is an open-source dataflow processing framework for batch and stream data [A114]. Flink uses the parallelization contract (PACT) programming model, a generalization of the MapReduce programming model, and second order functions to perform concurrent computations on distributed collections in parallel [Hü15]. Flink compiles submitted queries into DAGs that it optimizes and executes on a cluster of nodes. Flink relies on pipelining and buffering to avoid the materialization of intermediate results. Stream operators in Flink exchange intermediate results via buffers, i.e., an operator sends its buffer downstream when it is full or after a timeout. This enables processing data with high throughput and low latency. Furthermore, Flink provides operator fusion [Hi14]. This ensures that fused operators exchange tuples in a push-based fashion, whereas not-fused operators exchange buffers in a pull-based fashion. Back-pressure occurs in Flink when an operator receives more data than it can actually handle. Back-pressure is usually due to a temporary spike in the incoming workload, garbage collection stalls, or network latency.

2.3 Fault Tolerance and Checkpointing in Apache Flink

Flink's *Checkpointing Mechanism* consistently stores and recovers the state of a streaming query [Ca17]. The mechanism ensures fault tolerance in the presence of failures, meaning upon recovery, the program's state eventually reflects the stream state from before the failure. The checkpointing settings enable specifying message delivery guarantees, which ensure that every record from the data stream is processed exactly-once, at-least-once, or at-most-once. This mechanism lays the foundation for the *Savepoint Mechanism*, which enables deliberately stopping and resuming a running query. When restarting from a savepoint, Flink allows for updating aspects of the query itself, e.g., adding or removing

operators as well as adjusting their degree of parallelism. The checkpointing mechanism periodically triggers *checkpoints* of the streaming query, which act as independent snapshots to which the system can fall back in case of a failure. For queries with small states, the checkpoints have only negligible impact on the overall system performance. For queries with large state, checkpointing may drastically affect the query performance, mainly because of the time needed to copy the state on a persistent storage. In case of a program failure, Flink stops and restarts the streaming query by resetting the operator state to the one captured in the last successful checkpoint. The checkpointing mechanism requires a persistent stream source (e.g., Apache Kafka) that allows for rewinding the stream to a specific point in time and resend all subsequent messages for strict processing semantics. Non-replayable stream sources instead loose intermediate records upon query restart.

3 Protocol Description

This section describes the main features of the migration protocol that enables runtime modifications on a running streaming query. These are migrating stateful operator instances across nodes, introducing new operators into a running query, and replacing user-defined functions (UDFs) of operators.

3.1 System Model

Before we discuss the design of the migration protocols, we provide a description of our system model. We assume that our SPE ingests an infinite set of tuples r_1, r_2, \dots . A tuple $r_j = (A, t)$ consists of a set of attributes A and a timestamp t . Each operator in our system processes a tuple at a time through a UDF $f(r_j, S)$, where r_j is the input tuple and S is the current state of the operator. According to the semantic of its UDF, the operator emits zero or more output tuples upon processing an input tuple. Each operator runs with its own degree of parallelism. A logical query plan consists of a set of source, sink, and processing operators, which the SPE models as nodes of a DAG. A physical query plan contains all the parallel instances of all the operator. The SPE also represents this as a DAG. We refer to a job as a running physical plan in the SPE. The edges of a DAG act as communication channels between those machines and represent the data exchange among operators, which follows three patterns. A parallel instance of an operator can 1. forward a record to a single parallel instance of a downstream operator, 2. broadcast a record to all parallel instances of a downstream operator, and 3. send a record downstream based on some partitioning function. To support checkpointing, the SPE injects special markers m_i through the sources into the DAG, periodically or upon user request. Marker m_i triggers the i -th checkpoint for a job. Each parallel instance of an operator receives a marker on all its inbound communication channels and reacts by snapshotting its state and forwarding the marker downstream. As soon as all parallel instances successfully complete their snapshot, they asynchronously send the checkpointed state to persistent storage (e.g., a distributed file system). As soon as those asynchronous copies finish, the SPE marks the checkpoint as completed. Checkpoint markers logically divide the processing of an input stream in finite sets of tuples. The

SPEs guarantees exactly-once processing of each tuple in every set. The SPE consists of a coordinator process and several workers processes, which run on a cluster of nodes. Each worker process has a set of execution slots that determines the number of parallel instances of an operator it can run. In the rest of this section, we provide a thorough description of our proposed protocols based on the system model presented above.

3.2 Migration Protocol Description

This first protocol enables the migration of operator instances across machines, e.g., when detecting an upcoming hardware failure on a node in the cluster. Upon a migration request, the SPE retrieves all operator instances on the faulty node and allocates resources for these instances on others nodes. The SPE starts a migration by creating a special *modification marker* that contains all necessary information to perform the migration and ingests it into the DAG at the source operators (following the ideas of Del Monte [DM17]). Each operator instance eventually receives these markers and decides whether it needs to react on that migration marker. The key concept here is that a migration does not only affect the actual migrating instances but also the up- and down-stream instances. The upstream instances temporarily buffer their records during the migration duration as well as guarantee processing semantics and maintain FIFO order. The migrating instances store their state in persistent storage in order to consistently resume processing tuples once restarted on a different worker. The downstream instances rewire their inbound communication channels to correctly consume records from the migrated instances.

3.3 Modification Protocols Description

In addition to migrating operator instances, our protocol also enables the modifications of the data flow, e.g., the introduction of new operator instances or replacing the operator function in a running job. Both operations require the distribution of the UDFs in the cluster at runtime. The SPE starts each modification similarly to a migration by ingesting the modification marker at the source operators.

3.3.1 Introduction of New Operators

Upon receiving the modification marker, each operator checks how it needs to react. The upstream operators have to start buffering their outgoing records. Then, they broadcast the upcoming location of the newly introduced operator to all downstream operators. As soon as all upstream operators have successfully acknowledged buffering, all new operator instances will be started. Before the actual modification starts, the SPE distributes the UDFs compiled code to the target nodes such that the operators can immediately start instantiating those UDFs and processing records. The downstream operators instead attempt to connect to the machine of the newly instantiated operator instances or fail otherwise.

3.3.2 Changing the Operator Function

The last modification operation enables replacing the UDF of an operator at runtime. Similar to introducing an operator, the user first needs to provide the SPEs with the new UDFs compiled code for the operator to update. The SPE distributes the compiled code to all nodes on which the target operator runs. The SPE then introduces a special modification marker at the DAG sources that sets up all operator instances for the modification. Upon receiving the checkpoint marker for that modification, each instance of the target operator instantiates the new UDF and continues processing records.

4 System Architecture

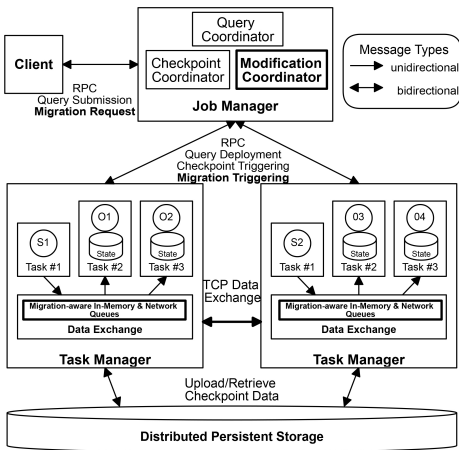


Fig. 1: System Architecture. Our contributions are marked in bold.

The client sends modification control messages to the coordinator (i.e., the Job Manager), which interacts with the Flink cluster. It delegates and distributes the actions between all workers (i.e., the Task Manager), that run the actual operator instances. These control messages are not part of the actual data flow and are realized as Remote Procedure Call (RPC) via the actor model [HBS73]. The actor model represents a system, in which all entities, namely actors, run concurrently and solely communicate via message passing. The client, the Job and Task Managers use the actor system to concurrently communicate via asynchronous messages. Apache Flink handles the data exchange among parallel instances of two operators in a produce-consumer relation through an internal protocol built on top of TCP. All operator instances on a Task Manager share some of their resources, such as TCP connections via multiplexing as well as common data sets and data structures. Each instance is responsible for establishing a connection to its upstream instances in order to retrieve the input tuples. Flink leverages either the Job Manager or a third-party storage system to persistently store the checkpoint data. In Figure 1, we present a conceptual view of the resulting system architecture.

This section deals with our system architecture that support the migration and modification protocol from the previous section. We explain the components that drive and supervise the modifications as well as the integration with the checkpointing mechanism of Apache Flink (Version 1.3.2). We first provide an overview of the existing components and then we illustrate our changes on the coordinator and worker sides.

4.1 Vanilla Components Overview

The client sends modification control messages to the coordinator (i.e., the Job Manager), which interacts with the Flink cluster. It delegates and distributes the actions between all workers (i.e., the Task Manager), that run the actual operator instances. These control messages are not part of the actual data flow and are realized as Remote Procedure Call (RPC) via the actor model [HBS73].

4.2 Our Changes on the Coordinator Side

Besides controlling the health of every Task Manager and monitoring queries execution, the Job Manager is responsible for the checkpoint mechanism. The existing *Checkpoint Coordinator* handles all functionality related to the checkpointing mechanism, i.e., it triggers checkpoints, supervises the lifecycle of a checkpoint, and may restart jobs based on past savepoints. Similarly, our newly introduced *Modification Coordinator* handles processing and validating modification commands as well as triggering and supervising the execution of these modifications. The validation includes whether the requested modification is applicable to the current, running job and some modification-specific checks. If successful, the Modification Coordinator prepares a modification-specific trigger messages, which it introduces in the data flow at source vertices of the target job. The coordinator keeps track of all vertices contained in the DAG and their modification life-cycle. Should an error occur during a modification, the coordinator aborts the current modification and notifies involved operators about its cancellation. Each operator may react to a certain trigger message or choose to ignore it, yet in any case it will acknowledge the reception to the Modification Coordinator. Depending on whether all vertices successfully acknowledge the trigger message as well as all subsequent, modification-specific state changes, the coordinator eventually marks a modification as *completed* or as *failed*. Normally, a task starts in the *Created*-state, gets *Scheduled* and *Deployed* by the Job Manager, processes all its input in the *Running* and eventually finishes by entering the *Finished* state. A task may fail for various reasons and subsequently enters the *Failed* state. Finally, if the SPE cancels a task due to an external reason, it enters the *Canceling* and *Canceled* state consecutively. We introduce two additional states, i.e., *Pausing* and *Paused*, which the task enters when the SPEs triggers a migration. In case of stateful operators, the tasks checkpoint and submit their state to a distributed persistent storage system.

4.3 Our Changes on the Worker Side

Each Task Manager executes *tasks*, which provide the environment for running the concrete operator instances. A task is mainly responsible for establishing the connection between Task Managers according to the up- and downstream operators, deserializing incoming records, processing them, serializing output records, and placing them in outgoing queues. The Job Manager determines the location of the up- and downstream operators in the job's initialization phase. The type of connections between operator instances depends on their relative location. In the case both producing and consuming instances are located on the same Task Manager, two instances transfer records through an in-memory queue. In the case they run on different Task Managers, the producer sends its output records via the network. When performing an operator migration, it is necessary to buffer intermediate records as long as the new consuming operator is not yet ready to receive records. Therefore, we implement a dedicated queue that is able to retain buffers while not blocking the upstream producers. It holds the buffers as long as possible in memory but it can also spill these buffers to disk. The spilling phase is asynchronous, i.e., writing to disk will not block. As

soon as a new consuming operator starts, it first reads all spilled buffers and only afterwards fetches newly-arrived buffers from memory to preserve FIFO semantic.

4.4 Query Plan Modifications

The protocol must at all times guarantee data integrity by not loosing any records or events and maintain the processing semantics. For each single operator instance to modify, we need to also consider its up- and downstream operators, since they should continue to process records. Therefore, instead of handling each operator individually, the modification message contains complete instructions for all operators. The logic for preparing this modification message is located in the Modification Coordinator, each operator simply follows the instructions contained in the modification message. We trigger the modification through RPCs to the source vertices, whereas we rely on the dataflow channels to propagate the modification message to remaining vertices with the same speed of records. Additionally, by relying on Flink data flow mechanism, all messages maintain FIFO semantics and are processed exactly once.

4.4.1 Upstream Operators

The checkpointing mechanism guarantees fault tolerance by aligning the checkpoint markers for each incoming data stream in each operator. Each operator instance receives records and events from the upstream operator. It also buffers pending records during the alignment phase of a checkpoint. This phase occurs when an instance has not received all the markers for its upstream instances and thus needs to buffer the incoming records for the blocked channels. Therefore, in case an operator wants to migrate to a different Task Manager, the SPE needs to migrate those buffers as well, increasing the state size. Because these buffers are not part of the internally-managed state, we need to handle them separately. To this end, the modification mechanism ensures that all upstream operators perform a custom alignment procedure on the sending side. Modification messages contain an upcoming checkpoint id. When the checkpoint with that id is triggered, the upstream operators broadcast checkpoint barriers along with a modification acknowledge message to the migrating downstream operators. This event signals to consumer operators that the upstream operator is spilling all further records and events to disk. Hence, it is safe to expect no new buffers from this operator instance. Synchronizing on a checkpoint on the sending side guarantees that no data are currently in-flight between the operators and the target operator has no buffered data. Therefore, it is safe to take further actions, such as pausing the operator for a migration. However, this also means that the modification has to wait until that specific checkpoint is acknowledged at the producing operators. The checkpointing interval determines how often a checkpoint is triggered by the Checkpoint Coordinator. The duration of taking a specific checkpoint depends on many factors, but most importantly on the number of operators in the job and the actual state size. Therefore, for jobs with small state and a low checkpointing interval, the SPE completes a modification trigger message in a matter of seconds. For jobs with large state and thus long checkpointing interval, the waiting time for the checkpoint to finish may be significantly longer than the duration of the actual modification itself.

4.4.2 Target and Downstream Operators

If an operator reacts to a trigger message, it will transition into a new state according to the specific modification and wait for further actions or events. Instances of downstream operators generally only wait for the target operators to react. In case the location of the target operator changes, such as during a migration, downstream instances receive the new location in the last record and then update the connection to that new upstream instance accordingly.

5 Protocol Implementation

The following section describes each modification operation in detail. These are migrating operator instances across Task Managers, introducing new operators into a running job and replacing user-defined functions of operators at runtime.

5.1 Operator Migration

In Figure 2, we provide an overview of the four steps involved in a migration. Upon a migration request, the Modification Coordinator retrieves all operators on that specific Task Manager and allocates new execution slots. It checks if enough resources are available, but also assigns these new slots to each migrating operator. This step is essential as each slot determines the location on which the operator instances will restart after state submission. The Modification Coordinator creates the trigger message that contains instructions for the involved operators and ingests it into the data flow through the source operators. This message determines the operators to migrate as well as the upstream operators that react by spilling their records to disk. Involved operators wait for the upcoming checkpoint marker to trigger the actual migration. When the Checkpoint Coordinator introduces that marker in Step 2, all upstream operators stop sending records to the migrating instances. Every operator starts the migration sequence by collecting and transmitting its current state to the Modification Coordinator as well as transitioning into the *Pausing* state. Additionally, it sends its upcoming new location to all downstream operators, as shown in Step 3. Finally, the task initiates the release of all allocated resources and enter the *Paused* state. As soon as the Modification Coordinator receives the confirmation of a successful transition to the *Paused* state, it restarts the operator. In Step 4, the Modification Coordinator attaches the state location and starts the execution in the previously assigned task slot. In addition, it will compute the new in- and outputs of the instance of the migrating operator based on the updated location of all other migrating operators instances. We rely on the checkpoint mechanism of Flink to collect and reassign the state of every operator.

5.2 Introduction of new Operators

The modification protocol also enables the modification of the data flow itself, in particular, the introduction of new operators into a running query. The requirement for this modification

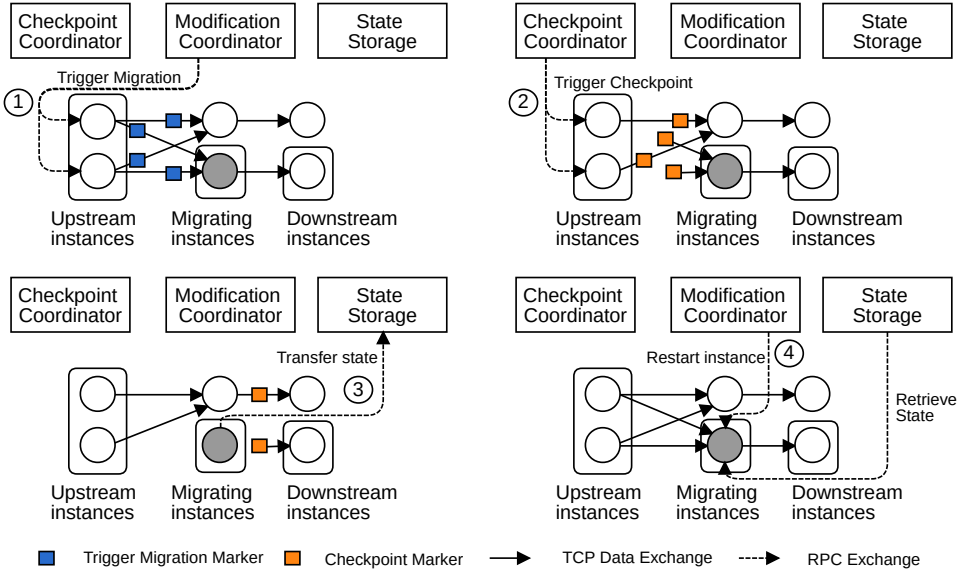


Fig. 2: Overview of modification protocol procedure

is to provide the SPE with compiled code for the new operators. Apart from that, introducing operators works similarly to migrating operator instances. The set of all upstream operators is formed by all operator instances directly before the new operator. Similarly, the set of downstream operators consists of all operators after the operator that should be inserted. As soon as all upstream operators successfully acknowledge the spilling phase, the Modification Coordinator deploys the new operator instances. The deployment payload also contains the compiled code for the new operators. Prior to a modification, our protocol enforces that new operators do not violate any type constraints of the up- and downstream operators.

5.3 Changing the Operator Function

This modification operation enables replacing the UDF of an operator at runtime. Similar to introducing an operator at runtime, the user needs to provide the new code for the target UDFs. The Modification Coordinator then introduces the modification message in the dataflow containing the checkpoint id and the reference for the new UDF. Through this reference, each Task Manager involved in the modification asynchronously fetches the actual new code once. Afterwards, each operator instantiates the new UDF and registers a callback. When the SPE completes the checkpoint with the ID specified in the modification message, every Task Manager triggers the callback locally. The purpose of the callback is to replace the operator function with the new one only when the SPE completes a global checkpoint. The rationale behind this choice deals with guaranteeing exactly-once processing semantic. We need to ensure that we consistently process each record with the new UDFs only after a specific point in time.

6 Evaluation

In the following, we present the empirical evaluation of our proposed modification protocols. We use the Nexmark and a custom benchmark to evaluate our protocols. We present the performance metrics for the operator migration, introduction a new operator; as well as the replacement of UDFs. Finally, we discuss the results and findings of our work.

6.1 Data Generator

In order to minimize the dependencies and not benchmark external systems, we implement a custom data generation and a message queuing system based on the work of Karimov et al. [Ka18]. We prefer an external generator over consuming from an internal Flink source because this may lead to unrealistic throughput and latency metrics as the source's record generation affects the overall system performance. In case of back-pressure, for instance, Flink is not able to produce new data because of the unavailability of network buffers. We use our data generator in all the following benchmarks to create a constant, non-fluctuating workload.

6.2 Cluster Environment and Benchmark Setup

We run all the experiments on an 8-nodes cluster, each with 48 cores and 48 GB of memory. Out of the 8 machines, one is dedicated to the Flink Job Manager, five run the Task Managers and the remaining two machines host three data generator instances each. This ensures that the generator instances do not interfere with neither the Job Manager nor the Task Managers. Our benchmarks have a warm-up phase of 10 minutes, after which each modification is triggered. For all benchmarks without windowing, we calculate the latency by subtracting the record's creation timestamp from its arrival time at the sink. If not mentioned otherwise, the filesystem state backend is used for the checkpointing mechanism.

6.3 Workloads

This section introduces the workloads with which we have evaluated the migration protocol.

6.3.1 Stateful Map Query

The first benchmark is a stateful map query (SMQ) with a source, a stateful map, and a sink operator. The sources read monotonically-increasing numbers along with a creation timestamp. Each map operator counts the number of elements this particular instance has received so far and appends that number to the output tuple. Finally, the sink computes the overall latency for each element by subtracting the event's timestamp from the current timestamp and writes everything to disk. This benchmark is used to demonstrate the correctness of the migration mechanism with small state size. In addition, it demonstrates the mechanism with many operators of varying degree of parallelism. The source operator, the map operator, and the sink operator have 60, 80, and 70 parallel instances, respectively.

6.3.2 Nexmark Benchmark

The modification protocol is suited for streaming queries with large state, which need longer to write and recover their state in case of failures. For this purpose, we select Query 8 (NBQ8) of the Nexmark Benchmark Suite [Tu18]. This benchmark represents a three-entity online auction system and is designed to measure the performance of various aspects of a streaming system. The three entities are the stream sources *persons*, *auctions*, and *bids*. The *person* source emits a registration event, every time a new user registers in the auction system. The *auction* source emits events for each newly created auction by a specific person. Finally, persons may bid on auctions, which creates a *bid* event emitted by the respective source. Query 8 finds those persons who created a new auction within a certain time frame after signing up at the auction service. We implement this query through a windowed join using a tumbling window of 20 minutes in event time. Since this benchmark generates a substantially larger state than the previous benchmarks, it is not feasible to write the state to disk every single time. For job with large state, writing gigabytes-sized state to disk simply takes too much time. Therefore, we leverage RocksDB³, i.e., the Flink-embedded key-value store, which offers *incremental checkpointing*. We set the checkpointing interval to 30 s.

6.4 Migration Protocol Benchmark

This section demonstrates the migration operation on the SMQ and NBQ8. We compared our migration technique against the Flink mechanism of canceling and restoring a job with a savepoint. As Flink has no special operation for simultaneously taking the savepoint and canceling the job, those operations can only occur subsequently. Therefore, Flink waits until all operators have successfully acknowledged the savepoint and then cancels the query. While taking a savepoint, the job continues to consume records, which means that all in-flight records are not actually part of the savepoint and are therefore lost upon restoring the job.

6.4.1 Stateful Map Job Performance Drill Down

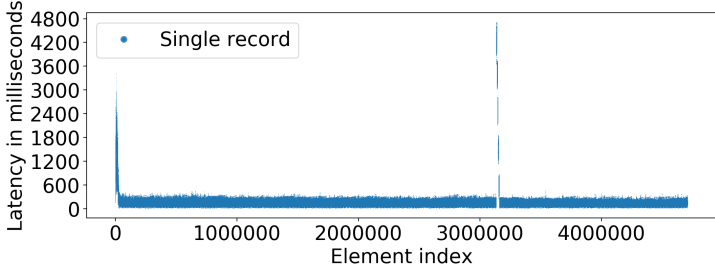
Savepoint. Figure 3a shows the duration for each event, when canceling and restoring the job from a savepoint. Initially, most of the time is spent on establishing the connection to submit the cancellation command to the job, whereas actually canceling takes only about 300 ms. The system stores savepoint (100 KB) on a shared file system. Restoring and scheduling the job takes roughly 3 s. This adds up to a total duration of around 7.5 s. The latency spike in Figure 3b represents the time in which the streaming job restarts and corresponds to the actual restarting duration of about 4.5 s.

Migration. Figure 4a shows the duration for each step when migrating all operator instances from one specific Task Manager. The initialization for sending the `TriggerMigration` messages takes a bit less than 3 s. The next 2 s are spent waiting for the upcoming checkpoint barriers to arrive at the job sources, upon which they will spill to disk and broadcast their

³ <http://rocksdb.org/>



(a) Event timeline for canceling and restoring the stateful map job via a savepoint



(b) Individual records latency

Fig. 3: Benchmark results for canceling and restoring the Stateful Map Job via a savepoint

new location. It takes almost another 1 s until the map operators are able to enter their migration. This happens once they receive either the spilling to disk events or the new location messages from all upstream sources. The sink migrations are fast and take less than 0.3 s. The long duration of almost 1.5 s might be explained by communication overhead introduced by the high degree of parallelism. In total, our mechanism migrates 34 operator instances, while 112 operator instances spill to disk during the migration. The migrating instances are 10 source, 12 map and 12 sink operators with a total migrated state size of 17 kB. We see the latency spike at the time of the migration of about 3500 ms, as shown in Figure 4b. This is because the actual duration of the migration without waiting for the checkpoint barriers takes about 3 s plus some additional delay for reconnecting to the operators. The migrated operators continue to process records after the migration, whereas the old operator instances stop their execution.

6.4.2 Nexmark Benchmark Performance Drill Down

Savepoint. Figure 5a shows the complete duration for canceling and restoring the job via a savepoint. The system needs about 161 s from submitting the savepoint command until the state has actually been stored in the state backend. We show this in Figure 5b as there is no decrease in sent records due to the fact that the job sources still consume data. The state size after running the Benchmark for about 12 min is about 13.4 GB. The complex setup is the cause of the long cancellation duration.

Migration. Figure 6a shows the complete duration for migrating all operators from one specific Task Manager. The system spends almost 1 min waiting for the upcoming checkpoint barrier to arrive. This is due to the 30 s checkpointing interval of which at least one full cycles needs to pass. Since two times the checkpointing interval is also the maximum time

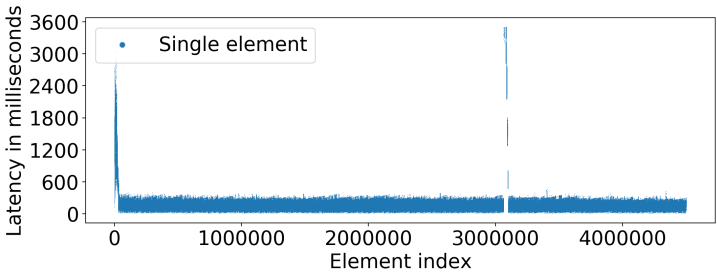
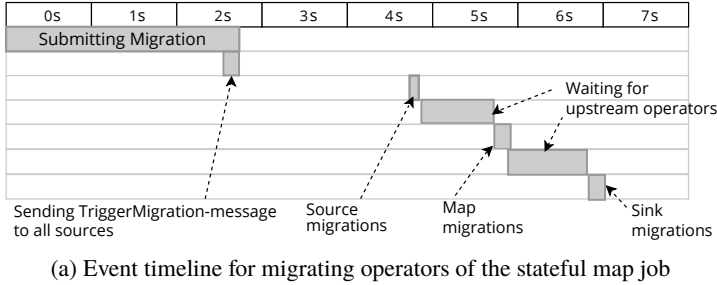


Fig. 4: Benchmark results for performing a migration on the Stateful Map Job

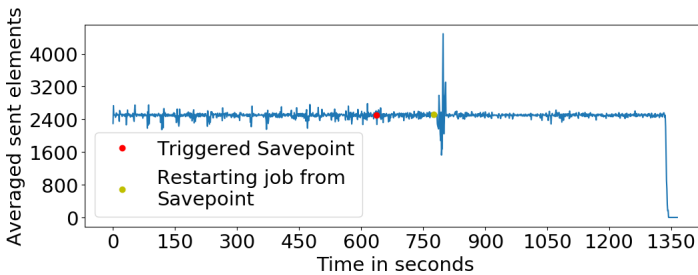
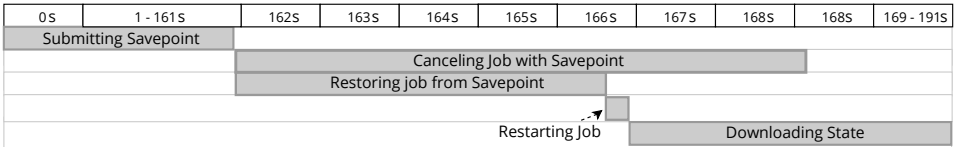
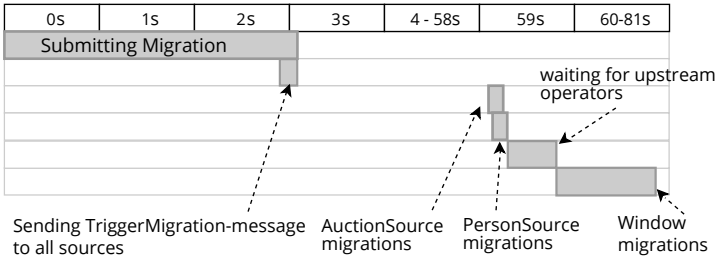


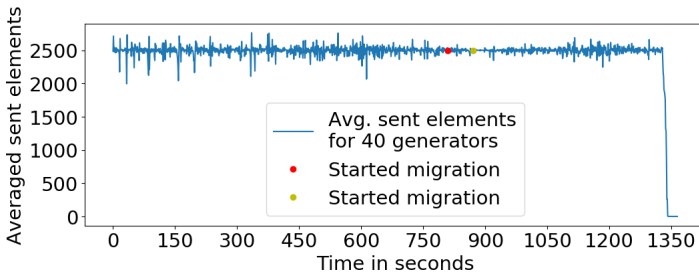
Fig. 5: Benchmark results for canceling and restoring NBQ8 via a savepoint

it takes until the modification mechanism triggers, this represents almost the worst case

waiting duration. The migration moves 24 operators in total across Task Managers, 8 person and 8 auction sources, as well as 8 window operators. The remaining 64 job sources are instead spilling to disk. The total state size for the job at the point of the migration is about 13.5 GB. However, only about 2.7 GB are actually stored and retrieved via the state backend for the window operators in about 21 s. Figure 6b shows that the throughput at the generators is not affected by the migration, since pausing and restarting the sources on different Task Manager takes less than 250 ms.



(a) Event timeline for migrating operators of NBQ8



(b) Average throughput at the 80 generators

Fig. 6: Benchmark results for performing a migration on NBQ8

6.5 Introducing new operators at runtime

To demonstrate the introduction of an operator at runtime, we use the SMQ (see Section 6.4.1). Its function filters the input stream by only letting every 5th record pass. We set the checkpointing interval to 1 s with concurrent checkpoints enabled. Figure 7 gives an overview of the modification duration. Most of the time for uploading the custom jar and submitting the command to Flink is spent establishing the connection. The actual preparation and sending of the trigger message takes less than 300 ms. It takes around 2 s for the corresponding checkpoint barrier to arrive, upon which all sources broadcast the new filter operator location. The source operators acknowledge the spilling to disk, while the map operators simultaneously update their inbound connections. As soon as all acknowledgments have been received, the new filter operator starts, which takes about 300 ms. In total, it takes only about 550 ms from the first spilling source to start and connect all 60 filter operators. The average throughput of the generators remains unaffected.

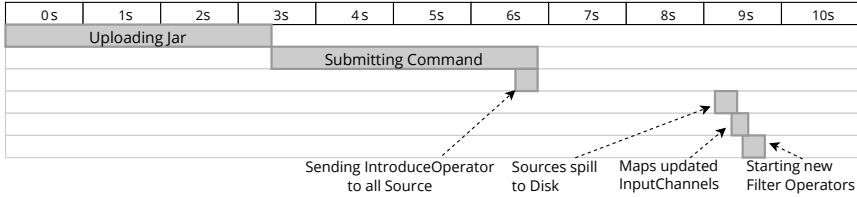


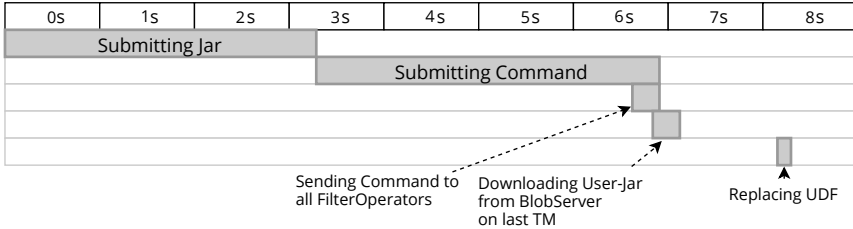
Fig. 7: Event timeline for introducing the filter function

6.6 Replacing the operator function at runtime

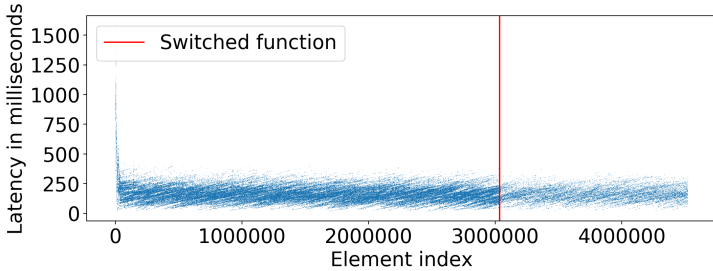
The last benchmark evaluates the replacement of a UDF of an operator at runtime. We reuse the SMQ with the filter operator already running. The source, filter, and map operators have 60 parallel instances, whereas the sink has 40. We set the checkpointing interval to 1 second with concurrent checkpoints enabled. Figure 8a shows the timeline of the introduction mechanism. Most of the time for uploading the new compiled code and submitting the command to Flink is spent on instantiating the client and establishing the connection. The actual preparation and sending of the trigger messages takes around 300 ms. Downloading the code on each Task Manager takes around 350 ms. Waiting for the upcoming checkpoint barrier, upon which all filter operators will replace their UDF with the one from the jar-file, takes around 1 s. In total, it takes a bit more than 8 s to replace the operator function. The synchronization on the checkpoint barriers may be omitted depending on the use case. Synchronizing only ensures that switching the function is aligned with the checkpoint mechanism, hence, it simplifies the recovery procedure. However, the synchronization is not necessary, e.g., all operators could immediately replace their function when receiving the trigger message. Replacing the operator function has nevertheless no impact on the throughput. Figure 8b shows the record latency for every 200th element in order to visualize the drop in incoming records after the function switch.

6.7 Discussion

The checkpointing settings of a given query have a big influence on the overall modification duration because of the eventual synchronization on every checkpoint barriers. For jobs with small state, fast checkpointing intervals are feasible, thus enabling the mechanism to react quickly on events, such as user input. For jobs with large state, the synchronization drastically limits the reaction time, however, the overall mechanism still performs better than the Flink baseline with savepoints. We are able to trigger all modification in less than 6 s for jobs with small state. For jobs with larger state, this duration increases up to about 60 s. This includes the heavy initialization overhead for the initial submission of the trigger as well as further messages. The experimental results show that it is feasible to migrate stateful operator instances at runtime with a negligible impact on the query performance. The operator state job demonstrated the migration of multiple, subsequent operators in 7.1 s. The modification duration grows with the number of subsequent operator instances, however, the impact is still low compared to the initial submission and the waiting duration. For the NBQ8, the migration mechanism showed big performance improvements, outperforming



(a) Event timeline for switching the filter operator



(b) Individual records latency

Fig. 8: Benchmark results for switching the filter function

the savepoint mechanism by a factor of 2.3, even including the long waiting and state transferring time. This is due to the long shutdown sequence of the savepoint mechanism, which transfers each operator state to the state backend. Table 1 summarizes the results of the migration operation. Additionally, Flink savepoint mechanism has the disadvantage of loosing in-flight records that flow from the sources during the savepoint procedure. When consuming from a persistent data source, such as Apache Kafka, we may still achieve exactly-once processing guarantees by replaying the missed records. However, the system loses records in the presence of non-persistent data source. Our migration mechanism prevents this from happening because stream sources never consume records that they cannot process. Furthermore, our mechanism does not introduce tangible throughput spikes at the generator because sources generally have very small state, which allows for quick migration.

Benchmark	Migrated State Size	Migrated Operators	Overall Duration	Waiting Time	Migration Duration
SMQ	17 kB	34	7.1 s	2 s	2.4 s
NBQ8	2.7 GB	24	81.8 s	54.9 s	23.7 s

Tab. 1: Migration overview for SMQ and NBQ8

Although introducing an operator into a running job suffers a few limitations, as mentioned in Section 5.2, it still enables many complex modification scenarios. Introducing operators enables altering the physical data flow of a DAG in under 10 s, depending on the checkpointing settings. With some further work, it would enable more sophisticated modification scenarios,

such as operator reordering. Replacing the operator function is a modification operation that enables fast tweaking of the streaming job in under 9 s with no performance impact. In the current version, users can leverage it to avoid restarting a running job, e.g., by adding additional logic to a UDF. It also enables more complex modification scenarios, such as tweaking the functionality of operator function based on internal or external triggers, such as runtime metrics.

7 Related work

Schneider et al. present an auto-parallelization mechanism for general-purpose, distributed data stream processing systems that deals with workload and resource aware scaling of operator instances as well as runtime state migration [Sc12]. We differentiate from their work as our migration mechanism is specific for queries that need exactly-once processing semantic as well large state. Wu et al. present ChronoStream, a distributed stream processing system specifically designed for elastic stateful stream computation [WT15]. ChronoStream achieves horizontal and vertical scalability in order to cope with workload fluctuation by treating internal state as a first-class citizen. The delta-compressed state of a task is separated into computational slices, which are duplicated and managed by a locality-aware placement mechanism across all machines. Through both mechanisms, they drastically lower the overhead caused by network I/O in case of migrations. Although their evaluation demonstrates linearly elasticity without sacrificing system performance or affecting colocated tenants, Ding et al. argue that their transactional migration protocol may cause incorrect results due to synchronization issues [Di15]. In contrast to our solution, they do not consider changes to the data flow and take placement decisions only based on the migration protocol, but not overall system performance. Heinze et al. [He14] deal with finding the right point in time to take scaling decisions through an online learning algorithm. They empirically compare their solution [He15] against local and global threshold-based mechanisms presented by Lorido et al. [LML14]. Although their results indicate that their auto-scaling technique performs better than the previous solutions, their solution only applies a simplistic migration protocol. In contrast, we design our protocols to cope with more demanding requirements, e.g., large state, exactly-once semantic. Nasir et al. [Na15b] propose the concept of *partial key grouping* in response to load imbalance caused by skewness in the key distribution of the input. Their approach monitors the number of tuples sent to two downstream instances. Each operator instance sends a tuple to the one with lower load estimation. As a result, tuples with the same key are routed to different parallel instances of the same operator. The authors extend their work to also include a mechanism for the “hottest” keys in the stream and assign more operator instances to those keys [Na15a]. In contrast to our approach, their solution does not alter the running query (i.e., no state migration) but only uses a streaming algorithm to determine the number of workers for “heavy hitter” keys, which is minimal yet sufficient for load balancing. Mai et al. introduce Chi [Ma18], a system that allows for dynamic reconfiguration of a running query. They use a migration mechanism similar to ours, however, they do not consider queries that result in large state and they do not further investigate such a specific scenario. Castro Fernandez

et al. also present a mechanism to scale up and down streaming topologies and to deal with operator failures [Ca13]. Our solution is different because of the explicit handling of large state and extended range of modifications, e.g., altering an execution plan at runtime. Systems based on micro-batching such as Apache Spark allow for reconfiguration at the end of each micro-batch, however, this results in higher latency and lower throughput because of synchronization. Finally, we point out that our approach is orthogonal to public and private cloud [Am] and resource managers [Hi11, Va13] because we tackle fault-tolerance and resource elasticity at application-level, whereas they isolate running applications through virtual machines or containers.

8 Conclusion

This paper examines the feasibility of modifying the execution of a running streaming job in Apache Flink. Our major contribution is the implementation of a modification protocol that enables three types of modifications, namely migrating operator instances, introducing new operator instances, as well as changing an operator's UDF. We evaluate the protocol using one custom benchmark and the Nexmark Benchmark. The results show that migrating operators for jobs with small state is as fast as using Flink's savepoint mechanism. Migrating operators of a job with 15 GB of state even outperforms the savepoint mechanism by a factor of 2.3. Furthermore, our migration mechanism solves the problem of data loss during job restart that arises when not consuming records from a persistent data source. Changing the data flow itself by either introducing new or replacing existing operators can be performed in less than 10 s and 8 s, respectively. The modification protocol opens the door for a variety of further enhancements to a running streaming job. It allows for online optimizations of the running job that were not possible before. An example is up- and down-scaling operator instances, which enables the SPE to dynamically adapt to changes in the incoming workload without halting the whole job. Lastly, removing the synchronization on the checkpoint barriers can significantly reduce the overall modification duration and increase the responsiveness of our mechanism.

Acknowledgment. This work was funded by the European Union through PROTEUS (ref. 687691) and STREAMLINE (ref. 688191).

References

- [Al14] Alexandrov, A.; Bergmann, R.; Ewen, S.; Freytag, .; Hueske, F.; Heise, A.; Kao, O.; Leich, M.; Leser, U.; Markl, V. et al.: The Stratosphere Platform for Big Data Analytics. The VLDB Journal, 2014.
- [Am] Amazon EC2. <https://aws.amazon.com/ec2/>.
- [Ca13] Castro Fernandez, R.; Migliavacca, M.; Kalyvianaki, E.; Pietzuch, P.: Integrating Scale out and Fault Tolerance in Stream Processing Using Operator State Management. ACM SIGMOD, 2013.
- [Ca17] Carbone, P.; Ewen, S.; F3ra, G.; Haridi, S.; Richter, S.; Tzoumas, K.: State Management in Apache Flink: Consistent Stateful Distributed Stream Processing. VLDB, 2017.

- [DG04] Dean, J.; Ghemawat, S.: MapReduce: simplified data processing on large clusters. USENIX OSDI, 2004.
- [Di15] Ding, J.; Fu, T.; Ma, R.; Winslett, Ma.; Yang, Y.; Zhang, Z.; Chao, H.: Optimal Operator State Migration for Elastic Data Stream Processing. CoRR, abs/1501.03619, 2015.
- [DM17] Del Monte, B.: Efficient Migration of Very Large Distributed State for Scalable Stream Processing. VLDB PhD Workshop, 2017.
- [HBS73] Hewitt, C.; Bishop, P.; Steiger, R.: A Universal Modular ACTOR Formalism for Artificial Intelligence. IJCAI, 1973.
- [He14] Heinze, T.; Pappalardo, V.; Jerzak, Z.; Fetzer, C.: Auto-scaling techniques for elastic data stream processing. In: IEEE ICDE Workshops. 2014.
- [He15] Heinze, T.; Ji, Y.; Roediger, L.; Pappalardo, V.; Meister, A.; Jerzak, Z.; Fetzer, C.: FUGU: Elastic Data Stream Processing with Latency Constraints. IEEE Data Eng. Bull., 2015.
- [Hi11] Hindman, B.; Konwinski, A.; Zaharia, M.; Ghodsi, A.; Joseph, A.; Katz, R.; Shenker, S.; Stoica, I.: Mesos: A Platform for Fine-grained Resource Sharing in the Data Center. In: USENIX NSDI. 2011.
- [Hi14] Hirzel, M.; Soulé, R.; Schneider, S.; Gedik, B.; Grimm, R.: A Catalog of Stream Processing Optimizations. ACM CSUR, 2014.
- [Hü15] Hüske, F.: Specification and Optimization of Analytical Data Flows. PhD thesis, TU Berlin, 2015.
- [Ka18] Karimov, J.; Rabl, T.; Katsifodimos, A.; Samarev, R.; Heiskanen, H.; Markl, V.: Stream Processing Performance in Online Game Scenarios. IEEE ICDE, 2018.
- [Ku15] Kulkarni, S.; Bhagat, N.; Fu, M.; Kedigehalli, V.; Kellogg, C. et al.: Twitter Heron: Stream Processing at Scale. ACM SIGMOD, 2015.
- [LML14] Lorido, T.; Miguel, J.; Lozano, J.: A review of auto-scaling techniques for elastic applications in cloud environments. Journal of grid computing, 2014.
- [Ma18] Mai, L.; Zeng, K.; Potharaju, R.; Xu, L.; Venkataraman, S.; Costa, P.; Kim, T.; Muthukrishnan, S.; Kuppa, V.; Dhulipalla, S.; Rao, S.: Chi: A Scalable and Programmable Control Plane for Distributed Stream Processing Systems. VLDB, 2018.
- [Na15a] Nasir, M.; Morales, G.; Kourtellis, N.; Serafini, M.: When Two Choices Are not Enough: Balancing at Scale in Distributed Stream Processing. CoRR, abs/1510.05714, 2015.
- [Na15b] Nasir, M.; Morales, G.; Soriano, D.; Kourtellis, N.; Serafini, M.: The power of both choices: Practical load balancing for distributed stream processing engines. IEEE ICDE, 2015.
- [Sc12] Schneider, S.; Hirzel, M.; Gedik, B.; Wu, K.: Auto-parallelizing stateful distributed streaming applications. ACM PACT, 2012.
- [ScZ05] Stonebraker, M.; Çetintemel, U.; Zdonik, S.: The 8 Requirements of Real-time Stream Processing. ACM SIGMOD, 2005.
- [To14] Toshniwal, A.; Taneja, S.; Shukla, A.; Ramasamy, K.; Patel, J.; Kulkarni, S.; Jackson, J.; Gade, K.; Fu, M.; Donham, J. et al.: Storm@ twitter. In: ACM SIGMOD. 2014.
- [Tu18] Tucker, P.; Tufte, K.; Papadimos, V.; Maier, D.: NEXMark - A Benchmark for Queries over Data Streams. 2018.
- [Va13] Vavilapalli, V.; Murthy, A.; Douglas, C.; Agarwal, S.; Konar, M. et al.: Apache Hadoop YARN: Yet Another Resource Negotiator. In: ACM SOCC. 2013.
- [WT15] Wu, Y.; Tan, K. L.: ChronoStream: Elastic stateful stream computation in the cloud. IEEE ICDE, 2015.
- [Za13] Zaharia, M.; Das, T.; Li, H.; Hunter, T.; Shenker, S.; Stoica, I.: Discretized Streams: Fault-tolerant Streaming Computation at Scale. ACM SOSp, 2013.

Text

A Hybrid Information Extraction Approach Exploiting Structured Data within a Text Mining Process

Cornelia Kiefer¹, Peter Reimann², Bernhard Mitschang³

Abstract: Many data sets encompass structured data fields with embedded free text fields. The text fields allow customers and workers to input information which cannot be encoded in structured fields. Several approaches use structured and unstructured data in isolated analyses. The result of isolated mining of structured data fields misses crucial information encoded in free text. The result of isolated text mining often mainly repeats information already available from structured data. The actual information gain of isolated text mining is thus limited. The main drawback of both isolated approaches is that they may miss crucial information. The hybrid information extraction approach suggested in this paper addresses this issue. Instead of extracting information that in large parts was already available beforehand, it extracts new, valuable information from free texts. Our solution exploits results of analyzing structured data within the text mining process, i.e., structured information guides and improves the information extraction process on textual data. Our main contributions comprise the description of the concept of hybrid information extraction as well as a prototypical implementation and an evaluation with two real-world data sets from aftersales and production with English and German free text fields.

Keywords: information extraction, clustering, text mining, free text fields

1 Introduction

Many data sets in research and industry capture information both in structured and unstructured data fields. Structured data fields are suitable if the data type and value domain fit the perceived purpose. For example, structured data fields are appropriate to store the duration of a downtime in a production line in seconds. Unstructured data fields are better if no suitable structured type is available or if one needs to express certain issues in natural language to be readable and understandable by human users. For example, unstructured free text fields are adequate when explaining how to repair a machine, since this information is complex and cannot be captured in structured data. Especially, humans tend to provide more complete information using natural language texts than using structured

¹ University of Stuttgart, Graduate School of Excellence Advanced Manufacturing Engineering, Nobelstr. 12, Germany cornelia.kiefer@gsame.uni-stuttgart.de

² University of Stuttgart, Graduate School of Excellence Advanced Manufacturing Engineering, Nobelstr. 12, Germany peter.reimann@gsame.uni-stuttgart.de

³ University of Stuttgart, Institute for Parallel and Distributed Systems, Universitätsstraße 38, Germany bernhard.mitschang@ipvs.uni-stuttgart.de

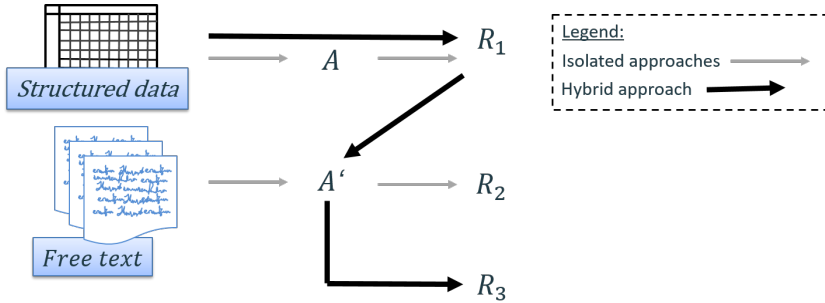


Fig. 1: Isolated information extraction approaches A on structured data and A' on unstructured text data yield results R_1 and R_2 . Our hybrid information extraction approach uses analysis result R_1 in the text mining process and yields result R_3 , thus extending results R_1 and R_2 .

information [HW96]. Thus, it is important to extract information not only from structured data, but also from unstructured, e.g., textual data as is mostly available in data sets from production, aftersales and research.

Standard approaches for information extraction from unstructured text data do not use structured data in text analysis (see Section 2). The result may be redundant information already available from structured data. Moreover, isolated approaches on structured data also miss crucial information. As illustrated in Figure 1, in contrast to these approaches, our hybrid information extraction approach exploits analysis results obtained from structured data in the text analysis pipeline.

The hybrid information extraction concept may be applied to all information extraction approaches on structured data with embedded or linked text data. In this paper, we present the concept and a prototypical implementation. In the evaluation, we present two real-world data sets and apply the method to them. Since for these data sets and use cases, no information on what to extract from the data is known beforehand, and no training data sets are available, we select a clustering information extraction approach for the prototypical implementation. Clustering is a robust and unsupervised means to extract information from text.

The goal of the suggested hybrid approach (yielding R_3 in Figure 1) is to increase the amount of new information, when compared to the information gained by the two isolated approaches (R_1 and R_2 in Figure 1). In our evaluation based on the two real-world data sets and the prototype, we denote the **'degree of new information'** with i_{new} and define it as shown in Formula 1.

$$i_{new} = (c_{new}/N) \quad (1)$$

where c_{new} is the number of cluster names not already known from the structured column and N is the number of all clusters considered.

By employing this purely quantitative metric, we are able to compare $R_1 - R_3$ in a straightforward way. Additionally, we can compare our results to the work of Ghazizadeh et al. [GML14]. In future work, we are going to consider more qualitative insights as well as additional quantitative metrics, e.g., based on entropy.

With an isolated approach on structured data, all information is extracted from the structured data fields only. Thus, based on the definition above, $i_{new} = 0$. Only, if information from free text is employed, i_{new} increases. For example, if half of the cluster names are not yet present in the structured information, i_{new} is 0.5. For R_2 and R_3 , i_{new} differs due to the exploitation of structured information available in the hybrid approach (R_3). By filtering out redundant information within the text mining process, the amount of new information increases. For our prototype and the two real-world data sets, for R_3 , i_{new} is 0.21 and 0.43 higher, than for R_2 .

The main contributions of this paper are:

- A description of the concept of hybrid information extraction.
- A discussion of design issues of a prototypical implementation of the approach for English as well as German free text fields.
- An evaluation of the hybrid information extraction approach. Here, we compare the results of isolated approaches (cf. R_1 and R_2 in Figure 1) with results of our hybrid approach (cf. R_3 in Figure 1) and we show that i_{new} is higher for the hybrid approach. For this purpose, we apply the prototype to an open dataset on problems with cars in aftersales (NHTSA data set⁴) and to a dataset on downtimes in a production line.

In the next two sections, we give an overview on work related to our approach and motivate hybrid information extraction with an example use case. In Section 4, we describe our method used for hybrid information extraction, and we discuss implementation details in Section 5. We illustrate the benefit of our approach with two data sets in Section 6 and conclude in Section 7.

2 Related Work

Plenty research works propose text mining approaches on free text. Compared to our work, these publications make no use of analytical results of structured data in the text mining process. Many approaches look at free text information in isolation. In the following, we present an excerpt of these approaches which work with real data sets: Carter et al. show a use case in the pharmaceuticals domain where they mine the Pillreports.com database using the k-means algorithm [CH14]. Gamon et al. apply clustering to mine opinions on cars in

⁴ <https://www-odi.nhtsa.dot.gov/downloads/>

the car reviews database⁵. The approach is based on a self-defined clustering algorithm [Ga05]. Brooks focuses on preventing industrial accidents [Br08]. In his approach, the SAS Text Miner Software is used to mine workers' compensation claims data. Clustering is based on the Expectation Maximization algorithm. Forman et al. assist technical support staff in a call center applying a self-developed clustering method on call logs [FKS06]. In many of the data sets used in these isolated approaches, also information in structured data fields is available. The main drawback of these approaches is however that they do not make use of this information source.

Many approaches use both structured and unstructured information in parallel. Yet, these approaches solely integrate the results of the isolated approaches. For many data sets like the data considered in this work such approaches are problematic, since valuable information may get lost (see Section 3). For example, Tan et al. mined data of a service center to get information on the expected processing times of service requests. Mining is based on structured data (the processing times) and case descriptions in free text fields [Ta00]. In their approach, they build a classification model which uses features induced separately from structured and text data. Chougule et al. speed up repair tasks of cars based on a framework, which combines association rule mining, case-based-reasoning and text mining [CRB11]. While the whole framework considers structured as well as unstructured data, the text mining component analyses the texts in isolation using hierarchical clustering algorithms.

Similarly, many approaches convert unstructured text data into structured data fields with the goal of merging structured and converted unstructured data and information. In contrast to our method, the information extraction methods work on the texts in isolation. For example, the DeepDive system structures free texts using statistical inference and machine learning [Ce15]. Gubanov et al. present the data tamer system [GSB14], where the structuring of textual data is based on external tools that are not described in more detail. After the conversion of the unstructured text data, modules such as schema integration and entity consolidation in data tamer may be applied.

Various approaches to information extraction are called hybrid, since they combine two machine learning algorithms. Silva et al. combine naive bayes, the PART algorithm and the k-nearest-neighbour with hidden markov models [SBP06]. Xiao et al. combine maximum entropy and maximum entropy markov models [XZZ08]. These approaches still analyze one type of data in isolation. They are not hybrid in the sense of using analytical results on structured data in the text mining process.

The work most related to ours is by Ghazizadeh et al. [GML14] and uses the same data set as we use (NHTSA data set, see Sections 3 and 6.1). Ghazizadeh et al. investigate reasons for fatal car accidents. They apply latent semantic analysis and hierarchical clustering to the free text fields in isolation. In difference to our approach, this work uses structured information in a first step only, before clustering takes place, to filter out the relevant part of the data. All structured information are ignored in the next steps of the information

⁵ <https://www.msn.com/en-us/autos/>

extraction process. Ghazizadeh et al. [GML14] present evaluation results which show that half of the cluster names correspond to vehicle components. These vehicle components represent information which is also available in a structured data field in the data set. Thus, only half of the clusters represent new and relevant information. In the hybrid approach suggested in this work we address this inconvenience.

While many approaches for the extraction of information from structured and free text data exist, the main drawback is that they are isolated: they do not employ structured information that is available and helpful within the text mining process. In this paper, we address this issue and show with two data sets from the product lifecycle that a hybrid approach to information extraction leads to new information that otherwise would be hidden behind redundant information.

3 Example Use Case

The department for National Highway Traffic Safety in the U.S. (NHTSA) wants to reduce the number of traffic crashes. For this purpose, they conduct recalls of unsafe vehicles and collect and analyze data on car crashes and problems with cars in a huge database since 1995. The data set contains structured information such as the car component affected. Customers filled this data field choosing the appropriate car component from a dropdown menu. Moreover, the NHTSA data set contains a free text field. The free text field describes the car crash or problem with the car. In Table 1 we show a small example data set containing information on the car component and a free text description.

Tab. 1: Example data set with structured (id, component) and unstructured information (description).

id	component	description
1	AIR BAG	AIR BAG FAILED DURING ACCIDENT (...)
2	AIR BAG	AIRBAG FAILED TWICE.
3	AIR BAG	AIR BAG LIGHT FAILED.
4	STEERING	VERY SENSITIVE STEERING AT HIGH VELOCITY (...)
5	STEERING	STEERING FAILED.
6	ENGINE	THE ENGINE SHUT OFF TWICE ON THAT DAY (...)
7	ENGINE	ALL ENGINE LIGHTS CAME ON (...)

An isolated analysis on structured data for example yields an ordered list of the most frequent car components involved in car crashes (cf. R_1 in Figure 1). An isolated analysis on the free text field also lists primarily car components (cf. R_2 in Figure 1). The results of Ghazizadeh et al. showed that half of the information in R_2 is not interesting to the analyst since it contains too much redundant information also contained in the structured field and i_{new} thus is comparably low ([GML14], see Section 2). They applied an isolated latent semantic analysis and hierarchical clustering approach to the NHTSA data set.

The hybrid approach tries to tackle this problem. It yields a list of frequently mentioned terms that are not deducible from the structured part of the dataset. R_1 and R_2 are oriented on car components, but R_3 mostly is oriented on issues. For example, in R_3 (cf. R_3 in Figure 1) the analyst finds new valuable information among the 5 highest ranked clusters (which will be discussed in more detail in Section 6.2): Many customers report problems in getting new secure car parts that the manufacturers need to change due to a recall. In isolated approaches the analyst misses this information since it is not present in R_1 , and in R_2 it is ranked as place 175 only (see Section 6.2 for more details on the prototypical implementation yielding R_2). This information may be crucial in preventing car crashes. Customers, while waiting for the secure car parts, might decide to drive the car anyway. Furthermore, the information in R_3 can be used to improve and extend the categories available in structured data in a feedback loop. The analyst may decide to add the 'unavailability of car parts' to the future structured data values available to the customers who file a complaint in the NHTSA data base.

4 Hybrid Information Extraction Approach

The goal of our approach is to extract more information from structured and free text data in terms of a higher degree of new information as defined in Formula 1 in the introduction to this work. In Figure 2, we illustrate an isolated approach to information extraction and show the resulting table in Figure 3. In Figures 4 and 5, we show an example illustrating the hybrid information extraction method and R_3 . Since we want to emphasize the difference between standard isolated approaches and our hybrid approach, we do not describe in this section preprocessing steps (such as tokenization) and vectorization, which both approaches have in common. We explain these steps in detail in the next section. Here, we focus on the steps special to the hybrid approach: (1) grouping and (2) removal.

In Figure 2 we illustrate an isolated approach. Here, free text fields are considered in isolation and all free texts are clustered. Finally, the name of the cluster in which a free text falls is added to the overall data set in the additional column 'cluster' as shown in Figure 3. The cluster name is based on the most frequent word in the cluster.

In Figure 4 we illustrate the first processing step of the hybrid approach, in which structured data is used to **group** free text fields. Here, the NHTSA data set is grouped by the structured data field on the car components into three groups (AIR BAG, STEERING and ENGINE) (step (1)). In Figure 5, we illustrate the next step, in which we **remove** all information that is already available in the structured data field on car components (step (2)). Only then, the free texts are **clustered** (step (3)). Finally we add a new column to the table which contains the name of the cluster, the result is shown in the last step in Figure 5. The isolated approach adds cluster information such as 'air bag' and 'steering'. This information is already available in the structured field 'car component' (see columns 'component' and 'cluster' in Figure 3). The hybrid approach results in clusters, such as 'light' and 'fail'. They represent new valuable information (see column 'cluster' and compare to column

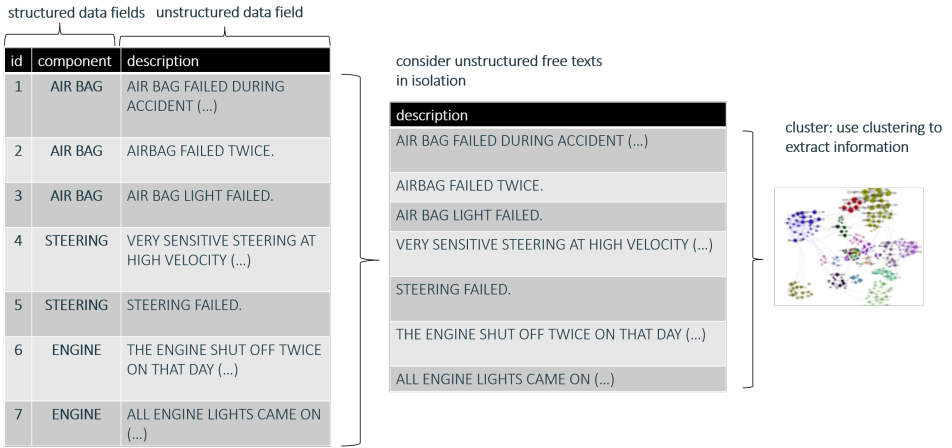


Fig. 2: Concrete example illustrating an isolated approach to information extraction from free text fields.

id	component	description	cluster
1	AIR BAG	AIR BAG FAILED DURING ACCIDENT (...)	air bag
2	AIR BAG	AIRBAG FAILED TWICE.	air bag
3	AIR BAG	AIR BAG LIGHT FAILED.	air bag
4	STEERING	VERY SENSITIVE STEERING AT HIGH VELOCITY (...)	steering
5	STEERING	STEERING FAILED.	steering
6	ENGINE	THE ENGINE SHUT OFF TWICE ON THAT DAY (...)	engine
7	ENGINE	ALL ENGINE LIGHTS CAME ON (...)	engine

Fig. 3: Concrete example illustrating the result of isolated information extraction from free text containing redundant information such as 'air bag' and 'steering'.

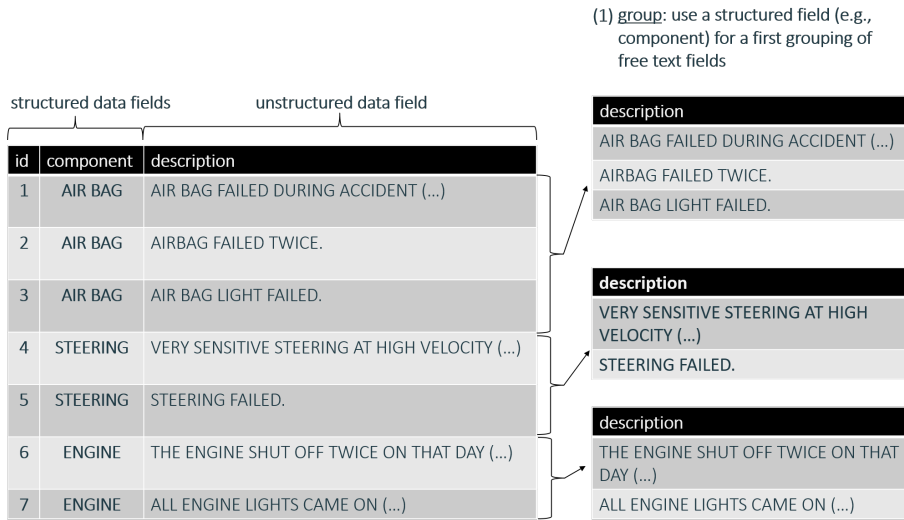


Fig. 4: Concrete example illustrating the distinguishing step 'group' of the hybrid approach to information extraction from free text fields.

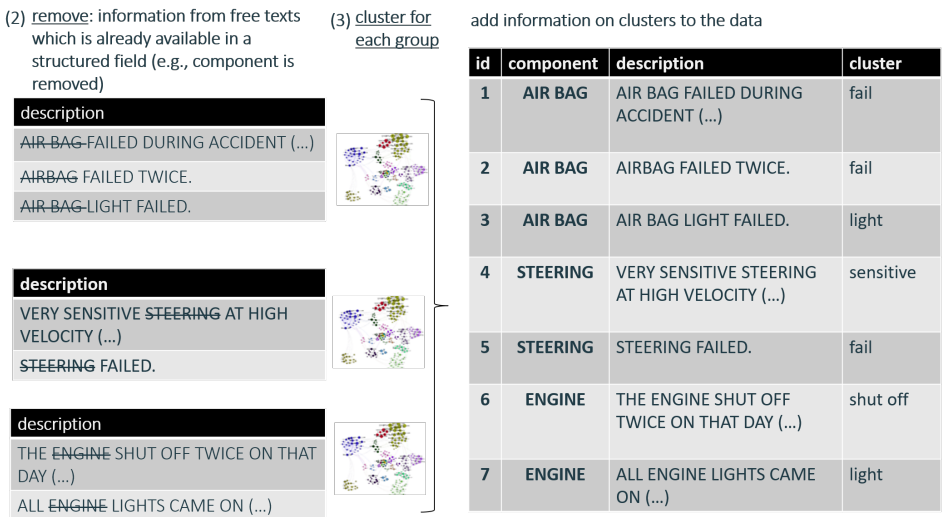


Fig. 5: Concrete example illustrating the distinguishing steps 'remove' and 'cluster for each group' of the hybrid approach to information extraction from free text fields and the result containing new valuable information such as 'fail' and 'light'.

'component' in Figure 5). We base our approach on three predominant characteristics of structured data sets with embedded free text fields, which we discuss in more detail in the remaining paragraphs of this section.

First of all, **free text fields store additional valuable information**. This was confirmed in many studies (see Section 2). Various methods, such as relation extraction, classification and clustering can extract valuable information stored in free text [ZM16]. We use clustering since it is suited best for our use cases. Moreover, Ghazizadeh et al. [GML14] also used a clustering approach, and we want to compare our results with their findings. However, the hybrid information extraction approach suggested in this paper is independent from the concrete information extraction method chosen. Also relation extraction and classification approaches may benefit from applying the concept to them. For example, a classifier which uses structured fields as well as unstructured free text fields in the feature generation, may benefit from removing information already present in structured fields from the free texts. The concrete effects on further information extraction methods need to be investigated in future work. Here, we focus on the validation of the core concept and employ a state-of-the-art clustering technique.

Second, in many data sets in industry and research, **we can group unstructured free text fields via information encoded in structured data fields**. For example, the NHTSA data set (see Section 6.1) may be divided into groups based on structured fields such as car component, year and car make. The hybrid approach uses this information for grouping. Thus, we do not end up extracting the same groups based on text mining free text fields. For a concrete example, see Figure 4, step (1).

Lastly, **if the same information can be extracted from either structured data or from free text fields of a data set, usually structured data is preferred**. In most research and industry data sets, the quality of structured data fields is estimated to be quite high. Pre-defined value ranges and quality control at the point of data entry lead to high quality structured data. However, the entry of texts is free and usually no pre-defined value ranges and quality control exist. Thus, free texts are oftentimes full of spelling mistakes, grammatical errors and abbreviations (compare e.g., [KM16] and [ZMZ16]). If an information is present in a structured field as well as in a free text field, we use the information from the structured field. Consequently, we do not want to extract this redundant information from the free text field with text mining. Thus, during preprocessing, we remove this information. E.g., in Figure 5 we remove the word 'steering' from all free texts in the respective group.

As we can see from Figure 5, in the hybrid approach, cluster names such as 'fail' or 'light' are added. After the grouping and removal step, these new cluster characteristics show up. Thus, compared to isolated approaches, our approach increases the amount of new information i_{new} available in the data set (see Section 6).

5 Design Decisions in the Implementation

The prototype is open source and can be retrieved from GitHub⁶. It is implemented in Python, since many Python programming libraries for natural language processing exist. The implementation is straightforward and helpful documentations are available (e.g., [BKL09] and [Pe14]). Furthermore, all tools and libraries chosen for the implementation of the prototype have industry-friendly licences. The prototypical implementation enables an easy integration of, e.g., new preprocessing methods, clustering algorithms and visualizations, as well as an easy adaptation to other languages. We designed the prototype that flexible so that both use cases with English and German free text fields may be covered easily. Other design decisions, such as on data types the prototype can read and preprocessing performed, are founded on the two use cases described in more detail in Section 6. In Figure 6, a schematic illustration of the prototype to our hybrid information extraction approach is shown. For the evaluation of the hybrid information extraction approach, we implemented a state-of-the-art clustering prototype as a baseline and a prototype for our hybrid approach. The two implementations are exactly the same, but for the two distinguishing processing steps 'group' and 'remove' (these two steps are bold-faced in Figure 6). In the following subsections we describe the processing steps from Figure 6 in more detail and state our implementation choices.

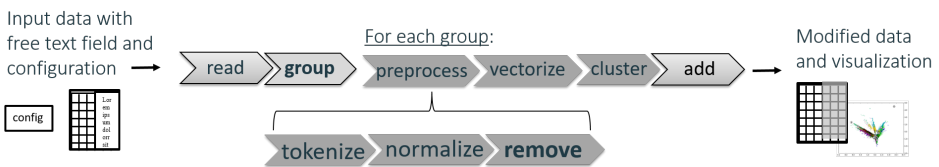


Fig. 6: Schematic illustration of the prototype. In the two boldfaced preprocessing steps 'group' and 'remove', structured data is used, which makes the approach hybrid.

For **reading** configurations, we used ConfigObj⁷. In the configuration, the user needs to state the column that contains the free texts and the column that contains the structured data that shall be used in the grouping and removal steps. If more than one structured categorical field is available and suitable, both may be applied to the 'removal' step. However, our use cases only require to select exactly one structured field for the 'grouping' step. Moreover, the processing steps can be freely defined by the user, or alternatively the default settings are used. With the default values, our prototype uses standard preprocessors, no synonyms in normalization, a tf-idf vectorizer and creates 12 clusters per group. The user may adapt these values if needed. ODBC data bases as well as CSV-formatted data sets may be read. Therefore, we use the library PyODBC⁸ and a CSV-standard tool in Python⁹. NumPy¹⁰ arrays represent the incoming and outgoing data.

⁶ <https://github.com/LinkMarco/PrototypeClustering>

⁷ <https://pypi.python.org/pypi/configobj/5.0.6>

⁸ <https://pypi.python.org/pypi/pyodbc/4.0.3>

⁹ <https://docs.python.org/3/library/csv.html>

¹⁰ <http://www.numpy.org/>

We base the **grouping** step on Python standard tools and SQL SELECT statements which are invoked from Python. All following steps are subsequently executed for each group. The free texts are grouped based on the structured data column as defined by the user in the configuration. For a concrete example of this processing step, see step (1) in Figure 4.

The **preprocessing phase** is a central part in text mining, since here the features are given by the words of the text (see, e.g., [MRS08]). Table 2 shows the main steps.

The detailed settings are flexible and can be determined in the configuration. The main library we use for natural language processing is the Natural Language Toolkit (NLTK)¹¹.

Tab. 2: Small example illustrating the preprocessing steps of our prototype: tokenization, normalization, and the removal of stopwords and redundant information as determined via analysis of structured data.

Preprocessing step	Sample text
Before preprocessing	AIRBAGS FAILED DURING ACCIDENT, BUT CAR PARTS ARE NOT AVAILABLE.
Tokenize	[AIRBAGS] [FAILED] [DURING] [ACCIDENT] [,] [BUT] [CAR PARTS] [ARE] [NOT] [AVAILABLE] [.]
Normalize	[air bag] [fail] [during] [accident] [but] [car part] [are] [unavail]
Remove	[fail] [accident] [car part] [unavail]

For **tokenization**, the Whitespace Tokenizer¹² or the Penn Treebank Tokenizer¹³ may be applied. In tokenization, the text is split into the smallest meaningful units such as words and compounds. We give an example in Table 2. Note that here the compound 'car parts', while being separated by a whitespace, was correctly selected as one token.

Then, we **normalize** all tokens. We provide plenty normalization methods in the prototypical implementation. These are optional and may be selected and adapted by the analyst for each use case, e.g., as described in Sections 6.2 and 6.3. In the normalization process, we may lowercase all texts. Spelling mistakes may be corrected using TextBlob¹⁴. Furthermore, contractions such as 'didn't' may be extracted. We use the multi-word expression tokenizer from NLTK¹⁵ for extraction. Then, white spaces may be normalized (two or more whitespaces are reduced to one). Moreover, we may remove urls, mail addresses, telephone numbers, numbers, punctuation marks, currency signs and accents using tools from Textacy¹⁶. Finally, we may stem the tokens, i.e., we delete affixes with the goal of normalizing and thereby grouping the tokens. For stemming, we apply the Porter Stemmer¹⁷ from NLTK. Additionally, different expressions which have the same meaning (=synonyms) may be consolidated. We

¹¹ <http://www.nltk.org/>

¹² http://www.nltk.org/_modules/nltk/tokenize/regexp.html#WhitespaceTokenizer

¹³ http://www.nltk.org/_modules/nltk/tokenize/treebank.html#TreebankWordTokenizer

¹⁴ <https://pypi.python.org/pypi/textblob>

¹⁵ http://www.nltk.org/_modules/nltk/tokenize/mwe.html#MWETokenizer

¹⁶ <https://pypi.python.org/pypi/textacy>

¹⁷ http://www.nltk.org/_modules/nltk/stem/porter.html#PorterStemmer

implemented the synonym consolidation based on standard tools in Python. In the small example in Table 2, we lowercase all words. Then we normalize 'NOT AVAILABLE' to its synonym 'unavailable'. Finally we stem the tokens, which e.g., transforms 'unavailable' to 'unavail' and 'failed' to 'fail'.

We base the **removal step** on the Word List Corpus Reader¹⁸ in NLTK. In this step, we remove stopwords. Stopwords are words that do not bear interesting information, but merely are present in the texts for grammatical reasons. In Table 2, *during*, *but* and *are* were identified as stopwords. Additionally we remove the information already present in the structured data column specified by the user. In our prototypical implementation, we add the string from the categorical structured field to the stopword list. Since these structured values usually are words in their base form, they match with the corresponding word occurrences in the stemmed free text fields. Depending on the use case and data set, further resolutions of synonyms and abbreviations need to be added, which are also supported by the prototype. For a concrete example of the 'remove' processing step, see Figure 5 step (2). In the small example in Table 2, the word 'air bag' was additionally removed.

We use the machine learning library Scikit-learn¹⁹ for **vectorizing and clustering** the free text fields. Vectorization means building a representation for each document that notes which words are present in the document and how these shall be weighted. The **vectorizer** can use two different weighting schemes: Either plain term frequencies (tf) or term frequencies times inverse document frequency (tf-idf). Tf-idf is a weighting scheme often used in information retrieval [MRS08], which leads to a proper baseline clustering prototype. Here, terms which are very frequent in the complete free text collection are downweighted and rare ones are upweighted. Thus, much redundant information is yet downweighted by means of the state-of-the-art weighting scheme. Thus, the state-of-the-art approach already extracts much new information and is a strong baseline. As we will show in Section 6, i_{new} still is higher for the hybrid information extraction approach than for that baseline. Several **clustering** algorithms are implemented in Scikit-learn. For its popularity and robustness, we chose the k-means algorithm for the prototype. This algorithm is a hard partitioning clustering algorithm, which means that each free text may only be put into exactly one of the clusters built. K-means is implemented following Lloyd's algorithm [L106]. We use NumPy for array representations and calculations in Scikit-learn. Thus, vectorization and clustering is fast. The prototype is built so that it is possible to calculate and compare i_{new} in the evaluation of the core concept. While we employ a robust and state-of-the-art clustering algorithm in the prototype, the concept is independent of the implementation chosen. Since the clustering step in our prototype is based on the Scikit-learn library, it is easy to add other clustering algorithms if needed.

Finally, **a new column is added** to the original data set. For each data instance it contains the name of the cluster to which the data instance belongs. The cluster name is based on the most frequent word in the cluster. This processing step uses NumPy arrays and SQL.

¹⁸ http://www.nltk.org/_modules/nltk/corpus/reader/wordlist.html

¹⁹ <http://scikit-learn.org/stable/>

Visualizations are optional. We implemented them using Matplotlib²⁰. The clusters as well as cluster quality metrics such as the silhouette coefficient may be visualized.

6 Evaluation of the Hybrid Information Extraction Approach

In the following subsections, we give details on two data sets from the product life cycle and explain how we apply the prototype to them. Furthermore, we compare the results of the hybrid approach with the results from isolated approaches. In Figure 1 these results are illustrated as R_1 - R_3 , where R_1 is the result of an isolated approach on structured data, R_2 is the result of an isolated approach on unstructured data, and R_3 is the result of our hybrid approach. For easy reference, we will denote the prototypes used in the experiments by R_1 - R_3 respectively. In the following we define the three prototypes tested:

- R_1 : The **isolated approach on structured data** is based on a simple SQL-query which we run on the databases. It is exemplified for the categorical structured data field 'component' in the NHTSA data set in the following SQL statement²¹. It simply groups the data for the structured data field 'component', counts the lines, and orders the result in descending order:

```
SELECT component, count(*) FROM nhtsa_table
GROUP BY component ORDER BY count(*) DESC;
```

- R_2 : We base the implementation of the **isolated approach on unstructured data** on the prototype described in Section 5. In fact, it is exactly the same, but the grouping as well as the removal of information from free texts based on the structured data field are omitted. Standard stopwords such as *and*, *the*, *it* are still removed. This ensures that the effect of the steps special to the hybrid approach can be viewed in isolation.
- R_3 : We described the implementation of the **hybrid information extraction prototype** in the previous section. It is adapted to the two use cases, i.e., as described below tailored configurations such as synonyms and preprocessors are defined.

By applying both R_2 and R_3 to the data sets, we can compare a state-of-the-art baseline clustering approach (R_2) with the very same approach plus the two distinguishing steps 'grouping' and 'removing' (R_3). Both R_1 and R_2 are oriented on the components, i.e., air bag, steering, power train, whereas R_3 mostly is oriented on issues, i.e., fail, noise, (unintended) acceleration, unavailable (car parts). In the presentation of our detailed evaluation results, we thus show for comparison the results R_1 representing component-based and R_3 representing issue-based results. Moreover, we report the difference between the three approaches in terms of the amount of new information i_{new} (as defined in Formula 1 in Section 1) and discuss the resulting clusters.

²⁰ <http://matplotlib.org/>

²¹ Note: the 'component' column is named 'compdesc' in the original NHTSA data set.

6.1 Data Sets

To evaluate the prototype, we apply it to two data sets. The first one is a freely accessible data set from the National Highway Traffic Safety Administration (NHTSA) in the United States of America. Since the data set²² as well as our prototype are freely accessible, all evaluation results with respect to the NHTSA data set are reproducible. The NHTSA complaint data set currently contains more than 1.3 million reports on incidents with cars.

The second data set used for illustration of our prototype comes from an industry partner in Germany. It contains 153k entries, comprises information on downtimes in a production line and contains German free text information. On that line, smaller, but complex parts of a car are manufactured. This data set allows us to see how the prototype may be applied to a use case in production. It contains structured information on the downtimes, such as error codes and the duration of a downtime in seconds. Furthermore, information on the reasons for downtimes and the actions that were taken to put the production line running again are noted in a free text field. The workers can fill the free text field via text entry into a tablet, directly on the shop floor. The text entries are in German and quite short (4.1 words per entry in average) and full of spelling mistakes and domain-specific abbreviations, which brings special challenges with respect to information extraction.

6.2 Data Analysis of the NHTSA Data Set

To apply the prototype resulting in R_3 to the NHTSA data set, at least the column that contains the structured field and the column which contains the free text field need to be given. We use the structured field with the affected car component in the grouping and removal steps. We stick with the default settings (see Section 5), but chose to add some synonyms and context synonyms. For example, all occurrences of 'not' or 'failed' and 'deploy' with no more than 3 words in between are normalized to 'not deploy'. Thereby, different ways of expressing the same concept are normalized to one term. The added synonym consolidations help in building semantically reasonable clusters. In this use case they have no influence on the 'removal' step of our core method. From this hybrid prototype adapted to the use case, the isolated prototype for R_2 is deduced. Both approaches are the same but for the grouping and removing steps and generate the same number of clusters (12 per group). R_1 is computed as defined above.

In the first analysis, we focus on how an isolated approach on structured data only (resulting in R_1) and the hybrid approach (resulting in R_3) differ. In Figure 7, we show the five most frequent car components (R_1 , left) and cluster terms (R_3 , right). For easier readability and comprehensibility, we name clusters by the most frequent word in that cluster in the base form. From the structured data, we extract the information on the most frequent car components affected. E.g., the electrical system, air bags, power trains of automatic transmissions,

²² NHTSA complaints data set: <https://www-odi.nhtsa.dot.gov/downloads/>

steering and power train. Using the hybrid approach (R_3), additional information can be extracted: We already know from structured data that problems with transmission are frequent. But we did not know that many complaints are about problems with noise, acceleration, unavailable (car parts) and problems where the car stalls. Accelerating the supply with car parts needed due to a recall (cluster 'unavailable') and investigating problems, where the car stalls (cluster 'stall') or accelerates (cluster 'acceleration') might help in preventing car crashes. The result R_3 completes the information already available. The user can use both in his analysis: structured as well as generated cluster information. In Table 3 we also note both, information from R_3 and R_1 (the latter is noted in brackets following the free text samples).

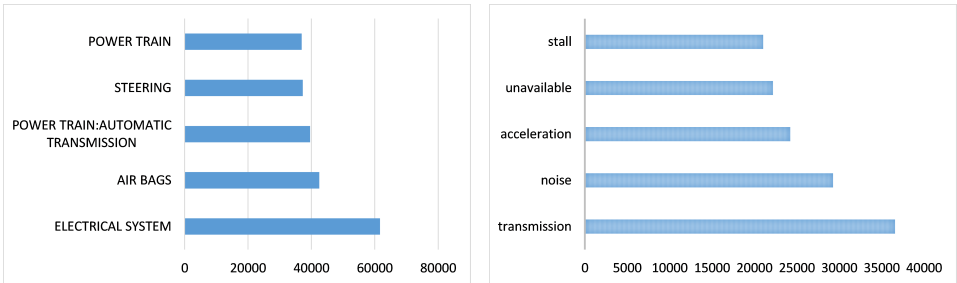


Fig. 7: Most frequent car problems based on an isolated analysis of structured data only (R_1 , left) and on the hybrid analysis (R_3 , right), all based on the NHTSA data source.

Tab. 3: Concrete examples of NHTSA complaints for most frequent clusters together with structured information as noted in the categorical field 'component' given in parenthesis

Cluster	Sample complaints
stall	VEHICLE STALLED DUE TO AN ELECTRICAL PROBLEM.('component': 'electrical system'); ENGINE STALLS WHEN APPROACHING A STOP. ('component': 'power train')
unavailable	THE PART TO DO THE REPAIR WAS UNAVAILABLE. (...) ('component': 'air bag'); (...) PARTS FOR RECALL IS NOT AVAILABLE SINCE REQUESTING (4) WEEKS AGO. (...) ('component': 'child seat')
acceleration	VEHICLE ACCELERATED BY ITSELF (...) ('component': 'vehicle speed control'); THE VEHICLE IS NOT ACCELERATING PROPERLY. (...) ('component': 'power train:automatic transmission')
noise	IT MAKES A LOUD NOISE AND NO 1 KNOW WHAT IT IS ('component': 'electrical system');WHINING NOISE WHEN TURNING STEERING WHEEL. (...) ('component': 'steering')
transmission	TRANSMISSION FAILURE AT 105,000 MILES (...)('component': 'power train:automatic transmission')

In a last experiment with the NHTSA data, we want to check how R_1 - R_3 differ in terms of i_{new} (see Formula 1 in Section 1). In R_1 no new information which goes beyond the structured information may be gathered, thus $i_{new} = 0$. We compare the degree of new information within the 100 biggest clusters in R_2 and R_3 . We illustrate the difference between

Tab. 4: Comparison of the degree of new information i_{new} for the three approaches R_1 - R_3 on the NHTSA data set

Result set	i_{new}
R_1 (only structured information)	0
R_2 (baseline)	0.55
R_3 (hybrid approach)	0.98

R_1 - R_3 with respect to this measure in Table 4. Here we see that R_3 contains significantly more new information than R_2 . Also see the results already described in Section 3. The new information might be crucial for manufacturers, e.g., to get better customer satisfaction. Thus, we were able to confirm and improve the results of Ghazizadeh et al. [GML14], who report half of the cluster names to correspond to vehicle components, which corresponds to an i_{new} value of 0.5.

6.3 Data Analysis of the Industry Data Set

For a second evaluation, we apply the prototypes to an industry data set with a German free text field. It contains information on causes and actions related to machine downtimes. We use a structured field with an error code description which indicates the group of errors of a downtime on the production line for grouping. The possible choices for filling the structured data field do not cover all types of errors and reasons for downtimes that may occur in reality. More choices are indicated in a free text field and may be deduced by our hybrid information extraction method, only.

To apply our prototype resulting in R_3 to the industry data set, details in preprocessing need to be changed in the configuration file due to German text: a German standard stopword list (from NLTK, see Section 5) and a German stemmer²³ need to be specified. We normalize some spelling mistakes and verbalizations and add a few synonyms and context synonyms. For example, if the german words 'strom' and 'leerlaufstrom' (both terms are on power/electricity) are mentioned near 'hoch' (english 'high') the main word is substituted by 'strom_hoch' (in english 'power_high'). Some of the added synonyms help ensure a good recall of the 'removal' step. Also, encoded umlauts such as 'ae' and 'ue' are normalized to 'ä' and 'ü' respectively. After adapting the hybrid prototype to the use case, R_2 is deduced from it. Both approaches yield the same number of clusters and only differ in terms of the two distinguishing steps 'group' and 'remove'. K in k -means is set as described above for the NHTSA data. For R_1 , the data is grouped by error code description with a SQL statement similar to the one described in the introduction of this section. For reasons of confidentiality, we use high-level abstractions of the German definitions. We furthermore translated them to English for the examples given in this work.

²³ e.g., http://www.nltk.org/_modules/nltk/stem/snowball.html#GermanStemmer

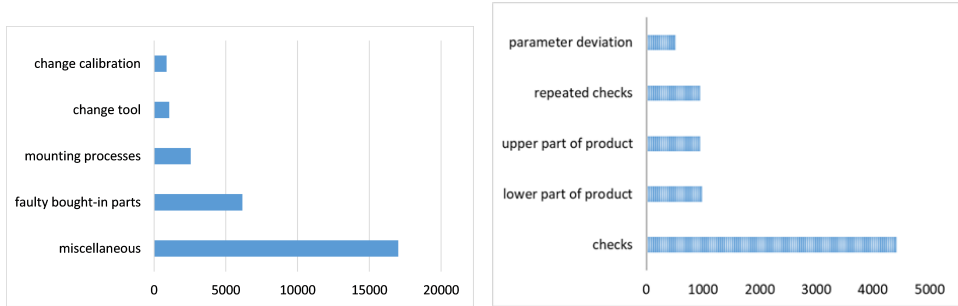


Fig. 8: Most frequent reasons for downtimes based on the structured data field containing an error code (R_1 , left) and based on the hybrid approach (R_3 , right), all based on the industry partner data source.

Finally, we compare the results of an isolated approach on structured data only (R_1) with the results of the hybrid approach (R_3). Using an isolated approach on structured data, the reasons for downtimes may be analyzed using the structured field with an error code and a table that defines these error codes. Following the hybrid approach, many fine-grained clusters are gained that represent new information. The results are illustrated in Figure 8. We see that the structured information on errors on the production line are very coarse-grained: problems due to faulty bought-in parts, mounting processes in general, change of tools and change of calibrations often lead to downtimes. The most frequent error code hints at 'miscellaneous' problems. This group is very big since many reasons for downtimes are not reflected in the given structured data values. Therefore the workers on the shop floor oftentimes chose the structured value 'miscellaneous' instead.

This is not helpful for the workers on the shop floor, who want to prevent or fix problems with downtimes of a production line. In contrast, the clusters resulting by our hybrid approach (R_3) are much more fine-grained. For example, in the big structured group 'miscellaneous', clusters such as 'problems with component parts' and 'problems with out-of-commission machines' were found. These give more detailed information to workers on the shop floor and to managers of the production line. From R_3 , more detailed and new information on reasons for downtimes may be extracted: From biggest to smaller clusters, detailed information on problems with checks, parts of the products which are produced, repeated checks and parameter deviations that lead to downtimes in the production line are reflected in the clusters. If this data was prepared for the shop floor workers, it might help in solving new downtimes of the production line faster. In a feedback loop, new reasons for downtimes in the production line may be added to the list of error codes in order to strengthen the significance of the structured fields.

We moreover compare the degree of new information in Table 5 with the same method as described for the NHTSA data set in the previous section (see Formula 1 in Section 1). In this case the baseline is even stronger, due to weaker structured information available in

Tab. 5: Comparison of the degree of new information i_{new} for the three approaches R_1 - R_3 and the industry data set

Result set	i_{new}
R_1 (only structured information)	0
R_2 (baseline)	0.78
R_3 (hybrid approach)	0.99

the industry data. Still, R_3 has a 0.21 higher i_{new} value than R_2 . The grouping step helps to reduce the number and size of big 'miscellaneous' clusters. Special attention needs to be paid to synonyms, spelling mistakes and abbreviations. If not addressed properly, these issues may lead to less exploitation of the benefits of the removing step. We discussed the different results with domain experts and found that the additional information contained in R_3 is relevant to the task of optimizing the process. Also, from the point of view of domain experts, the hybrid information extraction method has future potential: Before a new shift on the production line begins, a summary of current insights may be presented to the shop floor workers. Moreover, new staff could be assisted by a presentation of aggregated insights.

7 Conclusion and Future Work

We suggested a hybrid approach to the extraction of information from free text fields which yields more new information i_{new} from data with structured data fields and unstructured free text fields. The approach is based on natural language processing and k-means clustering and improves the results of two baseline isolated approaches by employing analytical results from structured data within the text analysis process. First, we group data based on a structured data field. Then, we preprocess data, while also redundant information, as determined via a structured data field, is removed. Then, data is clustered and a new column containing the cluster name is added to the data set. In this paper, we describe the concept and implementation of our approach for hybrid information extraction and discuss relevant design considerations. We based the prototype for the two baselines as well as the hybrid information extraction approach on free, open-source tools. The prototype is freely available on GitHub²⁴. The prototype for R_2 can be deviated from it and R_1 can be gathered by means of a SQL-query as shown in Section 6. Finally we evaluated our approach with two example data sets with German and English free text fields. While the data set from production is confidential, the NHTSA data set may be downloaded²⁵ and thus the results presented with respect to this use case are reproducible. We compared our approach to baseline isolated approaches on structured or unstructured data. We showed that isolated approaches to free text yield much redundant information already available in structured

²⁴ <https://github.com/LinkMarco/PrototypeClustering>

²⁵ <https://www-odi.nhtsa.dot.gov/downloads/>

data. The hybrid approach impedes this and yields more new information. For the two use cases, the degree of new information in R_3 is significantly higher than in R_2 .

In future work, we integrate our hybrid information extraction approach into a framework of methods for measuring and improving data quality of product lifecycle data. Then, we will apply the concept to further information extraction approaches. In future work, an efficient handling of big data may be enabled by transferring the prototype into text databases (e.g., [KLA15]). Moreover, we will address issues we encountered in analyzing production data such as synonyms, spelling mistakes and abbreviations. Furthermore, we will employ additional evaluation metrics, e.g., based on entropy.

Acknowledgements

The authors would like to thank the German Research Foundation (DFG) for financial support of this project as part of the Graduate School of Excellence advanced Manufacturing Engineering (GSaME) at the University of Stuttgart. Moreover, we thank Marco Link for crucial implementation work.

References

- [BKL09] Bird, Steven; Klein, Ewan; Loper, Edward: Natural Language Processing with Python. O'Reilly Media, 2009.
- [Br08] Brooks, Benjamin: Shifting the focus of strategic occupational injury prevention: Mining free-text, workers compensation claims data. *Safety Science*, 46(1):1–21, 2008.
- [Ce15] Ce Zhang: DeepDive: A Data Management System for Automatic Knowledge Base Construction. PhD thesis, University of Wisconsin-Madison, 2015.
- [CH14] Carter, B.; Hofmann, M.: An analysis into using unstructured non-expert text in the illicit drug domain. In: 2014 IEEE International Advance Computing Conference (IACC). pp. 651–657, 2014.
- [CRB11] Chougule, Rahul; Rajpathak, Dnyanesh; Bandyopadhyay, Pulak: An integrated framework for effective service and repair in the automotive domain: An application of association mining and case-based-reasoning. *Computers in Industry*, 62(7):742–754, 2011.
- [FKS06] Forman, George; Kirshenbaum, Evan; Suermondt, Jaap: Pragmatic Text Mining: Minimizing Human Effort to Quantify Many Issues in Call Logs. In: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD '06, ACM, New York, NY, USA, pp. 852–861, 2006.
- [Ga05] Gamon, Michael; Aue, Anthony; Corston-Oliver, Simon; Ringger, Eric: Pulse: Mining Customer Opinions from Free Text. In: Proceedings of the 6th International Conference on Advances in Intelligent Data Analysis. IDA'05, Springer-Verlag, Berlin, Heidelberg, pp. 121–132, 2005.

- [GML14] Ghazizadeh, Mahtab; McDonald, Anthony D.; Lee, John D.: Text Mining to Decipher Free-Response Consumer Complaints: Insights From the NHTSA Vehicle Owner's Complaint Database. *Human Factors The Journal of the Human Factors and Ergonomics Society*, (56, 6):1189–1203, 2014.
- [GSB14] Gubanov, M.; Stonebraker, M.; Bruckner, D., eds. Text and structured data fusion in data tamer at scale: 2014 IEEE 30th International Conference on Data Engineering, 2014.
- [HW96] Hogan, W. R.; Wagner, M. M.: Free-text fields change the meaning of coded data. *Proceedings of the AMIA Annual Fall Symposium*, pp. 517–521, 1996.
- [KLA15] Kiliyas, Torsten; Löser, Alexander; Andritsos, Periklis: INDREX: In-database relation extraction. *Information Systems*, 53:124–144, 2015.
- [KM16] Kassner, Laura; Mitschang, Bernhard: Exploring Text Classification for Messy Data: An Industry Use Case for Domain-Specific Analytics. In: *Advances in Database Technology - EDBT 2016, 19th International Conference on Extending Database Technology, Proceedings*. OpenProceedings.org, pp. 491–502, 2016.
- [LI06] Lloyd, S.: Least Squares Quantization in PCM. *IEEE Trans. Inf. Theor.*, 28(2):129–137, 2006.
- [MRS08] Manning, Christopher D.; Raghavan, Prabhakar; Schütze, Hinrich: *Introduction to information retrieval*. Cambridge University Press, New York, 2008.
- [Pe14] Perkins, Jacob: *Python 3 text processing with NLTK 3 cookbook: Over 80 practical recipes on natural language processing techniques using Python's NLTK 3.0*. Packt Pub., Birmingham, UK, 2014.
- [SBP06] Silva, E. F. A.; Barros, F. A.; Prudencio, R. B. C., eds. *A Hybrid Machine Learning Approach for Information Extraction: 2006 Sixth International Conference on Hybrid Intelligent Systems (HIS'06)*, 2006.
- [Ta00] Tan, Pang-Ning; Blau, Hannah; Harp, Steve; Goldman, Robert: Textual Data Mining of Service Center Call Records. In: *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD '00*, ACM, New York, NY, USA, pp. 417–423, 2000.
- [XZZ08] Xiao, Ji-Yi; Zhu, Dao-Hui; Zou, La-Mei, eds. *A hybrid approach for web information extraction: 2008 International Conference on Machine Learning and Cybernetics*, volume 3, 2008.
- [ZM16] Zhai, ChengXiang; Massung, Sean: *Text Data Management and Analysis: A Practical Introduction to Information Retrieval and Text Mining*. ACM, New York, NY, USA, 2016.
- [ZMZ16] Zhang, Yuhao; Mao, Wenji; Zeng, Daniel: A Non-Parametric Topic Model for Short Texts Incorporating Word Coherence Knowledge. In: *Proceedings of the 25th ACM International Conference on Information and Knowledge Management. CIKM '16*, ACM, New York, NY, USA, pp. 2017–2020, 2016.

Perceptual Relational Attributes: Navigating and Discovering Shared Perspectives from User-Generated Reviews

Manuel Valle Torre,¹ Mengmeng Ye Christoph Lofi²

Abstract: Effectively modelling and querying experience items like movies, books, or games in databases is challenging because these items are better described by their resulting user experience or perceived properties than by factual attributes. However, such information is often subjective, disputed, or unclear. Thus, social judgments like comments, reviews, discussions, or ratings have become a ubiquitous component of most Web applications dealing with such items, especially in the e-commerce domain. However, they usually do not play major role in the query process, and are typically just shown to the user. In this paper, we will discuss how to use unstructured user reviews to build a structured semantic representation of database items such that these perceptual attributes are (at least implicitly) represented and usable for navigational queries. Especially, we argue that a central challenge when extracting perceptual attributes from social judgments is respecting the subjectivity of expressed opinions. We claim that no representation consisting of only a single tuple will be sufficient. Instead, such systems should aim at discovering *shared perspectives*, representing dominant perceptions and opinions, and exploiting those perspectives for query processing.

Keywords: Perceptual Attributes; Modelling; User-Generated Attribute Values; Query-By-Example Navigation

1 Introduction

Social judgments like comments, reviews, discussions, or ratings have become an ubiquitous component of most Web applications, especially in the e-commerce domain. Now, a central challenge is using these judgments to improve the user experience by offering new query paradigms. Recommender systems have already demonstrated how ratings can be effectively used towards that end, providing users with proactive guidance within large item databases.

In this paper, we will discuss how to use unstructured reviews to build a structured semantic representation of such items, enabling the implementation of user-driven queries. Thus, we address one of the central challenges of Big Data systems: making sense of huge collections of unstructured user feedback. More specifically, we discuss the challenge of building structured, but latent representations of “experience items” stored in a relational database (like movies, books, music, games, but also restaurants or hotels) from unstructured user

¹ TU Delft, Web Information Systems, Van Mourik Broekmanweg 6, 2628 XE Delft, Netherlands; m.valltorre@tudelft.nl

² TU Delft, Web Information Systems, Van Mourik Broekmanweg 6, 2628 XE Delft, Netherlands; c.lofi@tudelft.nl

feedback. Such representations should encode the consensual perception of an item from the point of view of a large general user base. Consequently, this information can then be exploited for allowing semantically richer queries. In the following, we will use movies as a use case. However, the described techniques can easily be transferred to any other domain which has user ratings or reviews available.

While there have been previous works also aiming at representing items in a database based on social judgments (e.g., [LW15], [LN14]), we address one major yet unresolved problem: user judgments are inherently subjective as they represent a user’s perception. While rating-based systems are widely used, semantic quality quickly deteriorates when richer information sources like reviews are considered; they are less in quantity but richer in content and thus can express a wider variety of opinion. Furthermore, mining reviews is harder due to the complexity of natural language. Here, aspect-oriented sentiment analysis [YLT17] or document-embeddings [LW15] have been used to create structured representations which then later can be used for database query processing. However, the commonly chosen approach of combining the resulting representations into a single tuple (e.g., by averaging the document embeddings) is often not meaningful. Considering an example case of the movie “Twilight” (2008), typical reviews might express that the movie is a “beautiful romance full of alluring characters” or “an overall stupid movie and a disgrace to the vampire genre”. Other reviews might even be unrelated to the item itself, e.g. “my DVD was damaged on arrival, bad seller”. Clearly, a meaningful representation of the movie should incorporate all its relevant points of views, thus the aggregation of several user judgments has to be performed carefully. This problem is further aggravated by the current trend towards opinion polarization [MTT17]: publicly stated user opinions for example in reviews or comments are increasingly extreme, making it even more relevant than ever to respect different points of view when representing items.

Therefore, in this paper, we propose to represent each experience item in a database using multiple shared perceptual perspectives, with each perspective representing one major consensual opinion aggregated from multiple user judgments. Our contributions are:

- We present a large-scale *modelling experiment* outlining some of the challenges when modelling perceptual attributes
- We present the *foundations* of shared perspectives for experience items
- We provide an overview of the *design space* of different techniques and methods available to obtain and process such perspectives
- We introduce an adapted variant of the *query-by-example paradigm* intended to interact and query multiple shared perspectives
- We present our *prototype implementation* of a system using shared perspectives, and give insights into its query processing performance using simulations

- We conduct and present a *user study*, giving insights into the usefulness and semantic representativeness of the perspectives in our prototype system
- Based on the results of this study, we *identify shortcomings and challenges* with shared perspectives, and propose additional techniques to address some of them

2 Towards Shared Perspectives and Modelling Perceptual Properties

In the precursor work of this paper [LW15], we already explored preliminary concepts and implementations for encoding social judgments for capturing experience items like movies, books, etc. in relational databases. We used an e-commerce scenario where users can browse for experience items. This type of scenario is a prime application for user-generated judgments: user-friendly interaction with experience items is notoriously difficult, as there is an overwhelming number of those items easily available. Some of them are vastly popular and easily accessible mainstream items, but most of them are relatively unknown long tail products which are hard to discover without suitable support. Even more, the subjective user experience those products will entail (which, for most people, is the deciding factor for buying or consuming the product) is difficult to describe by typically available meta-data like production year, actor names, or even rough genre labels. Due to this problem, web services dealing with experience products enthusiastically embraced techniques for motivating the creation of user-generated judgments in the form of ratings, comments or reviews. In its most naïve (but very common) implementation, rating and review data are simply displayed to users without any additional processing (e.g., as seen in most current video streaming or shopping platforms). Querying and discovering items still relies on traditional SQL-style queries and categorizing based on non-perceptual meta-data (e.g., year, actor list, genre label, etc.). Manually reading these user judgments may help potential new customers to decide if they will like or dislike a certain item, but it does not really help them to discover new items beyond their expertise (i.e., manually reading user judgments works fine if a user knows what she is looking for, but has not yet come to a final buying decision). This led to the development of recommender systems [LSY03, BKV10], which proactively predict which items a user would enjoy. Often, this relies on collaborative filtering techniques [LSY03] which exploit a large number of user-item ratings for predicting a user's likely ratings for each yet-unrated item. While collaborative filtering recommender systems have been proven to be effective [KB11], they have only very limited query capabilities (basically, most recommender system offer just a single static query for each user).

2.1 Modelling Perceptual Properties

For enabling semantic queries like similarity exploration or query-by-example queries [LW15, LN14], the first step is to find semantically meaningful representations of database items going beyond simple available structured meta-data. As motivation for this work, we

argue that experience items are generally better characterized by their *perceptual properties*, e.g. their mood, their style, or if there are certain plot elements present – information which is rarely explicitly available and expensive to obtain. Furthermore, from a modelling point of view, it is often unclear which perceived properties describe items well.

Thus, in the following we investigate the question of “*How well can perceptual attributes be modelled manually by (semi-skilled) database schema designers*”. This modelling challenge can be approached in multiple different ways, as in [TNK10] where simply the Oscar Award categories are used (like “cinematography” “music” “costume design”) - a design decision which might represent some movies better than others, and might not necessarily represent how the general public would describe (and query) for movies.

For this paper, we conducted a *modelling experiment with 180 second year university BSc students*, and asked them to create a ranked list of perceptual attributes (i.e. attributes describing the consumption experience of a movie). The core task was “*Which attributes should be used to describe movies beyond typical structured meta-data as e.g., available in IMDB³.*” Also, a brief motivation why they think that these attributes would be relevant should be given. Practical limitations, like the challenge of how to obtain the values for such attributes, or the problem that certain attributes might be subjective, were to be ignored.

Tab. 1: Modelling Experiment Example Response

Attribute	Importance	Explanation
Quality of Acting	5	The quality of the acting in a movie can make a great difference in the overall quality. Believable acting can be a great help to a movies perceived quality.
Originality of Storyline	3	Most big movies seem to have the same storyline with some characters changing names, they are predictable and bore a lot of people. A lot of people are therefore interested in movies with original storylines
Amount of Explosions	2	A large amount of explosions will put off some viewers while others will really appreciate them.
Quality of Scenes	3	Simply put fight scenes can differ greatly in quality from the lameness that is Darth Vader vs Palpatine to the awesomeness that is gypsy danger rocket punching a giant monster in the face. People that watch action movies want to see people get punched in an awesome way, not thrown over a railing for example. This can really make or break these kinds of movies for certain viewers.

The students all possessed basic knowledge of database modelling techniques, and conducted

³ <http://www.imdb.com>

this task as part of their database education curriculum. Students were teamed into pairs, and performed this modelling experiment collaboratively to foster discussions and enforce at least a minimal degree of consensus on the relevance of attributes between the two team members. An example result created by one of the teams is shown in Table 1. That group claims that “acting” (as in “the quality of acting”) is the most important attribute while “storyline” or “scenes” are considered slightly less important.

The modelling teams provided between 4 and 10 attributes each, with an average of 6.5 attributes per team. An overview of all modelled attributes is shown in table 2. We manually grouped the participants’ responses (e.g., “quality of story”, “plot”, or “story” are all grouped into “storyline”). 90 teams participated in the experiment, and two attributes received a high degree of consensus: 74 teams mentioned “storyline”, and 72 mentioned “acting”. Several other attributes like “scenery”, “character”, or “directing” are only mentioned by roughly half of the teams, and several attributes were only mentioned once (like “Disney’ishness” describing how a much a movie feels like a Disney movie; or how friendly the movie seems to portray animals.)

As part of the experiment’s post evaluation, several students complained that it was not intuitive to model perceptual attributes. This also shows in the modelled attributes themselves: beyond attributes like acting and storyline, there is little consensus between the 90 data models created in the experiment, and for many modelled attributes it is debatable what they mean and how important they really are. Furthermore, importance of many attributes seems to be quite subjective. Thus, we conclude that explicitly modelling perceptual attributes is often unfeasible. Furthermore, even if such attributes are modelled, obtaining the actual attribute values for all modelled attributes and items in a database is far from trivial (e.g., [YLT17] uses aspect-oriented review mining for this with somewhat limited success). Thus, for the remainder of this work, we opt for fully automatically generated *latent* perceptual attributes as discussed in the next subsection.

- Perceptual attributes should be considered when describing experience items as they better capture how users see or query for items than traditional structured attributes.
- Perceptual attributes are hard to model explicitly as they are often fuzzy and subjective

2.2 Latent Representation of Perceptual Properties from Reviews

Due to the challenges regarding explicit modelling and mining of perceptual attributes, we investigate latent representations. Latent representations can be mined fully automatically from user judgments like reviews or ratings. However, while these attributes might have a real-world interpretation, that interpretation is typically unknown to us (for example, one attribute might represent how scary a movie is, but this attribute will simply have a generic name and we do not know that it indeed refers to scariness). We consider this an acceptable price to pay for the convenience of obtaining both the data model and item attribute

Tab. 2: Modelling Experiment Most and Least Important Attributes

Aspect	Sum Importance	Avg Importance	mentions
storyline	346	4.67	74
acting	341	4.73	72
scenery	197	4.10	48
character	186	4.53	41
directing	177	4.43	40
sound	162	2.74	59
humor	160	3.01	53
originality	67	2.48	27
pace	49	3.06	16
cinematography	46	2.70	17
child friendliness	3	3	1
babes	2	2	1
morality	2	2	1
animal friendliness	1	1	1
Disney'ishness	1	1	1

values fully automatically. A naive way of creating a latent representation is to embed each item in a high-dimensional vector space (therefore, such techniques are also sometimes called “embeddings”) with usually 100-600 automatically created dimensions where each dimension represents an (unlabeled) perceptual attribute (like scariness, funniness, quality of special effects, or even the presence of certain plot elements like “movie has slimy monsters” - but again, while the attributes likely do represent a real world aspect, the actual meaning is unknown to us).

Even without explicitly labeling the attributes resulting from embeddings, latent representations can already provide tremendous benefits with respect to the user experience: They can directly be used by most data analytic algorithms like clustering, supervised labeling, or regressions. Also, from a user’s perspective, such representations can be used with great effect to allow for semantic example-based navigation queries as we have shown in [LN14] for movies, or be used to increase the ease-of-consumption of review texts [YLT17].

To obtain such latent representations for e.g., movies, early approaches like [SLB12, LN14] relied on decomposing user-item-rating matrices using techniques not too unlike those also used in recommender systems [Sa01]. As the needed large user-rating corpora are hard to obtain nowadays due to privacy concerns, our later work [LW15] relied on the openly available reviews. Here, a very simple heuristic was used: *Similar movies should feature similar reviews, thus by embedding all user reviews into a latent space (by using e.g., techniques like Latent Semantic Analysis [Li12] or Document Embeddings [LB16]), and aggregating them into a single tuple with latent attributes, an effective representation*

of experience items can be created which can be used for example-based and similarity queries. However, this heuristic showed several shortcomings as discussed next.

- Perceptual attributes can be automatically extracted in an implicit latent form, e.g., using review or rating mining techniques
- Latent perceptual attributes have no clear explicit semantics, but can still be effectively used for navigational query-by-example queries

2.3 Towards Shared Perspectives

In [LW15], we relied on Amazon movie reviews to build representations of latent perceptual properties of movies. While we could show that the resulting attributes showed mostly comparable performance when used for a query-by-example system compared to a rating-based latent representation, we encountered several cases of unexpected behaviors. Thus, we manually inspected a selection of movies and their supposedly most similar titles. Here, it turned out that there are indeed many good matches in the similarity list. However, there are also some titles which are highly similar with respect to the latent perceptual attributes even though the movies themselves are very different (e.g., for the movie “Terminator 2”, both “Robocop” (a good match) and “Dream Girls Private Screenings” (a surprisingly bad match) are both considered similar). The reason for this irritating behavior seems to be that there are many “bad” reviews. “Bad” reviews are not discussing the movie itself, but other issues and do therefore not contribute to a meaningful representation. Typical examples are “I had to wait 5 weeks for delivery of the item! Stupid Amazon!”, “Srsly?! Package damaged on delivery?”, “I ordered the DVD version, got the Blue Ray!”. For “Dream Girls”, reviewers seem to be mostly concerned with the bad quality of the DVD version in comparison to the older VHS release. A similar issue is described in several reviews of the original DVD release of Terminator 2. Such reviews should therefore not be considered when building latent perceptual representations, however it is not a trivial process, since cleaning noisy web-data is a live line of research.

A second issue became apparent much later: for many movies, reviews often show very polarizing viewpoints, and aggregating them into a single tuple represents neither point of view well. Consider for example the teen vampire romance movie “Twilight” with the opinions “This is the worst movie of all times, with cheesy characters, dump story line, and really bad sparkling vampires” and “This movie is the best movie ever made, so romantic and beautiful, the love between the lead characters is so enjoyable.” (Note that most review for this movie follow either the first or second opinions - only few people believe that it is just an average movie...) By combining the (high-dimensional numeric vector) embeddings of such two highly polarizing opinions, the resulting combined embedding would likely not be useful. Therefore, it is essential to store strong opinions independently to each other and be considerate when aggregating judgments of users on perceptual properties.

However, we argue that while perceptual properties expressed by users are highly subjective, they are not necessarily unique and there is often a smaller set of shared point of views (or perspectives) when perceiving items: For example, out of 400 reviews for the drama “The Green Mile” in our Amazon dataset, essentially 180 express some variation of “this is a beautiful and touching movie, full of emotion” - while each review might be using slightly different words, the *perspective* is the same. Another 110 agree that the movie has “great acting and a good story that follows the book quite well”, and 80 reviews claim variants of “it is a good movie with famous actors, it is long but worth it”. The remaining reviews usually express some isolated fringe opinions, like “it is an unrealistic fairy tale in prison”, or “such a bad movie, Tom Hanks is so lame”, or indeed “I ordered this movie and the package was damaged” —opinions which are typically not shared by many others.

Therefore, we propose to group similar user judgments expressing perceptual properties into *shared perspectives*, and aggregate each shared perspective into one single latent representation. Thus, shared perspectives retain major potentially conflicting and/or polarizing opinions while still aggregating the underlying social judgments to minimize storage space and query complexity. Then we retain only major shared perspectives and discard lesser ones. Considering the “The Green Mile” example, this movie would then be represented by its traditional meta-data like title, actors, release year, etc., and in addition with three latent perceptual tuples representing the three major shared perspectives. This allows to perform similarity navigation from “The Green Mile” to other movies which are perceived as equally emotionally touching, or have equally good acting/plot following a novel. This is illustrated in figure 1: instead of traditional QBE which only relies on a single, hidden-to-the-user, item similarity, the users can choose to navigate along a shared perspective to discover new items *which are similar with respect to that perspective*. Also, several perspectives can be combined during query processing: “Twilight”, a teen vampire romance, is characterized by having two very distinct and conflicting shared perspectives (see above), one deeply embracing the movie for its romantic plot and great characters, the other perspective hating the movie for its plot and characters. A similar situation can be found for “Warm Bodies”, a teen zombie romance, which features nearly the same perceptions and perspectives.

- User perception are often highly opinionated, polarizing, or even contradicting. This needs to be respected when representing items, and a single perceptual representation of each item is usually insufficient.
- Still, there is often a smaller number of dominant perception which is consensual shared by larger user groups. Focusing on these *shared perspectives* is a good compromise for efficiently representing experience items while still respecting polarized opinions.

3 Shared Perspectives

This is the core section of this work, where the foundation and implementation of Shared Perspectives is described.

Figure 1: Query-by-Example Step using Shared Perspectives



3.1 Foundation and Terminology

In the following, we introduce new concepts relevant for representing shared perspectives. A database system wishing to use shared perspectives to represent a type of real-world items $I = \{I_1, I_2, \dots, I_{n_I}\}$ has a set of *factual attributes* $A_F = \{A_1, \dots, A_{n_{A_F}}\}$. These attributes A_F represent traditional relational attributes, and we use the term *factual* (as opposed to *perceptual*) to describe that there is at most one value for each item and attribute, and that the attributes have been agreed upon during the schema design phase. Each item $x \in I$ can be represented by a factual tuple $x_F \in D_F$, stored in the factual relation $R_F \subseteq D_F = A_1 \times \dots \times A_{n_{A_F}}$ using those factual attributes.

In addition, there is also the set of perceptual attributes $A_P = \{P_1, \dots, P_{n_P}\}$ with $D_P = P_1 \times \dots \times P_{n_P}$. In this paper, we showcase a system using latent perceptual features, thus the set A_P is not chosen by the schema designer, but is typically the result of some sort of algorithm *miner* automatically mining user judgments like reviews or ratings. For example, a document embedding algorithm will typically produce 100 to 300 such attributes. For each item x , there is in addition to the factual tuple x_F also a *set of shared perspectives* $x_{SP} = \{x_{SP_1}, \dots, x_{SP_{n_{x_{SP}}}}\} \subseteq D_P$. Finally, the item x is represented by (x_F, x_{SP}) . Such a tuple is not in the first normal form (as x_{SP} is a set). Therefore, a real-life application using traditional relational databases without special extensions would typically realize this by normalizing and having several relations using joins to build (x_F, x_{SP}) on the application side.

In order to obtain the shared perspectives for an item $x \in I$, each item x also features several user judgments $UJ_x \subseteq UJ$ like reviews, ratings, discussions, or just explicitly provided

user feedback. The aforementioned mining algorithm *miner* transforms such a judgment into perceptual tuples, i.e. $miner : UJ \rightarrow D_P$. By applying *miner* to each judgment in UJ_x , the set of all perceptual tuples $x_P \subseteq D_P$ is created, i.e. when a movie has 1000 reviews from different users, x_P will contain a perceptual tuple for each review. In our implementation, we do not store x_P , but only use it to discover the shared perspectives x_{SP} . Discovering shared perspectives is realized by the function *g&a* (group and aggregate), $g\&a : P(D_P) \rightarrow P(D_P)$. This function will group all perceptual tuples, and aggregate the bigger groups into a single tuple in order to produce the shared perspectives, i.e. $x_{SP} = g\&a(x_P)$.

We discuss the design space for implementing the functions *miner* and *g&a* in the following sections.

- Experience Items I can have factual A_F and perceptual attributes A_P . Perceptual attributes of each item x can be obtained from user judgments $UJ_x \subseteq UJ$ using a *miner* grouped into shared perspectives x_{SP} with an appropriate algorithm $x_{SP} = g\&a(x_P)$.
- The shared perspectives of an item $x_{SP} = \{x_{SP_1}, \dots, x_{SP_{n_{x_{SP}}}}\} \subseteq D_P$ can be compared to other shared perspectives of an item to find similar perspectives, and therefore similar items.

3.2 Query-By-Example using Shared Perspectives

Shared perspectives, especially the implementation chosen in this work with latent attributes, are hard to use with traditional SQL-style querying. Thus, we propose to use a variant of iterative query-by-example for allowing user to interact with the item space. This type of querying sits between SQL-style querying (where users have to specify exactly what they are looking for), and recommendations (where users specify little to nothing, and the system actively recommends). In [LN14], it has been shown that this type of querying works well with users who only have a vague idea of the items they want, and QBE querying was deemed an enjoyable and playful experience by the users in that study.

A core concept in querying is semantic similarity *sim* between two perceptual tuples, i.e. $sim : D_P \times D_P \rightarrow [0..1]$. While there are several approaches towards implementing this, we chose with a simple cosine-distance-based similarity.

The shared-perspective-enabled QBE process can be summarized as follows (also see figure 2): The user starts with an example item she likes. Then, for each shared perspective that item has, the system will discover up to n other items which have a shared perspective which is most similar to the current one (n is 3 in our system). Each of these items (n items per shared perspective) are then shown to the user, and the user can select from that display the item she likes most. Then, the process can be repeated until the user is satisfied (see algorithm 1).

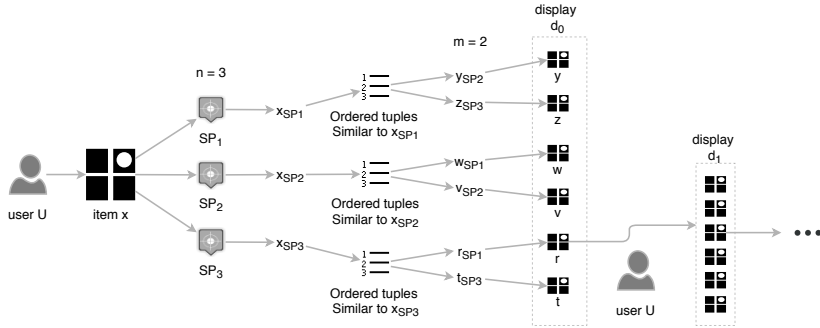
Algorithm 1 Query on SP

```

1: procedure X ITEM QUERY BY EXAMPLE
2:    $x \leftarrow UserProvidedExampleItem \in I$ 
3:   while user wants to continue do
4:      $display \leftarrow \emptyset$ 
5:     for each  $sp \in x_{SP}$  do
6:        $maxItemSim \leftarrow newRelation(item, similarity)$ 
7:       for each  $(y \in I) \wedge (y \notin display)$  do
8:          $maxSim \leftarrow \max(sim(sp, y_{SP}))$ 
9:          $maxItemSim.add(y, maxSim)$ 
10:       $display.add(top_n(maxItemSim))$ 
11:     $show(display)$ 
12:     $x \leftarrow UserSelectedItem \in display$ 
13:  end while

```

Figure 2: Query-by-Example: Shared Perspectives and Displays



- Each item has multiple shared perspectives using perceptual attributes. Shared perspectives are created by representing user judgments as perceptual tuples which are then grouped and aggregated by group.
- For querying, a variant of Query-By-Example can be used. After selecting a start example, we show the user several items which are similar with respect to the different shared perspective of the start item. Then, the user can select a new item she likes and the process starts anew until the user is satisfied.

4 Implementation Design Space

In this section, we outline some areas of the design space for implementing shared perspectives into an information system, and explain the choices of our prototype system.

4.1 Dataset

Underlying our prototype is an existing dataset of movie reviews crawled from Amazon [Mc15]. We only considered movies that have at least 100 reviews, and use maximum 300 (randomly selected) reviews per movie. We also discard reviews with less than 25 words as they are commonly uninformative. This results in a dataset consisting of around 375K movie reviews, for 2,041 movies overall.

4.2 Extracting Perceptual Attributes

In this section, we discuss the design choices for implementing the aforementioned function *miner* which translates reviews into perceptual attributes. In a good review, a user will take the time to briefly summarize the content of an item, and then expresses her feelings and opinions towards it - the task for *miner* is to represent these opinions in such a way that the similarity measures used by the QBE process work well.

In an earlier prototype we focused on explicitly modelling perceptual attributes and using aspect-oriented opinion mining to extract perceptual tuples [YLT17]. However, this had a considerable manual overhead and did result into only few usable attributes which could be extracted reliably. Thus, in this work, we focus on extracting a fixed number of latent perceptual attributes as in [LW15]. In [LW15] we discussed the advantages of different implementation techniques like LSA [DL05] or LDA [BNJ03]. However, in the last few years there has been a surge of approaches proposing to represent the semantics of text documents using neural language models. A prominent example is word2vec, representing the semantics of words as a fixed-length vector [Mi13], and the later version doc2vec which represents documents as vectors [LM14]. Studies in [LB16] suggest that the semantic performance of doc2vec in typical document tasks like clustering or retrieval seems to be quite good, outperforming other approaches based on bag-of-words, word vectors, or LSA/LDA. This has also been shown for the special case of reviews [LM14].

In our prototype, we used the document embedding implementation of doc2vec provided by Gensim ⁴, using distributed bag-of-words representation (dbow). Typically, such document embeddings need to be trained on a large corpus before they can be used. In scenarios where only a smaller amount of text is available, the solution is often to rely on models pre-trained on Wikipedia or Google News. However, as we had a large review corpus available [Mc15], we trained on all Amazon reviews without excluding anything.

The parameters for training and the embeddings were:

- Vectors are kept at 100 dimensions. Typically, 100 to 300 dimensions is considered good for document similarity tasks [LB16].

⁴ <https://github.com/piskvorky/gensim/>

- The training window is 10 since it showed good performance with documents of similar size (reviews)
- Frequent word subsampling seems to decrease sentiment-prediction accuracy, so it is not applied
- We do not consider any words which are mentioned only once in the whole corpus
- The learning rate is $\alpha = 0.025$ and is kept fixed

Example: *The Green Mile* review

"Ok, so it did not deserve best picture. It was still excellent. It has great performances in it. Particularly the guy who never was very famous Michael Jeter or whatever his name is. I love the visuals. I cried at the end. Michael Clarke Duncan is great."

→ Perceptual Tuple: (-0.640138, 0.422624, ..., -0.0350407, 0.192102)

- The *miner* selected for this work is doc2vec, applied to amazon movie reviews, with 100 dimensions.
- Hyper-parameter values were chosen in line with the recommendations of the authors of doc2vec and other implementations working with similar documents.

4.3 Aggregating Shared Perspectives

As part of the creation process of shared perspectives, individual perceptual tuples representing single reviews need to be aggregated and summarized to implement the function $g\&a$. We opted for spherical k-medoids clustering to group perceptual tuples for an item [DM01]. Alternatively, we also considered HDBSCAN, but this did not notably improve the results [Ca15]. The 'elbow method' was applied to select the best number of clusters. After clustering, we only retain up to 3 clusters. This is a simplifying design choice to keep the user interface easy and accessible. However, a quick inspection showed that most reviews would only show 2-3 clusters anyway (typically, this is reviews from people who hate the movie, reviews from people who love it, and balanced views). Thus this limitation will only sacrifice semantics in case of very diverse opinions. The shared perspective tuple is representing a cluster is chosen using the medoids. We prefer this solution over k-means, which would create an artificial shared perspective tuple which does not relate to a real user review.

- Spherical k-clustering was implemented as the choice of aggregating or $g\&a$ for this work.
- $k=3$ was decided since manual inspection of the 'elbow method' for several movies showed that most reviews would only show 2-3 clusters, plus it simplifies design and user interface for the evaluation in Section 5.3.

5 Evaluation

We evaluated our system in two ways: the first one is a query simulation akin to those performed in [LN14]. Here, the core idea is that we assume the existence of a hypothetical “target object” the user is searching for, and simulate the user interactions leading to that item. While this simulation is artificial and does not resemble a real-life user interaction, it still gives insights into the effectiveness of some design decisions. The second evaluation is using real users, and having them interact with the system to evaluate the perceived usefulness and semantic quality of the shown shared perspectives.

5.1 Evaluation: QBE User Simulation

The general effectiveness of QBE using perceptual attributes without shared perspectives has been shown in [LN14, LW15]. Therefore, in this section we focus on the effect the introduction of shared perspectives has on the QBE process. As a baseline, we force our system to only consider a single perspective (i.e., use the medoid of all perceptual tuples resulting from reviews as the only shared perspective). This resembles the setup in [LW15], which uses only a single tuple to represent a movie. We compare this to a version of the system in which we consider up to 3 shared perspectives.

The artificial evaluation scenario is as follows: choose a random start example movie, and choose a random target movie. Then perform query-by-example iterations choosing always the displayed movie which is closest to the target. As an evaluation metric, we count the number of iterations necessary to traverse from the start example to the target. We assume that using shared perspectives, fewer iterations are needed as the display selection has a wider semantic spread than when using only a single perspective, since SPs can help to ‘get out’ of dense similarity neighborhoods. For instance from “The Green Mile” to “Tinkerbell”, there is a perspective that relates them: “The Green Mile” has a “good story that follows the book” perspective, which leads to a display that includes “Matilda”, which has the perspective “beautiful family movie with a message”, and the next display contains “Tinkerbell”.

Note that this evaluation is very artificial from a semantic point of view: shared perspectives with QBE are designed to help users who have a vague idea of the style/type/general flavor of item they are looking for. They will not have a particular item in mind (if they had, they could simply retrieve it using SQL). Thus, we assume that users will choose a starting example which is in the proximity of their unclear target, and then clarify their preferences during the query process (e.g., “I know what I am looking for as soon as I see it.”)

The basic steps for this evaluation are:

1. For the currently selected movie x , the system generates a new display with 9 movies as described in section 3.2. When using shared perspectives, it is the 3 most similar

movies with respect to the 3 perspectives of x . This is, the 3 movies with a shared perspective tuple with the least cosine distance to x_{SP_1} , 3 more for x_{SP_2} , and 3 for x_{SP_3} . When not using shared perspectives, it is simply the 9 movies with least cosine distance to the single tuple x .

2. If target movie is in display, finish. If not, select the best option (i.e., movie which is most similar to the target) as new example movie from display
3. Repeat

Results: The average steps it takes from start to target movie for 175 different pairs of movies is 38 when using only a single representation, and 28.03 when using shared perspectives. Thus, shared perspectives reduce the average number of required interaction steps by roughly 30% (again, note, that in a real use case, there will be significantly fewer steps as start movies are chosen closer to the implicit “target”). The frequency distribution of the number of steps for this experiment is shown in Figure 3. This graph shows that by using SPs, around 80% of pairs reach the target movie in less than 50 steps, compared to 65% with SR. While this is an improvement, there is however also number of movie pairs for which the simulation takes more than 175 - something that does not happen when not using shared perspectives. This odd behavior will be investigated later in section 5.3.

Please note that these numbers seem high when compared to the best results reported in [LN14, LW15], where the results were between 10 and 17 steps. In those works, we employed an additional Bayesian probability user model which did not solely rely on similarity, but also provided shortcuts to the movie which is the predicted target of the user based on that model. Similar techniques could also be applied to shared perspectives. However, like mentioned in those works, this is beneficial when the goal is to reach a predefined target, which in reality is not known to user or system. Therefore in a real application, where there is no such target, displaying items that are highly informative for the system, like showing Terminator as similar to Finding Nemo, would be confusing to the user.

- For evaluation, we propose a simulation with a starting and target movie to mimic the behavior of a user.
- We compare the performance of a single representation of a movie against the multiple shared perspectives.

5.2 User Evaluation

The purpose of this user study is to obtain some insights on the quality of shared perspectives for querying from a user’s point of view. We do not consider this an exhaustive quantitative analysis, but rather a quicker exploration to obtain an intuition on the quality of the approach. Especially, we are interested in how useful different perspectives are perceived, and what

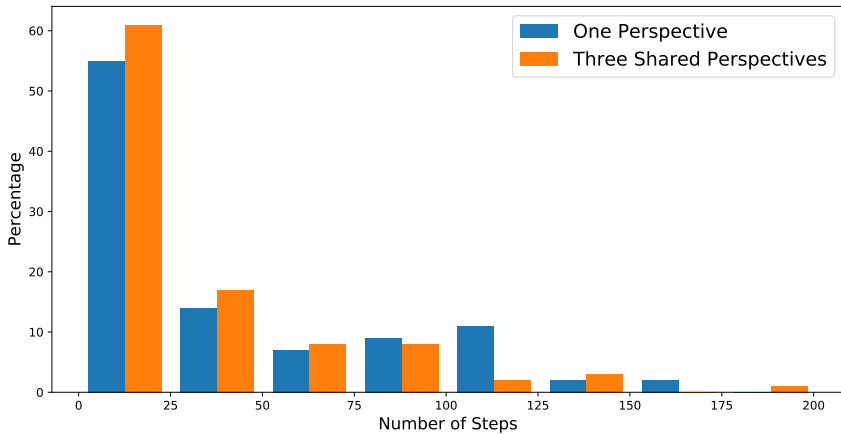


Figure 3: Histogram of Steps with 175 pairs

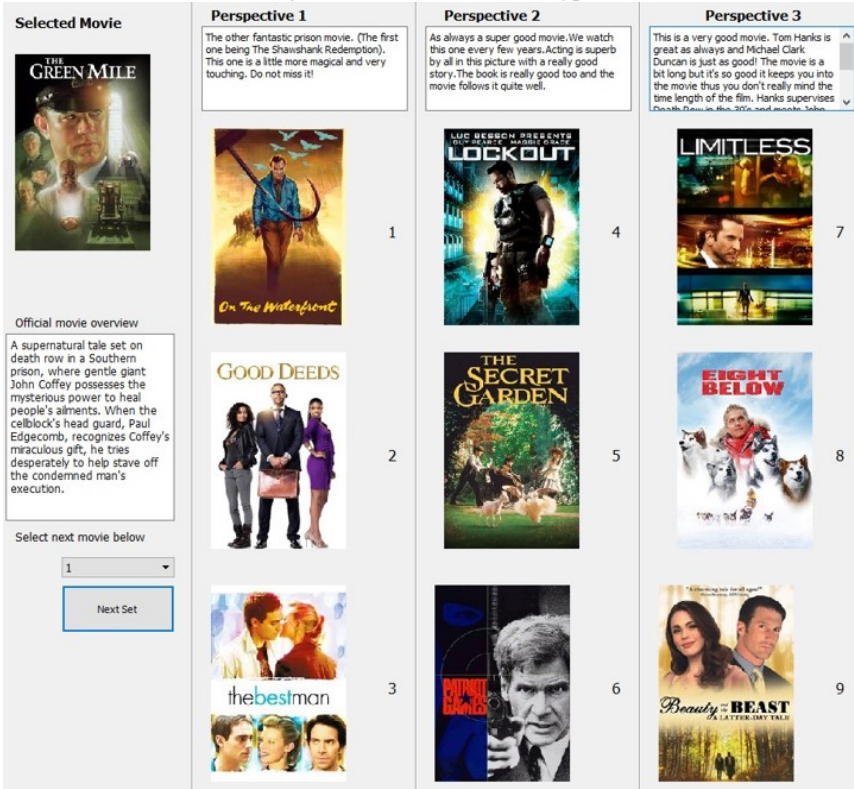
their semantics could be (remember: shared perspectives result from semantically clustering reviews. Thus the question is: What is the commonality of the reviews which contributed to a perspective? Is it a meaningful semantic?).

To this end, we asked seven participants to interact with the system and rate how useful a perspective is based on the constructed QBE display. The participants include people from 18 to 55 years old, from different countries and education levels. The setup of the experiment is the following:

- We sent a standalone desktop application to the study’s participants
- The application implemented the QBE workflow as outlines in section 3.2, with a display of 9 movies (3 for each shared perspective) in each iterations step.
- We showed the users the text of the medoid review which represents a perspective (see figure 4 for a user interface example)
- Instructions for the participants were to rate for each perspective how well the shown review snippet represents the selected movies on a 3-point scale: well represented (score 2), somewhat represented (1), not represented well (0).

All 7 participants evaluated the same set of 7 displays. For each shared perspective thus evaluated, we computed a “usefulness score” based on the user feedback. Here, some perspectives got higher ratings, like for “Real Steel” which had a perspective connecting to “The Last Starfighter” which can be summarized as “awesome sci-fi for the whole family”, or “My Dog Skip” related to “P.S. I Love You”, “The Good Life of Timothy Green” and “Bee Movie” because “My wife and I loved this movie. A heart warming story” with a usefulness score of 1.5. However, even when looking at this small-scale study involving only

Figure 4: User Interface Prototype



few users and example displays, some problems became apparent: “The Good, The Bad and The Ugly” relates to “The Towering Inferno”, “Shane” and “The Uninvited” because “The digital restoration looks really great”. This perspective was not perceived as useful with a score 0.5. This problem was already encountered in [LW15]. However, in that work, such reviews affected the whole item representation. However, when using shared perspectives, it usually results in only one bad perspective, while typically one or two useful ones remain. As such, this problem is less severe for shared perspectives than it was for [LW15], but still this is an unsatisfying result.

As an additional exploration, we provided users with the option to label shared perspectives with a few keywords. Even though we obtained only a smaller number of labels this way, some of them were quite descriptive: all participant for example noted that the second perspective for “The Jungle Book” represents “Disney classics”, or that the first perspective for “The Good, the Bad and the Ugly” should be something along the lines of “great western with very good actors”. In contrast, the first perspective in “The Green Mile” received labels along the lines of “good acting”, or “good story” with no clear consensus. Other

perspectives received no labels at all. In our future works, we will extend this preliminary inspection with a larger scale user study to provide reliable insights into the semantics of shared perspectives. For now, we remain confident that while not all perspective have a discernible meaning, many actually have one.

- In addition to the simulation for evaluation, a user-study is conducted to rate the usefulness of shared perspectives to find similar movies.
- The usefulness is for each shared perspective, and can be defined as: how good is $x_S P_1$ to say that movie x is related to movie w because of $w_S P_2$, to movie y because of $y_S P_1$ and z because of $z_S P_1$.

5.3 Predicting Usefulness Scores

Motivated by the results in the last section, we tried to look into the problem of not useful shared perspectives. The goal of this to learn from user feedback, and predict for each perspective a usefulness score.

In a production system, this could be a continuous process where users are given the option to down-vote a perspective as not useful, and the system is considering this feedback when computing usefulness scores. Then, perspectives could be presented ordered by their usefulness, or skipped all together if deemed very unuseful. Also, the calculation of all similarities could be modified by usefulness.

We manually analyzed all shared perspectives involved in the last experiment. For those, non-useful perspectives are closely similar to perspectives of a large number of movies. On the other hand, useful perspectives are only similar to perspectives of a smaller number of movies. This can be justified by the following intuition: useful perspectives should be specific to that movie, and only few other movies should share that perspective, like being a “heart-warming children movie” or being a “Disney classic”. If a perspective (i.e., a particular topic featured in reviews) is present in a large number of movies, it is likely not useful. This would for example be the case for “My packaging was damaged” or “Image quality is grainy due to HD upscale” - such sentiments can afflict any movie no matter it’s actual qualities or content.

To capture this intuition, we calculate the Pearson Median Skewness (PMS) score, also called Second Skewness Coefficient of a given shared perspective tuple to all other shared perspective tuples in the database, calculating a usefulness score for that perspective based on the difference between mean and median of PMS.

We extended the simulation experiment from section 5.1 by adjusting the similarity calculation with usefulness scores, thus de-emphasizing unuseful similarities. This improved the number of simulation steps from 38 when using only a single representation, to 28.03 when simply using shared perspectives, to 26.05 when using shared perspectives weighted

by usefulness. More importantly, while this average improvement is quite small, extreme outlier cases saw improvement: e.g., one pair of movies which took 82 steps now only takes 41; and others went from 205 to 170.

This result motivates us to investigate this approach more thoroughly in our future works, improving the way how we predict usefulness in a larger setup with more users and feedback. Especially, the calculation of usefulness scores should relate to actual user feedback instead of being based on intuitions as discussed in this section. In addition, modelling and calculating the score of perspectives can be used to remove noisy reviews, a current problem with applications of web-data.

- Leveraging the usefulness scores obtained before, a function of usefulness was modelled to then calculate the scores of all shared perspectives in the corpus.
- The scores are then evaluated in a simulation similar to Section 5.1 with promising results.

6 Conclusions

In this paper, we discussed the challenge of representing experience items like movies, books, or music. For such items, perceptual attributes should be modelled, which is an inherently difficult task during the design phase but also later when items need to be described with respect to these properties. To underline this claim, we presented a brief study with 180 students who were asked to model perceptual properties for movies. This task was shown to be hard, and the results between participants were not very consistent.

Therefore, in this paper we proposed the use of latent perceptual attributes which are automatically mined from user judgments like ratings of reviews. While such latent attributes have no explicit human-understandable semantics, they can be effectively used for query-by-example navigational queries. However, previous works revealed a shortcoming with this approach: typically, each item is represented using a single tuple. Especially when highly opinionated and polarized social judgments are used to generate the latent perceptual attributes, performance suffers as the resulting tuple represents neither viewpoint well. To rectify this, we introduced the concept of *shared perspectives*, perceptual tuples representing a dominant consensual point of view for an item. Each database item can have several shared perspectives, thus striking a balance between aggregating social judgments for storage and query efficiency, and still retaining conflicting opinions for better semantic representatives and querying.

We discussed our prototype implementation for a query-by-example system exploiting shared perspectives, and showed evaluations with synthetic queries but also a human user study. Based on the feedback gathered during evaluations, we suggested an improvement of our approach which takes the semantic usefulness of a perspective into account.

For future improvements, we aim at applying our approach on real-live problems in additional domains, as for example on large music repositories. Furthermore, the QBE query processing process can benefit from additional tuning by for example combining also the objective structured meta-data into the similarity measures, and also including additional user modelling to allow users to discover desired items even quicker (as e.g. in [LN14]). Also, a large-scale user study into the semantics of shared perspectives is planned.

References

- [BKV10] Bell, Robert M.; Koren, Yehuda; Volinsky, Chris: All together now: A perspective on the Netflix Price. *CHANCE*, 23(1):24–24, apr 2010.
- [BNJ03] Blei, David M; Ng, Andrew Y; Jordan, Michael I: Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.
- [Ca15] Campello, Ricardo J. G. B.; Moulavi, Davoud; Zimek, Arthur; Sander, Jörg: Hierarchical Density Estimates for Data Clustering, Visualization, and Outlier Detection. *ACM Transactions on Knowledge Discovery from Data*, 10(1):1–51, 2015.
- [DL05] Dumais, Susan; Landauer, Thomas: Latent semantic analysis. *Annual Review of Information Science and Technology*, 38(1):188–230, 2005.
- [DM01] Dhillon, Inderjit S.; Modha, Dharmendra S.: Concept Decompositions for Large Sparse Text Data Using Clustering. *Machine Learning*, 42(1):143–175, Jan 2001.
- [KB11] Koren, Yehuda; Bell, Robert: Advances in Collaborative Filtering. In: *Recommender Systems Handbook*, S. 145–186. 2011.
- [LB16] Lau, Jey Han; Baldwin, Timothy: An Empirical Evaluation of doc2vec with Practical Insights into Document Embedding Generation. *CoRR*, abs/1607.0:78–86, 2016.
- [Li12] Liu, Chien Liang; Hsiao, Wen Hoar; Lee, Chia Hoang; Lu, Gen Chi; Jou, Emery: Movie rating and review summarization in mobile environment. *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, 42(3):397–407, 2012.
- [LM14] Le, Quoc V.; Mikolov, Tomas: Distributed Representations of Sentences and Documents. In: *31st International Conference on Machine Learning*. Jgg. 32, Beijing, China, S. 1188–1196, 2014.
- [LN14] Lofi, Christoph; Nieke, Christian: Exploiting Perceptual Similarity: Privacy-Preserving Cooperative Query Personalization. In: *Web Information Systems Engineering – WISE 2014*. Springer International Publishing, Cham, S. 340–356, 2014.
- [LSY03] Linden, G.; Smith, B.; York, J.: Amazon.com recommendations: item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, jan 2003.
- [LW15] Lofi, Christoph; Wille, Philipp: Exploiting social judgements in big data analytics. *CEUR Workshop Proceedings*, 1458:444–455, 2015.
- [Mc15] McAuley, J.; Targett, C.; Shi, J.; van den Hengel, A.: Image-based recommendations on styles and substitutes. In: *ACM SIGIR Conf. on Research and Development in Information Retrieval (SIGIR)*. Santiago de Chile, Chile, 2015.

- [Mi13] Mikolov, Tomas; Sutskever, Ilya; Chen, Kai; Corrado, Greg S.; Dean, Jeff: Distributed Representations of Words and Phrases and their Compositionality. *Advances in Neural Information Processing Systems*, 21:3111–3119, 2013.
- [MTT17] Matakos, Antonis; Terzi, Evimaria; Tsaparas, Panayiotis: Measuring and moderating opinion polarization in social networks. *Data Mining and Knowledge Discovery*, 31(5):1480–1505, 2017.
- [Sa01] Sarwar, Badrul; Karypis, George; Konstan, Joseph; Reidl, John: Item-based collaborative filtering recommendation algorithms. *Proceedings of the tenth international conference on World Wide Web - WWW '01*, S. 285–295, 2001.
- [SLB12] Selke, Joachim; Lofi, Christoph; Balke, Wolf-Tilo: Pushing the boundaries of crowd-enabled databases with query-driven schema expansion. *Proceedings of the VLDB Endowment*, 5(6):538–549, 2012.
- [TNK10] Thet, Tun Thura; Na, Jin-Cheon; Khoo, Christopher SG: Aspect-based sentiment analysis of movie reviews on discussion boards. *Journal of information science*, 36(6):823–848, 2010.
- [YLT17] Ye, Mengmeng; Lofi, Christoph; Tintarev, Nava: Memorability of Semantically Grouped Online Reviews. In: *Semantics 2017*. Amsterdam, Netherlands, sep 2017.

Graphs

Graph Data Transformations in Gradoop

Matthias Kricke,¹ Eric Peukert,¹ Erhard Rahm¹

Abstract: The analysis of graph data using graph database and distributed graph processing systems has gained significant interest. However, relatively little effort has been devoted to preparing the graph data for analysis, in particular to transform and integrate data from different sources. To support such ETL processes for graph data we investigate transformation operations for property graphs managed by the distributed platform Gradoop. We also provide initial results of a runtime evaluation of the proposed graph data transformations.

Keywords: Graph analytics; Big Data; Graph transformations; Data integration

1 Introduction

The flexible and scalable analysis of large amounts of graph data has gained significant interest in the last decade and is supported by graph database systems (e.g., Neo4j), graph extensions in relational DBMS and a growing number of distributed platforms including those based on Apache Spark and Flink like GraphX, Gelly or Gradoop [Ju17a]. A largely neglected topic, however, is the support for ETL-like operations to prepare the graph data for analysis which requires to transform data sources into the supported graph format, to consolidate different graphs and to integrate them into a combined graph. As in traditional analysis platforms these steps can be highly complex and easily require the majority of time for graph analytics.

We have begun to investigate ETL and data integration for graph data for (extended) property graphs managed by the distributed open-source graph processing platform Gradoop [Ju16, Ju18]. Gradoop provides already different connectors to import data from relational databases or CSV files into property graphs. Furthermore, we provide initial match approaches within the FAMER system [SPR17, SPR18] to match and cluster graph vertices derived from multiple data sources. In this paper, we propose additional Gradoop operations to transform graphs to facilitate their integration with other graphs or to make them better suitable for analysis. For example, a bibliographic network with publications and their authors might have to be transformed for an easier analysis of co-authorships, e.g., by generating a graph with author vertices and co-authorship edges only. The proposed

¹ University of Leipzig, ScaDS Dresden Leipzig, Augustusplatz 10, 04109 Leipzig, Germany, {kricke, peukert, rahm}@informatik.uni-leipzig.de

operations are not only relevant for Gradoop but should be useful for other platforms supporting property graphs.

We thus extend Gradoop with a number of generic transformation operations that can be used to define advanced graph transformations on property graphs. Each graph transformation can implicitly trigger a series of low-level graph changes (e.g. vertex/edge/property additions and deletions) that do not need to be defined by the user. Composite transformations can be expressed from basic ones and the transformations we propose are implemented with Apache Flink for parallel execution and good scalability to large graphs.

After a discussion of related work, we briefly introduce Gradoop and outline the overall data integration process. We then describe and illustrate the new operations for graph data transformation. In Section 5, we present initial results of a runtime evaluation before we conclude.

2 Related Work

Graph preparation and transformation have received little attention in research so far especially with reference to integration and analysis of property graphs. On the other hand, there have been some algebraic, declarative and imperative approaches for graph transformation some of which have been considered for graph data processing. Algebraic approaches have been used for model transformation in software engineering [Lö93, Eh97] and more recently for the parallel execution with the vertex-centric processing model of Pregel [KTG14, Ar10, Ma10] or Map Reduce [Be15].

Declarative approaches rely on a declarative language like Cypher and Sparql to query, construct and transform graphs. Cypher and Sparql provide only limited support for graph transformation with its RETURN statement (Cypher) and CONSTRUCT statement (Sparql) but new language proposals like Open Cypher [Gr18] and G-Core [An18] provide extensions to express graph grouping and aggregations. BigGra [TH17] translates an SQL-like query language called UnQL+ to the Pregel processing model in GraphX. In this approach navigational queries and transformations are expressed as so-called structural recursions which seem complex to be defined by a user. Some additional proposals provide a declarative specification of graph extraction from relational databases, for example expressed with a Datalog-based language as done in GraphGen [XKD15] or Table2Graph[Le15] that is based on MapReduce.

Imperative methods provide languages for a step-wise definition of graph transformations. A well known representative is Gremlin/TinkerPop [Ro15] offering a set of low-level operators for navigation and traversal of graphs and for adding or removing vertices and edges. The GraphBuilder tool [Ja13] mainly focuses on the construction of graphs rather than their transformation based on the extraction of values from data sources. The support for the actual transformation is limited to filters. GraphGen [XD17], GraphX and Gelly[KVH18] also provide only limited support for transformation, letting the user add edges, vertices and attributes manually for each necessary transformation.

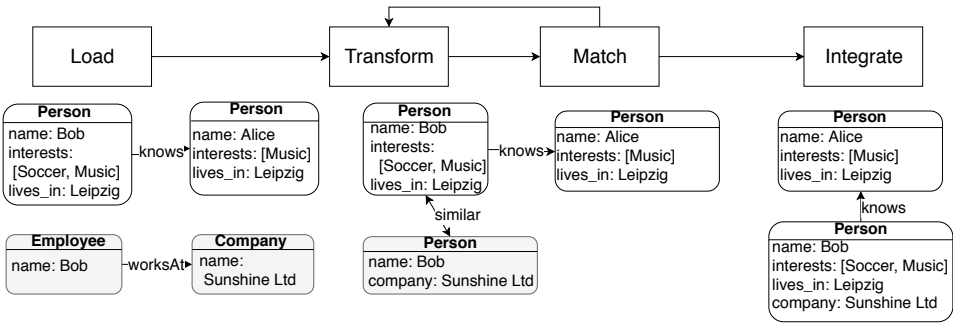


Fig. 1: Sample graph data integration pipeline combining person data from a social network and a company database.

3 Background

3.1 Graph Data Model

Gradoop [Ju16, Ju18] is based on an extended version of the *property graph model* (PGM) [RN10, RN12] which is widely used in graph database systems (e.g. Neo4j) and parallel processing systems such as Apache Spark GraphX. A property graph is a directed multigraph supporting properties and type labels for both vertices and edges. The properties are represented by key-value pairs (e.g. *name : Bob*). Properties are defined at the instance level and no schema definition is necessary. In extension to the PGM Gradoop supports storage and analysis of multiple property graphs called *logical graphs* that can also have a label and properties. Properties can be atomic (string, numeric, boolean, etc.) or collection-valued, e.g. lists. Gradoop also supports a number of generic operators on graphs and graph collections (for pattern matching, subgraph filtering, etc.) that can be used within workflows for graph analysis. The workflows can be specified in a declarative domain-specific language called GrALa. The implementation of the Gradoop operators is built on Apache Flink to achieve a parallel execution and scalability to large graphs.

3.2 Graph Data Integration

Figure 1 shows a typical data integration pipeline to integrate several sources into a graph for further analysis. Initially, already existing graphs are loaded or data from different sources such as databases or files of different formats (e.g. CSV, JSON, XML) are transformed into property graphs. The individual graphs may then have to be transformed to achieve a similar graph structuring and to facilitate further integration steps. In the example of Figure 1, we simplify the second graph by transforming the company vertices into properties of

Operator	GrALA
<i>Property To Vertex</i>	<code>graph.propertyToVertex(label, propertyName, newLabel, newPropertyName, edgeConfig, condense)</code>
<i>Vertex to Property</i>	<code>graph.propagateToNeighbor(label, edgeConfig)</code>
<i>Vertex To Edge</i>	<code>graph.vertexToEdge(vertexLabel, newEdgeLabel)</code>
<i>Edge To Vertex</i>	<code>graph.edgeToVertex(edgeLabel, newVertexLabel, edgeLabelSourceToNew, edgeLabelNewToTarget)</code>
<i>Connect Neighbors</i>	<code>graph.connectNeighbors(vertexLabel, edgeDirection, neighborVertexLabel, newEdgeLabel)</code>
<i>Invert Edge</i>	<code>graph.invertEdge(label, newLabel)</code>
<i>Cluster Fusion</i>	<code>graph.fuse(fusionConfig)</code>
Grouping	<code>graph.groupBy(vertexGroupingKeys, edgeGroupingKeys)</code>
Cypher Construct	<code>graph.query(patternQuery, constructionQuery)</code>

Tab. 1: Overview of structural graph transformation operators in Gradoop. Italic operations are new.

person vertices. To integrate the different graphs, we have to identify matching vertices and edges that need to be fused together. Given that the graphs may contain vertices and edges of many different types this is typically a complex process that is currently under investigation. What has already been implemented is the FAMER system [SPR17, SPR18] to link and cluster equivalent entities from multiple graphs, e.g., the vertices for the same person in the example. Clustered entities are fused together to create a single vertex in the integrated graph with the combined property values (e.g., for person *Bob* in the example). The integrated graph can be further transformed to support specific analytical purposes.

4 Graph Transformation

Table 1 gives an overview on the implemented transformations in Gradoop that change the graph structure. Additional simpler transformations include *property transformations* to change properties based on an UDF or basic functions like string splitting or concatenation. Due to space restrictions we only describe the first five transformations in more detail in the following. The grouping and Cypher operations have already been described in earlier work [JPR17, Ju17b]. Grouping allows us to determine structural graph aggregations with super-vertices and super-edges summarizing several vertices and edges based on common label and property conditions. Gradoop also has initial Cypher support to specify construction patterns such that the found instances for a query pattern can be transformed. We omit the description of the invert edge operation since it is simple and only inverts the edge direction plus the label of the edge. The cluster fusion operation combines several equivalent vertices into one and takes the union of different properties and combines different values for the same properties based on a specified function, e.g., to prefer the longest string or values from preferred sources (similar to fusion operations for relational data [BN09]).

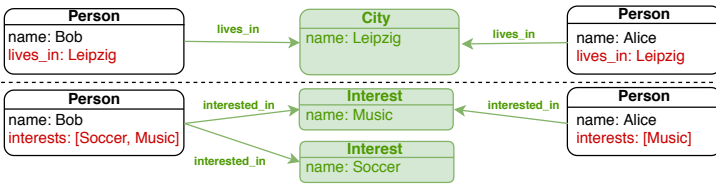


Fig. 2: PropertyToVertex examples for atomic (top) and collection (bottom) properties. Newly created/removed elements are shown in green/red, respectively.

In general, it is desirable that one can reverse the effect of graph transformations unless they lead to a reduced information such as deletes or grouping. We therefore have pairs of transformations and their inverse operations (`propertyToVertex` and `VertexToProperty`, `EdgeToVertex` and `VertexToEdge`). Furthermore we have to deal with both atomic and collection-based properties as well as with the creation / avoidance of duplicate information.

4.1 Property to Vertex & Vertex to Property

Property extraction is one of the most basic transformations and expressed in GrALA as: `graph.propertyToVertex(label, propertyName, newLabel, newPropertyName, edgeConfig, condense)`. It applies to all vertices of a given `label` and for the values of property `propertyName` it creates new vertices with label `newLabel` and property `newPropertyName`. For collection-based values a vertex is created for every value in the collection. Note that this operator is very beneficial to transform data imported from data files (e.g., in CSV format) into a property graph. In this case, one can generate a vertex per input record and then generate additional vertices and connecting edges for selected properties by applying `propertyToVertex`.

Since the same property value can occur many times, the creation of new vertices can lead to many duplicate vertices. We can thus choose to avoid such duplicate vertices (parameter `condense`). Such vertices can thus be connected to multiple originating vertices and thus represent shared information. The deduplication with the `condense` option is limited to equal values and thus only covers clean data sources. Hence, an additional deduplication for the created vertices may become necessary to fuse equivalent vertices with different names.

The user has several options to connect a newly created vertex to the original vertex with parameter `edgeConfig`: no edge, origin to new, new to origin and bidirectional. If an edge is created a user-defined label is set which is defined in the `edgeConfig`. The upper example in Fig. 2 shows the extraction of the atomic property `lives_in`. Alice and Bob are living in the same city and therefore the `City` vertex is deduplicated based on the value of the property `name`. All vertices Leipzig originates from are the start point of an edge with the label `lives_in` while the `City` vertex is the target of these edges. The `edgeConfig` for this



Fig. 3: The edge *promo_invited* is replaced (red) by the *Promotion* vertex with two edges (green) by using the *Edge to Vertex* operator. The *Vertex to Edge* operator inverts this operation.

example is: (label: *lives_in*; direction: *origin to new*). For the second example of Figure 2, we have a shared interest in *Music* so that this new vertex is connected to both *Bob* and *Alice*. This representation is obviously much better suited to analyze shared interests than the use of interest properties.

The inverse operation *vertex to property* is relatively straight-forward. Here *label* and *edgeConfig* are used to select the vertices to transform as well as the target vertices where the new properties should be added. As a result we can reverse both transformations shown in Figure 2.

4.2 Edge to Vertex & Vertex to Edge

The *Edge to Vertex* operator is beneficial for already existing graph structures. It creates a new vertex and two new edges for every edge with the desired label in the graph. GrALA code: `graph.edgeToVertex(edgeLabel, newVertexLabel, edgeLabelSourceToNew, edgeLabelNewToTarget)`. For the example in Figure 3 the GrALA call would be: `graph.edgeToVertex(promo_invited, Promotion, participated, invited)`. The operator turns the edge *promo_invited* between Alice and Bob with all its properties into a vertex and adds two user-defined edges.

The intuitive counterpart to *Edge to Vertex* is *Vertex to Edge*. It converts vertices with a specified label into edges between adjacent vertices. In GrALA it is defined as: `graph.vertexToEdge(vertexLabel, newEdgeLabel)`. To identify the necessary edges we select the "middle vertex" *v* with the specified label *vertexLabel* (which is to be replaced) and compose every direct neighbor with an edge going to *v* with every neighbor with an outgoing edge from *v*. With the *Vertex to Edge* operation we can revert the example given in Figure 3. The sole entry in the source set is Alice and in the target set Bob. Hence, an edge is created between Alice and Bob with the new label and all properties.

4.3 Connect Neighbors

The operator *Connect Neighbors* is designed to create relations between same-type vertices sharing a common neighbor vertex, e.g., employees of the same company (Fig.



Fig. 4: The example shows two *Persons* that got connected by there shared *Company*.

4) or authors of the same publication. In GrALA, the operator call is expressed by using: `graph.connectNeighbors(vertexLabel, edgeDirection, neighborVertexLabel, newEdgeLabel)`. Here, *vertexLabel* is the label of the central (shared) vertex and *neighborVertexLabel* the label of the vertices to be connected. Parameter *edgeDirection* is interpreted with respect to the shared vertex and can either be incoming, outgoing or undirected. For each pair of indirectly connected vertices of type *neighborVertexLabel* a new bidirectional edge of type *newEdgeLabel* is created. This is in contrast to the *Vertex to Edge* operation that creates directed edges and can be applied to vertices between neighbors of different type as shown in the example of Fig. 3.

Figure 4 shows an example where Bob and Marc share the same *Company* vertex. Therefore, the vertex and the two edges pointing to it are removed in favor of the *colleagues* edge which retains the properties of the removed vertex. If several edges are created, each of those edges is containing the property set of the removed vertex. The operation can also be used to create a co-author network from a publication network as discussed in the introduction.

5 Evaluation & Discussion

For the evaluation of the operators we used Flink version 1.5.0, Gradoop version 0.4.1 and a subset of the OpenAcademicGraph (as of 2017-06-09, MAG files 1 - 59)[Si15, Ta08]. OpenAcademicGraph contains two bibliographic datasets: the MAG dataset with 166 million and the the AMiner dataset with 155 million publication records. Each line in the datasets contains the Json representation of one publication.

Our subset contains 60 million publication records from the MAG dataset and comprises 102,5 GB of unpacked data. We read the data with the `JsonDataImport` of Gradoop that resulted in a so-called *Initial* evaluation dataset of 60 million vertices. However, some of our operators (*Edge to Vertex*, *Connect Neighbors*) require the data to already contain edges and relations between vertices. We thus created a second dataset called *Extended* using the *PropertyToVertex* operation. We extracted the properties author, affiliation, keywords and field of study all with `condense` option activated. This resulted in 100 million vertices and 74 million edges. Note that this graph may still include some deduplication problems not covered by *PropertyToVertex*, e.g., due to different variations of author or affiliation names.

We executed our benchmark operations on a Shared Nothing cluster of the Leipzig University Computing Center. We used 9 nodes of the cluster where each had 2 sockets equipped with

Configuration	Atomic Property Extraction			Collection Property Extraction		
	min	average	max	min	average	max
<i>no edge, no condense</i>	821s	838.2s	857s	851s	866.1s	894s
<i>no edge, with condense</i>	829s	846.7s	875s	853s	873.8s	899s
<i>create edge, no condense</i>	848s	853.3s	860s	910s	1066.7s	1235s
<i>create edge, with condense</i>	924s	940.6s	955s	905s	927.1s	955s

Tab. 2: Runtimes of atomic and collection property extraction.

6 core CPUs (Intel Xeon E5-2620 v3, 2.4 GHz, supports Hyperthreading), 128GB RAM, 6 SATA hard disks with 4 terabyte each and 10 Gigabit/s Ethernet interface. One node was designated to be the master node while all others were configured to work with 96 task executors in total (12 each). Each tested operator was executed at least 10 times and measurements contain I/O.

Table 2 shows the runtimes of the *PropertyToVertex* operator for atomic and collection-based property extraction. We consider the impact of whether or not edges are created between newly created and original vertices as well as whether or not property values are deduplicated (*condense* options). For the atomic case, we consider the *venue* properties and for the collection-based extraction we use the *FoS* properties (Field of Study) of publications. We observe that there are mostly only small differences between atomic and collection-based extraction. In the atomic case 22 million *venue* vertices and edges are created without deduplication and only 10 million vertices with deduplication. The collection-based property extraction created 40 million *FoS* vertices and edges without and 12.5 million vertices with deduplication. The highest runtime is in the collection-based scenario where more than 50 million new graph elements are created. Deduplication with the *condense* options typically incurs only a small additional runtime. For the collection-based extraction it is even faster since the much lower number of vertices to be created more than outweighed the deduplication effort.

Operator	minimum	average	maximum
<i>Vertex to Property</i>	1449s	1528.5s	1635s
<i>Vertex to Edge</i>	1055s	1088s	1147s
<i>Edge to Vertex</i>	69s	72,3s	76s
<i>Connect Neighbors</i>	1964s	2148.6s	2296s
<i>Invert Edges</i>	63s	66.65s	70s

Tab. 3: Runtimes of structural transformations.

The evaluation of the remaining transformations are based on the second dataset and Table 3 shows the resulting runtimes. We observe that edge-based transformations achieve the lowest runtimes favored by a small number of properties for edges. This makes communication in the cluster and creating new objects less expensive. Furthermore, the transformations can be implemented using only the vertex Ids. In contrast, operators like *Vertex to Edge*, *Connect Neighbors* or *Vertex to Property* rely on the graph structure and the whole graph needs to

be loaded. *ConnectNeighbors* turned out to be most expensive. For our evaluation this operator creates the co-author connections between authors of the same paper.

6 Conclusion & Future Work

We proposed structural transformation operations for property graphs with simple or collection properties to facilitate data integration and graph analysis. The operations have been implemented with Apache Flink and added to the open-source platform Gradoop. An initial evaluation for bibliographic data showed the applicability and relative efficiency of the operators. In future work we will further evaluate and optimize graph transformation operators and their use in real application scenarios as well as for graph-based data integration, in general.

7 Acknowledgements

This work was funded by the German Federal Ministry of Education and Research within project ScaDS Dresden/Leipzig (BMBF 01IS14014B). Computations for this work were done with resources of Leipzig University Computing Center.

References

- [An18] Angles, R. et al.: G-CORE: A core for future graph query languages. In: Proc. ACM SIGMOD. pp. 1421–1432, 2018.
- [Ar10] Arendt, T.; Biermann, E.; Jurack, S.; Krause, C.; Taentzer, G.: Henshin: advanced concepts and tools for in-place EMF model transformations. In: MODELS. pp. 121–135, 2010.
- [Be15] Benelallam, A.; Gómez, A.; Tisi, M.; Cabot, J.: Distributed Model-to-model Transformation with ATL on MapReduce. In: Proc. ACM SIGPLAN. pp. 37–48, 2015.
- [BN09] Bleiholder, Jens; Naumann, Felix: Data fusion. ACM Computing Surveys (CSUR), 41(1):1, 2009.
- [Eh97] Ehrig, H. et al.: Algebraic approaches to graph transformation. In: Handbook Of Graph Grammars And Computing By Graph Transformation: Volume 1: Foundations, pp. 247–312. World Scientific, 1997.
- [Gr18] Green, A.; Junghanns, M.; Kiessling, M.; Lindaaker, T.; Plantikow, S.; Selmer, P.: open-Cypher: New Directions in Property Graph Querying. In: Proc. EDBT. 2018.
- [Ja13] Jain, N. et al.: Graphbuilder: scalable graph ETL framework. In: Proc. 1st GRADES Workshop. 2013.
- [JPR17] Junghanns, Martin; Petermann, André; Rahm, Erhard: Distributed grouping of property graphs with GRADOOP. Proc. BTW conf., 2017.

- [Ju16] Junghanns, M.; Petermann, A.; Teichmann, N.; Gómez, K.; Rahm, E.: Analyzing extended property graphs with Apache Flink. In: Proc. SIGMOD Workshop on Network Data Analytics. 2016.
- [Ju17a] Junghanns, M. et al.: Management and analysis of big graph data: current systems and open challenges. In: Handbook of Big Data Technologies, pp. 457–505. Springer, 2017.
- [Ju17b] Junghanns, Martin; Kießling, Max; Averbuch, Alex; Petermann, André; Rahm, Erhard: Cypher-based graph pattern matching in GRADOOP. In: Proc. GRADES workshop. 2017.
- [Ju18] Junghanns, M.; Kießling, M.; Teichmann, N.; Gómez, K.; Petermann, A.; Rahm, E.: Declarative and distributed graph analytics with GRADOOP. PVLDB, 11(12), 2018.
- [KTG14] Krause, C.; Tichy, M.; Giese, H.: Implementing graph transformations in the bulk synchronous parallel model. In: FASE. 2014.
- [KVH18] K., Vasiliki; V., Vladimir; H., Seif: High-Level Programming Abstractions for Distributed Graph Processing. IEEE Trans. Knowl. Data Eng., 30(2):305–324, 2018.
- [Le15] Lee, S.; Park, H.; Lim, S.; Shankar, M.: Table2Graph: A Scalable Graph Construction from Relational Tables Using Map-Reduce. In: Proc. IEEE Conf. Big Data Computing Service and Applications. pp. 294–301, 2015.
- [Lö93] Löwe, M.: Algebraic approach to single-pushout graph transformation. Theoretical Computer Science, 109(1-2):181–224, 1993.
- [Ma10] Malewicz, G.; Austern, M.; Bik, A.; Dehnert, J.; Horn, I.; Leiser, N.; Czajkowski, G.: Pregel: a system for large-scale graph processing. In: Proc. ACM SIGMOD. 2010.
- [RN10] Rodriguez, M.; Neubauer, P.: Constructions from dots and lines. Bulletin of the American Society for Information Science and Technology, 36(6):35–41, 2010.
- [RN12] Rodriguez, M.; Neubauer, P.: The graph traversal pattern. In: Graph Data Management: Techniques and Applications, pp. 29–46. IGI Global, 2012.
- [Ro15] Rodriguez, M. A.: The Gremlin Graph Traversal Machine and Language. In: Proc. Symp. Database Programming Languages. pp. 1–10, 2015.
- [Si15] Sinha, A.; Shen, Z.; Song, Y.; Ma, H.; Eide, D.; Hsu, B.J.; Wang, K.: An overview of Microsoft Academic Service (MAS) and applications. In: Proc. WWW. 2015.
- [SPR17] Saeedi, A.; Peukert, E.; Rahm, E.: Comparative evaluation of distributed clustering schemes for multi-source entity resolution. In: Proc. ADBIS conf. pp. 278–293, 2017.
- [SPR18] Saeedi, A.; Peukert, E.; Rahm, E.: Using Link Features for Entity Clustering in Knowledge Graphs. In: Proc. ESWC. 2018.
- [Ta08] Tang, J.; Zhang, J.; Yao, L.; Li, J.; Zhang, L.; Su, Z.: Arnetminer: extraction and mining of academic social networks. In: Proc. ACM SIGKDD. pp. 990–998, 2008.
- [TH17] Tung, L.; Hu, Z.: Towards systematic parallelization of graph transformations over Pregel. Int. Journal of Parallel Programming, 45(2):320–339, 2017.
- [XD17] Xirogiannopoulos, K.; Deshpande, A.: Extracting and Analyzing Hidden Graphs from Relational Databases. Proc. ACM SIGMOD, pp. 897–912, 2017.
- [XKD15] Xirogiannopoulos, K.; Khurana, U.; Deshpande, A.: GraphGen: Exploring Interesting Graphs in Relational Data. PVLDB, 8(12):2032–2035, 2015.

Similarity

Efficient Bounded Jaro-Winkler Similarity Based Search

Jan Martin Keil¹

Abstract: The Jaro-Winkler similarity is a widely used measure for the similarity of strings. We propose an efficient algorithm for the bounded search of similar strings in a large set of strings. We compared our approach to the naive approach and the approach by Dreßler et al. Our results prove a significant improvement of the efficiency in computation of the bounded Jaro-Winkler similarity for querying of similar strings.

Keywords: Jaro-Winkler Similarity; Similarity Search; String Similarity

1 Introduction

The Jaro-Winkler similarity is a widely used measure for the similarity of strings. It was developed for the detection of duplicated persons in a dataset based on their name [Wi90]. Compared to other measures it provides both good results and fast computation [CRF03]. Nevertheless, the sequential calculation of the Jaro-Winkler similarity for the search of similar strings in large sets of strings is still a time-consuming task. Therefore, an optimized algorithm is needed for time-sensitive use cases like real time duplicate detection during data input, real time identification of named entities in input text (named entity linking), or real time fuzzy search.

We propose an optimized algorithm for the search of similar strings in a large set of strings. This work is structured as follows: In Sect. 2, we explain the Jaro-Winkler similarity and give an overview of related work, followed by the description of our approach in Sect. 3. In Sect. 4, we present the empirical evaluation of our approach. Finally, we conclude our work in Sect. 5.

2 Related Work

In this section we introduce the Jaro-Winkler similarity as well as other work on the efficient computation of this similarity measure.

¹ Heinz Nixdorf Chair for Distributed Information Systems, Institute for Computer Science, Friedrich Schiller University Jena, Germany, jan-martin.keil@uni-jena.de, <https://orcid.org/0000-0002-7733-0193>

2.1 Jaro-Winkler Similarity

The Jaro-Winkler similarity, often wrongly called *Jaro-Winkler distance*, is a similarity measure for two strings proposed in [Wi90]. It is based on the Jaro similarity

$$\text{Jaro}(s_1, s_2) = \begin{cases} \frac{1}{3} \cdot \left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m} \right) & : m > 0 \\ 0 & : \text{otherwise} \end{cases} \quad (1)$$

where $|s_1|$, $|s_2|$ are the lengths of both strings, m is the number of matching characters, and t is the number of transpositions. Matching characters are common characters in both strings with a maximum distance of $w = \frac{\max(|s_1|, |s_2|)}{2} - 1$ [DN17; Wi90]. Some sources give an alternative definition of $w = \frac{\min(|s_1|, |s_2|)}{2}$ [CRF03]. A character matches at most one character in the other string, selected by picking the first candidate during a nested iteration over both strings. The number of transpositions t is half the number of not equal positions in the concatenated strings of all matching characters in order of original occurrence [CRF03; Wi90]. Contrary to frequent assumptions, t is not equal to the number of permutations that is required to align the order of the matching characters. For example, $t(abc, bca) = 1.5$, even though two permutations are required to align the matching characters abc and bca .

The Jaro-Winkler similarity adds a boost for equal prefixes to high Jaro similarity values:

$$\text{JaroWinkler}(s_1, s_2) = \begin{cases} \text{Jaro}(s_1, s_2) + l \cdot p \cdot (1 - \text{Jaro}(s_1, s_2)) & : \text{Jaro}(s_1, s_2) \geq b_t \\ \text{Jaro}(s_1, s_2) & : \text{otherwise} \end{cases} \quad (2)$$

l is the length of the common prefix of both strings up to a maximum l_{bound} , b_t is the boost threshold, p is the prefix scale, and $l_{\text{bound}} \cdot p \leq 1$ must hold true. Implementations typically use the values $l_{\text{bound}} = 4$, $b_t = 0.7$, and $p = 0.1$ as in the original implementation².

2.2 Efficient Jaro-Winkler Similarity Computation

Dreßler et al. proposed an optimized algorithm to reduce the computation effort for pairwise similarities of strings from two large sets of strings given a similarity threshold [DN17]. They reduced the number of Jaro-Winkler similarity computations by applying filters to the pairs of strings. The first filter determines an upper bound of the Jaro similarity based on the lengths of both strings. The second filter determines an upper bound of the Jaro similarity based on the maximum number of matching characters using the character histograms of both strings. This approach provides an enormous performance improvement compared to a naive implementation for the matching of two large sets of strings. Therefore, it is useful for the matching of datasets or the detection of duplicates. However, if the task is to process user input or queries, i.e. compare one or a few strings with a large set of strings,

² <https://web.archive.org/web/19990822155334/http://www.census.gov:80/geo/msb/stand/strcmp.c>

a naive implementation outperforms this approach due to its overhead. Thus, this approach is not appropriate for the processing of user input or queries.

A recently published approach by Wang et al. addresses this issue [WQW17]. They developed an index for the Jaro-Winkler similarity search that contains special string signatures. Based on a lower bound of the number of matching characters and these signatures, they select candidate strings that might be similar to a query string. Further, the order of the signatures in the index allows to abort the scan of the remaining index at a certain point.

3 Approach

We propose another Jaro-Winkler similarity algorithm, which reduces the computational effort for the search of strings (*terms*) that are similar to a single string s_1 (*query*) in a large set of strings S_2 (*terminology*) given a similarity threshold θ . The terminology will be stored in a customized *PATRICIA tree* [Mo68] that additionally stores at each node the string lengths of all subjacent leaf nodes. This enables to skip irrelevant terms by skipping whole branches of the tree. Compared to a *trie* it avoids node chains without junctions at the bottom. This reduces the number of similarity computations. The maximum distance between matching characters (w) and thereby the number of matching characters (m) depends on the lengths of both strings. Therefore, the tree will be traversed once for each length of terms in the terminology, as in List. 1. Nodes without subjacent leaf nodes with this length will be ignored. Many traversals will stop at the root node, due to of the low maximum Jaro-Winkler similarity of strings with a notable difference of lengths.

During the traversal at each node the maximum Jaro-Winkler similarity will be computed based on the query string s_1 and the known prefix s_2^* of the term strings, as in List. 2. If the maximum Jaro-Winkler similarity is less then the threshold θ , the traversal of the current branch will be skipped, as in line 11 of List. 2. This requires a function for the maximum Jaro-Winkler similarity of a string s_1 and all strings S_2^* with prefix s_2^* and length $|s_2|$. JaroWinkler, Jaro, l , m , and t depend on s_1 and s_2 . Given $\max_{s_2 \in S_2^*}(\text{Jaro}) \geq b_t$, $0 \leq l \cdot p \leq 1$, and $0 \leq \text{Jaro} \leq 1$, then

$$\max_{s_2 \in S_2^*}(\text{JaroWinkler}) = \max_{s_2 \in S_2^*}(\text{Jaro} + l \cdot p \cdot (1 - \text{Jaro})) \quad (3)$$

$$= \max_{s_2 \in S_2^*}(1 - 1 + \text{Jaro} + l \cdot p - l \cdot p \cdot \text{Jaro}) \quad (4)$$

$$= \max_{s_2 \in S_2^*}(1 - (1 - \text{Jaro}) \cdot (1 - l \cdot p)) \quad (5)$$

$$= 1 - (1 - \max_{s_2 \in S_2^*}(\text{Jaro})) \cdot (1 - \max_{s_2 \in S_2^*}(l \cdot p)) \quad (6)$$

Therefore, the maximum Jaro-Winkler similarity is

$$\max_{s_2 \in S_2^*} (\text{JaroWinkler}) = \begin{cases} 1 - \left(1 - \max_{s_2 \in S_2^*} (\text{Jaro})\right) \cdot \left(1 - \max_{s_2 \in S_2^*} (l) \cdot p\right) & : \max_{s_2 \in S_2^*} (\text{Jaro}) \geq b_t \\ \max_{s_2 \in S_2^*} (\text{Jaro}) & : \text{otherwise} \end{cases} \quad (7)$$

The function for the maximum Jaro similarity is

$$\max_{s_2 \in S_2^*} (\text{Jaro}) = \begin{cases} \frac{1}{3} \cdot \left(\frac{\max_{s_2 \in S_2^*} (m)}{|s_1|} + \frac{\max_{s_2 \in S_2^*} (m)}{|s_2|} + 1 - \frac{\min_{s_2 \in S_2^*} (t)}{\max_{s_2 \in S_2^*} (m)} \right) & : \max_{s_2 \in S_2^*} (m) > 0 \\ 0 & : \text{otherwise} \end{cases} \quad (8)$$

List. 1: Search Initialization

```

1 FUNCTION search( $s_1, S_2, \theta$ )
2 treeRoot := tree( $S_2$ )
3 result :=  $\emptyset$ 
4 FOREACH lengths(treeRoot) AS  $|s_2|$ 
5    $w := \frac{\max(|s_1|, |s_2|)}{2} - 1$ 
6    $s := \text{newState}()$ 
7    $s.\text{minM} := 0$ 
8    $s.\text{minT} := 0$ 
9    $s.\text{maxL} := \min(4, |s_1|, |s_2|)$ 
10   $s.\text{saveCommonChars1} := 0$ 
11   $s.\text{assigned1} := \text{boolean}[|s_1|]$ 
12   $s.\text{assigned2} := \text{boolean}[|s_2|]$ 
13   $s.\text{commonChars2} := ""$ 
14  traverse(treeRoot,  $s_1, |s_2|, s_2^*, \theta, w, s, \text{result}$ )
15 RETURN result

```

List. 2: Tree Traversal

```

1 FUNCTION traverse( $\text{node}, s_1, |s_2|, s_2^*, \theta, w, s, \&\text{result}$ )
2 FOREACH characters of node AS  $c$ 
3   append( $s_2^*, c$ )
4   updateMaxL( $s_1, s_2^*, w, c, s$ )
5   updateMinM( $s_1, s_2^*, w, c, s$ )
6   updateMinT( $s_1, s_2^*, w, s$ )
7   updateMaxM( $s_1, s_2^*, w, s$ )
8   IF  $|s_2| = |s_2^*|$ 
9     finaliseMinT( $s_1, s_2^*, s$ )
10  maxJWS := using Eq. (7)
11  with  $|s_1|, |s_2|, s.\text{maxL}, s.\text{maxM}, s.\text{minT}$ 
12  IF  $\text{maxJWS} \geq \theta$ 
13  IF  $|s_2| \in |s_2^*|$ 
14    add( $\text{result}, <s_2^*, \text{maxJWS}>$ )
15 ELSE
16 FOR child  $\in$  children( $\text{node}$ )
17   IF  $|s_2| \in \text{lengths}(\text{child})$ 
18      $sc := \text{deepCopy}(s)$ 
19     traverse( $\text{child}, s_1, |s_2|, s_2^*, \theta, w, sc, \text{result}$ )

```

At each traversed node $\max_{s_2 \in S_2^*} (l)$, $\max_{s_2 \in S_2^*} (m)$, and $\min_{s_2 \in S_2^*} (t)$ will be computed. The effort can be reduced by reusing results from the parent node and updating them according to the new s_2 characters. For each new character it will be checked, if $\max_{s_2 \in S_2^*} (l)$ needs to be reduced, as in List. 3. To compute $\max_{s_2 \in S_2^*} (m)$, for each new character $\min_{s_2 \in S_2^*} (m)$ needs to be updated, as in List. 6: Each new s_2 character will be compared to the not matched s_1 characters in range. The first matching s_1 character according to reading order will be selected and $\min_{s_2 \in S_2^*} (m)$ will be updated. $\max_{s_2 \in S_2^*} (m)$ will then be computed once per node by adding the number of possible further matches to $\min_{s_2 \in S_2^*} (m)$, as in List. 7. $\min_{s_2 \in S_2^*} (t)$ can only be computed for s_1 characters that are already outside of the range of new s_2 characters, as new matching characters of s_1 might be located before earlier ones. Therefore, for each new s_2 character $\min_{s_2 \in S_2^*} (t)$ can be updated regarding s_1 characters at a new save position, as in List. 4. When s_2 is complete, t can be updated regarding the remaining characters of s_1 , as in List. 5. This algorithm overestimates $\max_{s_2 \in S_2^*} (\text{Jaro})$ if t and m can not achieve their extreme values at the same time. For example, for $s_1 = \text{abcd}$ and

$s_2 = \text{bound}$ the values are $\max_{s_2 \in S_2^*}(m) = 4$ and $\min_{s_2 \in S_2^*}(t) = 0$, but $\max_{s_2 \in S_2^*, t=0}(m) = 3$ and $\min_{s_2 \in S_2^*, m=4}(t) = 1$.

List. 3: Computation of $\max_{s_2 \in S_2^*}(l)$

```

1 FUNCTION updateMaxL( $s_1, s_2^*, w, c, s$ )
2 IF  $|s_2^*| \leq \text{bound}$  AND  $s_1[|s_2^*| - 1] \neq c$ 
3    $s.\text{maxL} := |s_2^*| - 1$ 

```

List. 4: Computation of $\min_{s_2 \in S_2^*}(t)$

```

1 FUNCTION updateMinT( $s_1, s_2^*, w, s$ )
2  $i := |s_2^*| - w - 1$ 
3 IF  $0 \leq i$  AND  $i < |s_1|$ 
4   IF  $s.\text{assigned1}[i]$ 
5     IF  $s_1[i] \neq s.\text{commonChars2}[s.\text{saveCommonChars1}]$ 
6        $s.\text{minT} := s.\text{minT} + 0.5$ 
7      $s.\text{saveCommonChars1} := s.\text{saveCommonChars1} + 1$ 

```

List. 5: Final computation of t

```

1 FUNCTION finaliseMinT( $s_1, s_2^*, w, s$ )
2 FOR  $i := \max(0, |s_2^*| - w)$  TO  $\min(|s_1|, |s_2| + w) - 1$ 
3   IF  $s.\text{assigned1}[i]$ 
4     IF  $s_1[i] \neq s.\text{commonChars2}[s.\text{saveCommonChars1}]$ 
5        $s.\text{minT} := s.\text{minT} + 0.5$ 
6      $s.\text{saveCommonChars1} := s.\text{saveCommonChars1} + 1$ 

```

List. 6: Computation of $\min_{s_2 \in S_2^*}(m)$

```

1 FUNCTION updateMinM( $s_1, s_2^*, w, c, s$ )
2 FOR  $i := \max(0, |s_2^*| - w - 1)$  TO
    $\min(|s_1|, |s_2^*| + w) - 1$ 
3   IF not( $s.\text{assigned1}[i]$ ) AND  $s_1[i] = c$ 
4      $s.\text{assigned1}[i] = \text{true}$ 
5      $s.\text{assigned2}[|s_2^*| - 1] = \text{true}$ 
6      $\text{append}(s.\text{commonChars2}, c)$ 
7      $s.\text{minM} := s.\text{minM} + 1$ 
8   BREAK

```

List. 7: Computation of $\max_{s_2 \in S_2^*}(m)$

```

1 FUNCTION updateMaxM( $s_1, s_2^*, w, s$ )
2  $\text{assignable1} := 0$ 
3 FOR  $i := \max(0, |s_2^*| - w - 1)$  TO
    $\min(|s_1|, |s_2| + w) - 1$ 
4   IF not( $s.\text{assigned1}[i]$ )
5      $\text{assignable1} := \text{assignable1} + 1$ 
6    $\text{assignable2} := |s_2| - |s_2^*|$ 
7    $s.\text{maxM} := s.\text{minM} + \min(\text{assignable1},$ 
    $\text{assignable2})$ 

```

The underlying strategy of our approach as well as the approach by Wang et al. is the early termination of the similarity computation for not similar strings. However, our approach successively approximates the similarity and extensively reuses earlier results. Conversely, the approach by Wang et al. once filters the strings by a rough upper bound of the similarity before computing the exact similarity. Both approaches skip computations for strings based on the intermediate results for other strings.

4 Evaluation

For the evaluation we used a Java implementation of each algorithm. *Our implementation* is publicly available³ and was used in version 0.1. For the *algorithm by Dresler et al.* we used their implementation⁴. To avoid a bias, we added a few modifications⁵, including the removal of a parallel execution management overhead during serial execution, and correction of bugs that skip parts of the result. While some of the changes decrease the runtime of the implementation, others increased them. However, to the best of our knowledge, the

³ https://mvnrepository.com/artifact/de.uni_jena.cs.fusion/similarity.jarowinkler/0.1.0

⁴ <https://github.com/kvndrsslr/SemanticWeb-QuickJaroWinkler>

⁵ <https://github.com/fusion-jena/QuickJaroWinkler>

modifications did not add any unnecessary increase of the runtime. For the *naive algorithm* we used the Jaro-Winkler similarity implementation provided in the Apache Commons Text library⁶ version 1.4. To correct the computation results we added a few modifications⁷, which became part of the 1.5 release of the library. A comparison to the *approach by Wang et al.* was not possible. The approach is not described in sufficient detail in the paper to reimplement it and the implementation is not publicly available. We are in contact with the authors, though, and aim to compare the approaches in the future.

All implementations require a preparation of the terminology, like building up the PATRICIA tree, but of different extent. We distinguished between the preparation and the actual similarity computation. To evaluate our approach we tested the following hypothesis:

Hypothesis 1 *Using our algorithm will improve the efficiency of the bounded Jaro-Winkler similarity computation between few queries and prepared large sets of terms, compared to the algorithm by Dresler et al. and the naive algorithm.*

4.1 Methods

We used the Java benchmark harness OpenJDK JMH⁸ to execute performance measurements of the three implementations. A collection of 1.429.572 names from the dataset “Person data” in the DBpedia dump 2016-10⁹ was used as test data. We used the following measurement parameters, which cover the intended use cases: (a) The *number of queries* (10^0 to 10^5 ; 10^6 was skipped due to long duration), (b) the *number of terms* (10^0 to 10^6), (c) the *threshold* of the Jaro-Winkler similarity (0.91, 0.95, 0.99). (d) the *overlap* of the set of query string and term strings (*full* means that all terms are contained in the queries, if possible for the given number of queries; *half* means that half of the terms are contained in the queries, if possible; *none* means that none of the terms is contained in the queries), and (e) the *preparation* of the terminology, specifying whether the time for preparation will be contained in the measurement (*unprepared*) or not (*prepared*).

Each configuration and implementation was executed on three machines with 20 different pseudo random subsets of names for the terms and the queries, resulting in 60 executions per configuration and implementation. We measured the throughput, which is the number of executions of all queries (= one operation) per second. The usage of the throughput results in a high precision of the measurement for short running computations, but decreasing precision for longer running computations. This fits to the intended use cases. The measurements were executed on 18 machines each equipped with two Intel Xeon Scalable 6140 18 Core 2,3 Ghz processors and 192 GB memory. Parallel computation was not used to avoid measurement

⁶ <https://commons.apache.org/proper/commons-text/>

⁷ <https://github.com/apache/commons-text/pull/87>

⁸ <http://openjdk.java.net/projects/code-tools/jmh/>

⁹ <https://wiki.dbpedia.org/downloads-2016-10>

errors. The measurement code, execution scripts, analysis scripts and result files are publicly available¹⁰.

4.2 Results

We used the Welch's t-test (unequal variances t-test) to compare the measurements, as we can not assume equal variance. First, we compared the three executions with equal configuration and of the same implementation. In 309 of 6570 cases (naive 6, Dresler 246, our 57) we found significant differences. Therefore, we hereinafter use the median values of the three corresponding execution. Then, we compared the corresponding measurements of different implementations. The overlap parameter caused only slightly differences between the comparison results. Therefore, we omit separate results. Tab. 1 shows the comparison results of the measurements of different implementations with equal configuration except the overlap parameter. Every triangle in the table represents the comparison results of 60 measurements for the column implementation and 60 measurements for the row implementation. The triangles point at the implementation with higher mean throughput. Comparisons above the diagonal involved unprepared measurements, comparisons under the diagonal involved prepared measurements. Bracketed comparisons were not significant. For example, for the naive approach and our approach and the parameters 1 term, 10 queries, threshold 0.91, and with preparation the mean throughput of the naive approach was insignificantly higher represented by (Δ). Fig. 1 shows the mean measurement for the implementations with 10^6 terms, depending on the number of queries, the threshold, and the preparation. All axes are log scaled.

The results presented in Tab. 1 prove that our approach significantly improves the computation efficiency of the bounded Jaro-Winkler similarity with 100 to 10^6 prepared terms, threshold ≥ 0.91 , and up to 10^3 queries, compared to the approach by Dresler et al. and the naive approach. Therefore, we accept the hypothesis. Our measurements were limited to 10^6 terms and 10^5 queries. This limitation fits to the addressed use case. The test dataset size and the time consumed by the measurement are further limiting factors. Due to the limitation, we can not provide valid results on the comparison of the approaches for larger string sets. However, the results presented in Tab. 1 indicate that the usage of our approach will improve computation efficiency for 100 or more terms in case of small query sets. Moreover, they indicate that the usage of our approach will improve the efficiency of computations with 10 up to 10^3 queries even if the terminology is unprepared. These limits will become worse by reduction of the thresholds. Due to the measurements visualized in Fig. 1d and Fig. 1f we expect that the approach by Dresler et al. will outperform our approach for larger query sets. However, this is not the use case our approach was developed for. We aim to support the search for similar strings of one or a few strings in a large set of string.

¹⁰ <https://github.com/fusion-jena/JaroWinklerSimilarityEvaluation> or DOI: 10.5281/zenodo.2269909

5 Conclusions

We presented a new approach for the efficient computation of the bounded Jaro-Winkler similarity. This approach has been evaluated by comparing it with the naive approach and the approach by Dreßler et al. [DN17]. Our results prove a significant improvement of the efficiency in computation of the bounded Jaro-Winkler similarity for querying of similar strings compared to these earlier approaches. In future work, we aim to also compare our approach with the approach by Wang et al. [WQW17], depending on the availability of the implementation or a comprehensive description of the approach. Further, we provide a ready to use Java implementation of our approach for easy application and adaptation into other languages. We are convinced, that this work opens up new application fields of the Jaro-Winkler similarity.

Acknowledgments. Part of this work was funded by DFG in the scope of the LakeBase project within the Scientific Library Services and Information Systems (LIS) program. The computational experiments were performed on resources of Friedrich Schiller University Jena supported in part by DFG grants INST 275/334-1 FUGG and INST 275/363-1 FUGG. Many thanks to Frank Löffler for very helpful advice on the evaluation setup. Likewise many thanks to the three anonymous reviewers and the shepherd Ingo Schmitt for very helpful comments on earlier drafts of this manuscript.

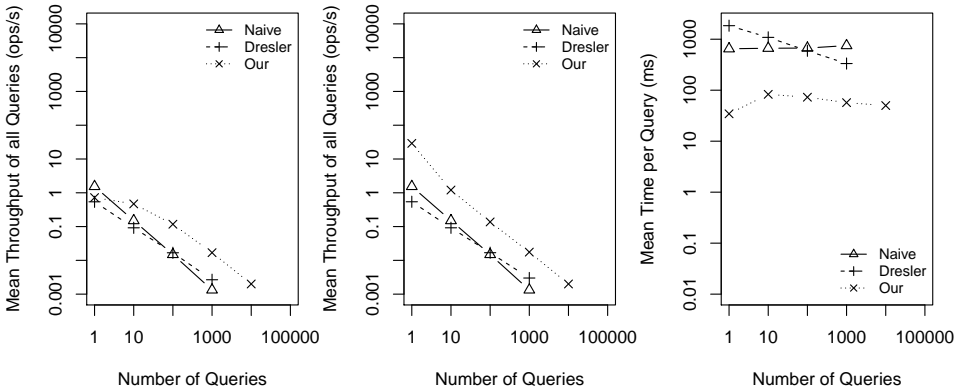
References

- [CRF03] Cohen, W. W.; Ravikumar, P.; Fienberg, S. E.: A Comparison of String Distance Metrics for Name-Matching Tasks. In (Kambhampati, S.; Knoblock, C. A., eds.): Proceedings of IJCAI-03 Workshop on Information Integration on the Web (IIWeb-03), August 9-10, 2003, Acapulco, Mexico. Pp. 73–78, 2003.
- [DN17] Dreßler, K.; Ngomo, A. N.: On the efficient execution of bounded Jaro-Winkler distances. *Semantic Web* 8/2, pp. 185–196, 2017, DOI: 10.3233/SW-150209.
- [Mo68] Morrison, D. R.: PATRICIA - Practical Algorithm To Retrieve Information Coded in Alphanumeric. *J. ACM* 15/4, pp. 514–534, 1968, DOI: 10.1145/321479.321481.
- [Wi90] Winkler, W. E.: String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage. In: Proceedings of the Section on Survey Research. American Statistical Association, pp. 354–359, 1990.
- [WQW17] Wang, Y.; Qin, J.; Wang, W.: Efficient Approximate Entity Matching Using Jaro-Winkler Distance. In (Bouguettaya, A. et al., eds.): *Web Information Systems Engineering - WISE 2017 - 18th International Conference, Puschino, Russia, October 7-11, 2017, Proceedings, Part I*. Vol. 10569. Lecture Notes in Computer Science, Springer, pp. 231–239, 2017, DOI: 10.1007/978-3-319-68783-4_16.

Tab. 1: Comparison results of mixed measurements with full, half, and zero overlap of terms and queries. ◀ row implementation significantly outperformed column implementation; (▶) row implementation insignificantly outperformed column implementation; Δ column implementation significantly outperformed row implementation; (Δ) column implementation insignificantly outperformed row implementation; ? missing or too imprecise values due to long running measurements; ≈ no difference measured, due to low precision

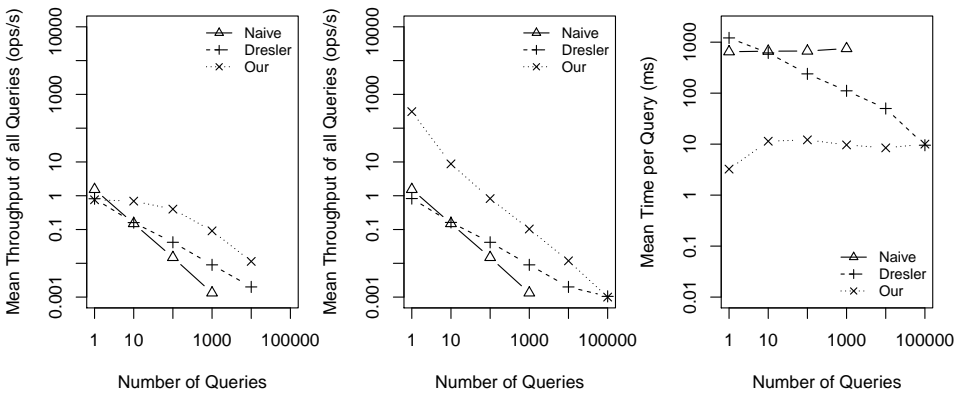
Queries \triangleright	Naive										Dresler										Our										standard
	1	10	10 ²	10 ³	10 ⁴	10 ⁵	10 ⁶	1	10	10 ²	10 ³	10 ⁴	10 ⁵	10 ⁶	1	10	10 ²	10 ³	10 ⁴	10 ⁵	10 ⁶	1	10	10 ²	10 ³	10 ⁴	10 ⁵	10 ⁶			
Threshold	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀		
Naive	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀		
Dresler	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀		
Our	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀	◀		

prepared



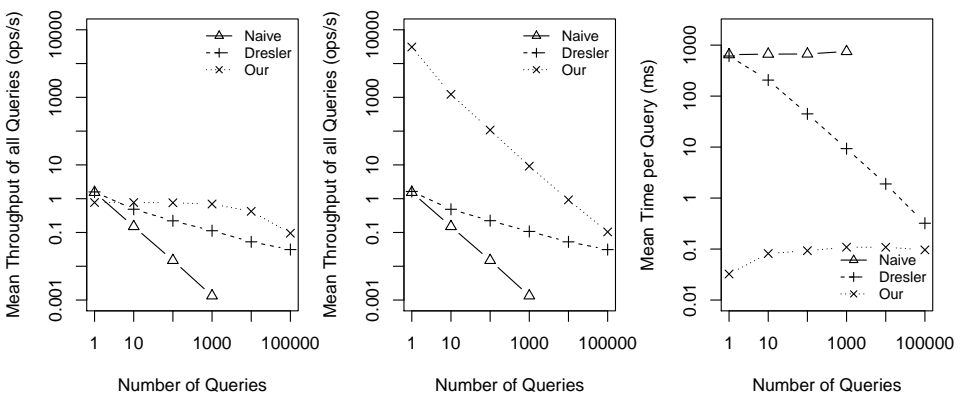
(a) Threshold 0.91, unprepared

(b) Threshold 0.91, prepared



(c) Threshold 0.95, unprepared

(d) Threshold 0.95, prepared



(e) Threshold 0.99, unprepared

(f) Threshold 0.99, prepared

Fig. 1: Mean of measurements with 10^6 terms and full, half or zero coverage.

The Best of Both Worlds: Combining Hand-Tuned and Word-Embedding-Based Similarity Measures for Entity Resolution

Xiao Chen^{1,2,3}, Gabriel Campero Durand^{1,2,3}, Roman Zoun^{1,2}, David Broneske^{1,2}, Yang Li^{1,2},
Gunter Saake^{1,2}

Abstract:

Recently word embedding has become a beneficial technique for diverse natural language processing tasks, especially after the successful introduction of several popular neural word embedding models, such as word2vec, GloVe, and FastText. Also entity resolution, i.e., the task of identifying digital records that refer to the same real-world entity, has been shown to benefit from word embedding. However, the use of word embeddings does not lead to a one-size-fits-all solution, because it cannot provide an accurate result for those values without any semantic meaning, such as numerical values. In this paper, we propose to use the combination of general word embedding with traditional hand-picked similarity measures for solving ER tasks, which aims to select the most suitable similarity measure for each attribute based on its property. We provide some guidelines on how to choose suitable similarity measures for different types of attributes and evaluate our proposed hybrid method on both synthetic and real datasets. Experiments show that a hybrid method reliant on correctly selecting required similarity measures can outperform the method of purely adopting traditional or word-embedding-based similarity measures.

Keywords: Entity resolution; Word embedding; Similarity measures; Learning-based entity resolution

1 Introduction

Entity resolution (ER) is the problem to identify digital records within one or among different data sources that refer to the same real-world entity. A typical solution is first to generate all candidate pairs, then calculate similarities between each pair [Ch12]. The high quality of ER results essentially relies on the correct selection of similarity measures, which is determined by domain experts based on an elaborate analysis of record attributes and their experiences. Except for the human efforts, traditional methods to calculate similarity between two records are mostly unaware of semantics, e.g., they determine how similar two

¹ Otto-von-Guericke-University of Magdeburg, Institute of Technical and Business Information Systems, Building 29, Universitätsplatz 2, 39106 Magdeburg, Germany

² E-Mail: {xiao.chen, campero.gabriel, roman.zoun, david.broneske, yang.li, saake}@ovgu.de

³ Acknowledgements: This work was partially funded by the DFG [grant no.: SA 465/50-1] and China Scholarship Council [No. 201408080093]. Authors would also like to thank the PC and reviewers for the valuable feedback.

strings look like in terms of edit distance. As a result, the accuracy may be limited for cases where words with similar semantics are expressed in different ways.

Recently, there have been several research papers that use word embedding for entity resolution, aiming to overcome the aforementioned limitations of traditional solutions [Eb18] [KAP18] [Mu18]. Word embedding maps words or phrases from the vocabulary to vectors of real numbers [Yo03] and makes it possible to extract semantic information conveniently and efficiently. It has become quite popular in various *Natural Language Processing (NLP)* tasks, after several neural word embedding models are introduced, such as word2vec [Mi13a], GloVe [PRS14] and FastText [Bo16]. Generally speaking, by using word embedding, attributes of records in ER are mapped to a vector space, no matter which kind of data it is, while capturing semantic similarities between attribute values of a record pair. Therefore, word embedding can be used to reduce the human efforts. In addition, the accuracy might also be further improved, since the approach captures semantic similarities between records.

However, adding an extra word embedding step to the similarity calculation phase will probably have negative effects on efficiency. For instance for attributes without semantic meanings, especially attributes with numerical values, it makes little sense to map them to a vector space using word embedding, because distances between them cannot be correctly managed due to the missing semantics. Still, numeric data play an important role for expressing records in financial business data.

Therefore, in this paper, we propose to combine traditional hand-tuned and popular word-embedding-based similarity calculations for ER, always choosing the most suitable similarity measures for each attribute in order to achieve the best accuracy for a given input dataset. Specifically speaking, with regard to attributes with a high appropriateness of word embedding techniques, word embedding is firstly applied to map the data to vectors using FastText pre-trained model, then cosine similarity is used to calculate similarities between record pairs. For other attributes, for which word embedding is not suitable, particularly numerical values, a tailored similarity function is used.

We summarize our contributions of this paper as follows:

- We propose a hybrid similarity calculation method for ER: the attribute-based selection between traditional hand-tuned similarity functions and word-embedding cosine similarity function to achieve a higher accuracy;
- We identify attributes that suit word embedding more than traditional similarity measures, which provides a guidance of choosing the most suitable similarity measure for ER;
- We design different combinations of adopted similarity functions and evaluate them on both real and synthetic datasets. Our results show that the hybrid solution can outperform other solutions which purely use traditional or word-embedding-based

similarity calculations, when the attributes of a dataset contain both semantic and non-semantic attributes.

The rest of this paper is organized as follows: We present related work in Section 2. Then we introduce our hybrid method in Section 3 and show the evaluation result in Section 4. At last, we conclude our work in Section 5.

2 Related Work

Entity resolution: Most of the ER research is pair-based and shares a common ER process, as surveyed in [Ch12] and [EPIV07]. In recent years, along with the increase of input data, solutions for ER are also asked to be scalable, which facilitates using parallel computing for ER, Chen et al. give an overview and classification on the research of parallel ER [CSG18].

Word embedding for entity resolution: As aforementioned, recent research has considered applying word embedding for ER. There are two main research questions of employing word embedding for ER. One is which embedding granularity to use for ER; the other one is how to get the vector of each attribute of a tuple after each word or sub-word has been embedded. Ebraheem et al. adopt word-level embedding using the GloVe pre-trained dictionary [PRS14] and propose two methods to get the vector representation of an attribute: an averaging method and an RNN-based method with LSTMs. Then a representation of a tuple is obtained by concatenating the vectors of all its attributes [Eb18]. Kooli et al. employ N-gram-level embedding using the the Fasttext library and then concatenate all vectors of all subwords [KAP18]. N-gram-level embedding should provide more accurate result when there is a large proportion of infrequent words in the input dataset [Mu18]. For ER tasks, data is often dirty and contains many infrequent words, therefore, in our work, we also use N-gram-level embedding. Mudgal et al. study several possible embedding choices from both the granularity of the embedding and adopted model, and sketch a design space for deep learning solutions [Mu18].

3 A Hybrid Approach for Entity Resolution

In this section, we introduce our ER process using hybrid similarity calculation methods. As we explained above, the work in this paper focuses on accuracy. Hence we do not take blocking (indexing) techniques into consideration to our process. With this, we get the most exact results since by blocking, we might block away matches in the worst case.

Figure 1 shows the entire process. The process first asks the user whether there is an available property file, which means the user knows the data quite well and knows the best choices of similarity functions to compare each attribute (excluding identifiers). If this is the case, we directly load the property file and divide all attributes into different groups based on the similarity measures defined in the property file. Otherwise, a property file will be

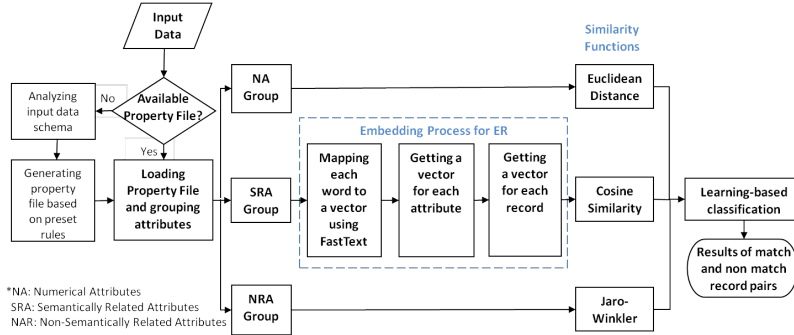


Fig. 1: Flowchart of our hybrid method

generated based on the input data schema and our preset rules. In order to achieve satisfied accuracy and provide guidance to choose suitable similarity measures, we classify common attributes into three groups, propose suitable similarity measures for each group and then the corresponding property file can be generated:

Numerical attributes (NA): Numerical attributes refer to those attributes, whose values can be compared, for example, the age. The Euclidean distance is used to provide a satisfactory accuracy for numerical attributes. However, not all numbers belong to this group. In contrast to numerical attributes, the other type of numbers are named as numerical strings, which act like strings and their values cannot be compared to be bigger or smaller. For example, telephone numbers or postcodes are considered as numerical strings. All numerical strings belong to the next group: non-semantically related attributes.

Non-semantically related attributes (NRA): These kinds of attributes are those whose values are without semantic meaning (e.g., people names or numerical strings) and usually with a short length. In ER tasks, the different values of this type of attributes of a match pair are usually caused by typos and formats. Using word embedding for this case may consider those values with a big distance, which can lead to lower accuracy. Therefore, for this group, we propose to use Jaro-Winkler similarity functions instead of word embedding. Moreover, Jaro-Winkler has a higher speed than the other traditional functions, such as Levenshtein similarity.

Semantically related attributes (SRA): The last group is semantically related attributes, whose values are strings with various meanings (often long), multiple strings or even whole sentences. For this group, word-embedding plus cosine similarity approach is chosen to calculate similarity. Traditional syntactical-based similarity measures are not used for semantically related attributes, because they only check the edit distances between values of attributes, and are not able to realize the meaning or the distributional semantic behind, so that they would probably provide unacceptable results. In terms of the predefined rules, we can calculate the similarity of each corresponding attribute pairs (cf. Section 3.1). Afterwards, the classification step is performed to divide all pairs into match and non-match groups (cf. Section 3.2).

3.1 Attribute Similarity

In this subsection, we specify how the similarity of each pair of attributes is calculated by the preset rules.

NA and NRA Similarity. According to the two rules regarding NA and NRA, the similarity between two attribute values of records calculated straightforwardly with Jaro-Winkler similarity and Euclidean metric, because the granularity for them is the entire attribute, which is calculated with the following formula:

$$\text{attrSim}(r_1.\text{attr}, r_2.\text{attr}) = \begin{cases} \text{Euclidean}(r_1.\text{attr}, r_2.\text{attr}), & \text{attr} \in \text{NA}; \\ \text{Jaro_Winkler}(r_1.\text{attr}, r_2.\text{attr}), & \text{attr} \in \text{NRA}. \end{cases} \quad (1)$$

SRA Similarity. Inspired by the work of [Bo16], we use FastText, an extension of the continuous skip-gram model [Mi13b] that is used to produce word embeddings (i.e., vectors), to obtain the similarity of semantically related attributes. The FastText model is chosen instead of word2vec and GloVe because use cases will range across diverse domains. Hence, we cannot guarantee coverage with only models on word-level.

Vector representation of words. The input of FastText is a text corpus. Given enough text data and contexts, FastText can achieve highly accurate meanings of the words appearing in the corpus and establishes a word's association with other words. The output is a set of vectors, that is, words in the corpus are transformed a vector representation in a semantic vector space. Moreover, FastText is capable of predicting the vector representation for words not occurring in the corpus, since it decomposes each word into a set of characters of n -grams. Hence, every word appearing in the attribute value can be converted into vector representation \vec{w} .

Vector representation of attributes. Considering that there may be two or more words in one attribute, we achieve the vector representations of attributes by computing the mean value of all the word vectors in the attribute:

$$\vec{\text{attr}} = \frac{\sum_{i=1}^n \vec{w}_i}{n} \quad (2)$$

Therein, n is the number of words in attributes. We compute the cosine similarity between two attribute vectors to evaluate their similarity. More formally, the similarity of SRA is represented by:

$$\text{attrSim}(r_1.\text{attr}, r_2.\text{attr}) = \text{cosine}(r_1.\vec{\text{attr}}, r_2.\vec{\text{attr}}), \quad \text{attr} \in \text{SRA}. \quad (3)$$

3.2 Learning-Based Classification

After the similarity for each attribute pair is calculated with their respective methods, a classification step is used to classify pairs to matches or non-matches. The simplest

approach is a threshold-based method, which compares the average similarity score with the preset threshold value. Those pairs whose average score is higher than the threshold are considered as matches, and vice versa. However, in order to get the average score, similarity scores for each attribute are firstly summed up, which loses detailed information contained in the separated attribute similarities [Ch18]. Therefore, in our ER process, we adopt a learning-based classification step to overcome this drawback. A learning-based classification method firstly trains a classifier on a training dataset with available match or non-match labels, then the trained classifier is used to classify pairs with match or non-match status. So far, there have been different classifiers proposed, how to select a suitable one is based on the input data properties. In Section 4, we will describe the three classifiers we employ for our experiments.

4 Evaluation

In this section, we show our designed similarity calculation methods and the evaluation of their F-measure on different real-world datasets and synthetic datasets. Next, we show the datasets used for our experiments, and represent designed combinations of similarity calculation methods, at last we describe the three classifiers we use for the evaluation.

Datasets: Table 1 shows the three datasets we used for our experiments.

The first dataset “DBLP-ACM Citation” include two parts, which are from the DBLP citation database and the ACM citation database, respectively. All of them have four attributes, including title, authors, venue and publication year. Therein, “title”, “venue” are considered as SRAs, while “authors” is NRA and “publication year” is NA.

The second dataset “Amazon-Google Products” includes two parts as well, which are from Amazon and Google. Both of them have four attributes, including id, name, description, manufacturer and price. Based on their properties, “name”, “description” and “manufacturer” are considered as SRAs, while “price” is NA. All above datasets are downloaded from [Le17], which are benchmark datasets often used for entity resolution.

The last dataset is synthetic and generated by a data generator called GECO [TVC13]. GECO consists of a GEnerator and a COrruptor, which is specifically designed for generating ER datasets. The generated dataset contains personal information with the following 13 attributes: gender, given-name, surname, postcode, city, sex, telephone-number, credit-card-number, income-normal, age-uniform, income, age, and blood-pressure. Therein, the last five attributes are considered as NAs, while “sex” and “gender” are SRAs and the other attributes are NRAs.

The first two real datasets are commonly-used benchmark datasets for ER. However, both of them only contain one numerical value, which we think the most useful attribute type to show differences of using different combinations of similarity calculations. Therefore, we involve the generated dataset as well, which contains five attributes with numerical values.

Tab. 1: Datasets used in experiments

Datasets		#Pairs (#DS1&#DS2)	#Matches	#SRA	#NRA	#NA
Real Datasets	DBLP-ACM	6001104 (2616&2294)	2224	2	1	1
	Amazon-Google	4400264 (1364&3226)	1300	3	0	1
Synthetic Dataset	Persons	551250 (1050&1050)	96	2	6	5

Combinations of similarity calculation methods:

Traditional hand-crafted similarity functions only: For this scenario, we use Jaro Winkler similarity function to calculate the similarity of all string attributes, i.e., NRAs plus SRAs, and use Euclidean distance to calculate the similarity of all NAs.

Word embedding and cosine similarity based method only: For this scenario, we use word-embedding-based method for all attributes.

Our proposed hybrid method: We propose to use word embedding for SRAs, Jaro Winkler for NRAs and the Euclidean distance function for NAs.

Classifiers: We employ three straight-forward classifiers which are intended to represent general solutions for classification in an ER context.

Tree Boosting (XGBoost): XGBoost consists of an ensemble of regression trees that uses additive optimization [CG16]. Such approach starts with a low variance, high biased solution, and gradually reduces the bias by decreasing the sizes of neighborhoods. We employ the specific XGBoost model, which introduces fine-grained improvements penalizing individual trees, leading to a competitive classification performance.

Random forest classifier (RF): This is an approach based on multiple decision tree algorithms for achieving a higher accuracy. It is intended to reduce overfitting risks.

K-nearest neighbor classifier (KNN): This approach considers the k (in our case k=5) closest training examples for each record to decide on the class membership of the record.

4.1 Results and Discussion

Tab. 2: Evaluation Results with Different Classifiers (F-measure)

Combinations		XGBoost	RF	KNN
Generated Dataset	Traditional	100	100	88.46
	WordEmbedding	100	100	100
	Hybrid	100	100	58.54
DBLP-ACM	Traditional	97.04	97.70	95.17
	WordEmbedding	92.56	94.82	93.94
	Hybrid	93.69	94.28	89.31
Amazon-Google	Traditional	20.19	25.35	21.11
	WordEmbedding	19.10	31.09	24.10
	Hybrid	29.72	38.32	19.78

Table 2 shows the results of our evaluation (results are reported for a random split of 66/34% training/test data, each including respectively 66/34% of the existing match and non-matches). It shows the F-measure values by using the three aforementioned classifiers XGBoost, KNN, RF under our three designed combinations of similarity measures on the three datasets. Next, we will discuss the result of each dataset in detail.

Generated dataset: As can be seen, the F-measure reaches its optimal for the generated dataset, with word embedding solutions performing consistently well across classifiers. This is a slightly surprising finding, as most attributes can be labeled as non-semantic. We deem the goodness of embeddings to be partly dependent on the coverage of the FastText learned representations. The purpose of using the generated dataset is it contains several NAs, which is promising to show the accuracy difference between different approaches. However, the generated dataset still lacks unpredictability and complexity compared to a real dataset, so that all F-measures are very high and we cannot get valuable conclusions from it. We also observe that the results of KNN are affected by the existence of irrelevant features, whereby the similarity of items on one dimension leads to misclassifications for close neighbors. For its less effective configurations, the model outputs 6 and 21 false positives out of 30, reducing the F-measure on this dataset.

DBLP-ACM dataset: For the relatively clean and easy citation dataset, the F-measure is observed to be consistently higher than 95% with traditional approaches outperforming others. The word-embedding-based and the hybrid approach show lower results. By careful consideration of its attributes properties, we found that the “title” attribute is, although having long strings, actually not semantic related. A paper title normally only has one version and its name usually only differs due to possible typos. Under this situation, word-embedding-based methods may fail. Therefore, for those attributes with long strings, careful consideration is needed about whether an edit-distance-based or semantic-based similarity measure is more suitable for them.

Amazon-Google dataset: For the more complex product dataset (various descriptions may express the same semantic meaning) we observe that hybrid solutions outperform the other two pure solutions, and achieve the best results with XGBoost and RF. For KNN we also observe some problems with the existence of irrelevant dimensions, which increase the number of false negatives. Overall, embedding-based approaches outperform traditional solutions (for classifiers RF and KNN), or provide comparable result with XGBoost, which indicates the necessity to use word embedding for datasets with real semantic attributes. The hybrid approach performs the best with XGBoost and RF, which indicates that by carefully choosing word-embedding-based or traditional similarity measure according to attribute properties (mainly basing on non-semantic and semantic) a better accuracy can be achieved than using only one of them.

We should also note that the F-measure values we report for this challenging dataset are much lower than many published results. We attribute this to the fact that almost all previous research adopts either blocking or thresholding techniques to reduce the amount of

non-matching pairs, so that the training data set is much more balanced, helping to achieve a higher F-measure. For this paper, our purpose is solely to test the benefit of our proposed hybrid similarity calculation approach. Therefore, we avoided blocking and thresholding, in order to highlight the specific impact of the approaches.

To conclude our discussion, a word-embedding-based approach works predominately better for semantic attributes. For non-semantic attributes it may also be possible to achieve a F-measure which is comparable (i.e., as seen for the generated dataset) or worse (i.e., as seen for the DBLP-ACM dataset) than traditional similarity measures. However, for numerical values, word embedding is not recommended since a hybrid approach shows obviously better F-measure than only using word embedding (based on the result of Amazon-Google dataset with XGBoost and RF classifiers). Therefore, the safest way for similarity calculations of ER problems is to use word embedding for semantic attributes and to use traditional similarity measures for non-semantic attributes. Besides, interestingly, in our findings we note that the choice of classifier seems to be secondary to the choice of similarity measures, with simple classifiers being able to outperform more complex ones, provided they are given adequate (i.e., descriptive, distinctive) similarity measures as input. We expect this to be partly explainable from the simplicity of the datasets (for the first two cases) and from the improvements brought by semantics (in the third case).

5 Conclusion

In this short paper we propose to use a hybrid similarity calculation solution for ER tasks and provide a practical evaluation of three different combinations of similarity measures with general machine learning classifiers. We find that embeddings are generally useful, though they are not a silver bullet, and both hybrid and traditional approaches can achieve superior results. We find that similarity measures can have a greater impact than the choice of classifiers in the resulting goodness of an ER process.

In summary, by using a prototypical workflow without blocking or thresholding and with general classifiers, we show that the current use of word embeddings alongside traditional measures for entity resolution opens-up a bundle of promising choices for practitioners, without lending itself easily to a one-size-fits-all solution. We envision two core challenges: On the one hand, previous work [KR08] has shown that thresholding, with a given blocking solution, improves the learning process; similarly work in entity resolution with embeddings [Eb18] shows good results without quantifying the precise impact of blocking and thresholding. In future work we seek to extend our current study by considering this factor for the case of hybrid solutions. On the other hand, the search for the optimal balance between the similarity measures used becomes essential, as we show in our current study. Though some guidelines can be adopted in this task for specific cases, as we propose, future work is required to achieve a general method for combining the approaches.

References

- [Bo16] Bojanowski, P.; Grave, E.; Joulin, A.; Mikolov, T.: Enriching word vectors with subword information. arXiv preprint arXiv:1607.04606, 2016.
- [CG16] Chen, T.; Guestrin, C.: Xgboost: A scalable tree boosting system. In: SIGKDD. ACM, pp. 785–794, 2016.
- [Ch12] Christen, P.: Data matching: concepts and techniques for record linkage, entity resolution, and duplicate detection. Springer Science & Business Media, 2012.
- [Ch18] Chen, X.; Rapuru, K.; Durand, G.; Schallehn, E.; Gunter, S.: Performance Comparison of Three Spark-Based Implementations of Parallel Entity Resolution. In: DEXA-BDMICS. volume 903. Springer, pp. 76–87, 2018.
- [CSG18] Chen, X.; Schallehn, E.; Gunter, S.: Cloud-Scale Entity Resolution: Current State and Open Challenges. OJBD, 4(1):30–51, 2018.
- [Eb18] Ebraheem, M.; Thirumuruganathan, S.; Joty, S. R.; Ouzzani, M.; Tang, N.: Distributed Representations of Tuples for Entity Resolution. PVLDB, 11(11):1454–1467, 2018.
- [EPIV07] Elmagarmid, A.; P. Ipeirotis, Panagiotis; Verykios, V.: Duplicate record detection: A survey. TKDE, 19(1):1–16, 2007.
- [KAP18] Kooli, N.; Allesiaro, R.; Pigneul, E.: Deep Learning Based Approach for Entity Resolution in Databases. In: ACIHDS. Springer, pp. 3–12, 2018.
- [KR08] Köpcke, H.; Rahm, E.: Training selection for tuning entity matching. In: QDB/MUD. pp. 3–12, 2008.
- [Le17] Leipzig, Database Group: , Benchmark datasets for entity resolution, 2017. Downloaded on 27.11.2017.
- [Mi13a] Mikolov, T.; Chen, K.; Corrado, G.; Dean, J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781, 2013.
- [Mi13b] Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G.; Dean, J.: Distributed Representations of Words and Phrases and their Compositionality. Curran Associates, 2013.
- [Mu18] Mudgal, S.; Li, H.; Rekatsinas, T.; Doan, A.; Park, Y.; Krishnan, G.; Deep, R.; Arcaute, E.; Raghavendra, V.: Deep Learning for Entity Matching: A Design Space Exploration. In: SIGMOD. ACM, pp. 19–34, 2018.
- [PRS14] Pennington, J.; R. Socher, Riand Manning, Christopher: Glove: Global vectors for word representation. In: EMNLP. pp. 1532–1543, 2014.
- [TVC13] Tran, K.; Vatsalan, D.; Christen, P.: GeCo: An Online Personal Data Generator and Corruptor. In: CIKM. ACM, pp. 2473–2476, 2013.
- [Yo03] Yoshua, B.; Réjean, D.; Pascal, V.; Christian, J.: A neural probabilistic language model. Journal of machine learning research, 3:1137–1155, 2003.

Fast Approximated Nearest Neighbor Joins For Relational Database Systems

Michael Günther¹, Maik Thiele¹, Wolfgang Lehner¹

Abstract: K nearest neighbor search (kNN-Search) is a universal data processing technique and a fundamental operation for word embeddings trained by word2vec or related approaches. The benefits of operations on dense vectors like word embeddings for analytical functionalities of RDBMSs motivate an integration of kNN-Joins. However, kNN-Search, as well as kNN-Joins, have barely been integrated into relational database systems so far. In this paper, we develop an index structure for approximated kNN-Joins working well on high-dimensional data and provide an integration into PostgreSQL. The novel index structure is efficient for different cardinalities of the involved join partners. An evaluation of the system based on applications on word embeddings shows the benefits of such an integrated kNN-Join operation and the performance of the proposed approach.

Keywords: approximated nearest neighbor search, product quantization, RDBMS, word embeddings

1 Introduction

Word embedding techniques are powerful to study the syntactic and semantic relations between words by representing them in dense vectors. By applying algebraic operations on these vectors, semantic relationships such as word analogies, gender-inflections, or geographical relationships can be easily recovered [LG14]. Due to the powerful capabilities of word embeddings, some recent papers proposed their integration into relational databases [BBS17, Gü18]. This allows exploiting external knowledge during query processing by comparing terms occurring in a database schema with terms stored in word embeddings. To give some examples: a user may query all products in a product database and rank them according to their mean similarity to terms like “allergen” or “sensitizer”. In the context of a movie database, a kNN-Search can be performed to return the top-3 nearest neighbors of each movie title (see q_1 in Fig. 1). Given the movie “Godfather” as input this might result in “1972” (the release year), “Scarface” (another popular movie in the same genre) and “Coppola” (the director). Our main observation is that most of these SQL database word embedding operations perform similarity search with *k nearest neighbor search* (kNN) as a common subtask. Furthermore, the domain of the kNN-Search often needs to be restricted to terms that also appear in the database relation, i.e. the domain modeled by the database schema. In this way arbitrary terms, having their origins in the

¹ Technische Universität Dresden, Institut for Systems Architecture, Dresden Database Systems Group, Nöthnitzer Straße 46, 01187 Dresden, firstname.lastname@tu-dresden.de

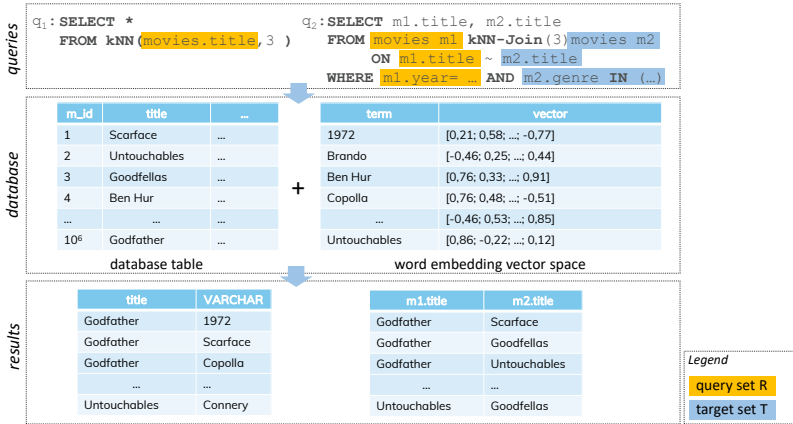


Fig. 1: Two Example Queries: kNN-Search and kNN-Join

large text corpora on which the word embedding models have been trained on, are filtered out. To give an example: if a kNN-Search is performed on an attribute *movie titles* the user usually expects to get the most similar movies but not release years, actors, directors or other terms that do not relate to the movie domain at all. Technically this boils down to a *k nearest neighbor join* (kNN-Join) that combines each element in a *query set R* with the *k* elements in a *target set T* that are closest to it. This is shown by q_2 in Fig. 1 that extends q_1 by a target set containing just movie titles and that returns movie titles only. Due to the high dimensionality of word embeddings (100 to 300 dimensions are a typical number) and large input data sets (query and target set), the kNN-Join is an extremely expensive operation. Therefore, we investigate approximation and indexing techniques based on vector quantization approaches, especially product quantization [JDS11], to accelerate kNN-Joins in the realm of RDBMSs. In particular, our contributions are the following:

- We identify two different kNN-Join query types with different needs regarding the supporting index structures.
- We propose a novel index structure which can cope with both query types and is flexible enough to deliver good performance on them.
- We detail how to efficiently process an approximated kNN-Join query that adapts to different query and target set sizes.
- We provide a practical implementation of the operator and our optimizations in PostgreSQL, which allows us to meaningfully evaluate the operator using high-dimensional data and fully-featured SQL queries regarding both, accuracy and runtime.

The remainder of the paper is structured as follows: In the next section, we define the kNN-Join problem and derive the challenges which arise by supporting this operation on high-dimensional data. In Sect. 3, we provide the fundamentals of the vector quantization techniques which form the foundation for our proposed index structure. In Sect. 4, we present our inverted product quantization index as well as our approximated and adaptive kNN-Join implementation. Given two real-world datasets, we evaluate the accuracy and response time of this novel operator regarding different input relation sizes in Sect. 5. Finally, we survey the related work in Sect. 6 and conclude the paper in Sect. 7.

2 Problem Description

Given two vector sets R and T in a d -dimensional Euclidean space \mathbb{R}^d and two elements \mathbf{r} and \mathbf{t} , with $\mathbf{r} \in R$ called query vectors and $\mathbf{t} \in T$ called target vectors, a kNN query is defined as follows:

Definition 2.1. The kNN query of \mathbf{r} over T , noted $kNN(\mathbf{r}, T)$, can be defined as:

$$kNN(\mathbf{r}, T) = \arg \min_{\{\mathbf{t}_1, \dots, \mathbf{t}_k\} \in T^{(k)}} \sum_{i=1}^k d(\mathbf{r}, \mathbf{t}_i).$$

Here d denotes the distance function between two elements. Typically, in the context of word embeddings, the cosine distance is used. However, in case of normalized vectors \mathbf{r} and \mathbf{t} , the cosine distance is proportional to the squared Euclidean distance. The normalization of the vectors does not change the cosine distance. Thus the $kNN(\mathbf{r}, T)$ for any \mathbf{r} and T can be computed using both metrics. If the query is not just one element but instead a set, the operation is denoted as kNN-Join.

Definition 2.2. The kNN-Join between a query set R and a target set T is defined as:

$$kNN(R \bowtie T) = \{\langle \mathbf{r}, \mathbf{t} \rangle \mid \mathbf{t} \in kNN(\mathbf{r}, T), \mathbf{r} \in R\}.$$

Challenges The aim of this paper is to provide a kNN-Join which is particularly suitable for high-dimensional data and varying target sets. In detail, we identify the following challenges:

1. Batchwise execution of large query sets: In contrast to a simple kNN-Search, it must be possible to execute large amounts of nearest neighbor queries at once for kNN-Joins. Most of the approximated kNN-Search (ANN-Search) approaches assume that the set of target vectors contains a very large number of vectors, however, they do not process large amounts of query vectors. In the case of kNN-Joins, the number of queries can be much larger than the target vector set.

2. High-dimensional data: Previous work on kNN-Joins for relational database systems focuses mostly on low-dimensional data [YLK10]. Because of the *curse of dimensionality*, the distances of pairs of sample vectors from a high-dimensional vector space tend to differ only little [Be99]. Therefore, techniques for exact kNN-Joins, trying to hierarchical

partition vector spaces, cannot be applied efficiently. Hence, the system must support suitable approximated search techniques to handle large vector sets.

3. Adaptive kNN-Join algorithm: An index for the kNN-Join stores all possible target vectors. However, a target set T often contains just a small subset of all vectors in the index. For example: target set for q_2 in Fig. 1 only contains vectors of movies published in a specific year out of millions of other vectors. The kNN-Join algorithm therefore must be adaptive to different target set sizes and should enable fast approximated search. With respect to the cardinality of R and T we identified two different kNN-Join query types in a database system:

kNN queries with small query set R and large target set T : This is the ordinary type of kNN queries, which most of the kNN frameworks assume.

kNN queries with large query set R and small target set T : This case is rather specific to the use in database systems and is currently not supported.

4. Different demands on precision and response time: Regarding the approximation of the vector similarity, it might be relevant for a user to specify how strongly the approximated nearest neighbors should correspond with the exact values. On the contrary, real-world systems need to comply with certain latency constraints, e.g., for exploratory data processing fast response times are crucial. Consequently, the approximated kNN-Join should provide features to configure such trade-offs. Providing this tunable trade-offs would also support query execution in an online aggregation manner, i.e., get estimates of a kNN-Join query as soon as the query is issued and steadily refine during its execution.

3 Vector Quantization

The index structure we propose is based on different *vector quantization* techniques. *Vector quantization* is able to transform vectorial data in an approximated compact representation [Gr84]. Furthermore, it enables fast approximated distance calculation which subsequently can be used to speed up kNN-Join operations. It is the basis of product quantization as well as the basis for inverted indexing techniques described in Sect. 3.2 and Sect. 3.3.

3.1 Quantization Function

Vector quantization can be implemented by a quantization function $q(\mathbf{y})$ which assigns a vector \mathbf{y} to a centroid $\mathbf{c}_j \in C$ where \mathbf{c}_j is the vector of C which has the lowest distance to \mathbf{y} . There are different ways to obtain such a quantization function, which is specified by the centroid set C and a distance function d . As a distance function, we use the *Euclidean distance*. The set C should be selected so that the distortion is minimal. The *k-means* algorithm is commonly used to achieve this goal for a given number of centroids $|C|$. An approximated representation of a vector dataset can be obtained by replacing every vector $\mathbf{y} \in Y$ (floating point values) with their centroid id j of its quantization centroid $\mathbf{c}_j = q(\mathbf{y})$ (integer values).

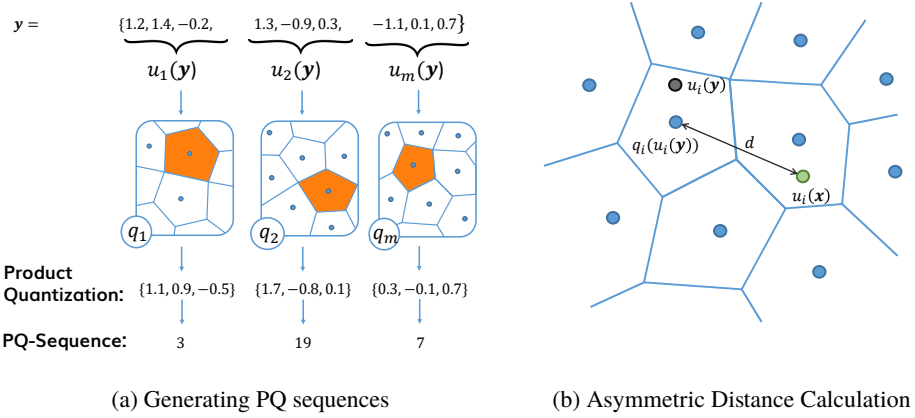


Fig. 2: Product Quantization

3.2 Product Quantization

A simple vector quantization approach might lead to a quite inaccurate representation of the vector dataset. For a more precise representation, huge numbers of centroids would be necessary that are impossible to process or even to store. For this reason, *product quantization* [JDS11] applies multiple quantizers on m subvectors $u_1(\mathbf{y}), \dots, u_m(\mathbf{y})$ of the original vector \mathbf{y} (see Fig. 2a.). Those quantizers are defined by quantization functions q_1, \dots, q_m with $q_i : \mathbb{R}^d \rightarrow C_i$. Typically, the cardinalities $|C_1|, \dots, |C_m|$ are equal. The product quantization is the sequence of centroids obtained by that process.

$$\underbrace{y_1, \dots, y_d}_{u_1(\mathbf{y})}, \dots, \underbrace{y_{(D-d)+1}, \dots, y_D}_{u_m(\mathbf{y})} \rightarrow q_1(u_1(\mathbf{y})), \dots, q_m(u_m(\mathbf{y})) \quad (1)$$

Using a dictionary denoted as codebook, the sequence of centroid vectors can be compactly represented as a sequence of centroid ids.

kNN-Search with PQ-Index Product quantization sequences can be utilized to accelerate the calculation of nearest neighbors by providing a fast way to compute approximated squared distances. Approximated square distances between a query vector \mathbf{x} and a vector \mathbf{y} for which a product quantization sequence is available can be calculated by Eq. (2).

$$\hat{d}^2(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^m d(u_i(\mathbf{x}), q_i(u_i(\mathbf{y})))^2 \quad (2)$$

Entry ID	Coarse ID	PQ u_1	Sequence u_3	u_2	u_4	ID	Word	Vector	ID	Centroid Vector	ID	Pos	Sub Vector Centroid
1	1	4	2	3	4	1	tee	[1.78, 3.22, -2.55, ...]	1	[1.89, 2.42, -1.78, ...]	1	1	[1.80, -0.43]
2	4	1	2	1	3	2	toast	[1.78, -1.35, 0.45, ...]	2	[3.78, -1.22, 2.55, ...]	2	1	[-2.49, -1.89]
3	1	4	3	3	3	3	coffee	[1.11, 2.22, -2.01, ...]	3	[-0.78, -3.28, -0.57, ...]
4	2	1	3	4	1	4	eggs	[0.78, -0.72, 5.12, ...]	4	[4.17, 0.22, 2.24, ...]	1	2	[1.07, 3.22]
...

Index Data
Original Vectors
Coarse Quantizer
PQ Codebook

Fig. 3: Index Data Structure

The squared distances $d(u_i(\mathbf{x}), q_i(u_i(\mathbf{y})))^2$ have to be precomputed at the beginning of the search process. For every subvector $u_i(\mathbf{x})$ there are $|C_i|$ distance values to calculate, since $q_i(u_i(\mathbf{y}))$ can be any value of C_i . The distance measure is denoted as asymmetric by [JDS11], since it is defined between quantized and non-quantized vectors as visualized in Fig. 2b. Despite the effort of the preprocessing the technique reduces the computational costs, since the number of index entries in large vector datasets is much higher than $m \cdot |C_i|$, the number of those squared distances. Furthermore, the compact representation makes it easier to provide fast access to the index entries which can also improve performance.

3.3 Inverted Index Structures for Approximated Nearest Neighbor Search

For the standard product quantization search, it is necessary to calculate $|T|$ distance values for every query vector against the target set T . To reduce the number of distance computations and achieve *non-exhaustive* search behavior one can divide the dataset into partitions of vectors called cells which are locally close to each other. After that, only vectors which are in the same cell as the query vector are considered as candidates for the nearest neighbors. One can also extend the search to a certain amount of nearby cells. There are several ways to define the cells: typically, vector quantization is employed by [JDS11] to build the so-called IVFADC index. Here, a cell is defined by the subspace which the quantization function assigns to the same centroid (*Voronoi cell*). In [BBS17] it is stated that either LSH or k-means is used for that. Babenko et al. use product quantization [BL12] to build a fine granular inverted index which is described in detail in Sect. 4.3.

However, these non-exhaustive methods prohibit the search in arbitrary subsets of the index entries which is needed for smaller target sets T . To give an example: if one queries only in a cell of the vector space and the target set is small it is very likely that the index might return an empty set of candidates. To solve this problem we propose an adaptive kNN-Join algorithm which determines a suitable number of cells and provides multiple lookups.

4 Adaptive Search Algorithm for Approximated kNN-Joins

We propose an adaptive search algorithm for kNN-Joins which can cope with arbitrary target sets T . The index structure used by this algorithm is described in Sect. 4.1 and the

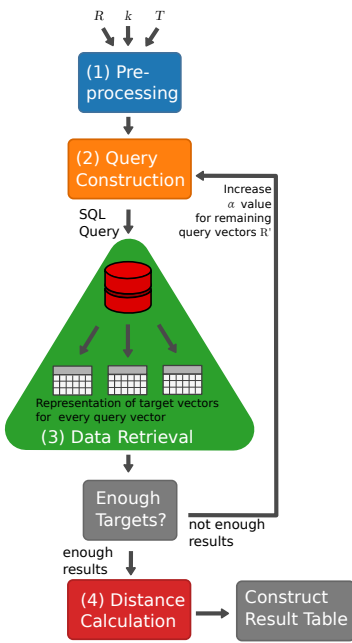
algorithm itself in Sect. 4.2. We employ the inverted multi-index described in Sect. 4.3 that is especially efficient for RDBMSs and propose an approach to estimate the number of targets read out from the inverted index in Sect. 4.3 which is necessary to use it for different target sets. The distance computation is outlined in Sect. 4.4. This is based on product quantization as described in Sect. 3.2. However, for the adaptation to different cardinalities of T , we employ a modification described in Sect. 4.5. Further optimizations are presented in Sect. 4.6.

4.1 Index structure

The data structure of our proposed index for word vectors is shown in Fig. 3. Every index entry in the `Index Data` table has an `id` to reference it (`Entry ID`). In addition, it consists of a `Coarse ID` referring to a partition the vector belongs to for the inverted search and a product quantization sequence (`PQ Sequence`). Every partition has a centroid which is stored in the `Coarse Quantizer` table. The centroids of subvectors for the product quantization are stored in a `Codebook`. Each of those PQ centroids has an `ID` which corresponds to codes in the product quantization sequences and a position $\text{Pos} \in \{1, \dots, m\}$ which refers to the position of the subvector it is calculated for (u_1, \dots, u_m). The original vectors are stored in a `Original Vectors` table.

4.2 Adaptive kNN-Join Algorithm

Fig. 4a shows a flow chart and Fig. 4b the pseudo code of our proposed kNN-Join algorithm. As input parameters, the algorithm gets a set of query vectors $R = \mathbf{r}_1, \dots, \mathbf{r}_n$, the set of target vectors T represented as subset of index entry ids and the desired number k of nearest neighbors. Furthermore, there are two configuration parameters: α and Th_{flex} . The value α determines the minimum number of targets per result that has to be considered for the search process. A higher value of α leads to a higher precision of the result set. The value Th_{flex} configures the calculation of distances with the product quantization which is discussed in detail in Sect. 4.5. The algorithm consists of four steps: At first, there is a *preprocessing step* (Line 3 to 6), which is necessary for the product quantization based distance calculation described in Sect. 3.2. There are two different types of distance calculations via product quantization based on either `SHORT_CODES` or `LONG_CODES`. The first one is suitable for large numbers of distance calculations per query whereas the second one is applicable for fewer distance calculations. Details are provided in Sect. 4.5. Th_{flex} determines the limit of distance calculation where the algorithm switches from `LONG_CODES` to `SHORT_CODES`, whereas the distance calculation depends on $\alpha \cdot k$. The precomputed distance values of subvectors are stored in \mathcal{D}_{pre} . In the *query construction step*, the retrieval of database entries from the inverted index is prepared. This involves the calculation of the coarse quantization C for every query vector \mathbf{r}_i in Line 9 which returns a sequence of the coarse centroid ids from the `Coarse Quantizer` table (see Fig. 3) in decreasing order according to the distance between



(a) Flow Chart of Algorithm

Require:

```

Selectivity:  $\alpha$ 
Threshold Flexible-PQ:  $Th_{flex}$ 
1: function ADAPTIVE-KNN-JOIN( $R, k, T, \alpha$ )
2:    $R' \leftarrow R, j \leftarrow 1$ 
3:   if  $\alpha \cdot k > Th_{flex}$  then  $\triangleright$  only for product quantization
4:      $\mathcal{D}_{pre} \leftarrow \text{PREPROCESSING}(R, T, \text{SHORT\_CODES})$ 
5:   else
6:      $\mathcal{D}_{pre} \leftarrow \text{PREPROCESSING}(R, T, \text{LONG\_CODES})$ 
7:   while  $R' \neq \emptyset$  do
8:     for all  $\mathbf{r}_i \in R$  do
9:        $C^* \leftarrow \text{COARSEQUANTIZE}(\mathbf{r}_i)$ 
10:       $\omega \leftarrow \text{ESTIMATEORDER}(C, T, \alpha \cdot k \cdot j)$ 
11:       $centr(i) \leftarrow \{c \in C \mid order(c) < \omega\}$ 
12:       $query \leftarrow \text{CONSTRUCTQUERY}(centr, T)$ 
13:       $T_{sub} \leftarrow \text{EXECUTE}(query)$ 
14:       $R' \leftarrow \{\mathbf{r}_i \mid |T_{sub}(i)| < \alpha \cdot k\}$ 
15:       $j \leftarrow 2 \cdot j$ 
16:    for all  $\mathbf{r}_i \in R$  do
17:      for all  $t \in T_{sub}(i)$  do
18:         $d \leftarrow \text{DISTFUNC}(\mathbf{r}_i, t, \mathcal{D}_{pre})$ 
19:         $\text{UPDATE}(top_k[\mathbf{r}_i], d)$ 
20:    return  $top_k$ 
    * ordered list of centroid ids
    
```

(b) Pseudo-Code of the Algorithm

Fig. 4: Adaptive kNN-Join Algorithm

the coarse centroid and the query vector \mathbf{r}_i . Every centroid id corresponds to a partition in the index. The number of partitions ω to be considered is estimated by the `ESTIMATEORDER` function (details are given in Sect. 4.3). Based on ω and C the set of the centroids to be retrieved from the index for \mathbf{r}_i is then added to $centr(i)$. After that, a single SQL query is constructed to retrieve the data for all query vectors from the index (*data retrieval step*, Line 13). Then, the query vectors R' for which not enough index entries could be retrieved are determined. For them, another query construction and retrieval iteration is done with a less conservative order estimation (modified by j). If the number of targets is sufficient, the distance values between every query vector \mathbf{r}_i and its respective index entries $T_{sub}(i)$ are calculated by a distance function `DISTFUNC` (*distance calculation step*). We elaborate more on the distance function in Sect. 4.4. The best candidates for the kNN operation are held in a sorted list top_k which is updated after every distance calculation.

4.3 Inverted Multi-Index and Partition Estimation

In Sect. 3.3 we described the advantages of inverted indexing for ANN. However, inverted indexing in general is poorly suitable when the target set T is only a subset of all vectors

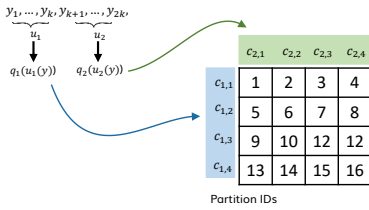


Fig. 5: Inverted Multi Index

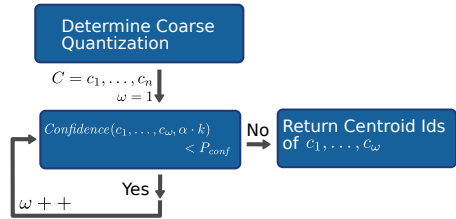


Fig. 6: Confidence Estimation

in the index T_I , i.e. $T \ll T_I$. For this case it is not obvious how many partitions should be considered. To solve this problem, we propose a method to estimate the number of targets observed by the search when a certain number of partitions is read out from the index. This allows us to pick the optimal number of partitions (Fig. 4b Line 11). To optimize the coarse quantization step (Fig. 4b Line 9) we use an inverted multi-index [BL12] which enables fast lookups even if large numbers of index partitions are needed.

Inverted Multi-Index A simple inverted index based on quantization could be created by clustering all possible target vectors T_I into n distinct partitions $P_1 \dot{\cup} \dots \dot{\cup} P_n$ which corresponds to the Voronoi cells of the centroids $\mathbf{c}_1, \dots, \mathbf{c}_n$. To determine the partitions in which to search for a query \mathbf{r} , one has to calculate all the distances $d(\mathbf{r}, \mathbf{c}_1), \dots, d(\mathbf{r}, \mathbf{c}_n)$. However, this could be time-consuming for database queries with a large query set R . To solve this problem, [BL12] propose to use product quantization to obtain more smaller clusters for the partitions with only a few centroids. Suppose the product quantization sequences which serve as labels for the partitions consist of two centroid indexes $c_1, c_2 \in \{1, \dots, n\}$, there are n^2 numbers of partitions (see Fig. 5). But, to determine the nearest clusters one has to calculate only the $2 \cdot n$ square distances between the subvectors of the query vector centroids stored in a codebook. Subsequently, an order of centroids in accordance with the distances to the query vector can be obtained by using the algorithm described in [BL12]. The data structure shown in Fig. 3 is designed for a simple coarse quantizer. If product quantization according to the inverted multi-index is used, the coarse quantization table is replaced by a second codebook relation and the ids c_1 and c_2 are represented by a single id $id_c = c_1 \cdot n + c_2$ in Coarse ID.

Estimation of the Number of Targets The overall objective of the estimation (Fig. 4b Line 10) is to determine a suitable number $\omega \leq n$ of nearest partitions in a way, that the probability P_{est} that it is necessary to run further database requests for the query vector \mathbf{r}_i is lower than a certain value $1 - P_{conf}$. This is done by iteratively incrementing ω until the confidence value obtained by a probabilistic model is higher than P_{conf} (see Fig. 6). The estimation relies on statistics about the distribution of the index. Those contain the

relative sizes of all partitions P_1, \dots, P_n compared to the whole index size (the total number of vectors). For the estimation, we consider the set of all index entries T_I , the target set of the current query T_i and a set of partitions $P_1 \dot{\cup} \dots \dot{\cup} P_\omega = T_p$ which are selected as the partitions with the nearest centroids to the query vector. We then want to estimate the probability $1 - P_{est}$ that T_i contains at least $\beta = k \cdot \alpha$ entries which corresponds to the condition in the algorithm of Fig. 4b in Line 14. It corresponds to the cardinality of $T_i \cup T_p$. For this purpose, we leverage a hypergeometric probability distribution (Eq. (3)) which describes the probability to get s successes by drawing M elements out of a set of N elements without replacement. In our case, s is the desired number of targets in $T_i \cup T_p$, M is the cardinality of T_p and N equals $|T_I|$. The probability of drawing at least β target vectors from the set T_I of all vectors in the index can be calculated with Eq. (4) by using the cumulative distribution function.

$$h(X = s ; |T_I|, |T_i|, |T_p|) = \frac{\binom{|T_p|}{s} \binom{|T_I| - |T_p|}{|T_i| - s}}{\binom{|T_I|}{|T_i|}} \quad (3)$$

$$\mu = |T_i| \cdot \frac{|T_p|}{|T_I|} \quad \sigma^2 = |T_i| \cdot \frac{|T_p|}{|T_I|} \cdot \left(1 - \frac{|T_p|}{|T_I|}\right) \cdot \frac{|T_I| - |T_i|}{|T_I| - 1}$$

$$1 - P_{est} = h_{cdf}(\beta - 1 ; |T_I|, |T_i|, |T_p|) = 1 - \sum_{s=0}^{\beta-1} \frac{\binom{|T_p|}{s} \binom{|T_I| - |T_p|}{|T_i| - s}}{\binom{|T_I|}{|T_i|}} \quad (4)$$

However, because of the complexity of the computation of the binomial coefficients, we have to use an approximation based on the normal distribution Eq. (5). To obtain the approximation, we use the mean μ and the variance σ^2 from the hypergeometric distribution (see Eq. (3)). In contrast to Eq. (3), which involves a cumulative distribution function, here, the number k is a *specific* number of targets which corresponds to $T_i \cup T_p$.

$$N(k; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot \exp \left[-\frac{(k - \mu)^2}{2\sigma^2} \right] \quad (5)$$

The approximation of the probability of getting at least $\beta - 1$ targets (Eq. (6)) is then obtained by its cumulative distribution function. The addition of 0.5 serves as a continuity error correction. It is added to the formula since the hypergeometric distribution is a discrete probability distribution. Since β is an integer value, $k < \beta - 1$ corresponds to $k < \beta - 0.5$.

$$\begin{aligned} h_{cdf}(\beta - 1; \mu, \sigma^2) &\approx 1 - \sum_{s=0}^{\beta-1} N(s; \mu, \sigma^2) \\ &\approx 1 - \frac{1}{2} \cdot \left(1 + \operatorname{erf} \left(\frac{(\beta - 1) + 0.5 - \mu}{\sqrt{2}\sigma} \right) \right) \end{aligned} \quad (6)$$

The probability h_{cdf} of getting enough targets can be increased by raising the number of partitions in T_p which corresponds to the coarse order ω in the algorithm in Line 10 of Fig. 4b. The algorithm chooses ω in a way that it is minimal and h_{cdf} is higher than a certain probability P_{conf} which is also termed as the *confidence value*. From experimental results, we noticed that 0.8 seems to be a good value to achieve high response time for the algorithm.

4.4 Distance Calculation

Beside the adaptive number of partitions and vectors to be considered by the kNN-Join algorithm, the trade-off between precision and runtime also depends on the distance function, namely: (1) *exact calculation*, (2) *product quantization* and (3) *product quantization with post verification*. Product quantization is the fastest one but also provides the lowest precision. It calculates distance values as described in Sect. 3.2. The exact calculation is too slow, especially for a large number of target vectors and a large α . Method (3) strikes a balance between both extremes and therefore represents the default distance function. In the first place it calculates the approximated distance values using product quantization for $k \cdot \alpha$ targets. Second, it refines the $k \cdot pvf$ best candidates with the exact method to obtain the final top-k. Here, the post-verification factor pvf is the major factor which influences the precision of the kNN computation. By adjusting it the user can control the trade-off between precision and runtime as desired in Challenge 3 of Sect. 2. For further details see the evaluation in Sect. 5.2.

4.5 Flexible Product Quantization

The product quantization index provides two parameters: the number of subvectors m and the number of centroids per quantizer $|C|$. The optimal setting of both parameters depends on the desired precision and response time as well as on the typical number of distance calculations $\alpha \cdot k$ which are performed for every query vector. In general, higher values of m and $|C|$ correspond to higher precision and higher response times.

If the target set size $\alpha \cdot k$ is large, the computation of the distances (Line 18) is the most time-consuming step whereas for small target sets, the computation time of the preprocessing step (Line 3) becomes more and more prevalent. Since a low value for m , i.e. a low number of subvectors, corresponds to a faster distance calculation, the product quantization speed for large target sets depends mainly on m . However, with a decreasing number of vectors $\alpha \cdot k$, the preprocessing step (Line 3) has also a high computational effort which is mainly influenced by $|C|$. A decreasing number $|C|$ corresponds to a faster search process.

To be efficient in both situations, we introduce a flexible product quantization search procedure. For product quantization search with a small number of distance calculations $\alpha \cdot k < Th_{flex}$ an index is created with a large number of subvectors $m = 2 \cdot m'$ but only a small number of centroids $|C|$ (see 4). This is called the LONG_CODES mode, since the pq sequences consist of a larger number m of ids. For a larger number of distance calculations,

the number of distances to sum up for each distance calculation (see Eq. (2)) can be reduced by precalculating squared distances for pairs of centroids $\langle \mathbf{c}_j, \mathbf{c}_{j+1} \rangle$ and pairs of subvectors $\langle u_j(\mathbf{r}), u_{j+1}(\mathbf{r}) \rangle$ (Sect. 4.2):

$$d(\langle u_j(\mathbf{r}), u_{j+1}(\mathbf{r}) \rangle, \langle \mathbf{c}_j, \mathbf{c}_{j+1} \rangle)^2 = d(u_j(\mathbf{r}), q_j(u_j(\mathbf{y})))^2 + d(u_{j+1}(\mathbf{r}), q_{j+1}(u_{j+1}(\mathbf{y})))^2 \quad (7)$$

where : $\mathbf{c}_j = q_j(u_j(\mathbf{y})), \mathbf{c}_{j+1} = q_{j+1}(u_{j+1}(\mathbf{y})), j \in \{2 \cdot i | i \in \mathbb{N}\}$

The distance calculation can then be expressed by the following equation:

$$\hat{d}(\mathbf{r}, \mathbf{y})^2 = \sum_{j=1}^{m'} d(\langle u_{2j-1}(\mathbf{r}), u_{2j}(\mathbf{r}) \rangle, \langle \mathbf{c}_{2j-1}, \mathbf{c}_{2j} \rangle)^2 \quad (8)$$

To efficiently calculate this, the product quantization sequences consisting of m numbers can be transformed into sequences of $m' = \frac{m}{2}$ numbers. Therefore, all centroid id pairs $\langle id(\mathbf{c}_j), id(\mathbf{c}_{j+1}) \rangle$ can be transformed into single ids:

$$id(\mathbf{c}_j, \mathbf{c}_{j+1}) = id(\mathbf{c}_j) \cdot |C| + id(\mathbf{c}_{j+1}) \quad (9)$$

This is called the `SHORT_CODES` mode. For a kNN-Join, this transformation process has to be done only once irrespective of the number of queries (see Fig. 4b Line 3 to 6). Optimal settings for the threshold Th_{flex} are discussed in Sect. 5.4.

4.6 Optimizations

Target List for Product Quantization Search A naïve way of doing the distance calculation via product quantization might be to calculate the distances directly by iterating through the targets instead of collecting the targets as it is done in Fig. 4b in Line 13. However, to execute product quantization efficiently it is important that the precomputed distances stay in the cache. Since the precomputed distances are specific for the query, it is necessary to collect all product quantization sequences and assign them to the query vectors in the first place. Afterward, the distance computation can be done query-wise. So, all precomputed distances specific for a query can stay in the cache. Moreover, the approach of [AKLS15] could be used to further improve memory locality to speed up the product quantization search. Thereby, product quantization sequences are compressed to fit into SIMD cache lines.

Prefetching As stated in Sect. 4.4, we collect the targets in Line 13 of the search algorithm (Fig. 4b) and assign them to the query vectors they should be compared to. This requires a lot of random memory accesses to the lists of targets. To speed up this step, we prefetch the target list entries which has to be updated next from time to time. We tested the effect of the prefetching with our algorithm on a query with 10,000 queries and 100,000 targets (300-dimensional vectors) and $\alpha = 100$ and $k = 10$. For this query the construction time of the target list could be reduced by $\approx 35\%$, from 1.4 seconds to 0.9 seconds. For more

details about that one can take a look at the code³.)

Fast Top-K Update In Line 19 of the algorithm in Fig. 4b the top_k gets updated after every distance calculation. If the distance value is lower than every other index entry, the new index entry has to be inserted into this array of current nearest neighbors. However, this can be time-consuming since every other array element with a larger distance has to be moved. For a large top_k , the updates can be accelerated by first adding new candidates in a buffer. If this buffer gets full or all distance calculations are done, all candidates are added to the top_k in one run. This is in particular useful if post verification (see Sect. 4.4) should be done and thus a large set of candidates is required in the first place. Alternatively, one can use a linked list instead of an array for the top_k . However, linked lists are space consuming which could become a problem for large query sets.

5 Evaluation

In this section, we first evaluate our adaptive kNN-Join implementation for varying post-verification factors and α values and compare them to the basic batch-wise product quantization approach (see Sect. 5.2). Moreover, we provide a detailed runtime investigation for the different sub-routines of the kNN-Join given different query and target set sizes (Sect. 5.3). The impact of short and long code sizes on the precomputation and distance calculation is shown in Sect. 5.4. In Sect. 5.5, we finally evaluate the accuracy of the target size estimator that was presented in Sect. 4.3.

5.1 Experimental Setup

We use two different datasets of word embeddings to evaluate our approach, the popular Google News dataset⁴ which is trained with the word2vec [Mi13] skip gram model and a dataset trained on data from Twitter⁵ with GloVe [PSM14]. We use python scripts to create the index structures for these datasets as shown in Tab. 1. The kNN-Join, that can be used for queries similar to the example in Fig. 1, is implemented as a user-defined function.

The index consists of entries with an entry_id and a product quantization sequence as well as a codebook storing the centroids. As a baseline, we use the exhaustive product quantization search as described in [JDS11], which can easily be generalized to a kNN-Join operation. Basically, it makes no use of inverted indexing and thus calculates approximated distance values between any query vector in R and any target vector in T to determine the kNN results. The method can simply reuse the index data table and the codebook of our adaptive index (Page 6 Fig. 3) while ignoring the Coarse ID column. To make the comparison fair we implemented a batch-wise search algorithm as UDF, like it is done for the

³ https://github.com/guentermi/postgres-word2vec/blob/master/freddy_extension/ivpq_search_in.c

⁴ <https://drive.google.com/file/d/0B7XkCwpI5KDYN1NUTT1SS21pQmM/edit?usp=sharing>, last access: 15.08.18

⁵ <https://nlp.stanford.edu/projects/glove/>, last access: 15.08.18

	Google News (GN)	Twitter (TW)
Size	3,000,000	1,193,514
Dimensionality	300	100
Coarse Centroids	$2 \cdot 32$	$2 \cdot 20$
Product Quantization	$m = 30, C ^* = 32$	$m = 10, C ^* = 32$
Confidence	$P_{conf} = 0.8$	$P_{conf} = 0.8$
Threshold (For Flexible Product Quantization)	$Th_{flex} = 15,000$	$Th_{flex} = 15,000$

* number of centroids for each quantizer

Tab. 1: Dataset and Index Characteristics

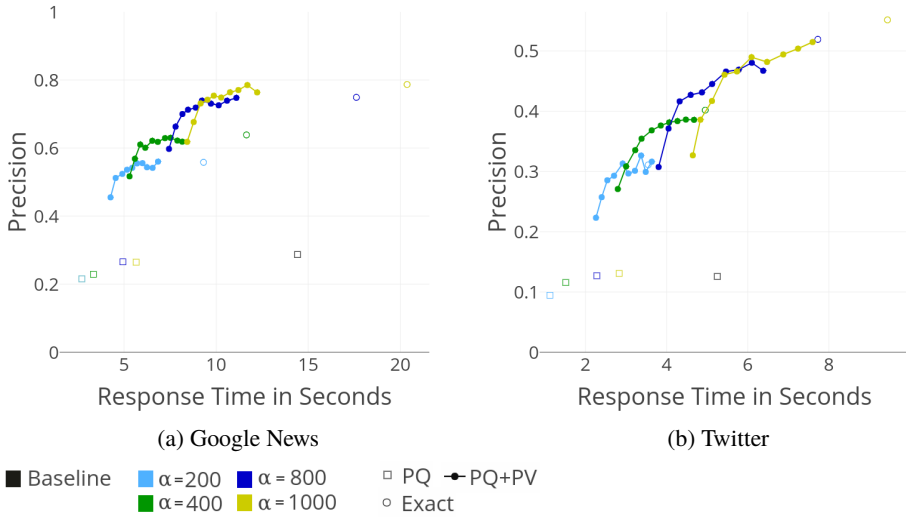


Fig. 7: Evaluation of Execution Time and Precision

adaptive search algorithm. To enable repeatability we have published the implementation⁶. The machine we run the evaluation on is a Lenovo ThinkPad 480s with 24GB main memory, an Intel i5-8250U CPU (1.6GHz) and a 512GB SSD. The computation runs only on a single core in a PostgreSQL instance on a Ubuntu 16.04 Linux System.

5.2 Influence of Index Parameters on Precision and Execution Time

In Fig. 7, the execution time and precision curves for different α values and increasing pvf values are shown. All the kNN-Joins are executed on 5,000 query and 100,000 target vectors with $k = 5$. The post verification factors used for the computation are 10, 20, ..., 100. The precision is determined by calculating the amount of nearest neighbor results of a query vector which concur with the exact results relative to the number of k . Since doing the exact

⁶ <https://github.com/guenthermi/postgres-word2vec>

calculation for all query vectors of a kNN-Join is very time-consuming, we draw bootstrap samples of the query vectors of size 100 to derive an estimation of the actual precision value by determining the precision of the samples results. The measurements for every configuration are done 20 times and the median values are determined. The value of $\alpha \cdot k$ is always lower than Th_{flex} . Thus, the LONG_CODES method is used.

As one can see, for most of the chosen values of pvf and α the adaptive search with PQ distance calculation has the shortest execution time and also outperforms the product quantization baseline method in terms of precision and runtime. Join operations with exact distance calculation have significantly longer execution times than the other methods, however achieving the highest precision value. Nevertheless, the post verification might be the better choice in most of the cases, since it achieves high precision values while being much faster than the exact computation. For increasing values of pvf the execution time, but also the precision generally increases. Regarding the α values, one can also observe that higher values lead to higher precision values at the expense of execution time.

The post verification method is significantly slower than the product quantization method, even though pvf has a low value. This is the case since the use of post verification requires to at least calculate k exact distance values for each query vector. Furthermore, it needs to retrieve the raw vector data for every target vector which has to be considered for distance calculation. Moreover, it is necessary to hold these vectors in memory until the distance computation starts. During the distance computation, the vectors of the currently best candidates have to be stored together with the product quantization sequences in a separate TopK list to apply the post verification step later. For high values of pvf , on one hand, the post verification step gets time-consuming while on the other hand more updates of the TopK lists are required during the distance calculation step.

5.3 Performance Measurements

We evaluate the performance of the search algorithm by measuring the execution time of certain subroutines of the algorithm denoted by numbers 1 to 4 in Fig. 4a. This is done for different cardinalities of query sets R and target sets T . The query and target vectors are sampled from the whole set of word embeddings of the Google News dataset. The results of our measurements are shown in Fig. 8 for different values of $|R|$ and $|T|$. For the measurements we set $\alpha = 100$, $pvf = 10$ and used a fixed target set size of 10,000 while increasing $|R|$ and a query set size of 10,000 while increasing $|T|$. All measurements are done five times and the average value is determined. The distance computation time increases with the query set size as well as with the number of target vectors. The query construction time only increases with an increasing query set size. If the query set size is fixed the query construction time slightly decreases with increasing target set size because a higher number of partitions has to be determined for every query vector in case the number of targets is very low. The main effort during the query construction is the calculation of the coarse quantization for every query vector. Since this process does not change with the number of target vectors, the execution time is rather constant. The data retrieval time

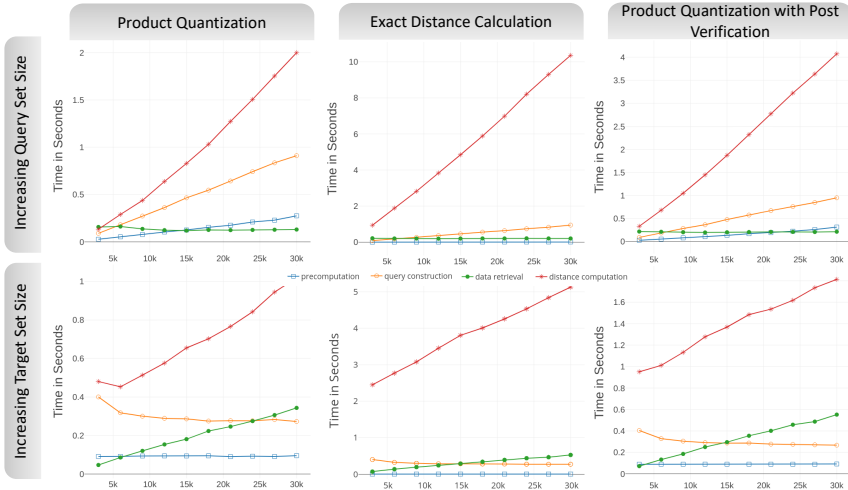


Fig. 8: Time Measurement for increasing sizes of query set R and target set T

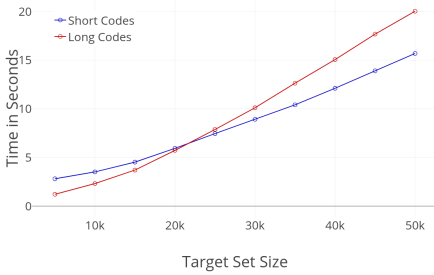
effort is nearly constant for an increasing number of query vectors while its execution time increases if the target vector set grows. The preprocessing has to be done per query vector. Therefore only the query set size influences its execution time.

5.4 Flexible Product Quantization

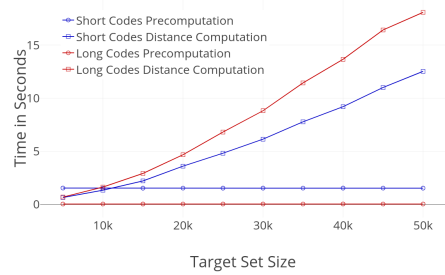
Flexible product quantization (Sect. 4.5) can adjust the product quantization distance calculation to smaller or bigger sets of vectors. The product quantization sequences in our index structure for the Google News dataset consist of codes $c_i \in \{0, \dots, 31\}$ of length 30 which can be combined to shorter codes $c'_i \in \{0, \dots, 1023\}$ with length 15. The overall execution time of a kNN-Join with product quantization distance calculation is shown for both methods in Fig. 9a. Fig. 9b shows the execution times for the precomputation and distance calculation step. We use query sets of size 5,000. The target set size $|T|$ is shown on the x axis. The value α is set to $\frac{|T|}{2 \cdot k}$. The measurements are done 10 times with randomly sampled query and target vectors and average values are determined. For small target sizes with $|T| \leq 20,000$ the computation via long codes is faster. For larger target sets the overhead of the distance calculation for long codes becomes prevalent such that the calculation with short codes is faster.

5.5 Accuracy of the Target Size Estimation

The number of targets determined in the retrieval step of the algorithm before the distance calculation can be estimated. For this purpose we leverage an approximation of the



(a) Execution Times of Both Methods



(b) Precomputation and Distance Computation

Fig. 9: Evaluation of Short and Long Codes Calculation

hypergeometric distribution as described in Sect. 4.3. The estimated number of targets derived from the index is μ as defined in Eq. (3). In Fig. 10a, a scatter plot of the estimated and actually derived number of targets is shown. For these measurements, kNN-Joins with a single randomly sampled query vector are executed and the number of targets obtained in the first retrieval step is determined inside the user-defined function. This was done for all $\alpha \in \{1, \dots, 100\}$, $k = 5$ and target sets of size $|T| = 1,000$. For each α value 10 queries are executed. The divergence of the estimation is higher if the desired number of targets per query vector gets higher. This can be noticed in the 4th-grade polynomial regression curve of the sample points in Fig. 10b. However, if the number of desired targets is near to $|T|$ it is apparently decreasing.

To prevent the system from executing a lot of database queries, one can adjust the confidence value P_{conf} . It represents how likely it is that only one database request is sufficient to derive the desired number of targets from the index. This was also evaluated by single query kNN-Joins with $\alpha = 10$ and the same search parameters as in the last experiment. The amount of queries where the condition is true (only one request was required) in relation to P_{conf} is shown in Fig. 10c. For each confidence value $P_{conf} \in \{0.05 \cdot i | i = 1, \dots, 20\}$ 1,000 queries are executed. As desired, the amount of queries where the condition is true rises up to 100%, if the confidence value increases up to 1. However, the actual confidence is quite higher than P_{conf} , since the confidence can only be increased step-wise by incrementing ω .

6 Related Work

There is already limited work done in integrating kNN operations in database systems. For instance, PostgreSQL can be extended by PostGIS [Po18] which allows running kNN queries for low dimensional (geographical) data. Index structures can be created with GiST (Generalized Search Trees) to speed up such operations. An integration of vector similarity search for high dimensional data into Spark has been done by [BBS17] for word embeddings. Here, LSH [Ch02] or spherical k-means [DM01] is used to partition

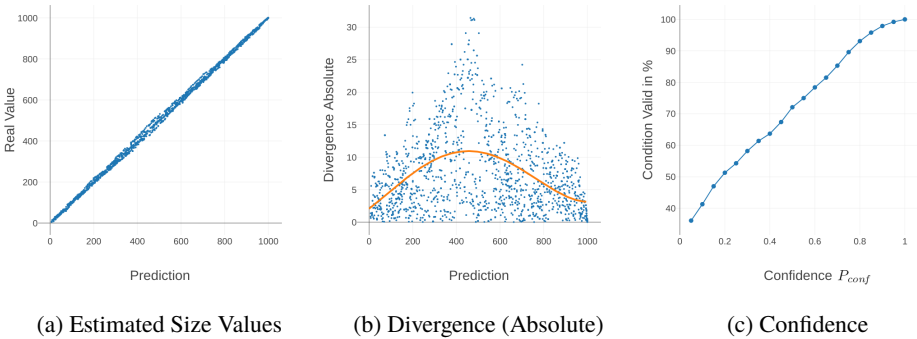


Fig. 10: Estimation of Target Set Size

vectors for filtering. However, the actual distance calculation might be done with exact methods. We proposed FREDDY [Gü18] which supports approximated kNN queries for high dimensional data, however, can not efficiently execute kNN-Join operations. A system called ADAM_{pro} [GAKS14] adds approximated kNN-Search techniques on top of a database system for multimedia retrieval. Furthermore, also approximated kNN-Joins are already integrated into a relational database system by [YLK10]. However, this is only applicable for low dimensional data. This work differs from previous work in the way that it employs state-of-the-art approximated nearest neighbor search techniques to support approximated kNN-Joins also for high dimensional data. To do so several modifications and optimization specific for kNN-Joins in RDBMSs have been done.

Approximated Nearest Neighbor Search The difficulty to find the nearest neighbors especially in high dimensional vector spaces has led to the development of several kinds of approaches for approximated nearest neighbor (ANN) search. However, not all of them are applicable for kNN-Join operations in RDBMSs. On one hand, recently graph-based methods are developed [Ha11] which are fast, however, do not allow online index updates. On the other hand, there are so-called cell probe methods which divide the search space into several cells. One of the most popular ones E2LSH [Da04] applies locality sensitive hashes (LSH) to achieve such a partitioning. Jegou et al. [JDS11] employ product quantization for partitioning. Additionally, their approach can be combined with inverted indexing similar to [SZ03] for even faster search. One advantage of product quantization is, that it is easy to add vectors during runtime in an online update manner which is particularly useful for the application in relational database systems. Thus, we based our index on such quantization techniques. In the context of word embeddings, vectors for text values of multiple tokens can be added during runtime by an averaging method [CJ15]. For such updates, the quantizations of the new vectors have to be calculated. Vectors can be removed by simply deleting the quantization entries from the index. For frequent inserts and deletions, the online product quantization approach proposed by [XTZ18] suggests

updating the centroids of the quantizer functions. In this way, the index can react to context drifts of its containing data.

7 Conclusion

We propose a novel index structure for approximated kNN-Joins on high dimensional data which is flexible enough to deliver good performance on different cardinalities for query and target vector set. It enables non-exhaustive search for different target sets. We have shown that the proposed index structure can achieve faster response times than product quantization as an instance of a state-of-the-art exhaustive search method. We provide a practical implementation of the operator and our optimizations in PostgreSQL.

Future Work By adjusting the post verification factor pvf one can influence the precision and response time of the kNN-Join. However, there are further search parameters, in particular α , which have an influence here. It might be hard for a user to find out which parameter configuration leads to a specific precision and response time. Thus, an oracle which can provide the optimal parameters to achieve a good precision by fulfilling constraints regarding the execution time would improve the usability of the system.

Acknowledgments

This work is funded by the German Research Foundation (DFG) within the Research Training Group “Role-based Software Infrastructures for continuous-context-sensitive Systems” (GRK 1907) and by the Intel® AI Research.

References

- [AKLS15] André, Fabien; Kermarrec, Anne-Marie; Le Scouarnec, Nicolas: Cache Locality is Not Enough: High-performance Nearest Neighbor Search with Product Quantization Fast Scan. Proc. of the VLDB Endowment, 9(4):288–299, 2015.
- [BBS17] Bordawekar, Rajesh; Bandyopadhyay, Bortik; Shmueli, Oded: Cognitive Database: A Step towards Endowing Relational Databases with Artificial Intelligence Capabilities. CoRR, abs/1712.07199, 2017.
- [Be99] Beyer, Kevin; Goldstein, Jonathan; Ramakrishnan, Raghu; Shaft, Uri: When Is “Nearest Neighbor” Meaningful? In: Proc. of the 7th International Conference on Database Theory. Springer, pp. 217–235, 1999.
- [BL12] Babenko, Artem; Lempitsky, Victor: The Inverted Multi-Index. In: 2012 IEEE Conference on Computer Vision and Pattern Recognition. IEEE, pp. 3069–3076, 2012.

- [Ch02] Charikar, Moses S: Similarity Estimation Techniques from Rounding Algorithms. In: Proc. of the 34th Annual ACM Symposium on Theory of Computing. ACM, pp. 380–388, 2002.
- [CJ15] Campr, Michal; Ježek, Karel: Comparing Semantic Models for Evaluating Automatic Document Summarization. In: International Conference on Text, Speech, and Dialogue. Springer, pp. 252–260, 2015.
- [Da04] Datar, Mayur; Immorlica, Nicole; Indyk, Piotr; Mirrokni, Vahab S: Locality-Sensitive Hashing Scheme Based on p-stable Distributions. In: Proc. of the 20th Annual Symposium on Computational Geometry. ACM, pp. 253–262, 2004.
- [DM01] Dhillon, Inderjit S; Modha, Dharmendra S: Concept Decompositions for Large Sparse Text Data Using Clustering. *Machine Learning*, 42(1-2):143–175, 2001.
- [GAKS14] Giangreco, Ivan; Al Kabary, Ihab; Schuldt, Heiko: ADAM - A Database and Information Retrieval System for Big Multimedia Collections. In: 2014 IEEE International Congress on Big Data. IEEE, pp. 406–413, 2014.
- [Gr84] Gray, Robert: Vector Quantization. *IEEE ASSP Magazine*, 1(2):4–29, 1984.
- [Gü18] Günther, Michael: FREDDY: Fast Word Embeddings in Database Systems. In: Proc. of the 2018 International Conference on Management of Data. ACM, pp. 1817–1819, 2018.
- [Ha11] Hajebi, Kiana; Abbasi-Yadkori, Yasin; Shahbazi, Hossein; Zhang, Hong: Fast Approximate Nearest-Neighbor Search with k-Nearest Neighbor Graph. In: Proc. of the 22nd International Joint Conference on Artificial Intelligence. volume 22, p. 1312, 2011.
- [JDS11] Jegou, Herve; Douze, Matthijs; Schmid, Cordelia: Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2011.
- [LG14] Levy, Omer; Goldberg, Yoav: Linguistic Regularities in Sparse and Explicit Word Representations. In: Proc. of the 18th Conference on Computational Natural Language Learning. pp. 171–180, 2014.
- [Mi13] Mikolov, Tomas; Sutskever, Ilya; Chen, Kai; Corrado, Greg S; Dean, Jeff: Distributed Representations of Words and Phrases and their Compositionality. In: *Advances in Neural Information Processing Systems*. pp. 3111–3119, 2013.
- [Po18] PostGIS, postgis.net, Last Access: 06.09.2018.
- [PSM14] Pennington, Jeffrey; Socher, Richard; Manning, Christopher D.: GloVe: Global Vectors for Word Representation. In: *Empirical Methods in Natural Language Processing (EMNLP)*. pp. 1532–1543, 2014.
- [SZ03] Sivic, Josef; Zisserman, Andrew: Video Google: A Text Retrieval Approach to Object Matching in Videos. In: Proc. of the 9th IEEE International Conference on Computer Vision-Volume 2. IEEE Computer Society, p. 1470, 2003.
- [XTZ18] Xu, Donna; Tsang, Ivor; Zhang, Ying: Online Product Quantization. *IEEE Transactions on Knowledge and Data Engineering*, 2018.
- [YLK10] Yao, Bin; Li, Feifei; Kumar, Piyush: K Nearest Neighbor Queries and kNN-Joins in Large Relational Databases (Almost) for Free. In: 2010 IEEE 26th International Conference on Data Engineering (ICDE 2010). IEEE, pp. 4–15, 2010.

Machine Learning

In-Database Machine Learning: Gradient Descent and Tensor Algebra for Main Memory Database Systems

Maximilian Schüle,¹ Frédéric Simonis,² Thomas Heyenbrock,³ Alfons Kemper,⁴
Stephan Günemann,⁵ Thomas Neumann⁶

Abstract: Machine learning tasks such as regression, clustering, and classification are typically performed outside of database systems using dedicated tools, necessitating the extraction, transformation, and loading of data. We argue that database systems when extended to enable automatic differentiation, gradient descent, and tensor algebra are capable of solving machine learning tasks more efficiently by eliminating the need for costly data communication.

We demonstrate our claim by implementing tensor algebra and stochastic gradient descent using lambda expressions for loss functions as a pipelined operator in a main memory database system. Our approach enables common machine learning tasks to be performed faster than by extended disk-based database systems or as well as dedicated tools by eliminating the time needed for data extraction. This work aims to incorporate gradient descent and tensor data types into database systems, allowing them to handle a wider range of computational tasks.

1 Introduction

Applications in the increasingly important domains of machine learning, data mining, and statistical analysis often use multi-tier architectures. This needlessly impedes the knowledge discovery process by separating data management from the computational tasks. The lowest tier consists of a database system that, often undervalued and considered as a basic data storage, typically uses SQL as established unified declarative data manipulation language.

When analyzing a taxi dataset, the database system continuously receives tuples containing taxi trip data. The data then has to be extracted so it can be analyzed in tools like Caffe, MATLAB, R, Spark, TensorFlow, or Theano. Even disregarding the various interfaces involved, we cannot avoid the expensive extract, transform, load (ETL) process needed to pull the data out of the data management tier. As modern main memory database systems—which combine online analysis processing (OLAP) and online transactional processing

¹ TU Munich, Chair for Database Systems, Boltzmannstraße 3, 85748 Garching, m.schuele@tum.de

² TU Munich, Chair for Database Systems, Boltzmannstraße 3, 85748 Garching, simonis@in.tum.de

³ TU Munich, Chair for Database Systems, Boltzmannstraße 3, 85748 Garching, thomas.heyenbrock@tum.de

⁴ TU Munich, Chair for Database Systems, Boltzmannstraße 3, 85748 Garching, kemper@in.tum.de

⁵ TU Munich, Chair for Database Systems, Boltzmannstraße 3, 85748 Garching, guennemann@in.tum.de

⁶ TU Munich, Chair for Database Systems, Boltzmannstraße 3, 85748 Garching, neumann@in.tum.de

(OLTP) [FKN12]—already allow data mining beside real time analysis, the integration of machine learning algorithms represents the next step of moving computation to the data.

Modern machine learning essentially transforms data in tensors until a loss function is applicable. These machine learning models are to the vast majority trained using gradient descent. We believe that integrating a gradient descent optimizer in the database system will encourage data scientists to perform more computational tasks within database systems. In the taxi data example, a gradient descent operator would allow online optimization using the latest data as soon as it is received without the need to extract or transfer it.

The main contribution of this work is to provide an architectural blueprint for integrating gradient descent into modern main memory database systems. To achieve that, we need an automatic differentiation framework, an architecture for fitting the optimizer into a database system’s query plan, a tensor data type, and a user-friendly language extension. We design gradient descent as a new relational operator that uses SQL lambda functions [Pa17] to specify model functions. When extending SQL, an already standardized and established language, by gradient descent and tensor algebra, we only require one new operator and one new data type, enabling existing database systems’ interfaces to be reused without the need of learning a new language.

The rest of this paper is structured as follows: First, after summarizing the existing research, we describe our automatic symbolic differentiation framework and the gradient descent operator together with the proposed tensor algebra to be integrated into database systems. Having implemented these operators, we then explain how they can be used for machine learning tasks. Finally, we compare the performance of our enhanced database management system with that of other machine learning systems using the Chicago taxi rides dataset.

2 Related Work

This section describes current state-of-the-art machine learning tools, which will provide a baseline in the later evaluation, as well as similar automatic differentiation and parallel gradient descent frameworks, and the current state of research on moving data analytics into database systems, which we continue in this work.

Dedicated machine learning tools, such as Google’s TensorFlow [Ab16] and Theano [Ba12], support automatic differentiation to compute the gradient of loss functions as part of their core functionality. As they are external to the database system, they require the extraction and transfer of data. To integrate gradient descent, we only need the automatic differentiation component of the tools. Here, we use TensorFlow’s expression tree for loss functions as a reference for the automatic gradient differentiation. Also, we use TensorFlow as a baseline in our experiments.

Automatic differentiation methods can be divided into numeric (approximating the derivative) or symbolic differentiation; that again can be split into forward and backward accumula-

tion [BPR15] depending on the direction in which the chain rule is applied. As existing gradient differentiation tools do not allow for tensor derivatives, Laue et. al. [LMG18] proposed a framework for automatically differentiating higher order matrices and tensors. This is based on the Ricci calculus and allows both forward and backward accumulation. The framework was built with the same aim as ours but—as their source code has not been published—we had to prototype a similar one.

If database systems are to support tensor derivatives, they will need a tensor data type. The work of Luo et al. [Lu17] integrated linear algebra based on matrices and vectors into a relational database system in the way we need. It demonstrated database systems to be an “excellent platform“ for distributed and scalable computations and also that integrated array or matrix data types are superior to their table representations. To execute linear algebra in database systems, Kernert [Ke16] deeply integrated linear algebra functionalities into an in-memory database system and adapted dense and sparse matrices to a column-oriented storage layer.

The MADlib [He12] analytics library extends existing database systems such as PostgreSQL to perform data mining and statistics tasks. It provides a matrix data type and table functions for linear and logistic regression, which we use as another baseline. EmptyHeaded [Ab17] and HyPer [Pa17] integrate data mining algorithms into database systems as operators. Whereas EmptyHeaded compiles algorithms and relational algebra together, HyPer provides data mining operators as part of its relational algebra. This work extends its relational algebra for machine learning. Another architecture for scalable analytics is XDB [Bi14] that unifies transaction guaranties from database systems, partitioning, and adaptive parallelization strategies for analytical workloads. Vizdom [Cr15] is an interface for Tupleware [CGK14] for visually sketching machine learning workflows. Both approaches also intend to combine the benefits of database and of analytics systems.

For the vision of a declarative framework for machine learning, Kaoudi et al. [Ka17] adapt database systems’ optimizers for gradient descent computation. Using a cost function it chooses a computation plan out of stochastic, batch, or mini-batch gradient descent. Another notable automatic differentiation framework is Hogwild [Re11]. It presents an approximative way of lock-free parallel stochastic gradient descent without synchronization.

3 In-Database Gradient Descent

This paper’s main contribution is to present a gradient descent operator that can be used within a database system to reduce data transfers. This means we need a way to represent loss functions in SQL, so we first introduce some necessary mathematical background and define the terms used in our implementation. As well as integrating the resulting operator into the query plan, we also need a framework for computing the gradient of a given loss function, so we introduce our prototype framework for automatic gradient differentiation based on partial derivatives. We conclude by integrating gradient descent into the relational

algebra via two operators, one for gradient descent (used for the training dataset) and one for labeling (used to label test datasets). Finally, we demonstrate how these operators fit into the pipeline concept used by modern database systems.

3.1 Mathematical Background

Many machine learning tasks can be expressed as minimizing a mathematical function. Parametrized *model functions* $m_{\vec{w}}(\vec{x})$ such as linear combination are used to predict the label y (on data \vec{x} and weights \vec{w} with m elements each). A *loss function* $l_{X,y}(\vec{w})$ measures the deviation (*residual*) on all n datasets X of the predicted values $m_{\vec{w}}(X)$ from the actual labels \vec{y} , for example, mean least squares:

$$m_{\vec{w}}(\vec{x}) = \sum_{i \in m} x_i * w_i \approx y,$$

$$l_{X,\vec{y}}(\vec{w}) = \frac{1}{n} \sum_{i \in n} (m_{\vec{w}}(\vec{x}_i) - y_i)^2.$$

Gradient descent, a numerical method, finds the loss function's minimum by moving in the direction of the steepest descent, which is given by $-\nabla l_{X,\vec{y}}(\vec{w})$. Starting with arbitrary initial weights \vec{w}_0 , each step updates these values by the gradient, multiplied by the scaled *learning rate* α , until the minimum is reached:

$$\vec{w}_{i+1} = \vec{w}_i - \frac{\alpha}{i} * \nabla l_{X,\vec{y}}(\vec{w}_i),$$

$$\vec{w}_{\infty} \approx \{\vec{w} | \min(l_{X,\vec{y}}(\vec{w}))\}.$$

Batch gradient descent considers all tuples $\vec{x} \in X$ per iteration step, whereas *stochastic gradient descent* takes one tuple for each step:

$$l_{\vec{x},y}(\vec{w}) = (m_{\vec{w}}(\vec{x}) - y)^2,$$

$$\vec{w}_{i+1} = \vec{w}_i - \frac{\alpha}{i} * \nabla l_{\vec{x},y}(\vec{w}_i).$$

3.2 Automatic Gradient Differentiation

As we need an automatic differentiation framework to compute the gradients, we develop a framework for automatic tensor differentiation based on expression trees. Our database system uses this as the computational core of the gradient descent operator but is also suited for other applications. For that purpose, we have made the source code online⁷.

⁷ <https://gitlab.db.in.tum.de/MaxEmanuel/autodiff>

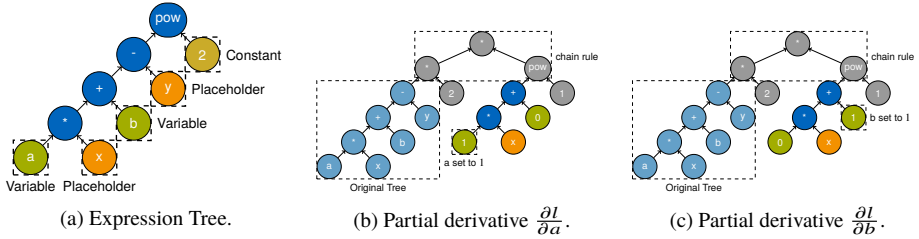


Fig. 1: Expression tree for (a) the loss function and the partial derivatives (b) $\frac{\partial l}{\partial a}$ and (c) $\frac{\partial l}{\partial b}$, used to compute the gradient: the nodes (blue) represent operations, the variables (green) represent weights or parameters to be optimized and therefore used for the derivatives, the placeholders are replaced by the incoming tuples, and constants (ocher) represent a fixed value.

The framework expects the loss function as input and builds an expression tree out of it. The expression tree allows to derive symbolically when optimizing the loss function by gradient descent. The gradient descent operator (introduced below) passes the loss function to the framework, obtaining the gradient in return. Since the operator forms part of the relational algebra, it can consume the input tuples natively as it evaluates the gradient to update the loss function’s weights at each iteration step.

The advantages of integrating gradient descent into the database system are that computationally intense tasks can then run on the database server and data transfers are reduced.

3.2.1 Basics

To carry out automatic gradient differentiation, we first build an expression tree based on the given loss function. This tree consists of *placeholders*, later replaced by the incoming tuples, *variables* representing the weights to be minimized, *constants* for constant numbers, and operation *nodes* (see Fig. 1a). Given a model function $m_{a,b}(x) = a * x + b$ that has the variables a, b and the placeholder x the loss function is:

$$l_{\vec{x},y}(a, b) = (a * x_i + b - y)^2.$$

After building the expression tree, we compute the partial derivations symbolically and combine them into a single gradient, based on the partial derivatives of all the variables:

$$\nabla l_{\vec{x},y}(a, b) = \begin{pmatrix} \partial l / \partial a \\ \partial l / \partial b \end{pmatrix} = \begin{pmatrix} 2(ax_i + b - y) * x_i \\ 2(ax_i + b - y) * 1 \end{pmatrix}.$$

3.2.2 Node Evaluation

The framework provides functions to evaluate single or multiple expression tree nodes. Nodes are implemented as structs, consisting of a numerical operator identifier and the child node identifiers. A compile-time registry stores the node types corresponding to each operator identifier as a binary function. To evaluate a node, the framework needs to handle node types listed above; *variables* and *constants* evaluate to their value, *placeholders* return the values of the corresponding entries of the placeholder mapping, while *operators* are evaluated by fetching all their child nodes, applying the appropriate evaluator function (drawn from the operator registry), and returning the result.

An internal associative cache with node identifiers as keys stores information about previously evaluated nodes. This enables us to avoid unnecessary work by reusing cached values when possible, only fully evaluating nodes (and adding the results to the cache) when there is a cache miss.

3.2.3 Deriving Nodes

The framework takes symbolic derivatives of some *operator* nodes (i.e., addition, multiplication, subtraction, and division) directly, by setting their respective *variable* nodes to one and the other variables to zero in the expression tree. We handle other operator nodes, such as the power function, according to the chain rule. Here, the unchanged term is implemented as a reference to the corresponding node of the original expression tree. In this way, we can calculate the derivatives of arbitrary-degree polynomials recursively. We use forward accumulation to compute the derivatives of multiple variables, thereby generating an expression tree for each derivative (see Fig. 1b, 1c).

When computing the gradient, multidimensional variables are treated as multiple single variables, with the derivatives being generated by combining the partial derivatives with respect to each variable into one gradient vector.

3.3 Integration in Relational Algebra

Now, we turn to this paper's main contribution, namely the creation of an in-database gradient descent operator. In this section, we specify the loss function using lambda functions and embed the resulting operator into the database system.

To begin, let us return the taxi data example. Here, the database stores data about previous trips, including information about the tip amount and how the fare was paid. This may be used as training data for assessing future taxi rides (i.e., as a test dataset) and hence may need to be labeled. Since data is added frequently, we do not have time to extract the

```

create table trainingdata (x float, y float); create table weights(a float, b float);
insert into trainingdata ... insert into weights ...

select * from gradientdescent(
   $\lambda(\text{data}, \text{weights}) (\text{weights.a} * \text{d.x} + \text{weights.b} * \text{d.y})^2$ , -- the loss function is specified as  $\lambda$ -expression
  (select x,y from trainingdata d), (select a,b from weights), -- training set and initial weights
  0.05, 100); -- learning rate and max. number of iteration

```

List. 1: Gradient descent operator using a lambda expression as loss function.

necessary data but still want to work on the most up-to-date information. We will need both a gradient descent operator (to minimize the parametrized loss function) and a labeling operator if we are to assess future rides.

First, we demonstrate how lambda functions can be used to express user-specified loss functions in SQL, then we explain how to integrate the operators needed for gradient descent into the relational algebra. Finally, we explain the concept of pipelines in database systems and explain how our operator enables parallelism.

3.3.1 Lambda Functions

Lambda functions are anonymous SQL expressions used “to inject user-defined code“ [Hu17] into analytics operators. Originally developed to parametrize distance metrics in clustering algorithms or edge weights in the PageRank algorithm, lambda functions are expressed inside SQL queries and allow “variation points“ [Pa17] in otherwise inflexible operators. In this way, lambda expressions broaden the applications of the default algorithms without the need to modify the database system’s core. Furthermore, SQL with lambda functions substitutes any new query language, offers the flexibility and variety of algorithms needed by data scientists, and ensures usability for non-expert database users. In addition, lambda functions allow user-friendly function specification, as the database system automatically deduces the lambda expressions’ input and output data types from the previously defined table’s attributes.

Here, we use SQL lambda expressions to specify the loss function whose gradient we wish to compute. Given two relations, $R\{[a, b]\}$ (containing the initial weights) and $S\{[x, y]\}$ (containing the data), the loss function (discussed above) can be specified, for example, as the following linear combination:

$$\lambda(R, S)(R.a * S.x + R.b - S.y)^2.$$

In this way, we can specify arbitrary loss functions in SQL (see List. 1). Lambda functions allow functions to be defined without providing further informations. The relation’s attributes implicitly define the *placeholder* or *variable* nodes’ type. There is no need to redefine the variable size, and we can also take the derivatives of higher-dimensional tensors.

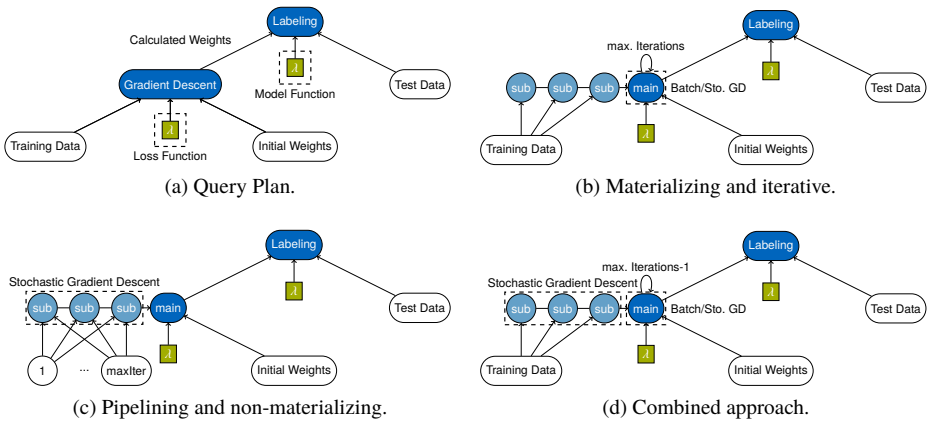


Fig. 2: Architecture for fitting gradient descent into database systems’ pipelines: (a) shows the basic query plan of the gradient descent operator: a binary operator with training data, initial weights as parameters, and the lambda expression for the loss function, it returns the optimized weights, that can be passed over to a labeling operator, that labels test data; (b) shows the parallelized variant, where the tuples are collected in sub threads, unified in the main thread, where the optimization happens in a parallel loop; (c) shows the sub threads computing local weights without materialization, there are as many input pipelines as iterations exist; (d) shows the combined approach, where local weights are computed initially.

3.3.2 Operator Tree

To integrate gradient descent in relational algebra, we need two operators, one gradient descent operator to calculate the weights and one to label the incoming tuples using the parametrized loss function. These are both binary operators as they expect two underlying operator trees each, one for delivering the input data (training data for the gradient descent operator and test data for the labeling operator) and one for the weights (the initial ones and the optimal ones respectively). Both operators use lambda expressions, introduced in the previous chapter, for specifying the loss function.

The gradient descent operator is applied to a training dataset, such as the dataset of previous rides in our taxi data example, to generate optimal weights and enable the labeling operator to label future examples. Alternatively, a subset of the original dataset can be used for training, by splitting the full dataset into training and testing datasets using standard SQL queries. Either way, we predict at least one attribute (the label) as a function of the other attributes (the features). The gradient descent operator first calculates the weights and passes them over to the labeling operator (see Fig. 2a). The gradient descent operator requires labeled training data, initial weights, and a lambda function for the model. The labeling operator consumes and materializes the optimized weights, computes labels for all incoming feature tuples, and adds them to the results set.

The lambda expression must be aware of the tensors' dimensionality. This is possible when the tensor size and the shapes of the input parameters are known at the compile time of the SQL query. Then, only one tuple containing all *variables* uses the pipeline for the initial weights. So, we suppose the incoming pipelines delivering the weights to contain only one tuple with all parameters. Another way is to enumerate the elements of every *variable* and to push every element as a single tuple. However, this is less user-friendly as the variables' dimensions would then need to be specified separately inside the lambda functions. This also applies to the pipeline between the two operators: either the weights must be sent in a fixed order or all the weights must be included in one tuple.

Both operators work independently and stand-alone. Normally acting together, they also operate disassembled, when the weights should be computed only once but used multiple times. Considering the taxi data example, we would materialize the optimized weights once a day, to obtain consistent data for the following estimates, but then use the same weights many times to label future taxi ride data. In addition, since the labeling operator simply evaluates the specified lambda function and adds a new attribute (label) to each tuple, it can be used with weights computed by other optimization methods.

3.3.3 Pipelining and Parallelism

We now investigate this architecture for integrating a gradient descent operator into a database system in more detail. In database systems, operators can be combined to form an operator tree with table scans as leaves. An operator tree may have multiple parallel pipelines and each tuple flows through one pipeline. We apply the producer-consumer concept [Ne11] to the operators to incorporate the algorithms in our main memory database system. It pushes tuples toward their target operators and provides parallelism by dividing the process into multiple execution pipelines. Operators can be classified as *pipeline friendly* or into *pipeline breakers* and *full pipeline breakers* by their behaviour of passing elements through directly, consuming all elements, or consuming and materializing all elements before producing output. Multiple execution pipelines are realized as sub-threads of the operator whereas the main thread is responsible for global computations in pipeline breakers.

This paper does not aim for comparing the performance of stochastic and batch gradient descent but for make them fit into a database system. Therefore, we discuss different strategies regarding the methods' needs inside a database system. To recap, batch gradient descent requires all tuples per iteration, whereas stochastic gradient descent expects one tuple at a time. The first method optimizes the weights by taking the average deviation as loss function, so all tuples needs to be materialized before. When expecting only one tuple at a time, the tuples can be consumed separately or even in parallel when a synchronization mechanism exists. Both methods work well when all tuples have been materialized before.

The intuitive way is to design gradient descent as a *full pipeline breaker* to consume all incoming tuples before producing the calculated weights. The tuples have to be materialized

before, so the design allows any optimization method to be called afterwards. For batch as well as for stochastic gradient descent, the operator’s main thread performs multiple iterations until the weights converge to the loss function’s minimum. Parallel loops grant distributed execution either by evaluating the gradient for each tuple independently (batch) or by updating weights as atomic global variables (stochastic). The weight synchronization for stochastic gradient descent is based on previous works on parallelization of optimization problems [Xi17; Zi10], which means taking the average weights after the computation is complete (and is not the focus of this work). Fig. 2b shows the parallelism of the materializing gradient descent operator. It consists of one main thread and one sub-thread per incoming pipeline. The sub-threads consume and materialize the incoming tuples, while the main thread collects them, consumes the input weights, and minimizes the loss function in parallel.

Specialized for database system’s pipelines, we devise a *non-materializing* operator suited for stochastic gradient descent only. While it still is a *pipeline breaker*, it computes stochastic gradient descent in each pipeline on partitioned sets of data without materialization. Multiple iterations are implemented by copying the underlying operator tree condoning the disadvantages of recompiling the whole subtree and the fixed number of iterations. Fig. 2c shows the gradient descent implementation with a separate input pipeline for each iteration. For each pipeline, multiple sub-threads compute local weights. The main thread is responsible for computing the global weights after each iteration is complete.

Finally, the *combined approach* combines the benefits of parallel pipelines with the advantages of only compiling the subtree once and being able to terminate when the process has converged. First, it precomputes the weights in separate pipelines handling subsets of the data and then iterates gradient descent on the materialized tuples until it converges. Fig. 2d shows the sub-threads only computing the local weights once for each input pipeline and materializing the tuples. The main thread computes the global weights of the first iteration, then performs the remaining iterations in parallel on the materialized tuples using either batch or stochastic gradient descent.

The labeling operator is *pipeline friendly* and highly parallel. It evaluates the parametrized model function for each tuple and adds the result as a new attribute without the need to materialize any of the tuples.

4 Tensors for Database systems

Since many mathematical problems are based on tensor algebra, we also extend PostgreSQL’s array data type to handle tensors by adding algebraic operations.

Here, we define dense tensors as a data structure whose size is evaluated at runtime but which can also be specified by a table attribute when a tensor is created. The number of dimensions is given first, followed by the width in each dimension. After that, the items themselves are stored sequentially, ordered beginning with the highest-numbered dimension.

4.1 Transpose

We define *transpose* as swapping the order of the first two dimensions, with further dimensions being treated as one huge block element. Here, `SELECT tensor_transpose(a) FROM tensors` produces the transpose of T^t of the tensor T as follows:

$$(t^t)_{i_1 i_2 i_3 \dots i_m} = t_{i_2 i_1 i_3 \dots i_m}.$$

4.2 Addition/Subtraction/Scalar Product

We handle addition or subtraction elementwise, meaning the tensors S and T must have identical dimensions, and the scalar product acts elementwise for a given scalar $r \in \mathbb{R}$:

$$\begin{aligned} T, S, T + S, T - S, r * T &\in \mathbb{R}^{I_1 \times \dots \times I_m}, \\ (t + s)_{i_1 \dots i_m} &= t_{i_1 \dots i_m} + s_{i_1 \dots i_m}, \\ (r * t)_{i_1 \dots i_m} &= r * t_{i_1 \dots i_m}. \end{aligned}$$

4.3 Product

Matrices like $A \in \mathbb{R}^{m \times o}$, $B \in \mathbb{R}^{o \times n}$ with equal inner dimensions can be multiplied to create the product C by summing up the product of m row elements with n column elements for each entry $c_{ij} = \sum_{k \in [o]} a_{ik} b_{kj}$. We can generalize this multiplication process to create products of tensors with equal inner dimensions where the new entries are sums of corresponding products:

$$\begin{aligned} T &\in \mathbb{R}^{I_1 \times \dots \times I_m = o}, U \in \mathbb{R}^{J_1 = o \times \dots \times J_n}, \\ S = TU &\in \mathbb{R}^{I_1 \times \dots \times I_{m-1} \times J_2 \times \dots \times J_n}, \\ s_{i_1 i_2 \dots i_{m-1} j_2 \dots j_m} &= \sum_{k \in [o]} t_{i_1 i_2 \dots i_{m-1} k} u_{k j_2 \dots j_m}. \end{aligned}$$

4.4 Tensor Usage for Equation Systems

Simple and multiple linear regression can be also computed without gradient descent by instead relying on tensor operations. If we set the gradient equal to zero, we will solve simple and multiple linear regression problems in terms of equation systems such as:

$$0 \stackrel{!}{=} \nabla l_{X, \vec{y}}(a, b) = \frac{1}{|X|} \sum_{i \in |X|} \begin{pmatrix} 2(ax_i + b - y_i) * x \\ 2(ax_i + b - y_i) * 1 \end{pmatrix}.$$

Using the helper variables $\hat{x} = \frac{1}{|X|} \sum_{x \in X} x$ and $\hat{y} = \frac{1}{|Y|} \sum_{y \in Y} y$ to represent the means of the labels and features, we obtain the following equations, which can easily be expressed in SQL (see List. 2):

$$a = \frac{\sum_{i \in |X|} (x_i - \hat{x})(y_i - \hat{y})}{\sum_{i \in |X|} (x_i - \hat{x})^2},$$

$$b = \hat{y} - a\hat{x}.$$

```
with means as (select avg(x) as mean_x, avg(y) as mean_y from datapoints),
sums as (select sum((x - mean_x) * (y - mean_y)) as nominator, sum(power(x - mean_x, 2)) as
denominator from datapoints, means),
a as (select 'a', nominator / denominator as value from sums),
b as (select 'b', mean_y - a.value * mean_x as value from means, a)
select * from b union select * from a;
```

List. 2: Simple linear regression in SQL.

To handle the increased number of variables involved in multiple linear regression, we can make use of the tensor operations defined above. Assuming we can invert the matrix

$$X' = \begin{pmatrix} 1 & x_{1,1} & \dots & x_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{m,1} & \dots & x_{m,n} \end{pmatrix} \in \mathbb{R}^{m \times (n+1)},$$

then we can obtain the weights that minimize the loss function from the following equation [LAC09], which can be expressed in our extended SQL (see List. 3):

$$\vec{w} = (X'^T X')^{-1} X'^T \vec{y}.$$

```
select (array_inverse(array_transpose(x)*x))*(array_transpose(x)*y)
from (select array_agg(x) x from (select array[1,x_1,x_2] as x from datapoints) sx,
(select array_agg(y) y from (select array[y] y from datapoints) sy) ty;
```

List. 3: Multiple linear regression in SQL using tensor operations.

4.5 Tensor Differentiation for Gradient Descent

Now, we combine the previous sections on gradient descent and tensors to show how minimization problems involving gradient descent can easily be handled using tensor differentiation with the help of tensor data types. As an example, we calculate k-Means cluster centers by using SQL tensor data types with our gradient descent operator.

Gradient descent allows us to compute the centers for the k-Means clustering algorithm based on a vector of initial weights. In List. 4, the two-dimensional points are provided as a set of two attribute tuples and the initial centers are given as one tuple consisting of two vectors. We then compute the centers via gradient descent by using a lambda function that expresses the minimum distance from each point to the currently nearest center. These distances are then optimized by adjusting the centers (weights). Afterwards, standard SQL-92 queries can be used to assign the points to their respective clusters.

```
create table points (x float, y float); create table weights(wx float[], wy float[]);
insert into points ... insert into weights ...
select * from gradientdescent(
  lambda(data, weights) min(0 <= i < length(wx, 1), (x - wx[i])2 + (y - wy[i])2),
  (select x,y from points), (select wx,wy from weights), 0.05, 100);
```

List. 4: Gradient descent for k-Means cluster computation.

5 Evaluation

In this section, we benchmark the gradient descent operator using the automatic gradient computation framework for various model functions and the different architectures of integrating gradient descent in HyPer, our in-memory database system.

These benchmarks are based on the Chicago taxi rides dataset⁸. For each ride, this included the duration (in seconds), distance (in miles), fare, and payment type used. We used simple linear regression to predict the fare based on the trip distance, and used multiple linear regression to also consider the ride time. Finally, we used logistic regression to infer the payment type (i.e., whether or not the fare was paid by credit card) from the fare cost.

We ran all tests using HyPer, our extended main memory database system developed for mixed OLAP and OLTP transactions. As a baseline, we considered two other database systems, namely MariaDB 10.1.30, PostgreSQL 9.6.8 with the MADlib v1.13 extension, as well as two dedicated tools, namely TensorFlow 1.3.0 (with GPU support enabled and disabled) and R 3.4.2. All experiments were run multi-threaded on a 20-core Ubuntu 17.04 machine (Intel Xeon E5-2660 v2 CPU) with hyper-threading, running at 2.20 GHz with 256 GB DDR4 RAM. An Nvidia GeForce GTX 1050 Ti was installed to enable TensorFlow computations on a GPU. We provide the test scripts for reproducibility with a data generator online⁹.

All tests were carried out five times and the average runtimes recorded. A Python script managed the program calls and measured the total user time spent on each one. The runtimes include the time taken to load the data into TensorFlow and into R, and also the TensorFlow session creation time. To assume the database system as the data storage tier, the data there was considered to have already been loaded.

⁸ <https://data.cityofchicago.org/Transportation/Taxi-Trips/wrvz-psev>

⁹ https://gitlab.db.in.tum.de/MaxEmanuel/regression_in_sql

5.1 Linear Regression Using Equation Systems

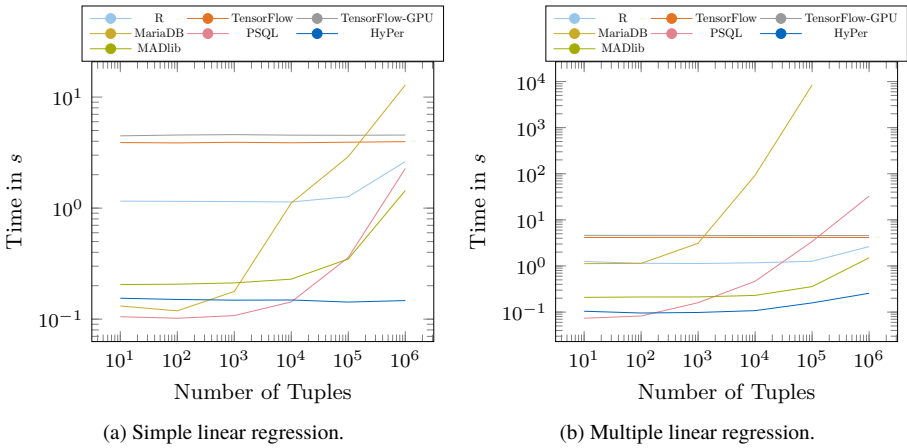


Fig. 3: Runtime for solving (a) simple or (b) multiple linear regression by equation systems.

Simple linear regression was solved using common SQL-92 queries in our main memory database system and the competitor database systems. In TensorFlow, we used tensor algebra for solving equation systems for linear regression. In MariaDB and PostgreSQL, we defined the matrix operations needed to solve multiple linear regression as a single query, while our main memory database system and R already support tensor algebra. The MADlib extension for PostgreSQL provides a predefined function for solving equation systems for linear regression that we used.

As Fig. 3 shows, our in-memory database was able to carry out both linear regression tasks in less than 0.3 seconds, significantly outperforming all the baseline methods. For small numbers of tuples, even the other classical database systems performed substantially better than TensorFlow, which required at least three seconds for every task.

5.2 Gradient Descent Benchmarks

For the gradient descent benchmark, we evaluated linear regression, multiple linear regression, and logistic regression models, minimizing a least squares loss function. In addition, we performed k-Means center computations with a specialized loss function. Each gradient descent optimization run consisted of 5,000 iterations, with a learning rate of 0.000000071.

For TensorFlow and R, we used gradient descent library functions to perform linear regression and logistic regression. For the baseline database systems, we implemented gradient descent for both models using PL/pgSQL in PostgreSQL and procedures in MariaDB. Additionally, we benchmarked the MADlib’s logistic regression function for

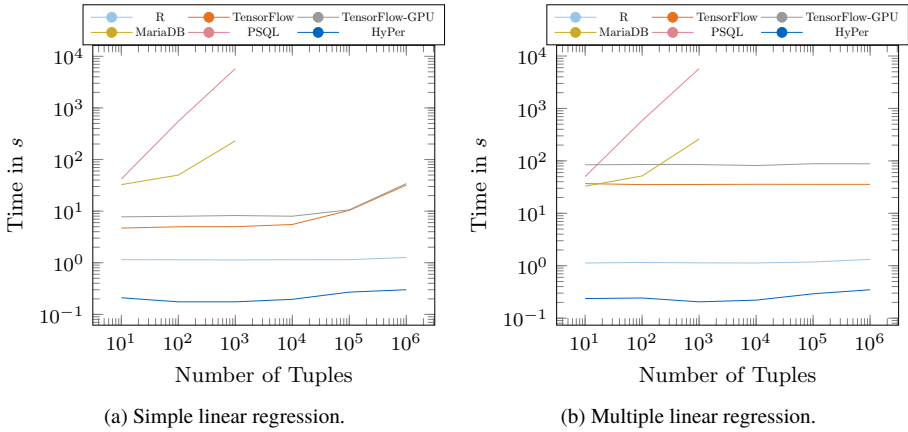


Fig. 4: Runtime for (a) simple and (b) multiple linear regression using gradient descent.

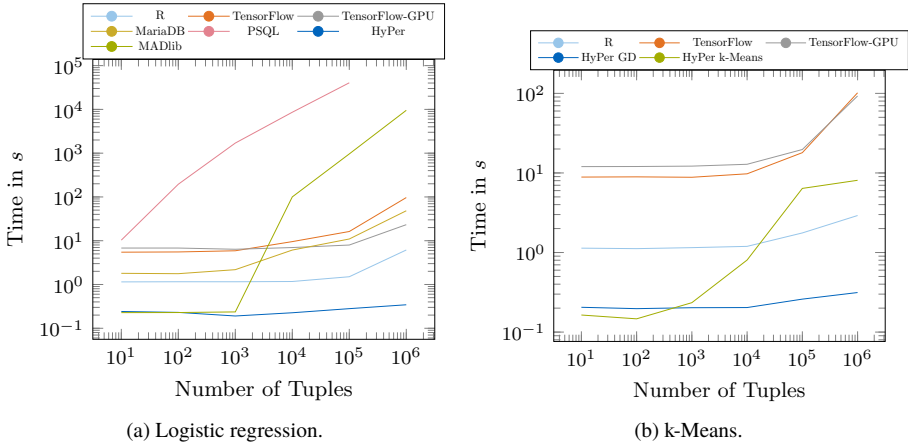


Fig. 5: Runtime for (a) logistic regression and (b) clustering using gradient descent.

PostgreSQL. We computed the centers for k-Means in TensorFlow using the corresponding library function, in our main memory database system using both the dedicated k-Means operator and the gradient descent operator, and in R using the k-Means clustering library.

The results for linear regression (see Figs. 4a, 4b) show that both the TensorFlow’s and R’s library functions run in constant time for small numbers of input tuples and perform better than the hardcoded in-database gradient descent functions, which scaled linearly.

Considering that linear regression can be solved by equation systems without gradient descent, all database systems performed better for linear regression than the TensorFlow’s

and R’s library functions. Nevertheless, our approach, gradient descent in a main memory database system, was as performant as dedicated tools using predefined models.

For the logistic regression tasks (see Fig. 5a), also dedicated tools’ library functions, still faster than the classical database systems, scaled linearly in the input size but were slower than our approach running in HyPer.

Fig. 5b also shows that our gradient descent operator was the fastest at computing the k-Means centers and even beating TensorFlow, although they both omitted the cluster assignment step. The R k-Means library function was faster than TensorFlow and slower than our database system (time for assigning points to clusters not excluded).

We further investigate on the time needed for data loading and for the actual computation. In Fig. 6, we see the runtime split up into CSV data loading, computation time, and TensorFlow’s session creation. Essentially is that the time needed for data loading represents the time to be saved by doing more computations inside of database systems. Also we observed that the time for TensorFlow’s session creation is more expensive when working on the GPU whereas the computation itself sometimes is faster. This explains the better results using TensorFlow without GPU support on smaller datasets.

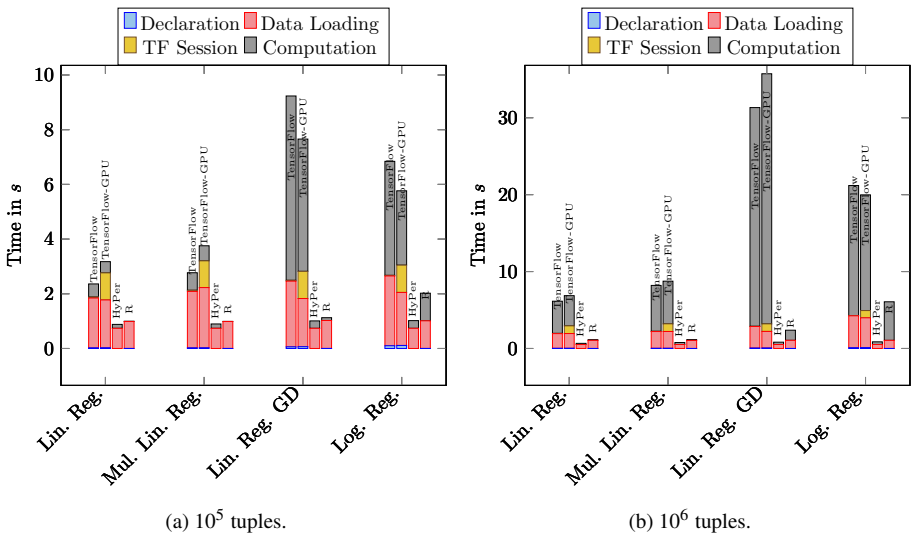


Fig. 6: Runtime split up into the different operations (declarations of variables, CSV data loading, TensorFlow’s session creation, actual computation) using (a) 10⁵ and (b) 10⁶ tuples for simple/multiple linear regression using equation systems and for simple and logistic regression by gradient descent: the time needed for data loading could be saved by doing more computations inside of database systems.

In summary, much time is spent on data loading when performing computations using dedicated tools. This time can be saved without any drawbacks by shifting computations

into the core of database systems. So it is worth to optimize various gradient descent tasks inside of database systems when such an operator exists.

5.3 Pipelining

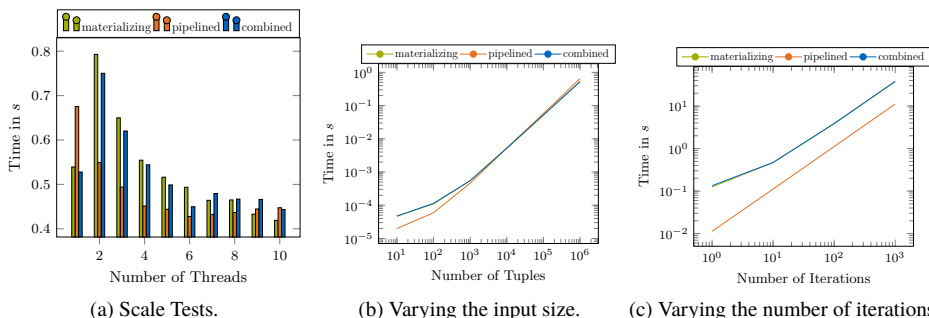


Fig. 7: Benchmarks for the different architectures of gradient descent in the database system’s pipelines by varying the number of (a) available threads, (b) the input size, and (c) the number of iterations.

Finally, we evaluated several different approaches to integrating gradient descent into the database systems’ pipelines. For this reason, we measured the performance of three different linear regression implementations: materializing, non-materializing (pipelined), and the combined approach. We varied the number of available threads (from one to ten, see Fig. 7a), the training set size (from ten to 10^6 , see Fig. 7b), and iterations (from one to 100, see Fig. 7c). The other parameters remained constant, defaulting to 10 iterations, 10^6 tuples and one thread. Parallelism is enabled for the scale tests only to avoid the overhead of preparing multiple pipelines during the other tests.

In all tests, the non-materializing implementation was the fastest when the methods could not terminate early because all iterations are precompiled as input pipelines. The materializing and the combined implementations performed very similarly but more slowly, but had the advantage of stopping when the weights converged or reaching a predefined threshold. All three implementations behaved similarly, the runtime depends linearly on both the size of the training set and the number of iterations.

All implementations scaled linearly with the number of threads with performance increasing by about 20 % percent for each additional thread. The parallelization of the non-materializing version scaled best whereas the overhead of synchronizing the weights for the materializing one allows the parallelization to be faster from the fourth added thread. When enabling parallelism but providing only one thread for execution, the non-materializing implementation performed worse than the others as it could not make use of the overhead of preparing sub-threads for multiple input pipelines.

The combination of both strategies performed slightly but not significantly better in all tests. As the tuples still have to be materialized and the weights have to be computed in

parallel loops, only one iteration could be saved by precomputing the weights in front. In our scenario, up to 10 % could be saved. But with increasing number of iterations the performance gain would be negligible.

To conclude, the non-materializing version appears to be the fastest when all iterations must be completed or when it is not possible to allocate enough memory for materializing the tuples. If gradient descent converges quickly, a materializing approach is the most suitable, due to its ability to terminate early. The combination of both methods is only suited when few iterations are needed as of little performance gains otherwise. Also the non-materializing architecture is only applicable in combination with stochastic gradient descent, whereas the materializing one works with batch gradient descent as well.

6 Conclusion

In this paper, we have presented in-database approaches to machine learning that use automatic differentiation, in-database gradient descent, and tensor algebra.

To achieve this, we extended recently introduced lambda functions to act as loss functions for gradient descent, and we also developed three strategies for integrating gradient descent into the pipelining concept of today's modern in-memory database systems. For our gradient descent, we developed an automatic gradient differentiation framework based on expression trees.

Together with tensor algebra, we showed that both extensions are sufficient for solving common machine learning tasks in the core of database systems without the use of external tools. A direct comparison with other relational database systems (both classical disk-based and modern in-memory systems) showed that this enabled linear regression tasks to be computed more rapidly, even by classical disk-based database systems. Moreover, pure approximation tasks like logistic regression can be solved as quickly by an extended main memory database system as by dedicated tools like TensorFlow and R, our strongest competitors.

This work was aimed at shifting more of the computational work to database servers. Given that modern machine learning has come to rely on tensor data and loss functions, we felt it was important to present an architectural blueprint for combining gradient descent with tensor algebra in a modern main memory database system. Here, we reused SQL with lambda functions as a language for specifying loss functions in a database system but, independently of the acceptance of SQL by data scientists, gradient descent will form one of the fundamental building blocks when database systems begin to take over more computational tasks, which modern hardware certainly allows. However, if in-database machine learning is to gain increased acceptance, a declarative meta-language for translation into SQL will be required.

References

- [Ab16] Abadi, M. et al.: TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. CoRR abs/1603.04467/, 2016, arXiv: 1603.04467, URL: <http://arxiv.org/abs/1603.04467>.
- [Ab17] Aberger, C. R.; Lamb, A.; Olukotun, K.; Ré, C.: Mind the Gap: Bridging Multi-Domain Query Workloads with EmptyHeaded. PVLDB 10/12, pp. 1849–1852, 2017, URL: <http://www.vldb.org/pvldb/vol10/p1849-aberger.pdf>.
- [Ba12] Bastien, F.; Lamblin, P.; Pascanu, R.; Bergstra, J.; Goodfellow, I. J.; Bergeron, A.; Bouchard, N.; Warde-Farley, D.; Bengio, Y.: Theano: new features and speed improvements. CoRR abs/1211.5590/, 2012, arXiv: 1211.5590, URL: <http://arxiv.org/abs/1211.5590>.
- [Bi14] Binnig, C.; Salama, A.; Zamanian, E.; Kornmayer, H.; Listing, S.; Müller, A. C.: XDB - A Novel Database Architecture for Data Analytics as a Service. In: 2014 IEEE Big Data, Anchorage, AK, USA, June 27 - July 2, 2014. Pp. 96–103, 2014, URL: <https://doi.org/10.1109/BigData.Congress.2014.23>.
- [BPR15] Baydin, A. G.; Pearlmutter, B. A.; Radul, A. A.: Automatic differentiation in machine learning: a survey. CoRR abs/1502.05767/, 2015, arXiv: 1502.05767, URL: <http://arxiv.org/abs/1502.05767>.
- [CGK14] Crotty, A.; Galakatos, A.; Kraska, T.: TUPLEWARE: Distributed Machine Learning on Small Clusters. IEEE Data Eng. Bull. 37/3, pp. 63–76, 2014, URL: <http://sites.computer.org/debull/A14sept/p63.pdf>.
- [Cr15] Crotty, A.; Galakatos, A.; Zraggen, E.; Binnig, C.; Kraska, T.: Vizdom: Interactive Analytics through Pen and Touch. PVLDB 8/12, pp. 2024–2027, 2015, URL: <http://www.vldb.org/pvldb/vol8/p2024-crotty.pdf>.
- [FKN12] Funke, F.; Kemper, A.; Neumann, T.: Compacting Transactional Data in Hybrid OLTP & OLAP Databases. PVLDB 5/11, pp. 1424–1435, 2012, URL: http://vldb.org/pvldb/vol5/p1424_florianfunke_vldb2012.pdf.
- [He12] Hellerstein, J. M.; Ré, C.; Schoppmann, F.; Wang, D. Z.; Fratkin, E.; Gorajek, A.; Ng, K. S.; Welton, C.; Feng, X.; Li, K.; Kumar, A.: The MADlib Analytics Library or MAD Skills, the SQL. PVLDB 5/12, pp. 1700–1711, 2012, URL: http://vldb.org/pvldb/vol5/p1700_joehellerstein_vldb2012.pdf.
- [Hu17] Hubig, N.; Passing, L.; Schüle, M. E.; Vorona, D.; Kemper, A.; Neumann, T.: HyPerInsight: Data Exploration Deep Inside HyPer. In: CIKM 2017, Singapore, November 06 - 10, 2017. Pp. 2467–2470, 2017, URL: <http://doi.acm.org/10.1145/3132847.3133167>.
- [Ka17] Kaoudi, Z.; Quiané-Ruiz, J.; Thirumuruganathan, S.; Chawla, S.; Agrawal, D.: A Cost-based Optimizer for Gradient Descent Optimization. In: SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017. Pp. 977–992, 2017, URL: <http://doi.acm.org/10.1145/3035918.3064042>.
- [Ke16] Kernert, D.: Density-Aware Linear Algebra in a Column-Oriented In-Memory Database System, PhD thesis, Dresden University of Technology, Germany, 2016, URL: <http://nbn-resolving.de/urn:nbn:de:bsz:14-qucosa-210043>.
- [LAC09] Lease, M.; Allan, J.; Croft, W. B.: Regression Rank: Learning to Meet the Opportunity of Descriptive Queries. In: ECIR 2009, Toulouse, France, April 6-9, 2009. Proceedings. Pp. 90–101, 2009, URL: https://doi.org/10.1007/978-3-642-00958-7_11.

- [LMG18] Laue, S.; Miterreiter, M.; Giesen, J.: Computing Higher Order Derivatives of Matrix and Tensor Expressions. In: NIPS, 2-8 December 2018, Montreal, Montreal, Canada. Pp. 2755–2764, 2018, URL: <http://papers.nips.cc/paper/7540-computing-higher-order-derivatives-of-matrix-and-tensor-expressions.pdf>.
- [Lu17] Luo, S.; Gao, Z. J.; Gubanov, M. N.; Perez, L. L.; Jermaine, C. M.: Scalable Linear Algebra on a Relational Database System. In: ICDE 2017, San Diego, CA, USA, April 19-22, 2017. Pp. 523–534, 2017, URL: <https://doi.org/10.1109/ICDE.2017.108>.
- [Ne11] Neumann, T.: Efficiently Compiling Efficient Query Plans for Modern Hardware. PVLDB 4/9, pp. 539–550, 2011, URL: <http://www.vldb.org/pvldb/vol4/p539-neumann.pdf>.
- [Pa17] Passing, L.; Then, M.; Hubig, N.; Lang, H.; Schreier, M.; Günemann, S.; Kemper, A.; Neumann, T.: SQL- and Operator-centric Data Analytics in Relational Main-Memory Databases. In: EDBT 2017, Venice, Italy, March 21-24, 2017. Pp. 84–95, 2017, URL: <https://doi.org/10.5441/002/edbt.2017.09>.
- [Re11] Recht, B.; Ré, C.; Wright, S. J.; Niu, F.: Hogwild: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent. In: NIPS, 12-14 December 2011, Granada, Spain. Pp. 693–701, 2011, URL: <http://papers.nips.cc/paper/4390-hogwild-a-lock-free-approach-to-parallelizing-stochastic-gradient-descent>.
- [Xi17] Xie, X.; Tan, W.; Fong, L. L.; Liang, Y.: CuMF_SGD: Parallelized Stochastic Gradient Descent for Matrix Factorization on GPUs. In: HPDC 2017, Washington, DC, USA, June 26-30, 2017. Pp. 79–92, 2017, URL: <http://doi.acm.org/10.1145/3078597.3078602>.
- [Zi10] Zinkevich, M.; Weimer, M.; Smola, A. J.; Li, L.: Parallelized Stochastic Gradient Descent. In: NIPS, 6-9 December 2010, Vancouver, British Columbia, Canada. Pp. 2595–2603, 2010, URL: <http://papers.nips.cc/paper/4006-parallelized-stochastic-gradient-descent>.

This work is part of the TUM Living Lab Connected Mobility (TUM LLCM) project and has been funded by the Bavarian Ministry of Economic Affairs, Energy and Technology (StMWi) through the Center Digitisation.Bavaria, an initiative of the Bavarian State Government.

Efficient Data-Parallel Cumulative Aggregates for Large-Scale Machine Learning

Matthias Boehm¹, Alexandre V. Evfimievski², Berthold Reinwald³

Abstract: Cumulative aggregates are often overlooked yet important operations in large-scale machine learning (ML) systems. Examples are prefix sums and more complex aggregates, but also preprocessing techniques such as the removal of empty rows or columns. These operations are challenging to parallelize over distributed, blocked matrices—as commonly used in ML systems—due to recursive data dependencies. However, computing prefix sums is a classic example of a presumably sequential operation that can be efficiently parallelized via aggregation trees. In this paper, we describe an efficient framework for data-parallel cumulative aggregates over distributed, blocked matrices. The basic idea is a self-similar operator composed of a forward cascade that reduces the data size by orders of magnitude per iteration until the data fits in local memory, a local cumulative aggregate over the partial aggregates, and a backward cascade to produce the final result. We also generalize this framework for complex cumulative aggregates of sum-product expressions, and characterize the class of supported operations. Finally, we describe the end-to-end compiler and runtime integration into SystemML, and the use of cumulative aggregates in other operations. Our experiments show that this framework achieves both high performance for moderate data sizes and good scalability.

Keywords: Cumulative Aggregates, ML Systems, Large-Scale Machine Learning, Data-Parallel Computation, Apache SystemML

1 Introduction

Large-Scale ML Systems: Machine learning (ML) and artificial intelligence in general are transforming our lives from recommendations and driving assistants to finance, health care, and enterprise ML. Large data collections are essential for learning these ML models [Co09; Da10], especially for complex models with many parameters. Thanks to feedback loops for label acquisition like active learning [CAL94; CGJ94; RMJ06] and weak supervision techniques [Ra16; Tr18], there is also abundant labeled data. These large labeled data collections in turn necessitate large-scale ML systems with the ability for distributed computation whenever necessary. State-of-the-art large-scale ML systems, therefore, follow either of two predominant distributed architectures: data-parallel operations for batch algorithms, often on top of data-parallel computation frameworks such as MapReduce [DG04], Spark [Za12], or Flink [Al14]; or parameter servers for mini-batch algorithms

¹ Graz University of Technology, Graz, Austria, m.boehm@tugraz.at

² IBM Research – Almaden, San Jose, USA, evfimi@us.ibm.com

³ IBM Research – Almaden, San Jose, USA, reinwald@us.ibm.com

[De12]. In SystemML [Bo14; Bo16], we support the general case of local and distributed data-parallel operations, task-parallel parfor loops, as well as local and distributed parameter servers, which can be seamlessly combined in a high-level language with R-like syntax.

Cumulative Aggregates: While data-parallel matrix multiplication [Bo16; Gh11; HBY13; Ro17; YSC15; Yu17], element-wise operations and aggregations [Bo18], meta learning [Sc15], and statistical functions [TTR12] have received lots of attention, important and challenging distributed operations like cumulative aggregates are largely overlooked in the literature. An example is the computation of column-wise prefix sums via

$$\mathbf{Z} = \text{cumsum}(\mathbf{X}) \quad \text{with} \quad \mathbf{Z}_{ij} = \sum_{k=1}^i \mathbf{X}_{kj} = \mathbf{X}_{ij} + \mathbf{Z}_{(i-1)j}, \quad (1)$$

where \mathbf{Z} and \mathbf{X} are $n \times m$ matrices, and output cells \mathbf{Z}_{ij} are defined as the column-wise sum from 1 through i , or recursively (with $\mathbf{Z}_{0j} = 0$). Other common cumulative aggregates include $\text{cummin}(\mathbf{X})$, $\text{cummax}(\mathbf{X})$, $\text{cumprod}(\mathbf{X})$ for cumulative minima, maxima, and products. Applications of cumulative aggregates include (1) iterative algorithms for survival analysis such as Cox hazard regression or Kaplan-Meier, (2) spatial and structural data processing via linear algebra, and (3) data preprocessing such as the sub-sampling of rows or the removal of empty rows, which both internally rely on cumulative aggregates.

Parallel Cumulative Aggregates: Although the recursive formulation in Equation (1) seems inherently sequential, computing prefix sums is a classic example of an operation that allows for efficient parallelization via aggregation trees [B193; CBZ90]. The basic idea is a logical tree over the input data, level-wise parallel aggregation (up sweep) and redistribution of offsets (down sweep) to compute the final output. This general approach of parallel cumulative aggregates is broadly applicable, but the integration into large-scale ML systems—with unordered distributed datasets—has received little attention so far.

Contributions: Our primary contribution is a holistic description of a framework for distributed, data-parallel cumulative aggregates and its integration in Apache SystemML, which is representative for state-of-the-art, large-scale ML systems:

- *Background:* In Section 2, we provide background on SystemML’s compiler and matrix representations, and introduce straw-man solutions for cumulative aggregates.
- *DistCumAgg Framework:* In Section 3, we then describe our self-similar framework for distributed cumulative aggregates. Besides basic aggregates, we also introduce complex cumulative aggregates for common sum-product recurrence expressions.
- *System Integration:* Furthermore, in Section 4, we describe the end-to-end compiler and runtime integration of the DistCumAgg framework in SystemML, and the use of cumulative aggregates in other operations such as the removal of empty rows.
- *Experiments:* Finally, in Section 5, we report on experiments in SystemML, including baseline comparisons with R, Julia, and MPI, micro benchmarks, as well as scalability experiments with increasing data and cluster sizes.

2 SystemML Preliminaries

To provide the necessary background of large-scale ML systems for data-parallel operations, we first describe Apache SystemML’s [Bo14; Bo16] compiler and distributed matrix representations. Subsequently, we introduce straw-man cumulative aggregates at script level and give an overview of support for cumulative aggregates in other ML and DB systems.

Script Compilation: SystemML provides a high-level language with R-like syntax to express algorithms via linear algebra such as matrix multiplications, aggregations, and statistical functions. These scripts are parsed into a hierarchy of statement blocks, delineated by control flow. For each basic block, we then compile directed acyclic graphs (DAGs) of high-level operators (HOPs) and perform various simplification rewrites. Size information is propagated from the inputs over the entire program and bottom-up through these HOP DAGs to compute memory estimates per operation. These estimates—together with driver and executor memory budgets—are in turn used to select physical operators for local and distributed operations, which eventually yield an executable runtime plan. For this purpose, we provide single-node CP (control program), as well as distributed Spark and MapReduce operators for all supported operations⁴ and computation primitives.

Distributed Matrix Representation: Large-scale ML systems for data-parallel operations commonly rely on blocked matrix representations, where matrices are stored as distributed collections of block indexes and fixed-size blocks [Bo16; Lu17; Ma16; Ro17; Yu17; Za16]. Individual blocks—also known as tiles [HBY13; ZHY09] or chunks [St11]—are then stored in dense, sparse, or ultra-sparse formats. The use of squared $b \times b$ blocks (e.g., with block size $b = 1\text{K}$ in SystemML) works very well in practice because it ensures aligned blocks during join processing and aggregations independent of the join dimension, transpose operations can be computed in a block-local manner, and even dense blocks (8 MB) still fit in L3 cache, while block overheads are usually amortized across many values. For local CP operations, the entire matrix is represented as a single block to reuse runtime operations, avoid unnecessary overheads, and simplify local operations.

Straw-man Cumulative Aggregates: ML systems with language support for loops and indexing can emulate cumulative aggregates at script level. In fact, we used these alternatives in various SystemML use cases before we added built-in support in 2014. Figure 1 shows two examples that both compute cumulative column sums \mathbf{B} of an $n \times m$ input matrix \mathbf{A} . We will use these straw-man implementations for baseline comparisons. Although both functions perform only $O(nm)$ additions, they have very different characteristics and thus, it is useful to understand the resulting time complexity. First, `cumsumN2` is the most obvious implementation, where we maintain a row vector of column sums `csums` as we make a single pass over the input and output matrices. However, due to copy on write semantics, each row-wise matrix left-indexing `B[i,]=` internally copies the entire matrix \mathbf{B} , which results in $O(n^2m)$ time complexity. Due to the absence of loop-carried dependencies over

⁴ There are few exceptions such as specific solvers and deep neural network operations for which distributed operations are rarely necessary and thus, not yet supported.

```

1: cumsumN2 = function(Matrix[Double] A)
2:   return(Matrix[Double] B)
3: {
4:   B = A; csums = matrix(0,1,ncol(A));
5:   for( i in 1:nrow(A) ) {
6:     csums = csums + A[i,];
7:     B[i,] = csums;
8:   }
9: }

```

```

1: cumsumNlogN = function(Matrix[Double] A)
2:   return(Matrix[Double] B)
3: {
4:   B = A; m = nrow(A); k = 1;
5:   while( k < m ) {
6:     B[(k+1):m,] = B[(k+1):m,] + B[1:(m-k),];
7:     k = 2 * k;
8:   }
9: }

```

Fig. 1: Straw-man Cumsum Functions (amenable to update-in-place).

B, this pattern qualifies for update in-place (with $O(nm)$), but only for local, in-memory operations whereas distributed RDDs are always immutable. Second, `cumsumNlogN` aims to overcome this issue by using a trick of shifted addition and exponentially increasing k , which results in a time complexity of $O(n \log n \cdot m)$ with copy on write. This shifting is similar to the data-parallel plus-scan by Hillis and Steele [HS86]. For local operations, SystemML again uses update-in-place because the data dependencies between right- and left-indexing correctly serialize the reads and writes on **B**. Finally, both script-level alternatives were unsatisfactory in practice—especially for distributed operations—which motivated the support for local and distributed built-in operations in SystemML.

Cumulative Aggregates in ML Systems: Numerical computing frameworks such as R, Matlab, and Julia all support `cumsum(X)`, `cummin(X)`, `cummax(X)`, `cumprod(X)`, while NumPy provides only `cumsum(X)` and `cumprod(X)`. For handling matrix inputs, Matlab, Julia, and NumPy use overloaded functions that specify the aggregation dimension with mostly column-wise defaults. In contrast, R users explicitly apply vector operations, for example, via `apply(X, 2, cumsum)` for computing column-wise prefix sums. R also provides parallel apply functions, but they require **X** to fit in memory of a single node. In contrast, SystemML provides built-in support for local and distributed cumulative aggregates, as well as complex aggregates like `cumsumprod(X)`, which we will discuss in Section 3.3.

Cumulative Aggregates in SQL: Given the importance in practice, we further review support for cumulative aggregates in SQL. Besides naïve approaches via self-joins and recursive formulations, the most practical approach is based on window functions as introduced in SQL:2003’s OLAP operations [Me03]:

```
SELECT Rid, V, sum(V) OVER(ORDER BY Rid) AS cumsum FROM X
```

which defaults to a row range of `ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW`. While cumulative aggregation with a running sum—as used in PostgreSQL [Le15]—works well for this scenario, Leis et al. further introduced a so-called segment tree to avoid unnecessary redundancy for semi-additive aggregation functions like `min` and `max` over variable-sized frames [Le15]. Interestingly, this segment tree is very similar to the concept of aggregation trees from the HPC literature [BI93; CBZ90]. In contrast to these settings with ordered input data and direct access, we focus on data-parallel cumulative aggregates.

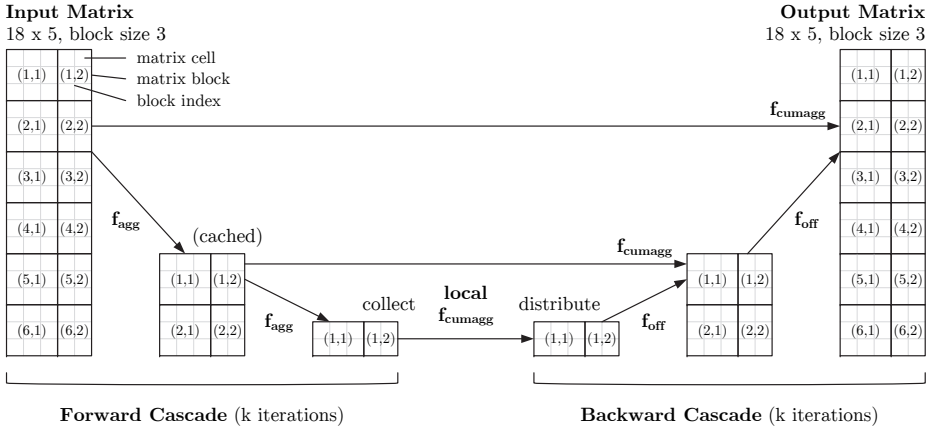


Fig. 2: Overview of the Distributed Cumulative Aggregate Framework (with $k=2$).

3 Data-Parallel Cumulative Aggregates

In this section, we describe DistCumAgg a framework for distributed, data-parallel cumulative aggregates. We first give an overview of the generic framework and subsequently discuss basic and complex cumulative aggregates as instantiations of this framework.

3.1 DistCumAgg Framework Overview

Framework Design: A major goal of our DistCumAgg framework is a seamless integration with blocked matrix representations, and hybrid runtime plans of local and distributed operations. At the same time, we do not make assumptions about the size of inputs or intermediates. The key idea to accomplish that is a self-similar operator chain as shown in Figure 2, consisting of a forward cascade of aggregations until the data fits into driver memory, a local cumulative aggregate over aggregates, and a backward cascade to produce the final result. Self-similarity allows for arbitrary lengths of cascades k (aggregates of aggregates) and the reuse of local cumulative aggregate operators as block operations.

Forward Cascade: Each iteration of the forward cascade applies an aggregation function f_{agg} to each matrix block to yield a row vector of column aggregates. These aggregates are then merged into an intermediate blocked matrix by mapping the input block index (i,j) to row i and column block j . Given typical block sizes of $b = 1K$, this aggregation reduces the data size by three orders of magnitude for dense input data. However, the data might still be too large. We, therefore, perform k iterations until the data fits into the driver memory. Assuming the aggregate intermediates are dense, estimating the size is simply done by scaling the input dimensions down by b^k . We do not perform these aggregations at once—as one might consider for optimization—because the intermediates are needed for the backward pass to ensure data-local operations without data shuffling.

Tab. 1: Instantiation of Basic and Complex Distributed Cumulative Aggregates.

Operation	Init	f_{agg}	f_{off}	f_{cumagg}
cumsum(\mathbf{X})	0	colSums(\mathbf{B})	$\mathbf{B}_{1:} = \mathbf{B}_{1:} + \mathbf{a}$	cumsum(\mathbf{B})
cummin(\mathbf{X})	∞	colMins(\mathbf{B})	$\mathbf{B}_{1:} = \min(\mathbf{B}_{1:}, \mathbf{a})$	cummin(\mathbf{B})
cummax(\mathbf{X})	$-\infty$	colMaxs(\mathbf{B})	$\mathbf{B}_{1:} = \max(\mathbf{B}_{1:}, \mathbf{a})$	cummax(\mathbf{B})
cumprod(\mathbf{X})	1	colProds(\mathbf{B})	$\mathbf{X}_{1:} = \mathbf{B}_{1:} \odot \mathbf{a}$	cumprod(\mathbf{B})
cumsumprod(\mathbf{X})	0	cbind(cumsumprod(\mathbf{B}) _{n_1} , prod(\mathbf{B} : ₂))	$\mathbf{B}_{11} = \mathbf{B}_{11} + \mathbf{B}_{12} \odot \mathbf{a}$	cumsumprod(\mathbf{B})

Local Cumulative Aggregate: The output of the last forward iteration is then collected into a local matrix at the driver, where we perform a local cumulative aggregate f_{cumagg} over the partial aggregates, yielding cumulative aggregates that can be used as offsets.

Backward Cascade: The backward cascade produces the final results based on the previously computed aggregates of all iterations and cumulative aggregate of the innermost iteration. We first parallelize (i.e., distribute) the innermost cumulative aggregate, and then perform k backward iterations inside-out. Each iteration splits the blocks of cumulative aggregates into rows and joins this collection with the corresponding input blocks of the forward cascade. The partial cumulative aggregates are then applied via f_{off} as offsets to the first row of each data block. Finally, we compute a block-local cumulative aggregate f_{cumagg} to yield the cumulative aggregate of the given iteration or final result, respectively.

Caching and Broadcasting: Regarding data-flow optimization, there are two simple techniques that greatly improved performance. First, for Spark distributed operations, we optionally cache aggregates of the forward cascade in storage level MEMORY_AND_DISK to avoid lazily recomputing these intermediates and thus, reading the main input multiple times. Second, to avoid unnecessary shuffling in the backward cascade, we perform a broadcast join—instead of a repartition join [B110]—of the data input and offsets, if the offsets fit into the broadcast memory budget. Using Spark, this applies to the innermost iteration because the broadcast variable must fit twice in memory of the driver as the broadcast operation serializes, compresses, and divides the in-memory object into chunks.

Instantiation: The generic DistCumAgg framework allows instantiating a variety of distributed cumulative aggregates by assigning concrete functions to f_{cumagg} , f_{agg} and f_{off} .

3.2 Basic Cumulative Aggregates

We now give an overview of instantiating the basic cumulative aggregates cumsum(\mathbf{X}), cummin(\mathbf{X}), cummax(\mathbf{X}), and cumprod(\mathbf{X}) for distributed operations. Table 1 shows the function mapping, where \mathbf{B} refers to a single matrix block of \mathbf{X} , \mathbf{a} is a row vector of cumulative aggregates joined to the data block \mathbf{B} , and \odot denotes an element-wise multiplication. Note that the functions f_{agg} , f_{off} , f_{cumagg} reuse existing unary aggregates,

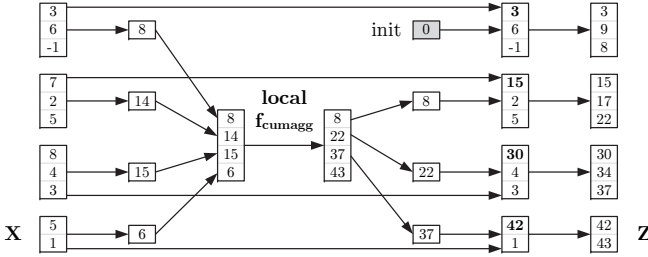


Fig. 3: Example Distributed Cumulative Sum (with $k=1$, $\mathbf{Z}_i = \mathbf{X}_i + \mathbf{Z}_{i-1}$).

element-wise operations, and cumulative aggregates, respectively. Since f_{off} is applied shifted by one block, we further need initialization values for the first block, which are—by operation semantics—0, ∞ , $-\infty$, and 1 for cumsum, cummin, cummax, and cumprod.

Example 1 (Distributed Cumulative Sum) Figure 3 shows a distributed cumsum with $k = 1$ iterations. We have an 11×1 input matrix, again with block size 3, resulting in 4 blocks. To compute the cumulative sum in a distributed manner, we first compute the block aggregates and merge them into a single in-memory matrix. Now we can perform cumsum(A) over the partial aggregates and distribute the intermediate. However, we need to shift these intermediates by one before the join because we want to use these as offsets for the next block. Accordingly, the first value is padded by the initialization value (0 for sum). We apply these offsets via element-wise addition to the first row of each block. Finally, we compute the block-local cumulative sum to produce the result. If we can broadcast the partial cumulative aggregates, the entire pipeline does not require any shuffle of the input matrix but only of partial aggregates, which are significantly smaller than the input.

3.3 Complex Cumulative Aggregates

Apart from basic cumulative aggregates, SystemML also supports the complex cumulative aggregate cumsumprod that is recursively defined as follows:

$$\mathbf{Z} = \text{cumsumprod}(\mathbf{X}) = \text{cumsumprod}(\mathbf{Y}, \mathbf{W}) \quad \text{with} \quad \mathbf{Z}_i = \mathbf{Y}_i + \mathbf{W}_i \odot \mathbf{Z}_{i-1}. \quad (2)$$

Common applications of cumsumprod are weighted (e.g., decayed) cumulative sums and frame-wise cumulative sums, where any $\mathbf{W}_i = 0$ acts as a frame boundary. The interleaved additions and multiplications seem to complicate preaggregation, but because multiplication distributes over addition we can still instantiate the DistCumAgg framework as shown in Table 1. Intuitively, we factor out the scaling per block, which allows again self-similar preaggregation. First, f_{agg} now returns a 1×2 matrix computed via $\text{cbind}(\text{cumsumprod}(\mathbf{B})_{n1}, \text{prod}(\mathbf{B}:2))$. Second, f_{cumagg} remains the local cumsumprod(B) operation which produces a scalar \mathbf{a} per block. And third, we reapply the scaling through f_{off} via $\mathbf{B}_{11} = \mathbf{B}_{11} + \mathbf{B}_{12} \odot \mathbf{a}$ before the final block-local cumsumprod(B).

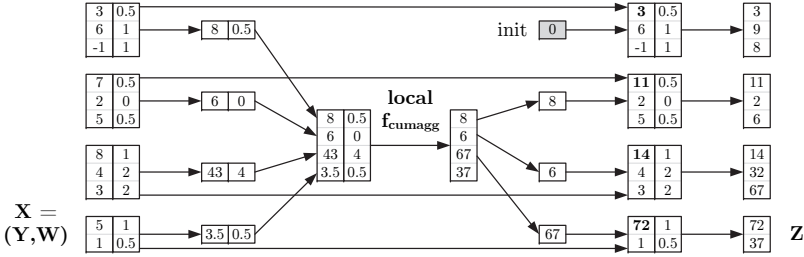


Fig. 4: Example Distributed Cumulative Sum-Product (with $k=1$, $\mathbf{Z}_i = \mathbf{Y}_i + \mathbf{W}_i \odot \mathbf{Z}_{i-1}$).

Data-Parallel Computation: To fit into the DistCumAgg framework for data-parallel operations, we restrict $cumsumprod(\mathbf{X})$ to the special case where \mathbf{X} is an $n \times 2$ matrix that concatenates the column vectors \mathbf{Y} and \mathbf{W} . This restriction ensures that all cumulative aggregates are unary operations and that related entries in \mathbf{Y} and \mathbf{W} appear in the same block, which enables efficient, block-local operations.

Example 2 (Distributed Cumulative Sum-Product) Figure 4 shows a distributed $cumsumprod$ with $k = 1$ iterations. Similar to Example 1, the input matrix has 11 rows, block size 3, and thus, 4 blocks. The second column is the weight vector \mathbf{W} . For a distributed $cumsumprod$, we first compute a 1×2 aggregate per block, holding $cumsumprod(\mathbf{B}_{n1})$ and $prod(\mathbf{B}_{.2})$. These aggregates are merged into a local matrix on which we apply a $cumsumprod$ to yield a vector of scaling factors. The factors \mathbf{a} are then applied to the first element of respective blocks via $\mathbf{B}_{11} = \mathbf{B}_{11} + \mathbf{B}_{12} \odot \mathbf{a}$ for computing the final result \mathbf{Z} .

3.4 Characterization of Applicable Operations

Following the instantiation of basic and complex cumulative aggregates, we aim to characterize the operations that are applicable to our data-parallel DistCumAgg framework.

Theoretical Foundation: The HPC literature has studied the parallel evaluation of recurrence equations extensively [B193; HK77]. Hence, we directly reuse known results from [B193] and impose additional restrictions for our DistCumAgg framework. Given a recurrence $x_i = a_i \oplus (b_i \otimes x_{i-1})$, it can be efficiently parallelized if (1) \oplus is associative, (2) \otimes is semi-associative (with an associative companion operator) or associative, and (3) \otimes distributes over \oplus . These conditions apply to first- and higher-order recurrences.

Additional Restrictions: Efficient, data-parallel computation over blocked matrices, and the integration into our self-similar framework further require two restrictions:

- *No Higher-Order Recurrences:* Due to limited data access across blocks, higher-order recurrences such $x_i = a_i \oplus x_{i-1} \oplus x_{i-2}$ are disallowed. We also limit the number of inputs to less or equal the block size to ensure co-partitioning of inputs.

- *Vectorized Operations:* We require that \oplus and \otimes have existing aggregate and element-wise operations, which enables the composition from existing block operations.

Strength Reduction: Note that $\text{cumsumprod}(\mathbf{X})$ uses $\text{cumsumprod}(\mathbf{B})_{n1}$ —i.e., the last block entry—as part of f_{agg} . Similarly, for $\text{cumsum}(\mathbf{X})$, we could use $\text{cumsum}(\mathbf{B})_n$. However, this simplifies to $\text{colSums}(\mathbf{B})$, which avoids materializing the cumsum output block.

4 System Integration

A robust and holistic system integration faces additional requirements. In this section, we discuss the end-to-end compiler and runtime integration of our DistCumAgg framework into SystemML as well as the use of cumulative aggregates in other operations.

4.1 Compiler and Runtime Integration

After script parsing and validation, a cumulative aggregate is represented by a single high-level operator (HOP), specifically a typed unary operator like $\text{u}(\text{cumsum})$. This HOP is then compiled into plans of physical operators as follows.

Simplification Rewrites: At HOP-level, we perform multiple rounds of size propagation and simplification rewrites. First, during intra- and inter-procedural analysis, we infer the output sizes of cumulative aggregates by propagating the input dimensions and assuming dense outputs. Second, we apply the following simplification rewrites in terms of transformation rules, which avoid unnecessary intermediates and data shuffling in distributed environments:

$$\begin{aligned}
 \text{rev}(\text{cumsum}(\text{rev}(\mathbf{X}))) &\rightarrow \mathbf{X} + \text{colSums}(\mathbf{X}) - \text{cumsum}(\mathbf{X}) \\
 \text{colSums}(\text{cumsum}(\mathbf{X})) &\rightarrow \text{colSums}(\mathbf{X} \odot \text{seq}(\text{nrow}(\mathbf{X}), 1)) \\
 \mathbf{X} \odot \text{cumsum}(\text{diag}(\text{matrix}(1, \text{nrow}(\mathbf{X}), 1))) &\rightarrow \text{lower.tri}(\mathbf{X}) \\
 \text{cumsum}(\mathbf{X}) &\rightarrow \mathbf{X} \text{ iff } \text{nrow}(\mathbf{X}) = 1.
 \end{aligned} \tag{3}$$

Examples of static—i.e., size-independent—rewrites are the computation of suffix sums $\text{rev}(\text{cumsum}(\text{rev}(\mathbf{X})))$ from the prefix sums, as well as the computation of aggregates such as $\text{sum}(\text{cumsum}(\mathbf{X}))$ or $\text{colSums}(\text{cumsum}(\mathbf{X}))$ from entries in \mathbf{X} scaled by their inclusion frequencies. Furthermore, the expression $\mathbf{X} \odot \text{cumsum}(\text{diag}(\text{matrix}(1, \text{nrow}(\mathbf{X}), 1)))$ selects the lower triangular matrix of a squared matrix \mathbf{X} (with $\text{nrow}(\mathbf{X}) = \text{ncol}(\mathbf{X})$) and thus, can be extracted via $\text{lower.tri}(\mathbf{X})$. In contrast, dynamic rewrites have additional size constraints: for example, $\text{cumsum}(\mathbf{X}) \rightarrow \mathbf{X}$ only applies if \mathbf{X} is a single-row matrix. Finally, we also support cumulative aggregates in SystemML’s code generation framework, which similarly requires size information for costing and validity constraints [Bo18].

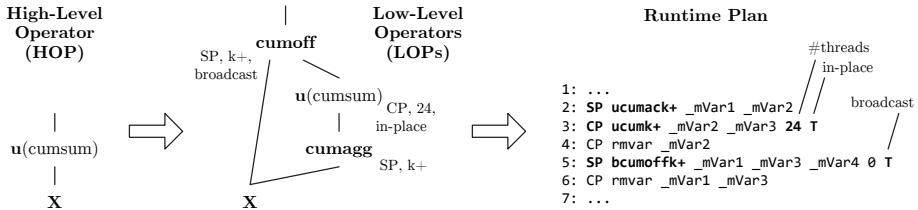


Fig. 5: Compilation Chain of Cumulative Aggregates.

Execution Plan Generation: After rewrites, we then use the size information for computing memory estimates. Compared to the local memory budget, these estimates are in turn used for execution type selection of local (CP) or distributed Spark (SP) operations. A CP operation maps to a multi-threaded physical operator, while an SP operation maps to a DAG of operators as shown in Figure 5. For a single-iteration $\text{cumsum}(\mathbf{X})$, we compile a unary SP cumagg , a CP cumsum , and a binary SP cumoff . These operators are annotated with the execution types, operation types (e.g., Kahan addition [TTR12]) and additional flags like broadcasting, number of threads, and in-place updates. This LOP DAG may also include checkpoints for caching if needed. In case of $\text{nrow}(\mathbf{X}) \leq b$, we compile only a cumoff of \mathbf{X} and a constant vector because the single row block does not require offsets. Finally, we generate runtime instructions from the LOP DAG in a depth-first traversal. Based on live variable analysis, this runtime plan also includes rmvar instructions for cleaning up temporary intermediates and their lineage-aware broadcasts and RDD variables [Bo16].

Runtime Operators: Important aspects of the runtime integration are the different physical operators and data transfer between local and distributed operations:

- *CP cumsum operator:* For local in-memory operations of f_{cumagg} , we provide a basic operator with copy-on-write semantics. This operator is parameterized with the specific f_{cumagg} and allows for single- or multi-threaded operations. In case of multi-threading, we compute offsets similar to the DistCumAgg framework but with static range partitioning in the number of threads. Optionally, this operator performs in-place updates—as shown in Figure 5—to avoid the expensive output allocation.
- *Spark Partial Cumulative Aggregate:* The data-parallel Spark ucumack+ aggregation (e.g., ucumack+ in Figure 5) is an operator for performing f_{agg} during the forward cascade. In detail, this entails (1) a data-local block aggregation into a row of partial column aggregates, (2) the insertion of this row into its position of an empty target block, and (3) the global merge of these partial blocks. For memory efficiency and scalability, partial blocks are communicated in sparse format.
- *Spark Cumulative Offset:* The data-parallel Spark bcumoffk+ offset aggregation, then applies the offsets via f_{off} and performs the final block-local f_{cumagg} . Performance is largely influenced by the join of data and offsets. If configured for broadcast, we use an efficient broadcast join that avoids data shuffling and overcomes Spark’s limitation

of 2GB broadcasts with partitioned broadcasts [Bo16]. Otherwise, we parallelize the offsets, split them into rows, and fall back to a repartition join. Similarly, if the data is hash partitioned, we also use a repartition join because Spark exploits the partitioning for shuffling only offsets [Bo16]. Finally, we perform a zero-copy offset aggregation during f_{cumagg} to avoid an additional copy-on-write, and thus, GC overheads.

The data transfer between CP and Spark operations is handled through SystemML's bufferpool [Bo16]. For example, each CP operation pins its inputs via `acquireRead`, which under the covers evaluates pending RDD operations and collects the output. To reduce latency, we overlap the RDD evaluation with the allocation of the local matrix.

4.2 Cumulative Aggregates in Other Operations

Cumulative aggregates are not just versatile tools at script level but also used as key primitives in other operations with complex dependencies. Examples are the removal of empty rows or columns via `removeEmpty`, the subsampling of rows and columns via permutation matrix multiplication, and the computation of cumulative distribution functions.

Semantics of `removeEmpty`: We use `removeEmpty` as an example. For instance, the expression `removeEmpty(target=X, margin="rows", select=rowSums(X!=0)>=thrU)` removes users—i.e., rows—with too few ratings ($< thrU$) from a ratings matrix \mathbf{X} in preparation for matrix factorization. Generally, `removeEmpty` selects non-empty rows or columns, or if a `select` vector is provided all rows or columns whose vector entry is non-zero.

Data-Parallel `removeEmpty`: Parallelizing `removeEmpty` is challenging over distributed, blocked matrices because the decision on the removal of a row depends on all columns of this row, and the output position of a row depends on the removal decisions of all previous rows. However, with cumulative aggregates, parallelization becomes simple. To explain this, we decompose `removeEmpty` into separate phases. Local operations exploit sparse formats and early-out opportunities, but here we focus on data-parallel operations only:

- *Selection Vector Computation* (No-op if user-provided): For distributed data-parallel operations, we simply compile distributed `rowMaxs($\mathbf{X} \neq 0$)` or `colMaxs($\mathbf{X} \neq 0$)` operations, respectively. These operations result in a 0/1 indicator vector \mathbf{s} .
- *Row/Column Selection*: Subsequently, we want to select rows/columns from the blocked representation of \mathbf{X} according to \mathbf{s} into the blocked representation of the output. The key to accomplish that is a potentially distributed $\mathbf{r} = \text{cumsum}(\mathbf{s})$, where \mathbf{r}_i holds the output position of the i th input row. This is similar to parallel packing of selected elements [CBZ90] and filtering through compaction [Ho05]. Finally, we simply reshuffle and merge relevant rows/columns into the target representation.

During compilation of `removeEmpty`, we construct a temporary HOP DAG including $u(\text{cumsum})$ to prepare the \mathbf{r} vector. This way, we reuse the entire compiler and runtime integration described in Section 4.1 including hybrid runtime plans and internal optimizations.

5 Experiments

Our experiments study both local and distributed cumulative aggregates. First, we compare SystemML with numerical computing frameworks such as R and Julia for singlenode operations. Second, we investigate the scalability of distributed cumulative aggregates, including the impact of internal optimizations, and a comparison with MPI.

5.1 Experimental Setting

Cluster Setup: We ran our experiments on a commodity 2+10 node cluster of two head nodes, and 10 worker nodes. All nodes have two Intel Xeon E5-2620 CPUs @ 2.1 GHz-2.5 GHz (24 virtual cores), 128 GB DDR3 RAM @ 1.3 GHz, six 3 TB disks, 1Gb Ethernet, a nominal peak memory bandwidth of 2×43 GB/s, and run CentOS Linux 7.2. We used OpenJDK 1.8.0_151, Hortonworks 2.5, Apache Hadoop 2.7.3, and Apache Spark 2.3.1, in yarn-client mode, with 10 executors, 19 cores per executor, 40 GB driver memory, 60 GB executor memory, and default memory fractions (0.5/0.6), which results in an aggregate cluster memory for data between $10 \cdot 60 \text{ GB} \cdot 0.5 = 300 \text{ GB}$ and 360 GB.

Baselines and Data: Our baselines are cumsum built-in operations and the straw-man scripts from Figure 1 in SystemML 1.2++ (12/2018) [Bo16], Julia 0.7 [Be17], and R 3.5, as well as a C-based MPI baseline using OpenMPI 3.1.3. All systems use double precision (i.e., FP64); SystemML is configured with a default block size of $b = 1\text{K}$ and a local memory budget of 70% of the maximum heap size. Finally, we use synthetic data for studying these baselines on a variety of scenarios with different data sizes.

5.2 Baseline Comparison

In a first set of experiments, we compare the script-level straw-man implementations and built-in support in SystemML, R, and Julia to provide a baseline for distributed operations. We use a single worker node, with `-Xmx96g -Xms96g -server` for SystemML, unlimited memory for R and Julia, and we report the average runtime of 100 operations.

Increasing Data Size: For a basic comparison, we fix the number of columns as $m = 256$ and systematically increase the number of rows. Figure 6 shows the results with log-scaled

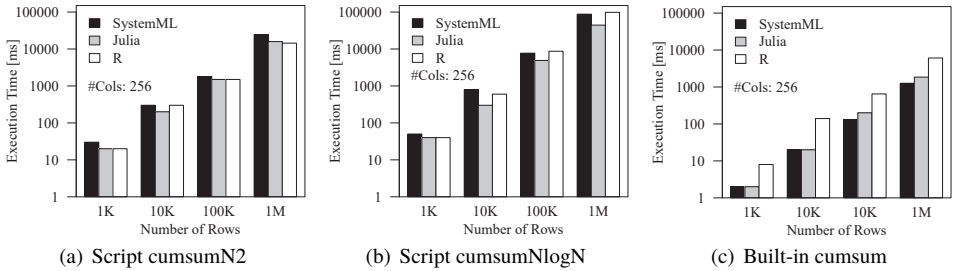


Fig. 6: Cumsum Performance with Increasing Number of Rows, Constant Number of Columns.

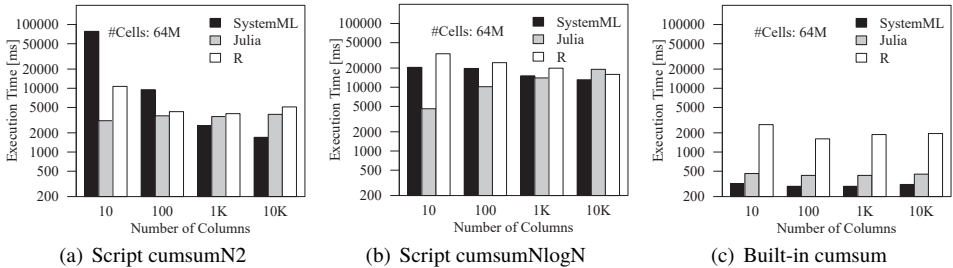


Fig. 7: Cumsum Performance with Increasing Number of Columns, Decreasing Number of Rows.

axes. First, although cumsumN2 and cumsumNlogN are amenable to update-in-place⁵, these script level functions are one to two orders of magnitude slower than the built-in functions. Here, cumsumNlogN shows higher overhead than cumsumN2 due to larger intermediates. However, without update-in-place rewrite in SystemML, cumsumNlogN largely outperforms cumsumN2 (e.g., 0.9 s vs. 86.7 s for 10K rows). Second, Julia performs best on cumsumN2 and cumsumNlogN because for SystemML and R, instruction interpretation overhead constitutes a bottleneck at $n = 256$ columns. Third and most importantly, the built-in functions of SystemML, and Julia show very similar performance, while R has additional overhead for its column-wise apply. SystemML slightly outperforms Julia due to multi-threaded operations, which are, however, dominated by result allocation.

Varying Matrix Shape: To validate the observation of high instruction interpretation overheads, Figure 7 shows the results for a constant data size of 64M cells (512 MB) but increasing number of columns (and thus, decreasing number of rows). First, Figure 7(c) shows that all built-in functions are rather invariant to the number of columns. Second, Figures 7(a) and 7(b) show that SystemML’s and R’s performance increases with increasing number of columns and thus decreasing number of operations, which is especially pronounced for cumsumN2 (Figure 7(a)) where the number of operations is linear in the number of rows. In

⁵ SystemML and R have copy-on-write semantics but apply update-in-place via rewrites and reference counting. In contrast, Julia uses update-in-place by default and requires an explicit $B = \text{copy}(A)$ if this is not desired.

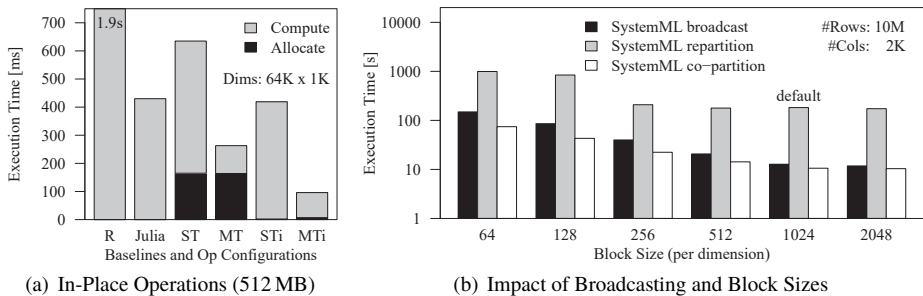


Fig. 8: Impact of Multi-Threading, In-place Updates, Broadcasting and Block Sizes.

fact, SystemML outperforms R and Julia for these script-level algorithms if the number of columns is sufficiently large. Surprisingly, Julia shows a slowdown with increasing number of columns, especially for `cumsumNlogN`, which is likely due to GC overheads.

In-Place Operations: For the sake of a better understanding of builtin operations performance, Figure 8(a) shows the impact of multi-threading and in-place updates, as well as a break-down of computation times. Here, ST/MT stand for single- and multi-threading, while the *i*-suffix indicates in-place updates. Multi-threading improves the compute time by 4.7x, but its runtime is then dominated by result allocation. Hence, in-place operations—as applied between forward and backward cascade—further significantly improve performance.

5.3 Impact of Broadcasting and Block Sizes

In a second set of experiments, we evaluate distributed cumulative aggregates in SystemML for the common scenario of dense matrices that fit in aggregate cluster memory and require only $k = 1$ iterations. We use a—randomly generated—dense $10M \times 2K$ input matrix (i.e., 160 GB) and study the impact of broadcasting, partitioning, and block sizes. To force Spark’s lazy evaluation, we report the average runtime of 100 `print(min(cumsum(X))` evaluations, including creating the Spark context once (≈ 15 s).

Broadcast vs Repartition Join: As described in Section 4.1, joining the data and computed offsets via a broadcast or co-partition join instead of a repartition join, allows for data-local computation without shuffling of the main input. Broadcast joins apply only to the innermost iteration, while co-partition joins apply to all iterations, except the outermost iteration unless the data is already partitioned. Here, we force the individual types and repartition the data for co-partition joins just once. Figure 8(b) shows significant improvements with broadcast/co-partition joins—whose differences will be analyzed separately—compared to a repartition join by up to 19.6x (17.3x at default block size of $b = 1K$).

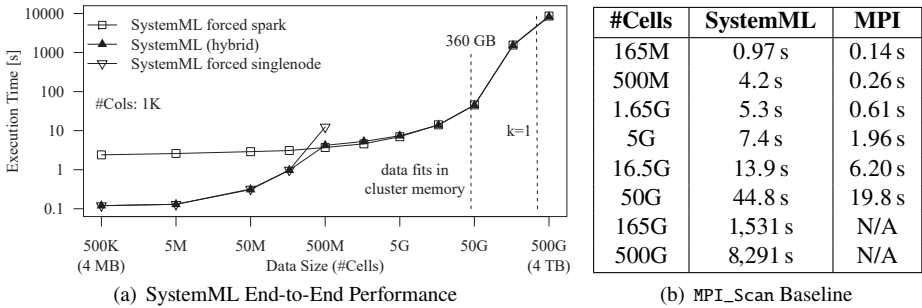


Fig. 9: Scalability with Increasing Data Size (from 4 MB to 4 TB).

Impact of Block Sizes: In SystemML, we use 1K as the default block size because it offers a great balance between (1) low per-block memory requirements and good cache behavior, as well as (2) amortized block overheads. For distributed cumulative aggregates, the block size also influences the size reduction per iteration, and thus, the number of partial aggregates, the number of iterations, and the broadcast overhead. In Figure 8(b), we observe—despite the moderate range of block sizes from 64 to 2048—significant performance impact of up to more than an order of magnitude. Repartition joins are less sensitive to block sizes because they are dominated by shuffling, but the default of 1K is generally a good compromise.

5.4 Scalability

In a third set of experiments, we test the scalability of cumulative aggregates over a spectrum of data and cluster sizes. We use a 10 GB driver and report the average runtime of `r print(min(cumsum(X)))` evaluations, including creating the Spark context once.

From 4 MB to 4 TB: Figure 9(a) shows the runtime of SystemML’s execution modes—with 10 GB driver and $r = 10$ repetitions—with increasing data size. Forced single-node operations are fast for small data but run out-of-memory at 40 GB. Already at 4 GB, we see a slowdown because pinning 2×4 GB in a 10 GB driver leads to evictions on every cumsum. In contrast, forced Spark operations show high overhead for small data due to Spark context creation, low parallelism in the number of partitions, and distribution overheads, but good performance for distributed, in-memory data and scalability for larger datasets. When the data size exceeds cluster memory at > 50 G cells, we see a 34.2x slowdown for 3.3x increase in data size. Similarly, multi-iteration, repartition-based operations cause another 5.4x slowdown for the next 3.3x size increase. SystemML’s default hybrid mode combines the advantages and provides efficiency for small as well as scalability for large datasets. Table 9(b) further shows an idealized MPI baseline (implemented in C) using 10 nodes and 19 slots per node. This baseline performs worker-local data generation, as well as $r = 10$ repetitions of a local f_{agg} , `MPI_Scan (MPI_SUM)`, local f_{cumoff} , f_{cumagg} , and aggregation, as

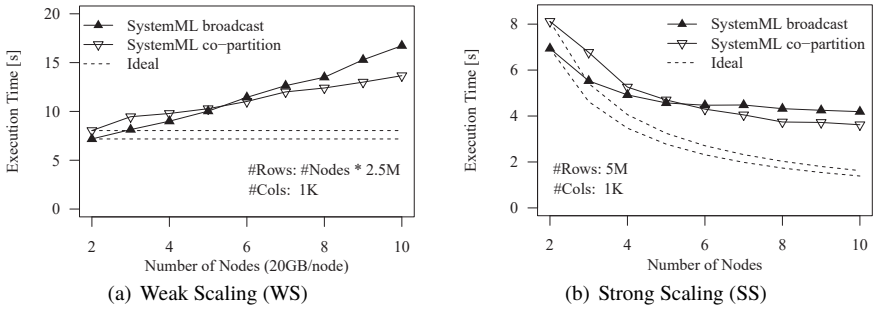


Fig. 10: Scalability with Increasing Cluster Size (from 2 to 10).

well as a final `MPI_Reduce (MPI_MIN)`. For sufficiently large data, SystemML shows—despite the handling of unordered blocks—only a 2.3x slowdown and thus, competitive performance.

Weak and Strong Scaling: Finally, we study the weak and strong scaling behavior of broadcast- and co-partition-based cumulative aggregates with $r = 100$ repetitions. First, for weak scaling (WS), we simultaneously increase the data and cluster size and thus, expect constant runtime. Figure 10(a) shows moderately increasing runtimes but generally good scaling behavior. Specifically, we have a 2.3x and 1.7x slowdown when increasing from (2 nodes/40 GB) to (10 nodes/200 GB). Second, for strong scaling (SS), we increase the cluster size but keep the data size constant at 40 GB. Comparing the runtime with 2 and 10 nodes, we expect a speedup of 5x but observe only 1.7x and 2.3x. This is due to the small data size of 40 GB, a moderate number of $\approx 40 \text{ GB} / 128 \text{ MB} = 312$ partitions (and thus, tasks), and a decreasing number of task waves $\lceil 312 / (19 \cdot \#Nodes) \rceil$. Overall, co-partitioning scales better because the initial partitioning is amortized over 100 repetitions, and piecewise broadcast fetching shows high latency over 1Gb Ethernet. To validate this explanation, we repeated the broadcast experiments on a 6 node cluster of similar HW but 10Gb Ethernet. When scaling from 2 to 6 nodes, we see a significantly better WS runtime overhead of 1.3x instead of 1.6x, and an SS speedup of 2.3x instead of 1.5x.

6 Related Work

We review related work from high-performance computing (HPC), database systems, recent large-scale ML systems, and the more general problem of sum-product optimization.

HPC Parallel Prefix Scans: Parallel prefix sums are a key building block in many HPC applications and therefore well-studied in the literature. Hillis and Steele presented an $O(n \log n)$ parallel scan⁶ (prefix) operation [HS86] for computing prefix sums—similar to `cumsumNlogN` in Figure 1—that generalized to associative operations. Blelloch later

⁶ The name *scan* was derived from the APL [FI73] plus-scan for computing vector prefix sums [BI93].

publicized more efficient parallel prefix scans, where for n data items and p processors, each processor sums up $\lceil n/p \rceil$ elements, and partial offsets are computed via an up- and down-sweep through a virtual aggregation tree [B189; B193; CBZ90]. These scan primitives are widely applicable to evaluate polynomials, solve recurrences or tridiagonal linear systems, and even implement radix- and quick-sort [CBZ90]. Sanders further analyzed and compared alternatives for MPI_Scan [ST06]. Parallel algorithms for first- and higher-order recurrences have also been theoretically investigated [CK75; HK77; KMW67; KS73]. More recently, several works introduced efficient scan primitives for GPU [Ho05; HSO07] and FPGA [AS14] devices. However, all of these works focused on traditional *message-passing* or *shared-memory* HPC systems with random data access. In contrast, we described efficient cumulative aggregates for data-parallel ML systems with blocked matrix representations on top of frameworks such as MapReduce [DG04], Spark [Za12], or Flink [A114].

Window Functions in Database Systems: Recently, the parallelization of cumulative aggregates has also received attention in the context of efficiently evaluating SQL window functions [Be13; Le15]. Database systems primarily parallelize over independent partitions of the PARTITION BY clause, but additional approaches exist for large partitions. First, Bellamkonda et al. introduced a method [Be13] that includes the ORDER BY keys into Oracle’s data distribution keys to create artificial partitions. Based on an existing range partitioning and sorting, the query coordinator then collects partial aggregates and distributes offsets back. Second, Leis et al. introduced the dynamic classification of partitions for inter- and intra-partition parallelism [Le15], where the latter relies on parallel sorting, and materializing a segment tree of partial aggregates, which can be computed in parallel. In contrast to our DistCumAgg framework, these works rely on partitioning and sorting, focus primarily on shared memory, and do not support complex cumulative aggregates.

Large-Scale ML Systems: Predominant architectures for large-scale—i.e., distributed—ML Systems are (1) data- or model-parallel parameter servers [De12; Li14], and (2) systems for data- and/or task-parallel distributed operations. Distributed cumulative aggregates are only relevant for the second category of data-parallel ML systems. In this context, systems such as RIOT [ZHY09], PEGASUS [KTF09], SystemML [Bo16], SciDB [St11], Cumulon [HBY13], Distributed R [Ma16], DMac [YSC15], Spark MLlib’s block matrices [Za16], Gilbert [Ro17], MatFast [Yu17], and SimSQL [Lu17] all rely on blocked matrix representations, which means that our DistCumAgg framework could seamlessly be applied in these systems. Yet, to the best of our knowledge, SystemML is the only ML system that provides built-in support for distributed cumulative aggregates.

Sum-Product Optimization: In contrast to most ML systems like OptiML [Su11], Theano [Be10], SystemML [Bo16], and TensorFlow XLA [Go] that perform simplification rewrites via heuristic pattern matching, sum-product optimization aims at a systematic exploration of rewrite opportunities. Existing work such as AMF [MP99] and SystemML-SPOOF [E117] use axioms and algebraic properties, or elementary transformation rewrites to systematically explore valid, alternative plans. Due to the complexity of reasoning about recurrences, none of these sum-product frameworks support cumulative aggregates yet. However, it is

an interesting direction for future work to automatically derive rewrites for these complex cumulative aggregates, especially regarding the interactions with other operations.

7 Conclusions

To summarize, we introduced the generic DistCumAgg framework for efficient, data-parallel cumulative aggregates over distributed, blocked matrix representations. We described the end-to-end compiler and runtime integration in SystemML, and physical operators that seamlessly fit into hybrid runtime plans of local and distributed operations. Our experiments have shown competitive performance for local and distributed in-memory operations, as well as almost linear scalability with increasing data size and degree of parallelism. In conclusion, many presumably sequential operations can be efficiently computed over distributed, blocked matrix representations on top of data-parallel frameworks like Spark. Cumulative aggregates are used in a variety of algorithms and for data preprocessing, which makes them valuable tools for data scientists. Interesting future work includes sum-product rewrites, dedicated GPU operators, and other recursive computations like time series analysis.

Acknowledgments: We thank José Molero for help with the MPI experiments and our anonymous reviewers for their valuable comments and suggestions.

References

- [Al14] Alexandrov, A. et al.: The Stratosphere platform for big data analytics. *VLDB J.* 23/6, 2014.
- [AS14] Arap, O.; Swamy, M.: Offloading MPI Parallel Prefix Scan (MPI_Scan) with the NetFPGA. *CoRR abs/1408.4939/*, 2014.
- [Be10] Bergstra, J. et al.: Theano: a CPU and GPU Math Expression Compiler. In: *SciPy*. 2010.
- [Be13] Bellamkonda, S. et al.: Adaptive and Big Data Scale Parallel Execution in Oracle. *PVLDB* 6/11, 2013.
- [Be17] Bezanson, J.; Edelman, A.; Karpinski, S.; Shah, V.B.: Julia: A Fresh Approach to Numerical Computing. *SIAM Review* 59/1, 2017.
- [Bl10] Blanas, S. et al.: A comparison of join algorithms for log processing in MapReduce. In: *SIGMOD*. 2010.
- [Bl89] Bletloch, G. E.: Scans as Primitive Parallel Operations. *IEEE Trans. Computers* 38/11, 1989.
- [Bl93] Bletloch, G. E.: Prefix Sums and Their Applications, tech. rep., CMU-CS-90-190, <https://www.cs.cmu.edu/~guyb/papers/Bl93.pdf>, 1993.
- [Bo14] Boehm, M. et al.: SystemML's Optimizer: Plan Generation for Large-Scale Machine Learning Programs. *IEEE Data Eng. Bull.* 37/3, 2014.
- [Bo16] Boehm, M. et al.: SystemML: Declarative Machine Learning on Spark. *PVLDB* 9/13, 2016.

- [Bo18] Boehm, M. et al.: On Optimizing Operator Fusion Plans for Large-Scale Machine Learning in SystemML. PVLDB 11/12, 2018.
- [CAL94] Cohn, D. A.; Atlas, L. E.; Ladner, R. E.: Improving Generalization with Active Learning. Machine Learning 15/2, 1994.
- [CBZ90] Chatterjee, S.; Bletloch, G. E.; Zaghera, M.: Scan primitives for vector computers. In: SC. 1990.
- [CGJ94] Cohn, D. A.; Ghahramani, Z.; Jordan, M. I.: Active Learning with Statistical Models. In: NIPS. 1994.
- [CK75] Chen, S.; Kuck, D. J.: Time and Parallel Processor Bounds for Linear Recurrence Systems. IEEE Trans. Computers 24/7, 1975.
- [Co09] Cohen, J.; Dolan, B.; Dunlap, M.; Hellerstein, J. M.; Welton, C.: MAD Skills: New Analysis Practices for Big Data. PVLDB 2/2, 2009.
- [Da10] Das, S. et al.: Ricardo: integrating R and Hadoop. In: SIGMOD. 2010.
- [De12] Dean, J. et al.: Large Scale Distributed Deep Networks. In: NIPS. 2012.
- [DG04] Dean, J.; Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. In: OSDI. 2004.
- [El17] Elgamal, T. et al.: SPOOF: Sum-Product Optimization and Operator Fusion for Large-Scale Machine Learning. In: CIDR. 2017.
- [FI73] Falkoff, A. D.; Iverson, K. E.: The Design of APL. IBM Journal of Research and Development 17/5, 1973.
- [Gh11] Ghoting, A. et al.: SystemML: Declarative machine learning on MapReduce. In: ICDE. 2011.
- [Go] Google: TensorFlow XLA (Accelerated Linear Algebra), tensorflow.org/performance/xla/.
- [HBY13] Huang, B.; Babu, S.; Yang, J.: Cumulon: Optimizing Statistical Data Analysis in the Cloud. In: SIGMOD. 2013.
- [HK77] Hyafil, L.; Kung, H. T.: The Complexity of Parallel Evaluation of Linear Recurrences. J. ACM 24/3, 1977.
- [Ho05] Horn, D.: Stream Reduction Operations for GPGPU Applications. In: GPU Gems 2. 2005.
- [HS86] Hillis, W. D.; Steele Jr., G. L.: Data Parallel Algorithms. Commun. ACM 29/12, 1986.
- [HSO07] Harris, M.; Sengupta, S.; Owens, J. D.: Parallel Prefix Sum (Scan) with CUDA. In: GPU Gems 3. 2007.
- [KMW67] Karp, R. M.; Miller, R. E.; Winograd, S.: The Organization of Computations for Uniform Recurrence Equations. J. ACM 14/3, 1967.
- [KS73] Kogge, P. M.; Stone, H. S.: A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations. IEEE Trans. Computers 22/8, 1973.
- [KTF09] Kang, U.; Tsourakakis, C. E.; Faloutsos, C.: PEGASUS: A Peta-Scale Graph Mining System. In: ICDM. 2009.
- [Le15] Leis, V.; Kundhikanjana, K.; Kemper, A.; Neumann, T.: Efficient Processing of Window Functions in Analytical SQL Queries. PVLDB 8/10, 2015.
- [Li14] Li, M. et al.: Scaling Distributed Machine Learning with the Parameter Server. In: OSDI. 2014.

- [Lu17] Luo, S.; Gao, Z. J.; Gubanov, M. N.; Perez, L. L.; Jermaine, C. M.: Scalable Linear Algebra on a Relational Database System. In: ICDE. 2017.
- [Ma16] Ma, E.; Gupta, V.; Hsu, M.; Roy, I.: dmapply: A Functional Primitive to Express Distributed Machine Learning Algorithms in R. PVLDB 9/13, 2016.
- [Me03] Melton, J.: ISO-ANSI Working Draft SQL/Foundation, 2003.
- [MP99] Menon, V.; Pingali, K.: High-Level Semantic Optimization of Numerical Codes. In: ICS. 1999.
- [Ra16] Ratner, A. J.; Sa, C. D.; Wu, S.; Selsam, D.; Ré, C.: Data Programming: Creating Large Training Sets, Quickly. In: NIPS. 2016.
- [RMJ06] Raghavan, H.; Madani, O.; Jones, R.: Active Learning with Feedback on Features and Instances. *Journal of Machine Learning Research* 7/, 2006.
- [Ro17] Rohrmann, T.; Schelter, S.; Rabl, T.; Markl, V.: Gilbert: Declarative Sparse Linear Algebra on Massively Parallel Dataflow Systems. In: BTW. 2017.
- [Sc15] Schelter, S. et al.: Efficient Sample Generation for Scalable Meta Learning. In: ICDE. 2015.
- [ST06] Sanders, P.; Träff, J. L.: Parallel Prefix (Scan) Algorithms for MPI. In: PVM/MPI. 2006.
- [St11] Stonebraker, M.; Brown, P.; Poliakov, A.; Raman, S.: The Architecture of SciDB. In: SSDBM. 2011.
- [Su11] Sujeeth, A. K. et al.: OptiML: An Implicitly Parallel Domain-Specific Language for Machine Learning. In: ICML. 2011.
- [Tr18] Tremblay, J. et al.: Training Deep Networks With Synthetic Data: Bridging the Reality Gap by Domain Randomization. In: CVPR Workshops. 2018.
- [TTR12] Tian, Y.; Tatikonda, S.; Reinwald, B.: Scalable and Numerically Stable Descriptive Statistics in SystemML. In: ICDE. 2012.
- [YSC15] Yu, L.; Shao, Y.; Cui, B.: Exploiting Matrix Dependency for Efficient Distributed Matrix Computation. In: SIGMOD. 2015.
- [Yu17] Yu, Y. et al.: In-Memory Distributed Matrix Computation Processing and Optimization. In: ICDE. 2017.
- [Za12] Zaharia, M. et al.: Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In: NSDI. 2012.
- [Za16] Zadeh, R. B. et al.: Matrix Computations and Optimization in Apache Spark. In: KDD. 2016.
- [ZHY09] Zhang, Y.; Herodotou, H.; Yang, J.: RIOT: I/O-Efficient Numerical Computing without SQL. In: CIDR. 2009.

Challenges in Data Processing

From Natural Language Questions to SPARQL Queries: A Pattern-based Approach

Nadine Steinmetz,¹ Ann-Katrin Arning,¹ Kai-Uwe Sattler¹

Abstract: Linked Data knowledge bases are valuable sources of knowledge which give insights, reveal facts about various relationships and provide a large amount of metadata in well-structured form. Although the format of semantic information – namely as RDF(S) – is kept simple by representing each fact as a triple of subject, property and object, the access to the knowledge is only available using SPARQL queries on the data. Therefore, Question Answering (QA) systems provide a user-friendly way to access any type of knowledge base and especially for Linked Data sources to get insight into the semantic information. As RDF(S) knowledge bases are usually structured in the same way and provide per se semantic metadata about the contained information, we provide a novel approach that is independent from the underlying knowledge base. Thus, the main contribution of our proposed approach constitutes the simple replaceability of the underlying knowledge base. The algorithm is based on general question and query patterns and only accesses the knowledge base for the actual query generation and execution. This paper presents the proposed approach and an evaluation in comparison to state-of-the-art Linked Data approaches for challenges of QA systems.

1 Introduction

Question answering (QA) is a research discipline at the intersection of natural language processing (NLP), information retrieval, and database processing aiming at answering questions formulated in natural language. Though, QA is a rather old research problem with early system solutions dating back to the sixties, the field has gotten great attention and research made a significant progress over the last few years. This can be exemplified by IBM's DeepQA system Watson which won the quiz show Jeopardy! in 2011. Another well-known example are the personal assistants based on voice recognition such as Apple Siri, Amazon Alexa or Microsoft Cortana which are able not only to execute spoken commands (e.g. "Put x on my shopping list") but also to answer (simple) questions in natural language. However, leveraging large knowledge bases to answer complex questions, supporting question types beyond factoid questions, and dealing with ambiguities are still challenging problems.

QA works usually in a sequence of the following steps: (1) question parsing and focus detection, (2) question classification, (3) query generation, (4) answer candidate generation (query execution), and (5) result ranking. In our work we focus on the steps (3) to (5) where

¹ TU Ilmenau, Databases & Information Systems Group, Ilmenau, Germany, *first.last@tu-ilmenau.de*

we consider structured databases / knowledge bases as sources for answers. Our goal is to provide a *generic* approach not hardcoded for a specific schema or database. For this purpose, we leverage a RDF database such as DBpedia and generate SPARQL queries. A RDF database allows a schema-agnostic approach where the schema has not to be known for query generation because all facts are represented by triples. In addition, it allows to exploit more advanced semantic concepts such as semantic equivalence, similarity or inference mechanisms. Finally, large collections of Linked Data based on a core set such as DBpedia or Wikidata represent a great source for answering a wide range of (not only) factoid questions.

Compared to existing works our query generation approach is independent from the underlying knowledge base by representing queries to answer questions through basic graph patterns whose mapping to knowledge base-specific properties or labels are determined at runtime. Our main contributions are (i) a pattern-based approach matching common (but also complex) natural language questions, (ii) loosely coupling to an underlying knowledge base, no training or specific information required, and (iii) first evaluation results showing similar or even out-performing results compared to specifically trained systems.

2 Problem Statement

The Open Challenge on Question Answering over Linked Data (QALD)² has been organized as an evaluation campaign as part of the Extended Semantic Web Conference (ESWC) and the CLEF Initiative (Conference and Labs of the Evaluation Forum) since 2011. The challenge focuses on bringing together scientists who work on question answering and compare new approaches according to a published dataset. The latest challenge – QALD 7 – has taken place at the ESWC 2017. For the evaluation of our approach we will use the training and test datasets provided for this challenge. Evaluation results will be presented in Section 4.

As a first summary the organizers of the QALD challenge published a survey on challenges in Question Answering over Linked Data [Hö17]. The authors give an overview of approaches that have been submitted to several conferences or challenges between 2011 and 2015 – overall they list 72 publications and 62 distinct systems. After review of these systems the authors identified seven challenges developers of QA systems are facing and that addressed in the respective publications. From these challenges, we address the following:

- **Lexical Gap:** As in every QA system the phrases of natural language require to be mapped to parts of the relevant ontology and knowledge base: resources, classes, properties. We try to overcome the lexical gap by creating a set of general questions and retrieve the potentially correct result by matching result type and question type and applying a specific ranking.

² <https://qald.sebastianwalter.org/index.php?x=home&q=home>

- **Ambiguity:** The mapping of phrases to the underlying knowledge to retrieve resources, classes and properties often results in a higher amount of output than desired. We apply a scoring at each mapping step and decide at the end about the correct query containing the correctly mapped phrases by applying also the ranking of the query result.
- **Complex Queries:** SPARQL in its latest version is able to manage queries containing different operators, such as GROUP BY, COUNT, or FILTER operations. We derive the necessity of such operators in a query by identifying the question type and comparing it to the result type of different generated queries.
- **Templates:** For our approach, several question types have been analyzed and we derived respective query transformation patterns that can be applied to any domain or knowledge base w.r.t. the identified question types.
- **Independence from Knowledge Base:** In addition to the ones listed by the authors of [Hö17] we take the challenge of developing a system that works independent from the underlying knowledge base. Our approach is based on general patterns and rankings and knowledge specific lookups.

3 System Architecture

Our presented approach processes given natural language questions in seven different steps, each fulfilling an individual task. The steps are the following (also shown in Figure 1):

1. Question parsing and focus detection
2. Generation of general queries with the phrases of the natural language question according to pre-defined patterns
3. Mapping subject/predicate/object of the general question to representations within the underlying knowledge base
4. Query execution
5. Result ranking
6. Output of the highest ranked SPARQL query and the corresponding result.

Within our algorithm only step 4 and 5 are dependent on the underlying knowledge base as the concrete properties, entities and ontology classes or categories are requested. All other steps are independent from the knowledge base and can be applied to any use case.

3.1 Preliminaries – Knowledge Base Transformation

As described in the previous section, our approach is only loosely dependent on an underlying knowledge base. This means, we are able to work with any knowledge base that fulfills a few preliminaries:

- the knowledge base is constructed in RDF(S)/OWL

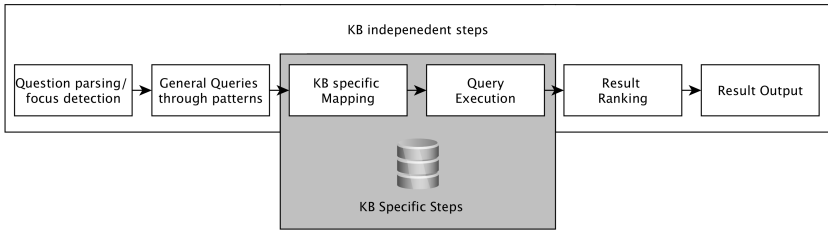


Fig. 1: Overall process of Question Answering

- there is terminological knowledge available about the used vocabulary/ontology (TBox)
- the actual facts are available as assertional knowledge (ABox).

To be able to use the knowledge base, some transformation processes have to be carried out – primarily for reasons of efficient search and lookup. The terminological part of the knowledge base (including OWL and SKOS) is analyzed for class and category labels (for the mapping/lookup process) and transferred to a easily accessible lookup store. Figure 2 shows the RDF(S) parts of a knowledge base which are used to build lookup structures for the knowledge base specific parts of our approach.

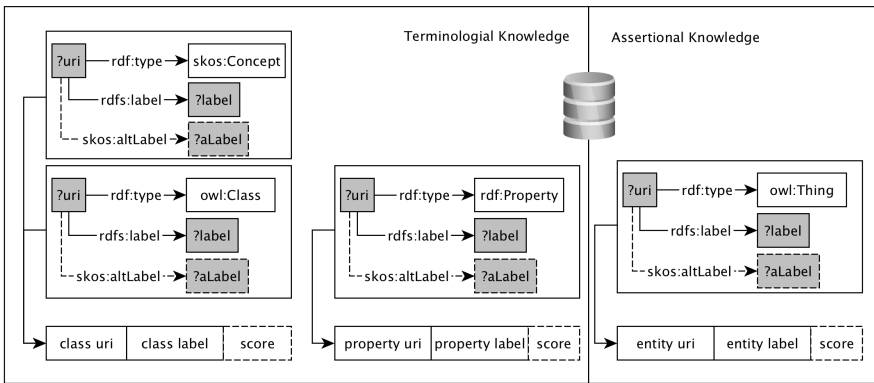


Fig. 2: RDF triples required for generation of lookup structures

We extract the class/category information from the knowledge base to be able to reference type information in questions, such as “Which university is located in Berlin?”. In some cases, the classes included in the ontology do not provide sufficient information about such type information, because the ontology is too general. For instance, for the question “Which Italian dessert contains coffee?” it is required to look up “Italian dessert” within the classes of the ontology. For instance, within DBpedia, the most specific class for this question

would be “Food”. But the entity `dbr:Tiramisu`³ (which would be a correct answer for the question) provides a fact that it is a subject of the category “Italian dessert”. Therefore, we also extract category information provided by `dc:subject`⁴ properties in some knowledge bases. The labels for properties are extracted to be able to find verb relationships within the knowledge base, such as a property for relationship between two persons: “Who supervised Alfred Kleiner?”. The labels for the entities are extracted to find the actual subject of a question within the knowledge base.

For these lookup stores (primarily indexed tables in a relational database) the original and – if available – alternative labels of classes/categories, properties (both terminological knowledge) and the actual entities (assertional knowledge) are extracted from the knowledge base.

As we have evaluated our approach based on the dataset provided by the QALD 7 challenge, the underlying knowledge would be DBpedia. For the labels of the entities DBpedia provides more information to be able to collect more synonyms for each entity. As DBpedia is derived from Wikipedia the labels of redirects and disambiguation pages can be used as additional labels. In this way, the entity `dbr:Diana,_Princess_of_Wales` are assigned 25 different labels, e.g. “lady di”, “princess diana”, “lady diana spencer” amongst others. Thereby, the probability of being able to find the correct entity mentioned in natural language is increased. However, the labels of the entities can be more or less relevant. The calculation of a relevance score helps to rank the retrieved entities for a natural language phrase. The same applies for provided alternative labels within the knowledge base (as provided by `skos:altLabel`⁵). Therefore, for each label a score is calculated between [0.0 ... 1.0]. Original labels achieve the highest score. The scores for all alternative labels are calculated according to similarity to the original label, type of the assigned entity (e.g. person’s family names), acronym format etc. The calculation for each score is described more in detail in [St14].

For the mapping of verb phrases from the question to DBpedia properties the ontology only provides original labels and no synonyms, similar to the labels for the ontology classes. This circumstance is an essential disadvantage for the mapping of properties. For instance, the fact that two people are/were married is represented by DBpedia property `spouse` while in natural language several other expressions are used, such as “wife/husband of”, “married”, “in a relationship” etc. Therefore, as an additional source for property labels we are using the PATTY dataset as described in [NWS12] to find potential properties for the verb phrases in the question. The dataset has been derived from a large collection of text documents and we calculated – similar to our entity lookup store – a score for each phrase-property mapping which represents a relevance value for the mapping of the phrase to the property. These scores are used for the ranking of the generated SPARQL queries to find the potentially most correct one for the given input question. As PATTY is a DBpedia specific dataset, our

³ DBpedia specific prefixes here and in the remainder of the paper stand for: `dbr` – <http://dbpedia.org/resource/>; `dbo` – <http://dbpedia.org/ontology/>

⁴ The Dublin Core vocabulary: <http://dublincore.org/documents/dcmi-terms/>

⁵ SKOS Core vocabulary: <https://www.w3.org/TR/2005/WD-swbp-skos-core-spec-20051102/>

approach is restricted to the label extraction as described in Figure 2 when using knowledge bases other than DBpedia.

3.2 Separate Steps of the Algorithm

As depicted in Figure 1 our algorithm consists of separate steps that are either dependent on or independent from the underlying knowledge base. The first parsing and generation of first triples is independent from the knowledge base and can be applied to any vocabulary. Afterwards, the general triples are transformed to knowledge base specific SPARQL queries and are executed on a SPARQL endpoint. In turn, the final ranking and result selection is independent from the knowledge base and builds upon previous ranking and the identified question type. The separate steps are described in detail in the following sections.

3.2.1 Knowledge Base Independent – Question Parsing and Pattern Matching

Question parsing and focus detection. The question is parsed using the Stanford lexical parser⁶ [MMM06]. The output of the parser is a parse tree (amongst others) as seen in the example of Listing 1. The parse tree reveals the sentence type as well as the word types (identified as Part-of-Speech (POS) tags) and the relations of the words among one another. In the example the sentence is of type “SBARQ” which means the sentence is “Direct question introduced by a wh-word or a wh-phrase.”⁷. To make the sentence type more specific, a second tag is classifying the actual question phrase. In our example it is “WHADVP” which means the question begins with a adverb phrase such as how or why. A full list of identified sentence/question types w.r.t. the dataset used for the evaluation of our system (cf. Section 4) is shown in Table 1. According to the question type the focus of the question is identified. Thereby, the subject that is used as result variable in the SPARQL query is identified. Within the actual question several patterns (combinations of word types) may occur which require to be translated to RDF triples for the SPARQL query. For instance, the phrases “the mayor of Chicago” and “Chicago’s mayor” result in different POS tag combinations: NP (the mayor) PP (of) NP (Chicago) and NP (Chicago’s) NN (mayor) respectively. Both combinations result in the same RDF triple: ?x onto:mayor res:Chicago .⁸ Such patterns are found in multiple question types and are dissolved independent from the identified question type. Therefore, we identified an extensive list of POS combinations which are translated to RDF triples as described in the following paragraph⁹.

List. 1: Sample parse tree for the sentence “When did princess Diana die?”

⁶ <https://nlp.stanford.edu/software/lex-parser.shtml>

⁷ <https://gist.github.com/nlothian/9240750>

⁸ The prefixes here and the remainder of the paper stand for: onto – namespace of the ontology of the underlying knowledge base; res – namespace of the resources of the underlying knowledge base

⁹ A list of example sentences, the identified patterns and the transformation to the respective SPARQL query pattern can be found here: <https://bit.ly/2R0wXPM>

```
(ROOT (SBARQ
  (WHADVP (WRB When))
  (SQ (VBD did) (NP (NNP princess) (NNP Diana))
    (VP (VB die))) (. ?)))
```

Tab. 1: List of identified question types

Sentence Type	Question Type	Example
SBARQ	WHADVP	When was the Battle of Gettysburg?
	WHADJP	How much did Pulp Fiction cost?
	WHNP	Who designed the Brooklyn Bridge?
	WHPP	In which city does the Chile Route 68 end?
S		Show me all books List all basketball players
SQ		Is Berlin the German capital?

Generation of general query triples. The natural language question then requires to be translated to RDF triples. Each RDF triple constitutes a fact which means that the phrases from the question are transferred to single facts. For instance, the question “Show me all books by Joanne K. Rowling.” includes two facts:

- the results have to be of type “book”, and
- the results are somehow created by “Joanne K. Rowling”.

The first fact results in a general triple `?x rdf:type onto:Book` . Here, the object requires to be a class from the underlying ontology. The second fact results in a general triple `?x onto:by res:Joanne_K._Rowling` .¹⁰ Here, the property requires to be included in the ontology and the object requires to be part of the knowledge base. These general triples are generated by analyzing the patterns of POS tags in the parse tree of the question. We identified an extensive list of patterns and assigned respective RDF triples. The respective subject, property and object in the RDF triple are represented by the phrases extracted from the question – as placeholders (except for pre-defined semantic properties, such as the property `rdf:type` which is part of the RDF vocabulary and used as property to state the class membership of a resource of the knowledge base). RDF facts are represented using properties to connect a subject and an object. Due to the fact that our approach is working independent from the underlying knowledge base, we do not know beforehand in which order properties connect subject and object. For instance, for the phrase “supervisor” the knowledge base could contain the property `onto:supervisor` which connects a student in the subject with the respective supervisor in the object. The knowledge base could also contain the property `onto:supervisorOf` which connects the supervisor in the subject with the student in the object. Therefore, we provide two versions for each triple generated from a POS tag combination resulting in more than one SPARQL query for each question. All queries are scored and ranked in a later step.

¹⁰ The triple looks like this before mapping the property and the object to the underlying knowledge base.

3.2.2 Knowledge Base Specific – Mapping and Execution

After the question is parsed and the general triples are generated the underlying knowledge base is taken into account to create the final queries including the knowledge bases specific entities, properties and classes/categories. Therefore, the extracted phrases from the parsed question are replaced URIs and finally the question specific aggregations and other operators are added. The queries are then executed on a SPARQL endpoint.

Subject/property/object mapping. In this step, the generated general queries are transferred to actual RDF triples as specific for the underlying knowledge base.

For the mapping of the subjects of the general triples the extracted phrases from the parse tree are directly looked up. For the objects, three different options are possible:

- direct lookup in case an entity is required according to the pre-defined pattern
- lookup for ontology classes in case a triple for `rdf:type` information is required
- the phrase is inserted as literal.

All mapping processes – for subject, property and object – may result in more than one result per phrase. For each result a query is generated in combination with all already derived queries. Taking into account the mapping scores from `PATY` and the entity lookup an overall score for each generated query is calculated by multiplying all derived scores.

The result of this step is a set of RDF triple combinations (as required within the `WHERE` clause of a SPARQL query) and the corresponding mapping score for each combination. In the next step the final queries are generated and executed.

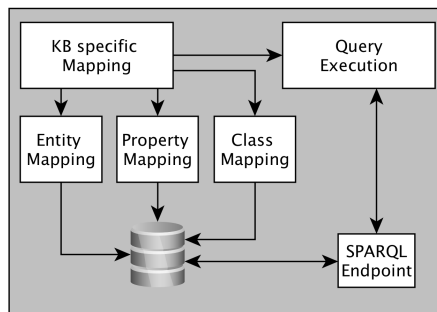
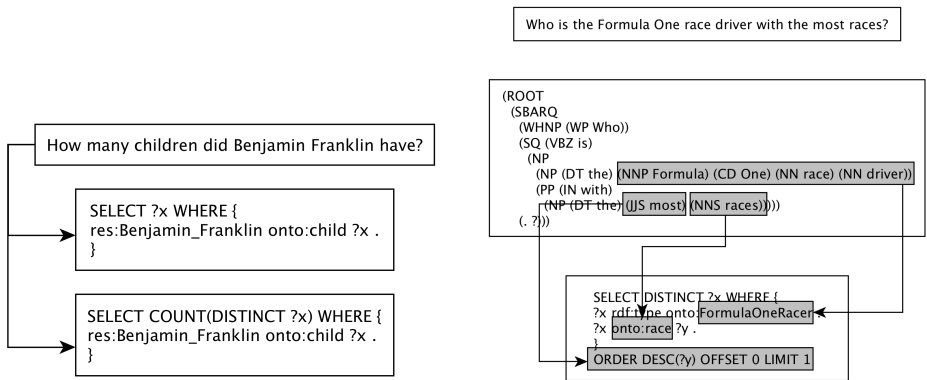


Fig. 3: Components specific for the Underlying Knowledge Base

Query execution. In this step the final queries are created and necessary operators added. The required operators are derived from the question type. For instance, for a `WHADJP` question – starting with “How many” or “How often” – the expected answer should be a

number. As the underlying knowledge base is unknown – as one of the challenges – to our system, the result type “number” can be derived in two different ways: either the range of the mapped property already provides a number or a COUNT operator has to be applied to count the number of resulting entities. Therefore, we create queries for each possibility, such as the example in Figure 4a depicts. In addition to the COUNT operator, other aggregation operators might be applied to the query. For instance, the question “Who is the Formula One race driver with the most races?” requires ORDER, LIMIT and OFFSET operators in the query. According to the POS patterns the required triples are generated and the respective variable used for the ORDER operation is identified. The type of ordering is identified according to pre-defined list of phrases: “most”, “highest”, “tallest” etc. for descending order or “least”, “smallest”, “youngest” for ascending order. Figure 4b shows the resulting SPARQL query for the mentioned example. After all queries are created and all required operators are applied the resulting queries are executed on the specific SPARQL endpoint.



(a) ... for questions where a number is expected as answer type. (b) ... for questions where ordering of results is required.

Fig. 4: Examples of query generation

3.2.3 Knowledge base Independent – Finalization and Result Output

Result ranking. For each query the results are derived and final ranking is applied according to the expected answer type (as pre-defined from the question type). For most of the question types the answer type is an entity or a list of entities. For question starting with “When” the resulting answer must be a date. And for questions starting with “How many” or “How often” the answer type is a number. The final ranking of the queries and the respective results is applied by comparing the expected answer type and the actual answer type. If the expected and actual answer type are matching the query receives the full score of 1.0. If the query does not produce any result the score naturally is set 0.0. If the query produces a result, but the answer type is not matching to the expected one, the mapping score is set to

0.5. We do not suppress answers with incorrect answer types, because of several reasons. On the one hand, knowledge bases (especially the ones that are automatically extracted or created) some times do not provide any literal types. Dates might not be formatted as date, but still be a correct answer. On the other hand, a question starting with "When" might often require a date as answer type, but a year ("1980") or a season ("Spring 1968") might also be a reasonable answer and contained in the underlying knowledge base like this (instead of a date).

The overall ranking of a query including results is calculated from the mapping score and the answer type score by multiplying both scores. Each query only receives one overall score. The set of results of a query (in case of a list of entities) is regarded as one answer. In this way, all queries are scored and sorted. The query with the highest score is assumed as potentially correct and the respective produced result is set as answer for the input question. At the current stage, our system assumes that a given natural language question actually has an answer. The case of a question that cannot be answered using the underlying knowledge – because of missing information¹¹ – is not taken into account, yet.

4 Evaluation

Evaluation Dataset. To evaluate our approach a dataset containing the natural language questions as well as the answers formatted as SPARQL endpoint results is required. As stated in [Hö17] only two challenges provide benchmarks with the required structure:

- Open Challenge on Question Answering over Linked Data (QALD) - as introduced in Section 6
- BioASQ – a challenge on large-scale biomedical semantic indexing and question answering¹²

The second challenge only provides questions from the bio-medical domain and the first challenge provides all-purpose QA benchmarks. The DBpedia constitutes the most well-known all-purpose knowledge base provided as RDF. We therefore choose to take into account the QALD challenge and use the dataset of the latest challenge to evaluate our approach¹³. The challenge provides a training dataset some weeks before the submission date and a test dataset to provide the final results of a submitting system. The latest challenge was the first one where participating systems were requested to provide their system as Docker image. In this way, the challenge organizers were able to evaluate the submitted systems directly – in contrast to submitted result files as XML/JSON the years before this last challenge. As the challenge already took place at the Extended Semantic Web Conference (ESWC) we are able to compare our results to the results of the participated systems.

¹¹ And according to the open world assumption, missing information might mean that something is not existent, but this is not compulsory.

¹² <http://bioasq.org/>

¹³ <https://project-hobbit.eu/challenges/qald2017/>

The QALD challenge is organized in four tasks. For our evaluation we chose “Task 1: Multilingual question answering over DBpedia” using the English version of the questions. The dataset for this task provides the following information for each question record:

- id – sequential order of the questions
- answertype – either “resource”, “string”, “number”, “date”, or “boolean”
- aggregation – true/false – stating if the required query contains aggregation operators
- onlydbo – true/false – stating if other knowledge bases than DBpedia are required
- question – the actual question provided in eight different languages, for each question additional keywords are provided
- query – a SPARQL query which provides the correct results
- results – the (list of) results – for each result the type and the value are given

For the evaluation of our system we only used the actual natural language question (without the given keywords) and the (list of) results. Everything else (answer type, result type, necessity of aggregation) is not used from the dataset and detected by our system (identified as pattern in the question).

Evaluation Measures. For the comparison of our approach to competing systems the measures Recall, Precision and F_1 -Score are used. For each question q recall, precision and F_1 -score are calculated as following:

$$\text{recall}(q) = \frac{\text{number of correct system answers for } q}{\text{number of benchmark answers for } q}$$

$$\text{precision}(q) = \frac{\text{number of correct system answers for } q}{\text{number of system answers for } q}$$

$$F_1\text{-score} = 2 * \frac{\text{recall}(q) * \text{precision}(q)}{\text{recall}(q) + \text{precision}(q)}$$

We calculated all measures as macro measures which means that they have been calculated separately for each question and all measures are averaged over all questions for the overall result.

Evaluation Results. For the QALD 7 challenge only two systems have been submitted providing results for English language: ganswer2 and WDAqua[UNC16]. Thus, we used only these two systems for comparison because only for these systems the results are reproducible. For the calculation of recall, precision and F_1 -score only questions have been taken into account for which the system was able to provide a result. We therefore calculated our measures for the evaluation in the same way. The results for the training and test dataset are shown in Table 2. Our approach achieves similar results or even outperforms the other competing system which participated in QALD 7 challenge. For QA systems the precision of the provided answers is more important than the recall: We would rather

provide an answer like “Amongst others correct answers are the following” – which means some correct answers might be missing – than proving something “Our answer is this, but it might incorrect”. Therefore, our future work will focus on an increased precision – of course in best combined with a reasonable high recall. Nevertheless, these first results show that our approach which is set up to be as independent as possible from the underlying knowledge base is able to compete with other systems that might be trained specifically for a domain and dataset.

Tab. 2: Evaluation results for the QALD 7 training and test datasets

Training Dataset	Recall	Precision	F_1 -Score
Our approach	0.588	0.570	0.578
ganswer2	0.592	0.557	0.556
WDAqua	0.540	0.490	0.510
Test Dataset	Recall	Precision	F_1 -Score
Our approach	0.665	0.584	0.622
ganswer2	0.498	0.487	0.492
WDAqua	0.160	0.162	0.161

As our approach consists of several separate steps we took a closer look at success and failure of our approach. Therefore, we evaluated each step separately and counted the questions where the step succeeded in comparison to the initial set of questions at the beginning of each step. The results are shown in Table 3. In this overview, two steps respectively the low success rates of two steps are striking: property mapping and final ranking. As described in Section 3.1 we are using the PATTY dataset for the property mapping. Unfortunately, the dataset only contains 58% of all properties contained in the DBpedia ontology. For some reason, very common properties, such as `dbo:deathDate` or `dbo:birthDate`, are missing in the dataset. This fact leads to the low success rate of 52.1% regarding the property mapping. The second lowest success rate is achieved by the overall ranking, which partly is again caused by the PATTY dataset. For each property mapping, a score is provided regarding relevance (cf. Sect. 3.1). Unfortunately, this ranking is sometimes misleading. As this ranking score (together with the scores from subject and property mapping as well as the validity check of the answer type) influences the final ranking, sometimes the wrong query is ranked highest although the used property does not match the phrase of the natural language question at all. The next paragraph will discuss the evaluation results in detail and regarding the proposed challenges.

Discussion. The evaluation results show that our system achieves similar or even better results than the systems that have been submitted to the 7th QALD challenge. However, the detailed analysis of the evaluation results reveals the weakest parts of our approach. As we are using the PATTY dataset to map natural language phrases to properties from the underlying DBpedia knowledge base, there are several questions that cannot be answered, because the dataset does not contain any property for the given phrase. This means, that

Tab. 3: Success rates of separate steps of our algorithm, evaluated on the QALD 7 training dataset

Algorithm step	Evaluation
Overall Questions	166
Focus detection successful	154 / 92.7%
Parsing successful	152 / 98.7%
Triple Generation successful	132 / 86.8%
Subject Mapping successful	127 / 96.2%
Property Mapping successful	62 / 48.8%
Object mapping successful	60 / 96.8%
Query Building successful	60 / 100%
Ranking successful	44 / 73.3%

for many questions no query can be generated, because the respective property is missing. The current DBpedia lists 1439 properties of which only 225 properties are listed in the PATTY dataset. Therefore, only 60% respectively 80% of the questions of the QALD 7 test and training dataset can be answered by our system. Thus, the lexical gap is an essential challenge for QA systems. Another problem constitutes the ranking of queries based on the separate rankings derived from property and subject/object mapping. For the relevance ranking of different properties provided by PATTY we use a co-occurrence measure of phrase and property within the dataset. Unfortunately, this results in some cases in wrong final rankings. For instance, Table 4 shows the rankings of generated queries for the questions “How often did Jane Fonda marry?” and “How often did Alan Rickman marry?”. According to the property relevance ranking the property `dbo:child` achieves a higher score than the property `dbo:spouse`. As all other ranking scores are identical, for the first question the wrong query is chosen as potential correct which delivers the wrong answer. This problem only occurs for entities who actually have children – as it is the case for Jane Fonda. Otherwise – as it is the case for Alan Rickman – the query with the property `dbo:child` does not provide a result (which results in a score of 0.0 for the answer type) and the query containing `dbo:spouse` is ranked highest. These problems result from the structure and quality of the underlying knowledge base respectively the transformed/deployed lookup stores. We discuss this topic further in Section 5. In addition, the evaluation results are influenced by the quality of the datasets. Unfortunately, the datasets contains questions where intuitively the provided answers are wrong. For instance, for the question “Which cities does the Weser flow through?” the entity `dbr:Fulda_(river)` is listed as expected correct result. But this entity is a river and not a city and it results from the missing type check in the SPARQL query provided in the dataset. By adding the type information in the SPARQL query the retrieved results are reduced which results in a lower recall for the evaluation.

Tab. 4: Comparison of Ranking of two queries

Query	Mapping		Answer Type	Final
	Subject	Property		
SELECT (COUNT(DISTINCT ?y) as ?x) WHERE { dbr:Jane_Fonda dbo:child ?y . }	1.0	0.23	1.0	0.614
SELECT (COUNT(DISTINCT ?y) as ?x) WHERE { dbr:Jane_Fonda dbo:spouse ?y . }	1.0	0.08	1.0	0.514
SELECT (COUNT(DISTINCT ?y) as ?x) WHERE { dbr:Alan_Rickman dbo:spouse ?y . }	1.0	0.23	1.0	0.614
SELECT (COUNT(DISTINCT ?y) as ?x) WHERE { dbr:Jane_Rickman dbo:child ?y . }	1.0	0.08	0.0	0.0

5 Challenges

In this section we discuss our experiences during the development of our system especially regarding the challenges described in Section 2.

Lexical Gap. As shown in the evaluation, the lexical gap is an essential factor when natural language questions cannot be answered. For DBpedia, there are multiple sources to extract synonymous labels for the entities, as described in Section 3.1. But for the terminological knowledge – classes/categories and properties – primarily only the original labels are provided. And this is the case for most knowledge bases. Therefore, the quality of the results using other knowledge bases than DBpedia depends on how the lookup stores can be complemented with additional synonymous labels. For this purpose we are following different strategies to be able to include various knowledge bases:

1. determine synonymous information from external vocabularies, such as WordNet¹⁴
2. extract mappings of natural language phrases and knowledge base properties from embedded RDFa¹⁵ information in text/web documents
3. extract additional information from mapped external ontologies

For the first case, WordNet provides synonymous phrases for each entry within the vocabulary. This information can be mapped to the original labels of the labels of the knowledge base to be complemented. The problem here is that the semantic character of the knowledge base is ignored and synonymous information might be applied to wrong resources. Therefore, this approach requires to be evaluated regarding ambiguity of the knowledge base and quality of the achieved mappings.

For the second strategy, web documents containing RDFa information are analyzed for the deployed terminological and assertional knowledge and thereby the natural language

¹⁴ <https://wordnet.princeton.edu/>

¹⁵ A markup language extension to HTML5 to enrich web documents with RDF to assign mentioned places, person, etc.: <https://www.w3.org/TR/xhtml1-rdfa-primer/>

information can be mapped to the used vocabulary. Web documents with RDFa annotations are emerging – especially in the field of educational content – and editors have been developed to provide user-friendly interfaces¹⁶. But the problem is that only a few popular vocabularies and knowledge bases are applied and it might be difficult to find documents containing very specific information as required for QA systems.

The third strategy is the mapping of separate parts of an ontology to other ontologies and get thereby additional information about the mapped parts. Different ontologies might use different labels for equivalent classes or provide additional mappings to further ontologies. Thus, synonymous information might be extracted and complemented in the original ontology.

Ambiguity. Our approach uses scores to rank different generated queries resulting from multiple entities achieved by the mapping process. As shown in Section 4 this separate mapping step is mostly successful (96,2% and 96,8% respectively for subject and object mapping). Therefore, our approach has proven to be successful regarding ambiguity of mapped entities. As discussed in the previous section, the problem of ambiguity is even less difficult for other knowledge bases than DBpedia because of mostly missing synonymous information.

Complex Queries. A natural language question that seems to be simple to be answered might result in a very complex formal query within the QA system. For instance, the question “What is the second highest mountain?” requires the actual triple to find heights of mountains, an operator to sort the heights in descending order, the limit of only one result and an offset to start with the second result in the descending order. All this information needs to be derived from the given natural language phrase. Our approach deduces this information from the question type and specific keywords from the lexically parsed question. The phrases “second” or “third” can be mapped to pre-defined patterns of operators (as described in Section 3.2.2). However, the pre-defined lists of keywords might be dependent on the underlying knowledge base where specific terms and facts might require specific aggregation functions. Therefore, we will explore specific characteristics of domains and topics with the application of our approach to further knowledge bases.

Templates. For the QALD 7 training dataset containing 216 questions we identified six different question types and 36 lexical patterns. Obviously, many lexical patterns are not specific to a question type and occur in various questions. We therefore do not expect the number of required patterns when applying different knowledge bases to increase excessively. It might be necessary to observe knowledge bases more in detail to derive specific patterns. For instance, property-object combinations that occur very often in a knowledge base might

¹⁶ <http://aksw.org/Projects/RDFaCE.html>

be a hint that this is a common fact and might be handled similar to class memberships – without the actual RDF triple of `rdf:type` and the URI of a specific class. For instance, the triple `res:some_City onto:isMetropolis "true"` would be equivalent to a class membership information such as `res:some_City rdf:type onto:Metropolis`. This class membership information must be derived to be able to answer questions like “Which metropolises are located in Europe?”. In addition, more general query patterns might be helpful to find the correct fact within the knowledge base. This applies for facts that might be distributed over more than one triple. For instance, within a knowledge base containing information about software projects the question “Who committed the most lines of code?” the information about lines of code might be assigned to a “commit event” including the original committing developer and the facts need to be aggregated over several commit events. This again requires knowledge about the deployed knowledge base which should be extracted during the preprocessing step.

Knowledge Base. Although our approach is set to be as independent from the knowledge base as possible, we are aware of the fact that available information and structure of the data contained in the knowledge base is an essential factor for the quality of a QA system. As described in the previous section, some information might be very domain specific and characteristic for knowledge base and is required to be detected and extracted or complemented before our system could answer specific questions. Another problem might be missing or wrong information in the knowledge base which is often the case for automatically extracted data, as applies for DBpedia. For instance, the information about the elevation of Mount Everest is missing in DBpedia although it is included in the respective Wikipedia article (this fact might be a hint that Siri is using DBpedia for its QA system – cf. Sect. 6). Furthermore, the type information is often missing for entities which leads to missing results when the type is checked in the SPARQL query. Otherwise, too many results might be provided when the type of the questioned subject is omitted. On the other hand, sometimes the type information is set incorrectly for certain entities. For instance, the actor “Terence Hill” is typed as `dbo:Mountain` (strangely along with other persons with the surname “Hill”). This again leads to wrong results when the SPARQL contains the type check. Here again, the solution might be to omit the type check in the query which demands an appropriate query generation algorithm to be able to rule out irrelevant results by additional facts in the query.

6 Related Work

Li et al. introduced an interactive natural language query interface that constructs SQL queries according to natural language questions [LJ14]. The interface therefore interacts with the user and requests feedback on chosen queries respectively concepts. The requesting user is involved in the query construction process at two different stages. First, the natural language phrase is parsed into a lexical tree. Afterwards nodes of the parse tree are identified

which can be mapped to components of the requested SQL schema. At this stage the first warning is given back to the user in case the mapping fails for one or more nodes. Also, the successful mappings will be presented to the user. After all nodes are interpreted correctly (with the help of the user) the linguistic parse tree is adapted to be valid for their system. Implicit nodes can be inserted, if necessary, and the user is able to support this process. The verified query tree is then transferred to a SQL statement containing joins, aggregate functions etc.

Deutch et al. presented a similar approach where a natural language query is transferred to a lexical query tree and the nodes are mapped to variables to be able to construct a conjunctive query (CQ) [DFG17]. The authors not only focus on the conversion process to be able to query a database for the requested natural language question, but also provide a natural language answer according to provenance tracking of tuples from the query tree and applying them to the answer tree.

Amongst others these two approaches present interfaces to convert natural language to SQL statements. In contrast to that our approach is built upon RDF knowledge bases respectively constructing SPARQL queries and does not use any user feedback for creating the formal queries. The following approaches are therefore all based on RDF knowledge bases and focus on translating natural language (NL) questions to SPARQL queries.

Freitas et al. developed a vocabulary independent approach for querying Linked Data so the user does not need to know about the vocabulary of the data sets. The main components of their system are (1) entity search, (2) spreading activation and (3) measuring semantic relatedness [Fr11]. The challenges their system is able to deal with are the lexical gap, ambiguity, complex operators and distributed knowledge. The essential question behind the key entity search is which components of the natural language query of the user can be mapped to classes or instances in the knowledge base. After detecting those key entities, the URI of the key entity is looked up and used as a pivot entity for the next step, which will be spreading activation. Before spreading activation can be started, the NL query has to be parsed in a form that can be mapped to the SPO form of data represented in RDF. After that, every pivot entity is used as a starting point for the spreading activation process. If there is more than one node that is higher than a given relatedness threshold, it will lead to a second path and in the end the path with the highest relatedness wins.

The system PARALEX introduced by Fader et al. in 2013 faces the challenges lexical gap, ambiguity as well as procedural, temporal or spatial questions and uses templates similar to our approach [FZE13]. However, PARALEX has a slightly different approach with focussing on a broad range of different questions asking for the same fact rather than on complex questions. It is only able to handle queries that can be represented as a triple, for example *portrayer(Emma Watson, Hermione Granger)*. PARALEX tries to map a given question to be answered by such a triple. Examples for those questions are: “Who portrayed Hermione Granger?”, “Who is the actress of Hermione Granger?”, “Who played the role of Hermione Granger?” or “Who is the actress that played the role of Hermione Granger in

Harry Potter?”. Although those questions may differ, they can all be answered by the factoid triple above. The system stores triple entries in a database that are of the form $r(e_1, e_2)$ and it can answer queries with one unknown variable that is either e_1 or e_2 . Therefore queries look like $r(?, e_2)$ for example for the question “Who portrayed Hermione Granger?” or $r(e_1, ?)$, which would answer the question “Which roles did Emma Watson play?” if we assume a triple like $\text{portrayed}(\text{Emma Watson}, \text{Hermione Granger})$ alongside others. To map the phrases of a question to an entry in the triple data store, a lexicon had been built by means of a learning algorithm which consists of two stages: first it is initially set up and then derivatives are added to increase the precision. The main disadvantage of the system is the missing ability to answer complex questions, because it is not able to build the query.

In 2014, Xu et al. introduced their system Xser [Xu14] and participated with their approach in the QALD 5 challenge (held at CLEF 2015 Initiative) achieving the highest recall and precision (recall=0.72, precision=0.74) among the competing systems. Their approach is based on two steps: first the NL question is analyzed for predicate argument structures using a semantic parser. In the second step, the actual queries based on the underlying knowledge base are generated. In contrast to our system, their approach requires to be trained using knowledge base specific training data.

Bast et al. presented an extensive survey in the field of semantic search on text and knowledge bases [BBH16]. Within this survey several combinations of the search input and the underlying data are explored. Obviously, a semantic search can be performed by analyzing a natural language as input. Therefore, the authors examine several approaches on understanding natural language and finding relevant search items for the input query.

Also in 2015, the system SemaGraphQA introduced by Beaumont et al. participated in the QALD 5 challenge [BGL15], achieving significantly worse results compared to the best performing system Xser as previously introduced (recall=0.32, precision=0.31). They participated again in the QALD 6 challenge achieving an increased precision but decreased recall (recall=0.25, precision=0.70). Their system uses a graph-based approach matching parts of the NL question to the knowledge base and building a syntactic graph.

The best performing system of the QALD 6 challenge (co-located with ESWC 2016) has been CANaLI introduced by Mazzeo et al. [MZ16]. CANaLI achieved an overall precision and recall of 0.89 for English questions of Task 4 of the challenge (“Multilingual question answering over DBpedia”). However, the system is working on a semi-automatic basis using user feedback to map to correct properties and entities in the knowledge base. We also use the datasets of Task 4 of QALD-6 to evaluate our system, but will not compare to CANaLI, because our system is working completely self-sufficient from the external help of the user.

When it comes to explaining the research field of question answering to people outside of the research community popular systems like Alexa, Siri & Co. perfectly fit to describe the challenges. These systems seem to know and understand everything and the research field might appear redundant. But, a closer look reveals the limits of the systems. The domain

of understanding a user is often very limited. For instance, you can ask for tomorrow's weather or where some place is or to call somebody from your contact list (using your smartphone). Additionally, some simple questions can be answered: The question "How high is Mount McKinley?" is correctly answered with "Denali is 20,308 ft above sea level" (even replacing the name with current correct one). But the similar question "How high is Mount Everest?" only answered by providing the article about Mount Everest on Wikipedia. The more complicated example (from the QALD challenge) "Who became president after JFK died?" only gives web resources regarding John F. Kennedy. These examples show that these systems are limited to popular and rather simple questions. The foundation of these systems are trained databases containing e.g. often requested web queries and the corresponding most frequent chosen web result.

7 Conclusion

In this paper, we presented our novel approach to question answering over Linked Data knowledge bases. In addition to the common challenges of QA systems, we designed our system to be as independent as possible from the underlying knowledge base. The preliminary transformation process extracts required information from the RDF(S) knowledge base and this information is used for a mapping step as part of the complete algorithm. Afterwards a SPARQL endpoint is required to execute the actual queries. A long-term goal is to provide a web interface where a knowledge base can be uploaded by a user, the transformation process is conducted and the SPARQL endpoint established. Subsequently, the user is able to ask natural language questions on the prepared knowledge base. Our approach has been evaluated on the latest datasets provided by the QALD challenge. First results show that our approach performs similarly or even better than the competing systems that participated in the last challenge. Our future work includes two main aspects: on the one hand we want to achieve a reasonable saturation of patterns for general purpose questions. This means, we will further explore common natural language questions for so far missing patterns without overfitting the assembled set of patterns and be as general as possible. On the other hand, we will transfer our approach to knowledge from other domains and further research the applicability of the detected patterns so far. Overall, we were able to show that a QA system is not necessarily required to be trained on a specific domain or knowledge base. Nevertheless, the discussed challenges provide sufficient input for further enhancements and developments in this complex research field.

8 Acknowledgements

This work was partially funded by the German Research Foundation (DFG) under grant no. SA782/26.

References

- [BBH16] Bast, Hannah; Björn, Buchhold; Haussmann, Elmar: Semantic Search on Text and Knowledge Bases. *Found. Trends Inf. Retr.*, 10(2-3):119–271, June 2016.
- [BGL15] Beaumont, R.; Grau, B.; Ligozat, A.-L.: SemGraphQA@QALD5: LIMS participation at QALD5@CLEF. In: *Working Notes of CLEF 2015 - Conference and Labs of the Evaluation forum*, Toulouse, France, September 8-11, 2015. 2015.
- [DFG17] Deutch, D.; Frost, N.; Gilad, A.: Provenance for Natural Language Queries. *Proc. VLDB Endow.*, 10(5):577–588, 2017.
- [Fr11] Freitas, A.; Oliveira, J.G.; O’Riain, S.; Curry, E.; Da Silva, J.C.P.: Querying Linked Data Using Semantic Relatedness: A Vocabulary Independent Approach. In: *Proc. of the 16th Int. Conf. on Natural Language Processing and Information Systems. NLDB’11*. Springer-Verlag, pp. 40–51, 2011.
- [FZE13] Fader, A.; Zettlemoyer, L.; Etzioni, O.: Paraphrase-driven learning for open question answering. In: *Long Papers, volume 1. Association for Computational Linguistics (ACL)*, pp. 1608–1618, 2013.
- [Hö17] Höffner, K.; Walter, S.; Marx, E.; Usbeck, R.; Lehmann, J.; Ngonga Ngomo, A.-C.: Survey on Challenges of Question Answering in the Semantic Web. *Semantic Web Journal*, 8(6), 2017.
- [LJ14] Li, F.; Jagadish, H. V.: Constructing an Interactive Natural Language Interface for Relational Databases. *Proc. VLDB Endow.*, 8(1):73–84, September 2014.
- [MMM06] Marneffe, M.; Maccartney, B.; Manning, C.: Generating Typed Dependency Parses from Phrase Structure Parses. In: *Proc. of the 5th Int. Conf. on Language Resources and Evaluation (LREC-2006)*. European Language Resources Association (ELRA), Genoa, Italy, May 2006.
- [MZ16] Mazzeo, G. M.; Zaniolo, C.: Answering Controlled Natural Language Questions on RDF Knowledge Bases. In: *Proc. of the 19th Int. Conf. on Extending Database Technology, EDBT 2016*, Bordeaux, France. pp. 608–611, 2016.
- [NWS12] Nakashole, N.; Weikum, G.; Suchanek, F.: PATTY: A Taxonomy of Relational Patterns with Semantic Types. In: *Proc of the 2012 Joint Conf. on Empirical Methods in Natural Language Processing and Computational Natural Language Learning. EMNLP-CoNLL ’12*, pp. 1135–1145, 2012.
- [St14] Steinmetz, N.: Context-aware semantic analysis of video metadata. Phd. thesis, Universität Potsdam, 2014.
- [UNC16] Unger, C.; Ngonga Ngomo, A.-C.; Cabrio, E.: 6th Open Challenge on Question Answering over Linked Data (QALD-6). In: *Semantic Web Challenges: Third SemWebEval Challenge at ESWC 2016*, Heraklion, Crete, Greece, May 29 - June 2, 2016, Revised Selected Papers. pp. 171–177, 2016.
- [Xu14] Xu, K.; Zhang, S.; Feng, Y.; Zhao, D.: Answering Natural Language Questions via Phrasal Semantic Parsing. In: *Proc. Natural Language Processing and Chinese Computing: Third CCF Conference (NLPCC)*, Shenzhen, China. Springer, pp. 333–344, 2014.

Industriebeiträge

High-Performance Queries

Domain Query Optimization: Adapting the General-Purpose Database System Hyper for Tableau Workloads

Adrian Vogelsgesang¹, Tobias Muehlbauer², Viktor Leis³, Thomas Neumann⁴, Alfons Kemper⁵

Abstract: The Hyper database system was started as an academic project at Technical University Munich. In 2016, the commercial spin-off of the academic Hyper database system was acquired by Tableau, a leader in the analytics and business intelligence (BI) platforms market. As a human-in-the-loop BI platform, Tableau products machine-generate query workloads with characteristics that differ from human-written queries and queries represented in industry-standard database system benchmarks. In this work, we contribute optimizations we developed for one important class of queries typically generated by Tableau products: retrieving (aggregates of) the domain of a column. We devise methods for leveraging the compression of the database column in order to efficiently retrieve the duplicate-free value set, i.e., the domain. Our extensive performance evaluation of a synthetic benchmark and over 60 thousand real-world workbooks from Tableau Public shows that our optimization enables query latencies for domain queries that allow self-service ad-hoc data exploration.

1 Introduction

The Hyper database system was started as an academic project at Technical University Munich. Hyper is developed as a general-purpose database system for transactional as well as analytical workloads, operating simultaneously on one state, embracing the SQL standard and ACID transactional guarantees, while delivering highest performance in both workload classes. In 2016, the commercial spin-off of the academic Hyper database system was acquired by Tableau, a leader in the analytics and business intelligence (BI) platforms market. The machine-generated query workload by Tableau's products greatly differs from the queries in publicly available industry-standard benchmarks that Hyper was initially optimized for [Vo18]. Not only let modern BI tools easily express very complex queries via an intuitive drag-and-drop interface, users of such tools also expect to be able to quickly analyze data of all sizes and of all types. Standardized benchmarks like TPC-H, TPC-C, TPC-DS, the CH-Benchmark, or the SQLite test suite are relevant, but do not represent the

¹ Tableau Software, avogelsgesang@tableau.com

² Tableau Software, tmuehlbauer@tableau.com

³ Tableau Software, vleis@tableau.com

⁴ Tableau Software, tneumann@tableau.com

⁵ Tableau Software, akemper@tableau.com

observed long tail in both, query and data set complexity. In addition, there is a growing demand to analyze the freshest state of the data as it becomes available and in interactive ways, where each result is the starting point of further questions. As such, it is infeasible to pre-compute results. To meet the low-latency query response time requirements of modern interactive BI, Hyper needed to be adapted to specific classes of queries commonly generated by BI tools like Tableau.

A common class of queries generated by modern BI tools are *domain queries*. In this context, domain refers to the set of distinct values that a column in a relation may contain. The domain of a column or an aggregate of the domain is retrieved in various contexts, e.g., to define filters or to properly size the axes in a multi-dimensional chart. What makes domain queries challenging is the fact that most real-world data sets are not properly normalized or are the result of a query that joined together different but related pieces of information into a single relation. Denormalized schemata inherently lead to duplicate values in columns and cannot efficiently be re-normalized on the fly.

In Hyper's storage layer, relations are horizontally partitioned into fixed-size chunks. Each chunk is again vertically partitioned into columns. Chunks can either be hot, i.e., containing tuples frequently modified by transactions, or cold, i.e., mostly scanned or accessed by point queries. Cold chunks are compressed using light-weight compression techniques and are called Data Blocks [La16]. To guarantee low-latency access to the freshest state of the data, de-duplication of columnar values is only feasible on a per-chunk level, since global de-duplication is inherently non-linear in the size of the data set. In this paper, we introduce optimizations for domain queries in the Hyper system that leverage de-duplication in Hyper's Data Blocks storage format. In particular, we contribute:

- A description of how domain queries can be detected.
- A way to efficiently execute domain queries leveraging the Data Blocks storage format.
- An extensive evaluation of domain query optimization in Hyper on synthetic and real-world data sets, including an analysis of 60,000 workbooks from Tableau Public.

2 Background

2.1 Challenges in the Context of Tableau

Tableau generates SQL queries for the Hyper database systems triggered by user input from and scheduled background tasks.

In this paper, we show how we addressed the two key aspects of the observed SQL workload:

- **Denormalized storage:** Many underlying data models in Tableau uses are flat, denormalized tables. This is due to several factors, including for performance reasons, similar to Blink [Ra08].
- **Machine-generated queries:** In contrast to most standard benchmarks for database systems, the queries issued by Tableau are machine-generated. For a database system, this has both up- and downsides: On the positive side, all queries are following a similar structure. The database can be optimized to detect some of the common patterns in the queries and provide optimized implementations for those queries. On the other hand, the machine-generated queries tend to be more complex than hand-written queries.

2.2 The Data Blocks Storage Format

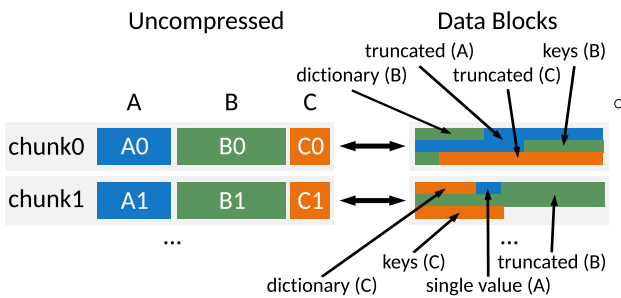


Fig. 1: The DataBlocks storage format

In this paper, we present how to apply domain query optimization to Hyper’s primary storage format, Data Blocks. Nevertheless, this optimization is applicable to other data storage formats. The Data Blocks format was originally introduced by Lang et al. [La16]. Figure 1 schematically depicts how a relation with three columns (A , B , C) is represented in the Data Blocks format.

The table is split into multiple chunks of a fixed maximum size, e.g., 2^{17} tuples. Each Data Block is self-contained and stores one or more attribute chunks in a byte-addressable compressed format. The attributes within a Data Block are stored in a columnar fashion. Columns are compressed using lightweight compression methods such as dictionary encoding or single-value encoding. Optimal compression methods are chosen per column within a block based mostly on the expected compression ratio. Note that the same column might be compressed through different compression methods within different blocks. In the figure, e.g., the first chunk of column A is stored using the compression *truncated* while the second chunk is stored as a single value. In addition, each block contains a small materialized aggregate (SMA) [Mo98] storing the minimum and maximum value contained in the block for each column.

The goal of Data Blocks is to conserve memory while allowing a high OLTP and OLAP performance. By maintaining a flat structure without pointers, Data Blocks are also suitable for eviction to secondary storage [FKN12]. A Data Block contains all tuple data, but no metadata, such as schema information.

Compression is employed to optimize processing speed by better exploiting the available memory bandwidth. Smaller storage size is an additional benefit of compression but not the primary goal of Data Blocks. Hence, Data Blocks are compressed using light-weight compression methods which still allow efficient scans as well as point accesses for OLTP workloads. In particular, Data Blocks currently use the following compression formats:

For dictionary-compressed columns, a small dictionary containing the domain of the Data Block is built. The tuple values are represented as tightly-packed offsets into that dictionary. The width of the offset values varies between 1 and 8 bytes depending on the size of the dictionary.

If not only the domain size, but also the domain range is small, it is efficient to store the tuple values as relative offset to a base value. The values are truncated by subtracting the minimum value within the chunk from each tuple value. Thereby, the storage space for the dictionary can be saved. In other literature, this compression method is also referred to as *frame-of-reference* compression.

In case a column contains the same value for all rows in a chunk, it is *single-value* compressed, i.e., the value is only stored exactly once.

3 Domain Query Optimization

Domain queries are a special type of query issued by Tableau in order to obtain the domain of a column, i.e., the set of different values that occur in the column. The obtained domain is used, e.g., for populating filter lists and drop down menus. In this section, we describe how Hyper leverages duplicate-free structures in the storage layer (e.g., dictionaries, single value compression) to speed up domain queries. Furthermore, we show how we extend the applicability of the domain-query optimization (DQO) to additional queries which are not issued as domain queries, but happen to be applicable for this optimization, too. Finally, we evaluate the impact of DQO both on a synthetic test set and on real-world data.

3.1 Use Case

Tableau allows ad-hoc interactive data exploration. One of the most basic interactions is filtering, which allows users to drill down into a subset of their data. By using filters, users can restrict the set of data being visualized based on arbitrary predicates. For this purpose, Tableau provides the user controls with which they can zoom in on or exclude parts of the

data set. Figure 2 shows some of the available filters. Using the *Order Date* slider, the data can be filtered to a certain date range. The *Segment* and *Region* filters allow to include and exclude specific values from the domain. As can be seen, for all 3 filters Tableau offers all values available in the data set. In order to display this list of available data to the user, Tableau needs to know the domain of the filtered values. For the *Order date* slider, it must know the range, i.e., the minimum and maximum date present in the *OrderDate* column. For the *Segment* selection list and the *Region* drop-down menu, it must know all distinct values from the underlying column, i.e., its domain.

```
SELECT column_x FROM user_data GROUP BY column_x;
```

Listing 1: SQL query issued to populate a drop-down list

```
SELECT MIN(column_x), MAX(column_x) FROM user_data;
```

Listing 2: SQL query issued to populate a drop-down list

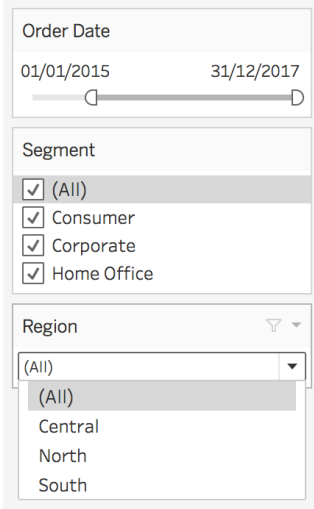


Fig. 2: Filters in Tableau

For both types of filters, Tableau issues queries to the underlying database system. Depending on the type of filter, Tableau issues a query in the form of either Listing 1 (for drop-down menus and selection lists) or 2 (for sliders).

Since the creation of filters in order to drill down into a visualization is part of the daily workflow of data analysts, answering the corresponding SQL queries quickly is important in order to provide a real-time experience. In the research context, Hyper's optimizations were geared towards standard benchmarks such as TPC-H, TPC-DS and TPC-C. Since those benchmarks do mostly not include domain queries, Hyper was initially not optimized for such queries.

In an ideal world, domain queries would not be challenging to process. On a normalized schema, a domain query can usually be answered by scanning a column that consists only of distinct values. Many real-world data models however are denormalized into a single

flat table and cannot easily be re-normalized. As such, it is not uncommon to have many duplicates in a column that need to be filtered to return a duplicate-free domain.

3.2 Overview Of Approach

Both query types issued by Tableau to populate its filter controls can be answered without reading all row values, just by looking at the column's domain. For the query in Listing 1 this is obvious since the query explicitly asks for the column's domain. The query in Listing 2 can also be answered using only the column's domain since the result of a MIN/MAX aggregation is not influenced by duplicated values.

In some cases, domain queries can be removed as part of the logical optimization. If the query from Listing 1 is issued against a column which is known to be unique, the *GroupBy* operator will be eliminated by Hyper. For denormalized data models, however, initially unique values from the domain tables are duplicated by the joins that computed the denormalized, flat table representation. Hence, this logical optimization is in many cases not applicable. With no possibility to remove the *GroupBy* during logical query optimization, Hyper leverages optimizations during the execution phase to speed up domain queries.

Without DQO, Hyper uses a normal table scan over the queried column. This table scan produces one column value for each row. Doing so it, e.g., expands single value compressed columns by duplicating the column's static value for each row. Each of those tuple values is then passed to a *GroupBy* operator which removes duplicate values again.

Obviously, the execution time of this query can be improved by skipping the generation of the duplicates in the table scan. In order to speed up this query, we give a hint to the table scan operator and make it aware of the fact that duplicates will be discarded by its parent operator. The table scan then has the freedom to skip producing duplicated tuples altogether. It can take advantage of the underlying physical data representation, e.g., single value compression, dictionaries or run length encoding, to reduce the number of produced values.

Note that the *GroupBy* operator remains in place even if the underlying table scan is instructed that it should not produce duplicate values. This means that the table scan has the freedom to skip duplicated tuples but is not forced to do so. E.g., if no dictionary is available the table scan is still allowed to produce duplicated tuples. It is even encouraged to do so instead of de-duplicating values on its own. After all, the *GroupBy* operator constitutes the most efficient de-duplication logic in Hyper. Table scan operators can only be more efficient than the *GroupBy* operator at de-duplicating values when they are able to exploit the underlying physical storage which would not be accessible to the *GroupBy* operator.

3.3 Detecting Applicability of DQO

In order for domain query optimization to be applicable, two conditions must be fulfilled:

- The consumer of the scanned tuples must be duplicate-agnostic, i.e., its output must not depend on the number of times tuples with identical values are emitted.
- The table must be stored in a format which allows one to easily suppress the generation of duplicates.

We decided to address the first requirement during logical optimization and set a flag on each table scan operator which is eligible for DQO. Only during query runtime the table scan then inspects the actual storage structures and suppresses duplicates as applicable. This split allows us to keep the strong separation between the logical query optimizer and the storage layer in tact. Hyper's query optimizer is agnostic of the storage layer and storage level knowledge is not being exposed to the logical optimizer. Furthermore, in many cases an ahead-of-time decision on the storage format is not even possible. During the compilation and optimization of prepared queries, for example, the data contained in the scanned columns and the used compression formats during the execution of the query might not yet be available.

We detect if a table-scan is in a duplicate-agnostic context by inspecting the algebra representation of the query plan. We do so after applying all other algebraic optimizations, most notably after constant folding and join reordering. Detecting domain queries after constant folding allows us to apply DQO in more cases since the algebra tree was already simplified and, e.g., tautological selection predicates are already removed.

Detecting domain queries only after join reordering comes with one potential drawback: The cardinality estimates used for join reordering are unaware of the potential cardinality reduction achieved through DQO. Hence, the join optimizer might pick a sub-optimal join order. We made this design decision in the sake of robustness: Even if we detect the applicability of DQO on the logical level, it is not yet clear if this can be exploited on the physical level. Hence, we prefer to use the non-DQO-aware cardinality estimates for join reordering since those estimates represent the worst-case cardinalities which we will experience if DQO cannot be applied, e.g., due to the absence of adequate physical storage structures.

The easiest set of conditions to identify either of the two queries shown in Section 3.1 would be:

1. The top level operator must be a GroupBy operator.
2. Its child operator must be a table scan.
3. The GroupBy operator must either a) be a grouping on only one column from the base table and contain no aggregates, or b) contain only MIN/MAX aggregates applied to one column from the base table.

While those conditions match both domain queries shown so far, they limit the applicability of DQO unnecessarily. Hence, we went for a more general approach to detect domain

queries: The optimizer tracks for each algebraic operator in which context it will be executed. We distinguish between two different types of contexts:

An operator in a **duplicate-sensitive context** needs to produce all instances of duplicated tuples. In contrast, in a **duplicate-insensitive context** the operator can decide to skip instances of duplicated tuples.

Duplicate-sensitivity is propagated top-down within the query tree. Most operators such as σ , \bowtie , Π just pass the duplicate-sensitivity down: if the operator itself occurs in a duplicate-insensitive context, all its inputs are also duplicate-insensitive and vice versa. Some operators, such as the set variants of UNION, INTERSECT and EXCEPT always evaluate their inputs in a duplicate-insensitive context. Another example for this case are semi-joins and anti-joins which pass down the duplicate sensitivity for one side while unconditionally evaluating the other input in a duplicate-insensitive context.

For domain queries as issued by Tableau, the most important operator is the GroupBy operator Γ . The GroupBy operator determines the input duplicate sensitivity independently of its output duplicate sensitivity. It only takes into account the computed aggregates. If all aggregates are duplicate-insensitive, the input operator is in an duplicate-insensitive context. For that purpose, we annotated all duplicate-insensitive aggregation functions, among others: MIN, MAX, BOOL_OR and all DISTINCT variants of aggregate functions such as COUNT(DISTINCT ...).

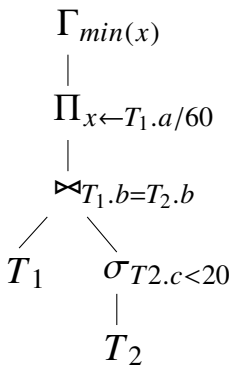


Fig. 3: Algebra tree of a more complex domain query

Figure 3 contains an exemplary algebra tree. The top-level Γ is evaluated in a duplicate-sensitive context as the result of a SQL query is by definition a multiset and hence duplicate-sensitive. Since the *GroupBy* only computes a MIN aggregate, its result does not depend on the absence or presence of duplicated input tuples. Hence, the Π operator is evaluated in a duplicate-insensitive context. The duplicate insensitivity is passed down to the join which can be rewritten into a semi-join given that no columns from its right side are accessed by the upper parts of the plan. Independent of replacing the join by a semi-join, the duplicate insensitivity propagates down through the whole tree. Hence, both table scans can skip duplicated tuples.

However, the scan on T_1 cannot benefit from DQO. Two columns from T_1 , a and b , are accessed. Data Blocks always compress columns individually and there is no data structure which allows to de-duplicate (a, b) tuples. Other storage formats might contain data structures such as dictionaries spanning multiple columns and hence could benefit from DQO. We are not aware of any such storage format which is widely adopted.

For T_2 , the query also accesses multiple columns (b and c). In this case, the selection is a SARGable predicate and the query still benefits from DQO thanks to the stored SMA synopses (see Section 3.4).

If a table scan occurs in a duplicate-insensitive context, a flag on that table scan is set. This flag will be picked up during execution of the compiled query. Except for this, the operator tree stays unchanged.

```

SELECT TITLE_CASE(c2s.segment_name)
FROM orders
JOIN customer_to_segment c2s
ON orders.customer_id = c2s.id
WHERE customers.order_date > '2015-01-01'::date
GROUP BY TITLE_CASE(c2s.segment_name)

```

Listing 3: Complex domain query generated for the Segment filter from Figure 2

Note that Tableau actually issues such complicated queries. Listing 3 shows a SQL query which translates to a query tree of the shape shown in Figure 3. In this exemplary scenario, the `TITLE_CASE` function is issued since the Tableau user created a custom computation to cleanup the inconsistent casing of the segment names. The Join was added by the customer using Tableau's data modelling capabilities to map a customer to the corresponding segment. The `WHERE` clause is generated since the Segments list is implicitly dependent on the selected range of order dates. The Segment list only shows segments for which there exist orders in the selected date range. Thereby, it is ensured that users cannot accidentally filter down the input data so that no more data points qualify.

3.4 Efficient execution of domain queries

After detecting a domain query, it is up to the table scan to exploit the optimization potential of the physical storage format. If a DQO was detected, i.e., if the consumers of the table scan ignore duplicates, the table scan operator is free to suppress duplicate values. However, the table scan is by no means forced to do so. It can just as well produce partially de-duplicated tuples or just the original tuples, since its parent operator stays unmodified and still eliminates duplicates which might get produced.

Figure 4 shows the unoptimized execution of a simple domain query against the Data Block format. The `GroupBy` operator receives its input tuples from each individual block. The individual tuple streams are implicitly unioned before passing them to the aggregation operator. In the example the first block is compressed using a dictionary. When scanning this Data Block, Hyper looks up every token from the data stream in the dictionary and passes the resulting values to the `GroupBy` operator. The second Data Block contains only one distinct value and this value is stored using single-value encoding. When the table scan scans this block, it duplicates this single value once for every row stored in the Data Block,

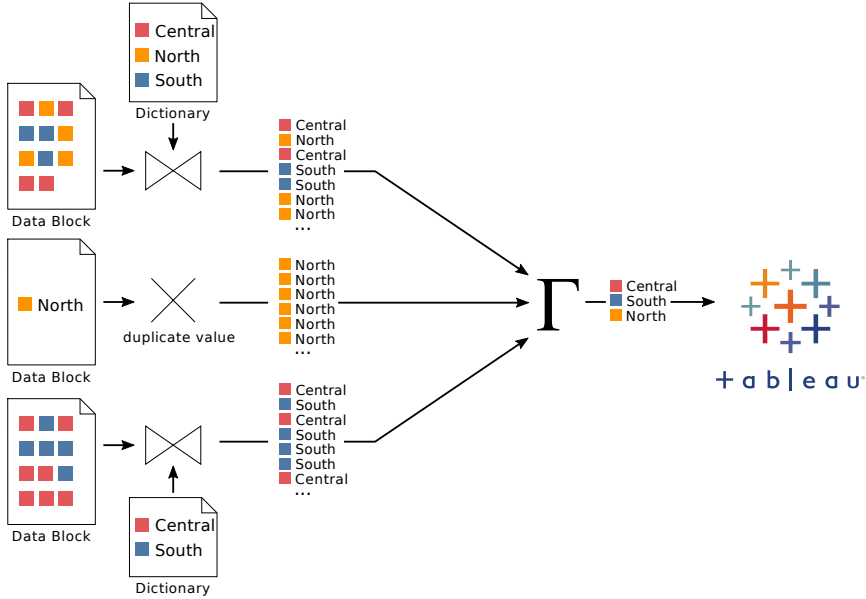


Fig. 4: Unoptimized execution of a domain query against Data Blocks with different compression methods

i.e., 2^{17} times. The third block of the column is stored using a dictionary again. This time however, the block does not contain any tuple with the region being *North*. Accordingly, the corresponding dictionary does not contain an entry for it. All in all the Γ operator receives 3×2^{17} tuples. It eliminates all duplicates and produces the domain consisting of only 3 tuples. Those 3 tuples are then sent to Tableau.

Comparing this to the optimized execution shown in Figure 5, the benefit from domain query optimization becomes obvious. Thanks to DQO, the Γ operator receives only 5 tuples instead of approximately 40 thousand tuples. Since every Data Block is a self-contained unit, there is no way to leverage the Data Block format to de-duplicate values across block boundaries. Hence, there still exist duplicated values which needs to be de-duplicated by the Γ operator.

Within each block, as many duplicates as efficiently possible are suppressed. For each block, the applied strategy depends on its compression method. For dictionary-compressed blocks (the first and the third block in Figure 5 the dictionary is scanned instead of the individual tuple values. For single-value-compressed blocks (the second block in our example), the duplication step is simply skipped and instead only the one single value is generated.

In addition, to those two compression methods, we also implemented DQO for frame-of-reference-compressed data if the offset from the reference value is stored using 1 byte. For

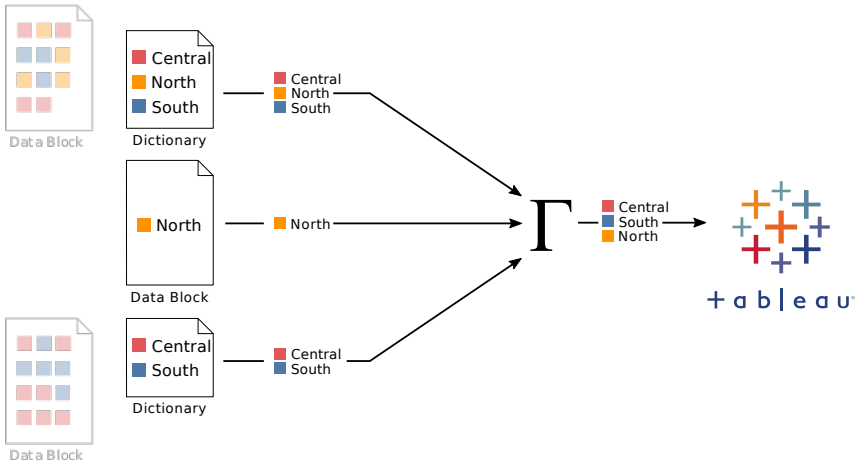


Fig. 5: Optimized execution of the query in Figure 4

this compression type, the values are de-duplicated on the fly: A small bit set is used to mark values which were already seen. Initially, all bits are set to false. Before passing a value to the parent operator, the corresponding bit in the bit set is checked. If it is set, the same value was already seen before and the duplicated value is skipped. Otherwise, the value is seen for the first time and gets produced. In this case, the corresponding bit in the bit set is set so that additional duplicates of the same value are suppressed.

We mentioned before that table scan operators should not do de-duplication since the GroupBy operator already provides the most efficient general purpose de-duplication algorithm. However, for 1-byte-offset-compressed blocks, this general guideline does not apply. Since the offset is stored in 1 byte, there are at most 256 distinct values among the 2^{17} rows of the Data Block. Therefore, it is known upfront that the de-duplication will remove a significant percentage of tuples. The upstream Γ operator implements a general purpose hash based algorithm. In this case, however, we know that all values are within a dense range of 256 integer values. We can exploit this knowledge and outperform Γ by using a small bit set instead of a more heavy-weight hash table.

For the other flavors of frame-of-reference compression, i.e., for 2-byte and 4-byte offsets, it is not clear upfront that de-duplication using a byte array will be helpful. The corresponding bit sets would require 8KB and 512MB of RAM, respectively. Since it is not clear if the additional effort to de-duplicate the values on the table scan level would pay off, we do not de-duplicate values at the table scan level for frame-of-reference encoding with 2 and 4 byte offsets.

Furthermore, the Data Block format allows to evaluate SARGable additional restrictions on

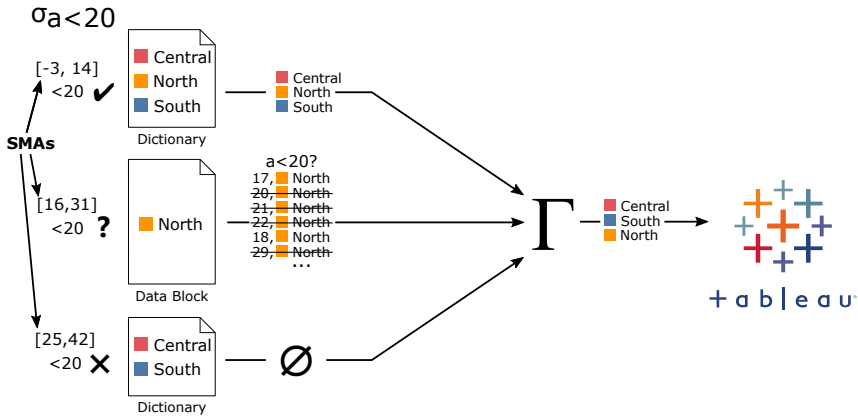


Fig. 6: Optimized execution of a domain query with an additional predicate

columns other than the column whose domain is being queried. By doing so, we allow the query from Listing 3 to also profit from DQO.

Figure 6 shows the evaluation of a domain query. The restriction is first evaluated against the SMA of the restricted column. In case, this comparison yields that the condition always evaluates to true for all values in the block's domain, the block's domain is scanned instead of the individual tuples. If the SMA indicates that the condition evaluates to false for all values in the block, the whole block is skipped. Only if the evaluation of the restrictions on the SMA is inconclusive, the block must be scanned in a normal fashion, i.e., tuple-by-tuple. The fallback to a normal scan is done on a block-by-block basis. Thereby, even if individual blocks cannot take advantage of DQO, other blocks can still profit from it.

In the special case of MIN/MAX aggregates, we can even go a step further and directly use the minimum and maximum values from the SMA.

3.5 DQO-aware selection of compression method

Since not all available compression schemes support domain queries well, we also adjusted the heuristic for picking the compression. Earlier, the compression method was chosen purely based on the size of the resulting compressed data. In general, this heuristic also aligns with scan performance: Incidentally, the compressions providing better compression ratios, also provided the better scan performance. With DQO, this statement no longer holds true, though. E.g., Frame of reference compression with 2 byte offsets is smaller than dictionary compression with 2 byte token indices since dictionary compression needs additional space to store the dictionary. On the other hand, domain queries on dictionary-compressed data

are more efficient than the same query on frame-of-reference-compressed data. While one can simply scan the dictionary to answer a domain query on dictionary-compressed data, the frame-of-referenced compressed data would require a full scan of all tuple values within the block.

To mitigate this effect, we adapted the heuristic to take the impact on domain queries into account. We still select compressions primarily based on the achieved compression ratio. However, compression methods which support domain queries get a 16KB head start. In other words, dictionary compression with 2 byte tokens is preferred over Frame of reference compression with 2 byte offsets as long as the dictionary is not bigger than 16KB. In theory this could regress performance, as we are using more instructions to unpack each tuple. However, even with the additional indirection through the dictionary, we are still limited by memory bandwidth and not instruction count.

4 Evaluation

To illustrate the impact of domain query optimization, we provide performance numbers for hand-crafted domain queries against artificial input data. We controlled the duplication ratio within this data set. In this set-up, we measured the end-to-end performance within Hyper including all phases of query processing (parsing, semantic analysis, query optimization, query compilation and query execution). Thereby, this evaluation provides us a notion of the overall impact of DQO on query time for individual SQL queries issued against Hyper. We did not include the time spent by Tableau to generate the SQL queries.

Furthermore, we evaluated DQO on our test set introduced in [Vo18]. This test set contains over 60 thousand real-world Tableau visualizations demonstrating the real-world impact of DQO. We first present statistics on the data set which show the potential impact of DQO. Next, we measured the impact of DQO on the query times for loading a complete visualization without taking user interaction into account.

In all experiments, we also monitored changes in file size. As described in Section 3.5, the heuristic for choosing the compression was adapted to prefer compression schemes which lead to a slightly larger on-disk size if it thereby enabled domain scans. However, in practice we did rarely see an increases in file size caused by the changed compression heuristic. The additional space consumption was masked by already existing padding and alignment in the database file.

4.1 Performance Improvements on Synthetic Data

In order to gauge the impact of DQO in isolation, we measured the execution times for the 3 queries shown in Listing 4.

```
SELECT column_x FROM user_data GROUP BY column_x;  
SELECT column_x FROM user_data  
  GROUP BY column_x ORDER BY column_x;  
SELECT MIN(column_x) FROM user_data;
```

Listing 4: SQL queries used for measuring DQO performance benefits.

The first query (subsequently referenced as *distinct*) requests the domain of a column in its most straightforward way. It represents a minimal domain query. Removing any part from it would make it no longer eligible for DQO. This query allows the most targeted performance measurement focused only on DQO. The second query (referenced as *distinct sorted*) is similar but sorts the data before returning it to the client. Thereby, it requires some additional processing to the performance measurement and the gains from DQO are expected to be smaller in comparison to the overall execution time. Domain queries issued by Tableau are posed in this exact form. The third query (referenced as *min*) focuses on measuring the time spent in the table scan. In contrast to the other queries, it does not require de-duplicating the values and thereby does not require building a hash table. Instead, the minimum is already computed while iterating over the tuples. Since keeping the minimum updated is cheap, this query measures mostly the time spent in the table scan. For this reason, this query is expected to be the fastest one among the test queries.

The queries were executed against generated data sets with a fixed number of 5 duplicated values within an increasing number of rows. The columns contain strings with moderate string lengths of up to 30 characters. We vary the number of tuples between 2^{17} and 1024×2^{17} . Thereby, the smallest tested relation consists of 1 Data Block while the largest one consists of 1024 Data Blocks.

By varying the number of tuples while keeping the distinct count constant, this experiment measures the most important axis along which domain queries must scale for Tableau: Tableau users are usually interested in filtering on a small domain, e.g., the regions in which they sold their products. The set of different regions rarely increases and tend to be quite small. At the same time, the number of orders and hence the overall number of tuples in the table increases over time.

All queries were run 5 times and we report the average times of the last 4 runs. The first run was excluded since it exposed large variance due to disk caching effects. Compilation times and execution times were measured separately. *Compilation time* refers to the time needed for parsing the SQL query, turning the abstract syntax tree into an operator tree, logically optimizing the operator tree, generating & optimizing the corresponding LLVM and compiling the LLVM code to machine code. It does not include the time needed to receive the SQL text over the network. *Execution time* includes the time needed for the actual query execution and for serializing the results into network buffers. However, it does not include the time, until the client actually received the result, i.e., the network buffers

were not flushed. All reported times were measured on a Ubuntu Desktop machine with 64GB RAM a 16-core Intel Xeon E5-2630 CPU clocked at 2.40 GHz with all 32 hyper threads enabled.

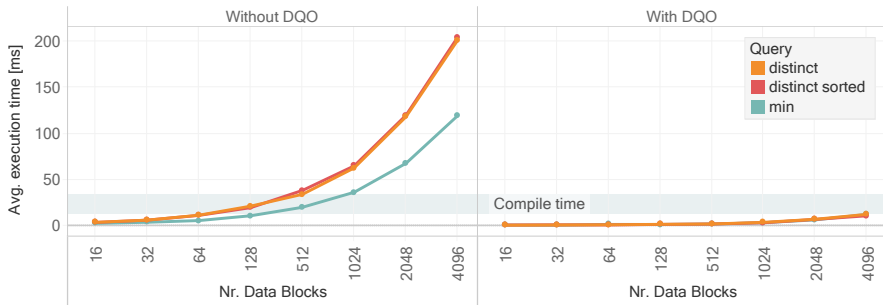


Fig. 7: Execution times for the domain queries with and without DQO on an increasing number of tuples with the number of distinct values being fixed to 5

Figure 7 shows the measured execution times. As expected, the *min* query represents a lower bound on the execution time for all other queries. Looking at the numbers without DQO, comparing execution times for the *distinct* and *distinct sorted*, they are mostly on-par. This means that the sort operation which is part of the *distinct sorted* has a negligible overhead and that the main contributor to query time for domain queries as issued by Tableau is the actual duplicate elimination.

With DQO enabled, execution times for all scenarios are by orders of magnitude faster. The improvement is larger as the data sets get larger. With DQO, compilation time dominates query time in all measured experiments. Query compilation accounts for approximately 0.11 seconds independent of scanned input data. Compared to the execution time which are still less than 1 millisecond for the largest measured this input data, query compilation is an order of magnitude slower.

Note that even with DQO, the query time is still linear in the table size and not in its domain size. Although DQO significantly reduces the number of tuples produced by the table scan, the number of tuples still depends linearly on the number of tuples in the base table. This is because Data Blocks build per-block dictionaries. Duplicates are only getting de-duplicated within each Data Block, and each Data Block passes on the deduplicated values from its dictionary. Hence, the number of scanned tuples and thereby also the query execution time is still linear in the number of Data Blocks. Since the number of Data Blocks is linear in the number of overall tuples, the number of tuples produced by the table scan is still linear in the number of tuples.

4.2 Impact of DQO on Tableau Public

To judge the potential impact of DQO, we first did a static analysis of all 60 thousand visualizations from Tableau Public. Those 60 thousand visualizations are backed by approximately 120 thousand database files.

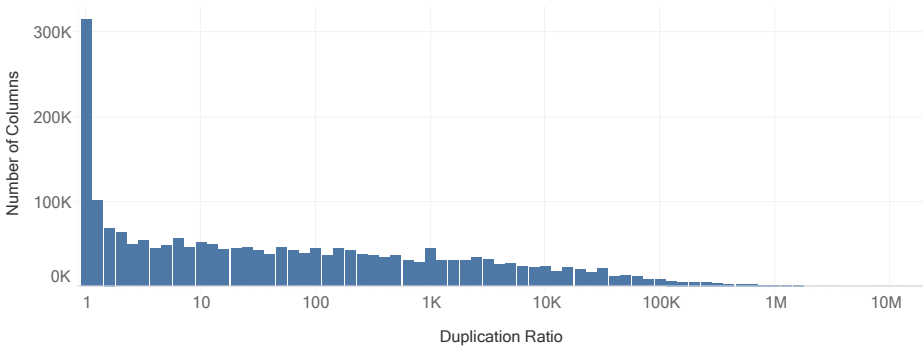


Fig. 8: Duplication ratio on Tableau Public visualized as the number of columns grouped by their duplication ratio

Figure 8 shows a histogram binning the two million columns contained in the database files by their duplication factor. The duplication factor is calculated as the number of tuples divided by the number of distinct values in the column. It thereby gives an upper bound on the number of tuples eliminated by DQO. It is depicted on the depicted on the abscissa using a logarithmic scale. The ordinate axis shows the number of columns with a given duplication factor. Highly duplicated columns are not shown in the figure since they would have distorted the overall scale of the chart. The long tail reaches a duplication factor of 116 million, i.e each unique value appears 116 million times.

The first bar of the histogram represents all columns with less than 5% duplicate values. 250k columns fall in this category. Among those columns 201.5k columns are completely duplicate free. For the duplicate-free columns, i.e., for 9.5% of all columns, DQO cannot provide any benefit. Also for the remaining 48.5k columns from this category the gain from DQO is only marginal, since the duplication factor is less than 5%. Hence, 12% of the columns are not profiting from DQO. The remaining 88% could potentially profit from DQO. In fact, for 71% of the columns at least 50% of the values are duplicates. Besides the presences of duplicates, DQO also requires one of the supported compression methods to be used. To our advantage, the compression heuristic tends to select eligible compression methods for duplicated data: The higher the duplication, the more likely a DQO-suitable encoding is chosen.

Most of the visualizations hosted on Tableau Public are based on only a small data set. Half of all columns contain less than thousand rows. If this data was saved using Data Blocks,

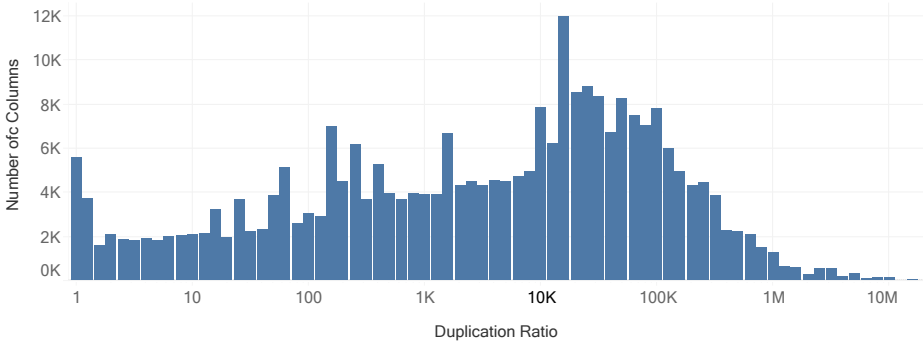


Fig. 9: Number of columns with more than one Data Block grouped by the duplication ratio.

for the vast majority of visualizations not even one Data Block would be filled. Queries on such small data sets are answered in a few milliseconds and most time is spent in the communication layer, for SQL parsing and for scheduling overheads. Focusing on columns with a significant number of rows, i.e., filling at least one Data Block, the numbers shift considerably. Figure 9 shows the same data as Figure 8 again, this time focusing on columns with a larger number of tuples.

We observed that larger datasets tend to contain more duplicate values. Our working theory to explain this correlation between data set size and duplication is the often denormalized data model: We suspect that many of the duplicates would not exist if the data was represented in a normalized schema and are only introduced during denormalization. In contrast, most small datasets were generated from Excel files or CSV files and did not join any data. Hence, for those small datasets Tableau did not denormalize the data and did not introduce the duplication which can be found in most larger extracts.

With this overview of the data in mind, the logical follow-up question is: How often are domain queries posed? In order to answer this question, we manually inspected the queries posed against 88 data sets uploaded to Tableau Public. The data sets were chosen based on their size, focusing on large data sets. The selected data sets have a zipped size between 300 megabytes and 1 gigabyte. In the process of rendering the visualizations based on those data sets, 2,940 queries were posed against Hyper. Most of those queries, in total 1,742, were meta data queries asking, e.g., for the contained tables and their columns. The remaining 1,198 queries resulted in 1,240 table scans on base tables. Approximately 12% of them (in absolute numbers: 157) would be theoretically eligible for DQO. Not all of those table scans were able to profit from DQO, however. This was because all of those table scans were operating on a column stored with a suitable encoding/compression. 128 of the table scans were able to profit from DQO for at least one of the scanned Data Blocks.

To measure the impact of DQO on an actual workload, all 2,940 queries from the examined visualizations were run against Hyper. We used the Ubuntu hardware described in Section 4.1. Without DQO Hyper needed 94.6 seconds on average to execute all queries including non-domain queries and meta data queries against a given dataset. DQO brings this time down by 1.5% to 93.2 seconds. These times include all time spent for query processing in Hyper, including both query compilation and query execution time. They do not include the time needed by Tableau to generate the queries and to interpret and visualize the results. To simulate a moderate load situation, we were always running two queries were run in parallel. The whole experiment was repeated 10 times and we report the arithmetic average over all 10 runs. Given the previous numbers on the duplication factor usually encountered in extracts and that about 6% of the queries are domain queries, the performance benefit of only 1.5% might be surprising at first. In most cases other queries dominate processing time.

To see how the measured 1.5% performance improvement generalizes beyond those 88 data sets, we measured the query times on all 60 thousand visualizations in our test set. Note, that we used a different hardware setup for this last experiment. We used a machine with a Xeon E5-2699 CPU clocked at 2.2GHz with 2 processors and 44 cores. The machine is equipped with 512GB RAM and is running under Windows Server 2012. We chose this differing test machine configuration to make the measurements finish in a feasible time. We do not think, the different hardware to influence the relative benefit of DQO significantly.

Among the already fast visualization (i.e., the query time was already smaller than 200ms), there was only a marginal gain as DQO improved the geometric mean of the query times by 0.15%. For those workbooks, the impact of DQO is small mostly because the computation of the visualization was already fast anyway and most time was spent in other stages of query processing. In general, we are targeting our performance features to address slowness experienced by the customer. As such, we usually focus on the visualizations taking more than than 0.2s to compute. For fast scenarios a customer usually does not notice improvements. Focusing on the slower visualizations, we measured an improvement of 1.69% in the geometric mean which aligns with the 1.5 percent improvement measured in the previous experiment.

When taking a closer look at the visualizations profiting from DQO, we saw that for 1,454 visualizations out of the 60,000 visualizations, the overall query time increased performance by more than 10%. In 2 cases we even saw performance improvements of over 100%. In the most extreme case we saw, performance was increased by slightly more than 250%, reducing query time from 386ms to 110ms.

5 Related Work

Tableau is based on Stolte’s dissertation work on interactive data visualization [St03; STH08]. In earlier work, we analyzed the database workload of typical Tableau applications and found that it differs very much from the well-known analytical TPC benchmarks [Vo18].

Before being supplanted by Hyper, data storage and query processing in Tableau was performed by the Tableau Data Engine (TDE) [Te15; WET11; WT14]. In TDE, domain queries are optimized as part of logical query optimization. TDE exposes dictionary lookups to the logical optimizer as joins between a table containing the dictionary tokens and the dictionary [WET11]. Domain query optimization is thereby subsumed by join culling: The optimizer notices that the table contents itself are not required to answer the query. Therefore, it removes the table scan from the query plan and only the dictionary scan remains.

Hyper was designed as a hybrid OLTP and OLAP main-memory database system [KN11]. It compiles queries and stored procedures to machine code via the LLVM compiler framework [Ne11], and uses morsel-driven parallelism for execution on multi-core CPUs [Le14]. Hyper uses Datablocks [La16] as its columnar storage format using lightweight compression. The tradeoffs of various compression schemes have been studied by Damme et al. [Da17]. Column sketches [HKI18] are a recently proposed form of lossy compression of columns that allows one to carry out the query processing directly on the compressed data. Mörkotte [Mo98] introduced Small Materialized Aggregates as lightweight synopses of data columns. The same idea was later also utilized in IBM Netezza as zone maps⁶. Binnig et al. [BHF09] developed SAP HANA’s ordered dictionary, which allows processing range queries on compressed columns.

6 Conclusion

In this paper, we introduced the notion of domain queries in the context of BI platforms such as Tableau. Domain queries are mostly used to populate filters with a list of all distinct values available in a data set. We have also shown how these domain queries can be answered efficiently in the Hyper database system, that is the data engine in Tableau products for data extracts. For this purpose, we first showed how to detect domain queries on relational algebra trees. After this, we went on to demonstrate how to optimally execute domain queries against Hyper’s Data Block storage format.

We evaluated our implementation of domain query optimization (DQO) on synthetic data sets. Those experiments have shown that DQO can easily reach query time speedups of multiple orders of magnitude. To show the real-world impact of DQO, we further analyzed a set of 60 thousand real-world Tableau visualization from Tableau Public [Vo18]. A static analysis of these workbooks was conducted to get a feeling for the potential impact of DQO.

⁶ https://www.ibm.com/developerworks/community/blogs/Wce085e09749a_4650_a064_bb3f3b738fa3/entry/understanding_netezza_zone_maps?lang=en

We manually inspected the queries posed against the largest data sets in our test set: 12% percent of the inspected queries qualified as domain queries. Measuring the impact of DQO on the query times for the queries issued during rendering those visualizations, we saw a performance improvement of up to 250%.

References

- [BHF09] Binnig, C.; Hildenbrand, S.; Färber, F.: Dictionary-based order-preserving string compression for main memory column stores. In: SIGMOD. Pp. 283–296, 2009.
- [Da17] Damme, P.; Habich, D.; Hildebrandt, J.; Lehner, W.: Lightweight Data Compression Algorithms: An Experimental Survey (Experiments and Analyses). In: EDBT. Pp. 72–83, 2017.
- [FKN12] Funke, F.; Kemper, A.; Neumann, T.: Compacting Transactional Data in Hybrid OLTP & OLAP Databases. PVLDB 5/11, pp. 1424–1435, 2012.
- [HKI18] Hentschel, B.; Kester, M. S.; Idreos, S.: Column Sketches: A Scan Accelerator for Rapid and Robust Predicate Evaluation. In: SIGMOD. Pp. 857–872, 2018.
- [KN11] Kemper, A.; Neumann, T.: HyPer: A Hybrid OLTP & OLAP Main Memory Database System Based on Virtual Memory Snapshots. In: ICDE. Pp. 195–206, 2011.
- [La16] Lang, H.; Mühlbauer, T.; Funke, F.; Boncz, P. A.; Neumann, T.; Kemper, A.: Data Blocks: hybrid OLTP and OLAP on compressed storage using both vectorization and compilation. In: SIGMOD. Pp. 311–326, 2016.
- [Le14] Leis, V.; Boncz, P.; Kemper, A.; Neumann, T.: Morsel-driven parallelism: a NUMA-aware query evaluation framework for the many-core age. In: SIGMOD. Pp. 743–754, 2014.
- [Mo98] Moerkotte, G.: Small Materialized Aggregates: A Light Weight Index Structure for Data Warehousing. In: VLDB. Pp. 476–487, 1998.
- [Ne11] Neumann, T.: Efficiently compiling efficient query plans for modern hardware. Proceedings of the VLDB Endowment 4/9, pp. 539–550, 2011.
- [Ra08] Raman, V.; Swart, G.; Qiao, L.; Reiss, F.; Dialani, V.; Kossmann, D.; Narang, I.; Sidle, R.: Constant-Time Query Processing. In: ICDE. Pp. 60–69, 2008.
- [St03] Stolte, C.: Query, analysis, and visualization of multidimensional databases. PhD thesis, Stanford University/, 2003.
- [STH08] Stolte, C.; Tang, D.; Hanrahan, P.: Polaris: a system for query, analysis, and visualization of multidimensional databases. Communications of the ACM 51/11, pp. 75–84, 2008.

- [Te15] Terlecki, P.; Xu, F.; Shaw, M.; Kim, V.; Wesley, R.: On improving user response times in Tableau. In: SIGMOD. Pp. 1695–1706, 2015.
- [Vo18] Vogelsgesang, A.; Haubenschild, M.; Finis, J.; Kemper, A.; Leis, V.; Muehlbauer, T.; Neumann, T.; Then, M.: Get Real: How Benchmarks Fail to Represent the Real World. In: Proceedings of the Workshop on Testing Database Systems. 2018.
- [WET11] Wesley, R.; Eldridge, M.; Terlecki, P.: An analytic data engine for visualization in Tableau. In: SIGMOD. Pp. 1185–1194, 2011.
- [WT14] Wesley, R.; Terlecki, P.: Leveraging Compression in the Tableau data engine. In: SIGMOD. Pp. 563–573, 2014.

Text

Einsatz kognitiver Verfahren am Deutschen Patent- und Markenamt

Mark Reinke¹, André Kischkel², Volker Jahns³, Uwe Crenze⁴, Olga Beltcheva⁵

Abstract: Die Begutachtung von Patentanträgen ist ein aufwändiger Prüfprozess, dessen Ziel es ist, eine Entgegenhaltung zu finden. Dabei stellt die Verschleierung des Patentanspruchs durch gezielte Umschreibungen eine große Herausforderung dar. In enger Zusammenarbeit zwischen dem Deutschen Patent- und Markenamt und der interface projects GmbH entstand ein modernes Recherche- und Klassifikationssystem auf Basis von durch neuronale Netze gelernten Distributed Word Embeddings. Der Beitrag stellt verschiedene Verfahren zum Lernen von Word Embeddings vor und bewertet diese hinsichtlich ihrer Eignung für die Prüfung von Patentanmeldungen.

Keywords: DPMA, Patentprüfung, Recherche, Klassifikation, kognitive Verfahren, Word Embeddings

1 Einleitung

In den letzten fünf Jahren konnten erhebliche Fortschritte beim Einsatz von maschinellen Lernverfahren, basierend auf künstlichen neuronalen Netzen, verzeichnet werden. Im Bereich der Text- und Multimedia-Analyse wird in diesem Zusammenhang typischerweise von kognitiven Verfahren gesprochen. Hierzu zählen insbesondere Verfahren zur Klassifikation, Verschlagwortung, Erkennung relevanter Entitäten und zur Unterstützung einer semantisch-assoziativen Suche (kognitive Suche). Insbesondere profitieren systematische Rechercheprozesse und die Erschließung unbekannter Dokumente von solchen Verfahren.

Wie in kaum einem anderen Bereich gehört das systematische Recherchieren zur Kernaufgabe der Prüfer in Patentämtern. Die stark ansteigende Anzahl von Patentanmeldungen auf nationaler und internationaler Ebene stellt eine große Herausforderung bei der Bearbeitung der Schutzrechtsverfahren dar. Gleiches gilt für Patentabteilungen in Unternehmen und Forschungseinrichtungen. Erschwerend kommt hinzu, dass bei der Erstellung der Patentschriften gezielt Umschreibungen verwendet werden, welche die Nähe der angemeldeten Innovation zu bereits erteilten Patenten verschleiern und eine Entgegenhaltung zum angemeldeten

¹ interface projects GmbH, Zwinglistraße 11/13, 01277 Dresden, mark.reinke@interface-projects.de

² interface projects GmbH, Zwinglistraße 11/13, 01277 Dresden, andre.kischkel@interface-projects.de

³ Deutsches Patent- und Markenamt, Referat 2.3.1, Zweibrückenstraße 12, 80331 München, volker.jahns@dpma.de

⁴ interface projects GmbH, Zwinglistraße 11/13, 01277 Dresden, uwe.crenze@interface-projects.de

⁵ Deutsches Patent- und Markenamt, Referat 2.3.1, Zweibrückenstraße 12, 80331 München, olga.beltcheva@dpma.de

Patentanspruch verhindern soll. Die durch kognitive Verfahren durchgeführte Analyse der Patentschriften unterstützt die Patentprüfer bei der Aufdeckung solcher Verschleierungen, indem besser nach inhaltlich ähnlichen Patentschriften für entsprechende Entgegenhaltungen recherchiert werden kann. Eine andere Herausforderung besteht in der gezielten Zuordnung eines hinsichtlich der fachlichen Aspekte der Patentanmeldung erfahrenen Prüfers. Hier unterstützen kognitive Verfahren bei der Klassifikation der eingehenden Patentanmeldungen.

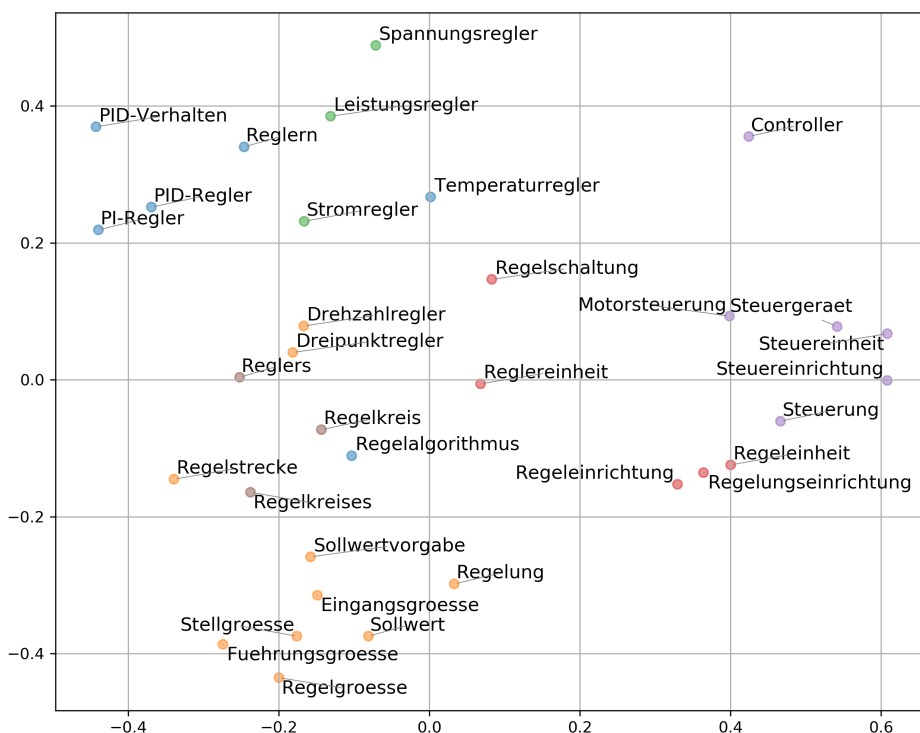
In diesem Beitrag werden verschiedene kognitive Verfahren vorgestellt, die im Rahmen der Zusammenarbeit zwischen dem Deutschen Patent- und Markenamt (DPMA) und der Firma interface projects GmbH auf der Basis des Produktes *intergator*⁶ entwickelt und am DPMA eingeführt wurden.

1.1 Die Evolution der Text-Indexierung: von Term-Vektoren zu Distributed Word Embeddings

Für ein effizientes Text-Information-Retrieval wurden in den letzten Jahrzehnten zahlreiche Verfahren entwickelt. Weit verbreitet sind insbesondere sogenannte Vector-Space-Modelle (VSM), die nahezu jede Text-Suchmaschine verwendet - oft unter Anwendung des TF-IDF-Maßes zur Bestimmung der Relevanz der einzelnen Terme. Die Dimension der Vektoren wird durch die Anzahl unterschiedlicher Terme in dem zu durchsuchenden Korpus bestimmt. Ein Vektor repräsentiert ein Dokument im Korpus. Die Ähnlichkeit von Dokumenten kann leicht durch den Kosinus zwischen den Dokument-Vektoren bestimmt werden. Die Nachteile von Term-VSM liegen in der hohen Dimensionalität des Vektorraums. Zudem sind die Vektoren sehr schwach besetzt. Verfahrensbedingt geben diese Vektoren auch keinerlei Information über die semantische Ähnlichkeit von Dokumenten. Erfolgt eine Suchanfrage mit einem Begriff, der in den gesuchten Dokumenten nicht enthalten ist, erhält man keine Treffer. Trotz einer breiten Palette von weiteren Methoden des Natural Language Processing (NLP) können viele linguistische und semantische Aufgabenstellungen meist nicht befriedigend gelöst werden. Hinterlegte Wörterbücher und sonstige Begriffssysteme, einschließlich aufwendig erstellter Ontologien, können von den Nutzern in der Regel weder bereitgestellt, noch aktuell gehalten werden.

Mit den Arbeiten von Mikolov bei Google und später bei Facebook wurden die grundlegenden VSM unter Anwendung künstlich neuronaler Netze auf eine neue Stufe gehoben. Das unter *word2vec* [Mi13] 2013 bekannt gewordene Verfahren für *Distributed Word Embeddings* verwendet neuronale Netze zum Lernen von Zusammenhängen zwischen Wörtern. Ein Vektor repräsentiert dabei ein Wort in einem Vektorraum mit wenigen hundert Dimensionen. Räumlich benachbarte Vektoren beschreiben Wörter mit einem ähnlichen Kontext. Daraus ergeben sich vielfältige Anwendungsgebiete, wie die automatische Ermittlung von sinnverwandten Suchbegriffen, die Suche in fremdsprachigen Dokumenten und die Klassifikation von Dokumenten.

⁶ <https://www.intergator.de>

Abb. 1: Verwandte Begriffe zu *Regler*

Die Grafik Abb. 1 visualisiert den semantischen Kontext zum Begriff *Regler* in Form einer 2D-Projektion eines aus ca. 300.000 Patenten trainierten 300-dimensionalen Word-Embedding-Vektorraums.

Die Besonderheit dieser Word-Embedding-Vektoren liegt darin, dass ein Wort über die gelernten Kontexte, in denen es auftritt, beschrieben wird. Darüber hinaus sind Word-Embedding-Verfahren robust gegenüber unbekanntem Begriffen in Dokumenten, die nicht in das Modell-Training einbezogen waren. Auf diese Weise können auch falsch geschriebene Wörter durch ihren Kontext erkannt und richtig eingeordnet werden. Word Embeddings liefern somit eine optimale Strategie für eine semantisch-assoziative (unscharfe) Suche. Letztendlich bildet die Gesamtheit der gelernten Word-Embedding-Vektoren ein Sprachmodell. Aus großen Korpora (z. B. Wikipedia, Zeitungs- und Nachrichtenarchiven u. ä.) lassen sich allgemeine Modelle vortrainieren und im Zusammenhang mit völlig anderen Textbeständen nutzen. Auch lassen sich Wort-Vektoren von einem sprachspezifischen Modell in ein anderes transformieren, um Suchen in fremdsprachigen Dokumentenbeständen zu ermöglichen.

Interessanterweise werden beim Erstellen der Word-Embedding-Vektoren auch linguistische Beziehungen zwischen Begriffen gelernt. Über Vektoroperationen lassen sich somit gezielt semantische Relationen ermitteln und für Frage-Antwort-Systeme oder Empfehlungssysteme einsetzen. Ein typisches Beispiel sind die schon in der ersten Veröffentlichung von Mikolov zu Word2Vec beschriebenen Wort-Analogien. Beispielsweise führt die Operation $\langle \text{König} \rangle - \langle \text{Mann} \rangle + \langle \text{Frau} \rangle$ zum Vektor $\langle \text{Königin} \rangle$. Gleichmaßen lässt sich aus zwei Ländernamen und einer der Hauptstädte die Hauptstadt des zweiten Landes bestimmen.

Das maschinelle Lernen von Word Embeddings ist grundsätzlich unüberwacht. Das heißt, es müssen keine manuell kategorisierten Dokumente bereitgestellt werden. Über spezielle Erweiterungen von Word2Vec (z. B. bei fastText [Jo16]) lassen sich auch klassenspezifische Modelle trainieren, die im Anschluss von einem entsprechenden Klassifikator verwendet werden.

1.2 Lernen von Word Embeddings

Die Grundidee von Word Embeddings besteht darin, dass sich die Bedeutung eines Wortes aus den Wörtern erschließt, in deren syntaktischer Nähe es benutzt wird. Die Implementierung von Word-Embedding-Verfahren basiert hauptsächlich auf diesem linguistischen Phänomen. Einen umfassenden Überblick über das Lernen von Word Embeddings mit Hilfe neuronaler Netze wird in [Ko16] gegeben. Zum Erlernen solcher Kontext-Wort-Beziehungen werden zunächst Wort-Paare gebildet. Als Beispiel-Korpus dient folgendes Pangramm:

Zwölf Boxkämpfer jagen Viktor quer über den großen Sylter Deich.

Dieses Pangramm besteht aus 10 Wörtern, die unser Vokabular bilden. Der Kontext eines Wortes wird über ein Fenster aus Wörtern links und rechts um das Fokuswort gebildet. In unserem Beispiel gehen wir der Einfachheit halber von einem Wortfenster von 1 aus, d. h. der Kontext wird durch jeweils ein Wort links und eins rechts vom Fokuswort gebildet. So erhalten wir folgende (Kontext, Wort)-Paare:

([Zwölf, jagen], Boxkämpfer), ([Boxkämpfer, Viktor], jagen), ([jagen, quer], Viktor), ...

Im Folgenden wird das Lernen von Word Embeddings am Beispiel von Word2Vec erläutert. Word2Vec stellt zwei verschiedene Netzwerkarchitekturen bereit. Die erste Variante *Continuous Bag of Words* (CBOW) sagt zu einem Kontext das Fokuswort voraus. Die zweite Architekturvariante *Skip-Gram* funktioniert genau umgekehrt und sagt zu einem Wort den wahrscheinlichen Kontext voraus. Der Vorteil der Skip-Gram-Architektur liegt in einer höheren Robustheit bei der Erstellung von Wortvektoren für seltene Wörter. Dies wird allerdings mit einer gegenüber CBOW wesentlich höheren Berechnungszeit erkaufte. Da im

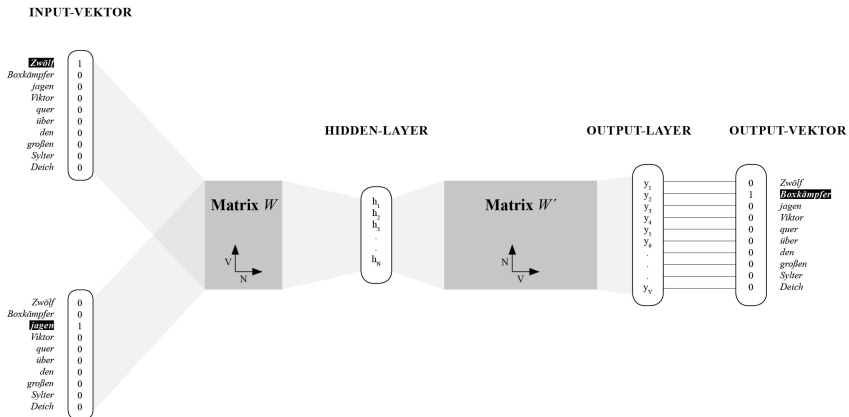


Abb. 2: Neuronales Netz mit CBOW-Architektur

vorliegenden Anwendungsfall zwischen CBOW und Skip-Gram kein signifikanter Unterschied in der Qualität der Ergebnisse festgestellt werden konnte, wurde aus Performance-Gründen CBOW verwendet.

Bei word2vec CBOW besitzen die neuronalen Netze je einen Input-Vektor für die Kontextwörter, einen Hidden-Layer mit N linearen Neuronen (Dimension des Modell-Vektorraums) und einem Output-Layer, der durch einen Softmax-Klassifikator ergänzt wird und das Fokuswort in einem Output-Vektor repräsentiert. Die Dimension der Input- und Output-Vektoren wird durch die Größe des Vokabulars (V) des zu untersuchenden Korpus bestimmt.

In unserem einfachen Beispiel besteht der Korpus aus einem einzigen Dokument mit einem Vokabular von lediglich 10 Wörtern. Normalerweise besteht ein Korpus aus vielen Dokumenten, die insgesamt ein Vokabular von tausenden von Wörtern umfassen. Die Input-Vektoren für die beiden Kontextwörter und der Output-Vektor für das Fokuswort werden durch sogenannte *One-Hot-Vectors* gebildet, die bis auf die Position des jeweiligen Kontext- bzw. Fokuswortes ausschließlich aus Nullen bestehen.

Für die abstrakte Repräsentation der aus den Kontextwörtern gelernten Bedeutung des Fokuswortes wird ein Vektor in einem N -dimensionalen Vektorraum erzeugt, wobei N typischerweise zwischen 300 und 500 liegt. Hierfür besitzt das neuronale Netzwerk einen Hidden-Layer, dessen N Neuronen der Spalten einer Gewichtsmatrix $W_{(V \times N)}$ entsprechen. Die Matrix besteht aus V Reihen (für jedes Wort des Vokabulars).

Der Output-Layer besteht aus einer zweiten Gewichtsmatrix $W_{(N \times V)}$ und einem Softmax-Regressionsklassifikator für die Rekonstruktion des Output-(Fokuswort)-Vektors. Jedes Output-Neuron besitzt einen Gewichtsvektor, der mit dem korrespondierenden Vektor im Hidden-Layer multipliziert wird. Das Ergebnis ist die Wahrscheinlichkeit, mit der das dem Output-Neuron zugeordnete Wort des Vokabulars ein Fokuswort zum gegebenen Kontext ist. Da ähnliche Wörter in ähnlichen Kontexten stehen, werden dementsprechend auch ähnliche Gewichtsvektoren erlernt.

1.3 Weiterentwicklung von Word Embeddings

Durch die Möglichkeit von Word Embeddings die Semantik und Syntaktik von Wörtern als Vektoren zu repräsentieren, eignen sie sich hervorragend als Grundbausteine für verschiedenste NLP-Anwendungen.

Durch Abwandlung und Erweiterung der ursprünglichen Architektur und Einbeziehung weiterer Techniken des maschinellen Lernens inklusive geschickter Optimierungen können Word Embeddings auch als Basis für Information-Retrieval-Tasks, wie Textklassifikation oder Textähnlichkeitsaufgaben, dienen. Facebook AI Research stellt mit *fastText* eine effiziente Erweiterung von *Word2Vec* zur Verfügung. Durch die Möglichkeit, Vektoren von *N-Grammen* auf Zeichen- oder Wortebene zu bilden, können auch unbekannte Wörter (*Out-of-vocabulary*) und Mehrwortbegriffe einbezogen werden. Weiterhin überträgt *fastText*, durch eine leichte Abwandlung der CBOW-Architektur, das Konzept der Embeddings auf Klassifikationsprobleme.

Im Folgenden wird die Anwendung und Evaluierung der hier dargestellten Verfahren anhand der Patentrecherche und Patentklassifikation am Deutschen Patent- und Markenamt vorgestellt.

2 Patentrecherche

Die Patentrecherche ist ein wesentlicher Bestandteil des Patenterteilungsverfahrens am DPMA. Im Rahmen der Patentprüfung ermittelt der Prüfer zu einer Patentanmeldung den relevanten Stand der Technik. Dies ist eine Liste relevanter Dokumente der Patent- und Nichtpatentliteratur, die sogenannten Entgegenhaltungen. Der Prüfer entscheidet anhand des Rechercheergebnisses, ob die Grundvoraussetzung für eine Patenterteilung, das Beruhen der angemeldeten Erfindung auf Neuheit und erfinderische Tätigkeit, gegeben ist.

Die Patentrecherche mit dem hauseigenen Deutschen Patentinformationssystem (DEPATIS) basiert bis dato auf der Suche nach Termen und bibliographischen Daten von Patentschriften. Die Suchtechnologie beruht auf einer Booleschen Suche. Die Suchbegriffe werden exakt oder trunkiert eingegeben und verwandte Begriffe müssen aus manuell gepflegten

Wortlisten übernommen werden. Die Suche nach bibliographischen Daten, wie zum Beispiel diversen Klassifikationssymbolen (IPC, CPC oder FI), Namen von Erfindern oder Anmeldern, Datumsbereichen wie Anmeldedatum, Publikationsdatum oder Prioritätsdaten, Familienmitgliedern, Entgegenhaltungen und zitierenden bzw. zitierten Schriften, liefert Dokumente, in denen exakt diese bibliographischen Daten vorkommen.

In der Praxis ist die Bildung von komplexen und langen Suchanfragen notwendig, weil die Kombination von Begriffen und Synonymen explizit angegeben werden muss. Die Suche soll eine Liste von Dokumenten liefern, die in einer angemessenen Zeit vom Prüfer gesichtet und bewertet werden kann. Das schnelle Anwachsen der Anzahl publizierter Patentschriften erschwert die Lage des Prüfers. Die Formulierung solcher Suchanfragen ist sehr zeitaufwändig, fehlerträchtig und erfordert langjährige Erfahrung auf dem Prüfgebiet. Da Anmeldungen oft allgemein formuliert werden, um den Wirkungsgrad des Patentes zu erhöhen (*Kleinfahrzeug* anstatt *Fahrrad*) und der Anmeldegegenstand durch die Verwendung abseits vom Stand der Technik liegender Begriffe, Umschreibungen (*im Kreis bewegen* anstatt *rotieren*) oder Wortneuschöpfungen (*Müllentsorgungsdrohne*) meist verschleiert ist, muss der Prüfer den Text inhaltlich erschließen und die Bedeutung des Textes intellektuell erfassen.

2.1 Kognitive Suche

The screenshot displays the 'intergator: cognitive search' interface. At the top, it shows the search filters: 'Patents (DE), 2000-2015' and 'Patents (US), 2000-2015'. The main search area shows a search result for 'DE 102012214867A1 (20140227)' titled 'Elektronisch gesteuertes Federsystem, Verfahren zur Steuerung eines...'. Below this, there are search filters and a search result summary: 'Search Result: 1 - 24 of 1,415,489' with a hashtag '#118f6b7e'. A bar chart shows the distribution of results across various terms like 'fahrrad', 'dämpfung', 'fahrader', etc. The main content area displays five search results, each with a star icon and a brief description:

- DE 1** ★: DE102012214867A1 (20140227) Elektronisch gesteuertes Federsystem, Verfahren zur Steuerung eines...
- DE 2** ★: DE202011110168U1 (20130704) Sensorsystem sowie Fahrzeug mit einem Sensorsystem
- DE 3** ★: DE102013212949A1 (20140130) Verfahren und Vorrichtung zur Steuerung einer Dämpfung eines durch den Fahrer...
- DE 4** ★: DE102013214517A1 (20150129) Fahrrad mit elektrischem Lenkeingriff sowie Verfahren zur Stabilisierung des...
- DE 5** ★: DE102011030405A1 (20130726) Stoßdämpfer für Fahrrad
- DE 6** ★: DE20001033557A1 (20040527) Umschaltvorrichtung

On the right side, a detailed view of the search result for 'Stoßdämpfer für Fahrrad' is shown. It includes the title, abstract, and claims. The abstract describes a shock absorber for a bicycle with an electronic damping device. The claims list five specific features of the device.

Abb. 3: Benutzeroberfläche der kognitiven Suche für die Patentrecherche

Die kognitive, auf semantischer Textähnlichkeit beruhende Suche reduziert den manuellen Aufwand und ermöglicht eine einfache, effiziente und effektive Recherche. Es werden folgende Funktionen bereitgestellt:

1. Die kognitive Suche ist in der Lage, inhaltlich ähnliche Dokumente zu einer Patentanmeldung ohne manuelle Eingabe zu finden. Es wird eine Auswahlliste von Dokumenten als nächstliegender Stand der Technik präsentiert. Eine automatisierte Patent-Versuche wird damit möglich.
2. Es ist sowohl möglich, relevante Abschnitte einer Anmeldung für die Suche auszuwählen, als auch in Kombination mit anderen relevanten Druckschriften zu suchen. So ist es beispielsweise möglich, ausgehend von einzelnen Ansprüchen der Anmeldung oder in Kombination mit dem in der Anmeldung zitiertem Stand der Technik zu suchen.
3. Bei der Suche werden Synonyme oder sinnverwandte Begriffe gefunden, die in weiteren Recherchen verwendet werden können. Die intellektuelle Ermittlung von Synonymen für die Formulierung der Suchanfrage bzw. Erstellung und Pflege von Synonym-Listen ist nicht mehr erforderlich.
4. Die Bewertung der Suchergebnisse wird durch Hervorheben von Begriffen, die eine semantische Nähe zur Eingabe aufweisen erleichtert.

Das Neuartige an dieser semantischen Suchtechnologie ist die Identifizierung von Patentliteratur mit ähnlicher Bedeutung. Ein ähnliches Dokument kann auch dann von der Suchmaschine als relevant angesehen werden, wenn die Begriffe der Sucheingabe nicht darin vorkommen. Diese Funktionalität ist für die Patentrecherche besonders wichtig, da für die Neuartigkeit eines Patentes die darin beschriebenen Ideen entscheidend sind und nicht die Begriffe, die verwendet werden.

Word Embeddings repräsentieren semantische und syntaktische Eigenschaften von Worten und sind damit eine geeignete Basistechnologie für die kognitive Suche. Die Idee der Word Embeddings ist auch auf größere Textabschnitte anwendbar. Diese sogenannten *Sentence Embeddings* bilden die Bedeutung von Phrasen oder Sätzen auf Vektoren ab. Damit lassen sich, wie bei Word Embeddings, Zusammenhänge als mathematische Operationen ausdrücken. Einige solcher Verfahren werden im Folgenden untersucht und es wird gezeigt, dass sich die Idee der Sentence Embeddings auch auf ganze Dokumente erweitern lässt (*Document Embeddings*). Eine Suchoperation kann dann als Vektorähnlichkeit zwischen Sucheingabe-Vektor und Dokumentvektoren im Suchraum ausgedrückt werden.

Die *Sentence Embedding*-Verfahren lassen sich in zwei Kategorien unterteilen: einfache, flache neuronale Netzwerke (Word2Vec, SIF [ALM17], Sent2Vec [PGJ17]) und komplexere, tiefe neuronale Netze (Skip-Thoughts Vectors [Ki15], InferSent [Co17]).

Wir beschränken uns hier auf flache neuronale Netze, da für das Training auf großen Datenmengen ein effizienter, ressourcenschonender Algorithmus benötigt wird und einfache Netzwerke bei ähnlichen Problemstellungen vergleichbar gute Ergebnisse erzielen konnten wie komplexere Architekturen (Arora et al. [ALM17], Pagliardini et al. [PGJ17], Hill et

al. [HCK16]). Die Entwicklung im Bereich *Deep Learning* ist jedoch rasant und eine zukünftige Betrachtung solcher Verfahren sinnvoll.

Für die kognitive Suche werden Embeddings verwendet, die mit dem *Sent2Vec*-Algorithmus erlernt wurden. Einzelne Wörter und ganze Dokumente werden in denselben Vektorraum projiziert und können durch einfache räumliche Entfernungsmetriken in Verbindung gesetzt werden. Mit diesem universellen Ansatz eröffnen sich neue Möglichkeiten. So realisieren wir damit, als Teil der kognitiven Suchanwendung, eine Reihe unterschiedlichster Information-Retrieval-Aufgaben:

Suche (Wörter \Rightarrow Dokumente): Hierbei macht man sich die Tatsache zu Nutze, dass der Zentroid der Wortvektoren der Sucheingabe räumlich nahe bei den semantisch ähnlichen Dokumenten liegt.

Dokumentähnlichkeit (Dokument \Rightarrow Dokumente): Semantisch ähnliche Dokumente liegen im Vektorraum nahe beieinander.

Automatische Synonyme (Wort \Rightarrow Wörter): Semantisch ähnliche Worte liegen im Vektorraum nahe beieinander.

Automatische Verschlagwortung (Dokument \Rightarrow Wörter): Schlagwörter eines Dokuments befinden sich in räumlicher Nähe zu den Dokumentvektoren.

Durch Matrix-Rotation werden die Vektorräume verschiedener Sprachmodelle zueinander ausgerichtet und lassen sich dadurch kombiniert verwenden [Co18]. Alle beschriebenen Aufgabenstellungen sind somit auch sprachübergreifend möglich.

2.2 Evaluation

Es gibt einige Veröffentlichungen, in denen *Sentence Embedding*-Verfahren für Aufgaben wie Sentimentanalyse, Textklassifikation oder Satzähnlichkeit evaluiert wurden. Soweit uns bekannt ist, wurden Sentence Embeddings noch nicht auf ihre Eignung für Suchanwendungen getestet. Es wurden in diesem Projekt zwei Tests entwickelt - der Titel-Test, um die Suche und der Entgegenhaltungstest, um die Dokumentähnlichkeit zu evaluieren. Folgende Verfahren werden miteinander verglichen:

T-VSM TF-IDF: Als Baseline dient das *Term Vector Space Model* (T-VSM) mit TF-IDF-Maß. Es wird die *More Like This Query* von Elasticsearch⁷ verwendet, um die relevantesten Begriffe eines Textes zu ermitteln und ODER-verknüpft nach diesen zu suchen.

⁷ https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-mlt-query.html#_how_it_works

Word2Vec: Die einfachste Methode Sentence Embeddings zu erhalten, ist die Mittelung aller mit Word2Vec gelernten Word Embeddings eines Textes. Eine Suche nach ähnlichen Dokumenten lässt sich als Suche nach den ähnlichsten Dokument-Vektoren zum Durchschnittsvektor der Sucheingabe ausdrücken.

SIF: Bei *Smooth Inverse Frequency* (SIF) erfolgt die Durchschnittsbildung mit gewichteten Wortvektoren, bei der relevante Wörter bevorzugt werden. Zudem werden die ersten Prinzipalkomponenten der Dokument-Vektoren entfernt (*Common Component Removal*), um semantisch unbedeutende Aspekte aus den Embeddings zu entfernen.

Sent2Vec: Sent2Vec ist eine Erweiterung von fastText. Ähnlich wie bei CBOW werden Wortvektoren trainiert, indem Wörter auf Grund ihres Kontextes vorhergesagt werden. Word2Vec verwendet ein Kontextfenster mit parametrisierbarer Maximallänge, Sent2Vec den gesamten Satz unter Erhalt aller Wort-N-Gramme. Die trainierten Wortvektoren eignen sich besonders gut für die Durchschnittsbildung langer Eingabetexte.

2.2.1 Titel-Test

In Ermangelung eines Goldstandards wurde ein automatisiertes Testverfahren konzipiert, mit dem eine „unscharfe“ Stichwortsuche simuliert wird. Beim *Titel-Test* werden die Patentschriften in Titel und Volltext (Zusammenfassung, Ansprüche, Beschreibung) separiert. Es wird nach dem Titel gesucht und evaluiert, ob das System das zugehörige Dokument als relevantesten Suchtreffer zurückliefert. Um zu testen, ob ein Text aufgrund seiner semantischen Ähnlichkeit zur Suchanfrage gefunden werden kann, werden in einem Vorverarbeitungsschritt alle Begriffe der Titel aus dem Volltext entfernt. Für diesen Test werden 10.000 deutsche Patentschriften als Datenbasis verwendet. Tab. 1 zeigt für jedes Verfahren den erreichten Recall in Prozent. Bei *Recall @1* wird getestet, ob das erste gefundene Dokument das gesuchte ist. Bei *Recall @10* muss das gesuchte Dokument unter den ersten 10 Treffern sein.

Verfahren	Recall @1	Recall @10
T-VSM TF-IDF	-	-
Word2Vec	10,8	29,6
SIF	28,9	61,0
Sent2Vec	44,1	76,6

Tab. 1: Die Ergebnisse des Titel-Tests

Sent2Vec ist den anderen Verfahren deutlich überlegen. Der Test (Tab. 1) zeigt, dass eine kognitive Suche, die nicht vom Vorkommen der Suchwörter in den Dokumenten abhängt, sondern auf inhaltlicher Ähnlichkeit basiert, möglich ist. Eine Suche mit Term-VSM kann keine korrekten Ergebnisse liefern, da die Eingabewörter aus den Dokumenten entfernt wurden.

2.2.2 Entgegenhaltungstest

In diesem Test wird evaluiert, wie gut ein System zu einer vorgegebenen Patentschrift ähnliche Schriften finden kann. Dafür wird die Tatsache genutzt, dass zitierte Literatur in den bibliografischen Daten der Patente hinterlegt wird. Dieser Test spiegelt den realen Anwendungsfall der Patentrecherche wieder. Für die Evaluation werden ca. 1,5 Millionen deutsche Patent- und Offenlegungsschriften der Jahrgänge 2000 - 2015 verwendet. Als Eingabe werden 2.630 Schriften betrachtet, für die alle mindestens eine zitierte Schrift im Datenbestand enthalten ist.

Verfahren	Recall @10	Recall @100	Recall @1000	Trainingszeit	Laufzeit
T-VSM TF-IDF	10,4	21,1	31,2	-	1h 52m
Word2Vec	0,6	12,3	20,6	18h	1m 39s
SIF	16,6	41,0	69,8	18h	3m 10s
Sent2Vec	17,6	43,3	69,4	30h	4m 40s

Tab. 2: Ergebnisse des Entgegenhaltungstests

Die Eignung von Sent2Vec für die Patentrecherche wird durch diesen Test (siehe Tab. 2) bestätigt. Wie beim Titel-Test bringt eine einfache Mittelung von Word2Vec-Vektoren keine guten Ergebnisse und schneidet schlechter ab als die Baseline. Mit SIF und Sent2Vec können hochwertige Dokumentvektoren erzeugt werden, die die semantischen Eigenschaften von Dokumenten abbilden.

2.3 Fallbeispiel

Am Beispiel der Patentschrift „*Elektronisch gesteuertes Federungssystem, Verfahren zur Steuerung eines Federungssystems und Computerprogramm*“ (DE102012214867) soll veranschaulicht werden, wie mittels kognitiver Suche das entgegengehaltene Patentedokument „*Stoßdämpfer für Fahrrad*“ (DE102011009405) trotz unterschiedlicher Terminologie gefunden werden konnte. Während mit einer klassischen Stichwortsuche die Entgegenhaltung nicht ermittelt werden kann, ist die Ähnlichkeit der Dokument-Vektoren hoch (unter den Top-3 Treffern im Entgegenhaltungstest). In der qualitativen Betrachtung wird ersichtlich, dass sich, obwohl keine große textuelle Übereinstimmung der Patente besteht, die Texte dennoch inhaltlich ähnlich sind, wie im Folgenden anhand der Textauszüge zu sehen ist.

Auszug aus DE102012214867:

Die Erfindung betrifft ein **elektronisch gesteuertes Federungssystem** für ein **Fahrrad** (1), enthaltend zumindest einem **Federelement** (3, 4), welches zwischen einem ersten Teil (10) des **Fahrrades** und einem zweiten Teil (14, 15) des **Fahrrades** (1) angeordnet ist, welche beweglich miteinander **verbunden** sind, wobei zumindest eine Kenngröße des **Federelementes** veränderbar ist, und zumindest einen Aktor (431), welcher auf das **Federelement** (3, 4) einwirkt, um die zumindest eine Kenngröße zu verändern, und ein **Elektronikmodul** (6), mit welchem ein Ansteuersignal für den zumindest einen Aktor (431) erzeugbar ist, wobei weiterhin ein **Steuerelement** (2, 63, 66) vorhanden ist, mit welchem das vom **Elektronikmodul** (6) erzeugte Ansteuersignal beeinflussbar ist, wobei das **Steuerelement** (2, 63) mit dem **Elektronikmodul** (6) über ein **Funksignal** (64) verbindbar ist und/oder der Aktor (431) mit dem **Elektronikmodul** (6) über ein Funksignal (64) verbindbar ist. Weiterhin betrifft die Erfindung ein entsprechendes Verfahren zur **Steuerung** eines **Federungssystems** für ein **Fahrrad** und ein Computerprogramm zu dessen Durchführung.

Auszug aus DE102011009405:

Stoßdämpfer für ein **Fahrrad** mit einer **Dämpfereinrichtung** mit einer ersten und einer zweiten **Dämpferkammer**, die über ein steuerbares Drosselventil in Verbindung stehen. Dabei ist eine wechselbare **Elektronikeinheit** vorgesehen, welche eine **Steuereinrichtung** umfasst, um mittels der **Steuereinrichtung** das elektrisch steuerbare Drosselventil zu steuern, um die **Dämpfereigenschaften** zu beeinflussen.

Dieses Beispiel ist typisch für die Vielfältigkeit der im Patentwesen verwendeten Fachsprache. Die folgende Tabelle zeigt die Verwendung unterschiedlicher, aber sinnverwandter Begriffe in den zwei Patenten:

DE102012214867	DE102011009405
Elektronikmodul, elektronisch	Elektronikeinheit, elektrisch
Steuerelement, Steuerung, gesteuert	Steuereinrichtung, steuerbar
Federungssystem, Federelement	Stoßdämpfer, Dämpfereinrichtung, Dämpferkammer

Tab. 3: Unterschiede im Vokabular zweier ähnlicher Patentschriften

3 Patentklassifikation

Die Patentklassifikation dient dem einheitlichen Klassifizieren von Patentdokumenten nach technischen Gebieten. Durch die ordnungsgemäße Einordnung von Patentdokumenten in eine Klassifikationsstruktur wird dem Patentprüfer der Zugriff zu der darin enthaltenen technischen Information erleichtert. Es existieren verschiedene Klassifikationsschemata, wie zum Beispiel die internationale Patentklassifikation (IPC), die Cooperative Patent Classification (CPC), die Klassifikation des Japanischen Patentamts (FI) und weitere. Alle Klassifikationsschemata besitzen eine hierarchische Baumstruktur. Im Prüfungsverfahren am DPMA werden Patentdokumente einheitlich nach der IPC klassifiziert, aber es werden auch andere Schemata berücksichtigt.

Die Klassifikation dient dem DPMA vor allem zur Unterstützung der Recherche, um Patentdokumente wieder aufzufinden, damit für die technischen Offenbarungen in Patentanmeldungen die Patentfähigkeit festgestellt werden kann. Eine korrekte und möglichst feingranulare Klassifizierung der Patentliteratur ist für die Arbeit des Patentprüfers somit entscheidend. Mit dem ständig wachsenden Stand der Technik wird auch die Patentklassifikation fortlaufend erweitert.

Die international harmonisierte Klassifikation IPC unterteilt sich von der obersten zur untersten Hierarchiestufe in Sektionen, Hauptklassen, Unterklassen, Hauptgruppen und Untergruppen. Ein Klassifikationseintrag besteht aus einem IPC-Symbol und einer textuellen Beschreibung [DP18]. Die IPC mit dem Revisionsstand 2018.01 enthält über 74.000 vollständige IPC Symbole: 642 Symbole bis zur Unterklasse und über 8.000 Symbole bis zur Hauptgruppe.

Sektion A: Täglicher Lebensbedarf
Sektion B: Arbeitsverfahren; Transportieren
Sektion C: Chemie; Hüttenwesen
Sektion D: Textilien; Papier
Sektion E: Bauwesen; Erdbohren; Bergbau
Sektion F: Maschinenbau; Beleuchtung; Heizung; Waffen; Sprengen
Sektion G: Physik
Sektion H: Elektrotechnik

Abb. 4: Die acht Fachgebiete (Sektionen)

Die Klassifikation von Patentanmeldungen in die IPC kann durch ein automatisiertes Klassifikationssystem erfolgen [BG11]. Der technische Gegenstand des Textes muss durch diesen Klassifikator möglichst genau erfasst und dann zum richtigen Knoten in die hierarchische Struktur der IPC klassifiziert werden.

Die möglichen Einsatzgebiete im DPMA werden im Wesentlichen durch folgende Anwendungsfälle beschrieben:

Elektronische Vorklassifikation: Die Verteilung der Patent- und Gebrauchsmusteranmeldungen im DPMA erfolgt anhand der vergebenen IPC-Klassifikation. Diese Vorklassifikation ist der Schritt im Patentverfahren, bei dem bereits möglichst genau die IPC-Stelle und somit das genaue Themengebiet der Anmeldung ermittelt werden soll. Zudem können weitere Nebenklassen der IPC falls notwendig vergeben werden. Die manuelle Vorklassifikation ist intellektuell sehr aufwändig. Der Inhalt des Anmeldungstextes muss von der klassifizierenden Person erfasst und sehr gut verstanden werden. Weiterhin muss eine große Vertrautheit mit der IPC-Klassifikation gegeben sein, um einen möglichst genauen Klassifikationsvorschlag zu liefern.

Der Schritt der Vorklassifikation kann durch einen automatischen Klassifikator unterstützt werden. Dazu muss die Anmeldung in elektronisch verarbeitbarer Form vorliegen. Im Jahre 2011 wurden die bisher an die Papierform gebundenen Patentakten durch ein vollelektronisches Verfahren ersetzt. Damit werden sämtliche in Papierform eingehenden Patentanmeldungen vollständig digitalisiert. Weiterhin können Patentanmeldungen über die webbasierte Plattform DPMAdirekt in elektronischer Form eingereicht werden. Diese Grundlage ermöglicht es einem automatischen Klassifikator, die in elektronischer Form vorliegenden Texte zu analysieren und Klassifikationsvorschläge zu erstellen. Anhand des IPC-Vorschlags wird die neu angelegte elektronische Patentakte automatisch an einen Patentprüfer weitergeleitet, welcher den IPC-Vorschlag prüft.

Interaktive Klassifikation: Ein weiteres Einsatzgebiet eines elektronischen Klassifikators ist die interaktive Klassifikation. Der Klassifikator übernimmt eine Assistenzfunktion und unterstützt die Patentprüfer bei der Vergabe der IPC-Symbole. Hierbei muss der Klassifikator verschiedene Dokumente interaktiv verarbeiten und Klassifikationsvorschläge erstellen. Durch die direkte Interaktion des Prüfers mit dem Klassifikator sollte es auch möglich sein, die Vorschläge auf bestimmte Bereiche der IPC, beispielsweise auf bestimmte Unterklassen, einzuschränken und Rückmeldungen zur Qualität einzugeben.

Umklassifizierung: Die IPC befindet sich in ständiger Überarbeitung. Jährliche Revisionen machen es erforderlich, die bestehende Klassifizierung von Patentdokumenten zu korrigieren. Für diesen Zweck werden bei jeder IPC-Revision Konkordanzlisten herausgegeben, welche ein Umschreiben der Klassifikation erleichtern. Häufig kommt es jedoch vor, dass IPC-Klassen zu umfangreich geworden sind oder sich die bisherige Einteilung als zu grob herausgestellt hat. In diesem Fall werden IPC-Symbole in weitere Untergruppen untergliedert oder teilweise sogar auf unterschiedliche Klassen aufgeteilt. Solche Fälle können nicht mehr regelbasiert umklassifiziert werden, sondern erfordern eine manuelle Neuklassifizierung. Hierfür muss, ähnlich wie bei der Vorklassifikation, der Inhalt des Dokuments von einem Prüfer beurteilt werden, um anschließend eine geeignete neue Klasse vergeben zu können. Zudem kann bei der Patentprüfung auffallen, dass die Anmeldung mehrere technische Aspekte hat, die während der ersten Patentklassifikation übersehen wurden oder tatsächlich anders zu gewichten sind.

Prüfstoffpflege: Als Prüfstoff wird der Gesamtbestand des im DPMA archivierten Standes der Technik bezeichnet. Dazu zählen sowohl sämtliche nationale Patent- und Gebrauchsmusterschriften, als auch internationale Patentdokumente von anderen Patentämtern. Gepflegt und verwaltet werden diese Dokumente in elektronischer Form im Deutschen Patentinformationssystem (DEPATIS). Die Informationen werden der Öffentlichkeit über den Onlinedienst DEPATISnet zur Verfügung gestellt. Die Qualität des Prüfstoffs mit seinen Volltexten und bibliographischen Daten ist sowohl für das DPMA selbst, als auch für seine Anmelderschaft und Informationskunden von großer Bedeutung. Die Prüfer müssen im Rahmen ihrer Tätigkeit den Prüfstoff aus deutschen, europäischen, aber auch asiatischen und amerikanischen Patent- und Offenlegungsschriften sichten, eingruppieren und können gegebenenfalls eine Änderung der IPC-Hauptklasse vornehmen. Die beschriebene Prüfstoffpflege ist eine kontinuierliche Aufgabe, die von jedem Prüfer Sorgfalt und einen gewissen Zeitaufwand fordert, da er jedes zu verifizierende Dokument intellektuell durchdringen muss, um die korrekte Einordnung in die IPC vornehmen zu können. Hier kann ein automatischer Klassifikator eine unterstützende Funktion zur Validierung, Ergänzung oder Umklassifikation der bestehenden Klassifizierung bieten.

Weitere Anwendungsfälle: Einige ausländische Ämter klassifizieren nicht nach IPC. Hier ist eine Neuklassifikation nach IPC notwendig, ähnlich dem Anwendungsfall der interaktiven Klassifikation. Da die internationalen Schriften in unterschiedlichen Sprachen abgefasst sind, ist es erforderlich, dass der Klassifikator auch nach Fremdsprachen, wie Englisch oder Französisch, klassifizieren kann. Neben der Klassifikation nach dem IPC-Schema kann der Prüfer die Dokumente auch in dem von der IPC abgeleiteten erweiterten Klassifikationsschema des DPMA (DEKLA) einordnen, welches am DPMA für die Suche im Prüfstoff verwendet wird. Zusätzlich zur Vergabe eines IPC-Symbols für eine deutsche Patentanmeldung sollte deshalb als eine weitere Funktionalität eines automatischen Klassifikators auch ein DEKLA-Symbol vergeben werden können.

Die beschriebenen Anwendungsfälle erfordern einen Text-Klassifikator, der die IPC-Klasse einer gegebenen Patentanmeldung oder -schrift vorhersagen kann. Verglichen mit anderen Klassifikationsproblemen, gibt es hier besondere Herausforderungen:

1. Der Wortschatz der Patentliteratur ist auf Grund seiner technischen Natur sehr umfangreich und mehrdeutig.
2. Potentiell sind sehr viele Kategorien zu unterscheiden. Diese sind hierarchisch angeordnet, so dass sie thematisch überlappen und sich in tieferen Ebenen nur noch durch spezifische Teilaspekte unterscheiden.
3. Die unterschiedliche Relevanz von Themengebieten im Stand der Technik spiegelt sich in der IPC-Hierarchie wieder. Die Patentschriften sind daher ungleich über die

Kategorien verteilt. So gibt es Kategorien, denen nur ein Patent zugeteilt ist, während andere tausende Patente umfassen.

Der folgende Auszug aus der IPC-Hierarchie zeigt, wie die thematische Unterscheidung beim Abstieg in den Baum zunehmend feiner wird und auf unterster Ebene meist nur noch begriffliche Feinheiten eine Rolle spielen (z. B. Vorderrad, Hinterrad).

B: Sektion B Arbeitsverfahren; Transportieren
B62: Gleislose Landfahrzeuge
B62K: Fahrräder; Motorräder; Rahmen; Lenkvorrichtungen;
 vom Fahrer betätigte Steuerungsvorrichtungen; Achsaufhängungen;
B62K 25/00: Achsaufhängungen
B62K 25/04: . . zum Anbau von Achsen federnd am Fahrradrahmen oder an der -gabel
B62K 25/06: . . mit Teleskopgabel, z.B. einschließlich schwingender Hilfsarme
B62K 25/08: . . . für das Vorderrad
B62K 25/10: . . . für das Hinterrad

Abb. 5: Auszug aus der IPC-Hierarchie

Unser Klassifikationssystem beruht auf dem `fastText Supervised`-Verfahren [Jo16]. Es handelt sich dabei um eine Abwandlung des `Word2Vec CBOW`-Modells. Während `Word2Vec` ein unüberwachtes Lernverfahren ist, das nur Text als Eingabe erhält, ist `fastText Supervised` ein überwachtes Verfahren, das auf Sätzen und den zugehörigen Kategorien trainiert wird. Bei `CBOW` besteht die Lernaufgabe in der Vorhersage eines Zielwortes unter Berücksichtigung der umgebenden Wörter. Bei `fastText Supervised` hingegen wird nicht ein Zielwort, sondern die Kategorie zu einem gegebenen Satz vorhergesagt.

Das Klassifikationssystem adressiert die gegebene Aufgabenstellung in folgender Weise:

1. `Word Embeddings` reduzieren die Dimension der für die Klassifikation benötigten Features um Größenordnungen im Vergleich zum *Term Vector Space Model*, das klassische Klassifikationsalgorithmen verwendet.
2. Mit dem *Negative Sampling*-Trick ([Mi13], [Jo16]) wird die Zeit-Komplexität reduziert und bleibt mit zunehmender Anzahl an Klassen konstant. *Negative Sampling* ist eine Annäherung der *Softmax*-Funktion. Bei jedem Trainingsschritt wird nur eine zufällige Auswahl an Negativ-Beispielen (Kategorien) betrachtet.
3. Die Trainingsdokumente werden über die IPC-Hierarchie ausbalanciert. Dabei werden die IPC-Symbole den Trainingsdokumenten dynamisch zugewiesen: Je mehr Dokumente in einen Zweig des IPC-Baums fallen, desto tiefer wird in den Baum abgestiegen. Der Klassifikator betrachtet die Hierarchie nicht. Die Dokumente werden flach klassifiziert, da sich eine hierarchische Klassifikation für die IPC nicht bewährt hat [Ba13].

3.1 Evaluation

Für die Klassifikation werden ca. 300.000 deutsche Patent- und Offenlegungsschriften der Jahrgänge 2010 - 2015 verwendet. Es werden nur der Titel und die Textfelder (Zusammenfassung, Ansprüche, Beschreibung) der Schriften herangezogen. Die Dokumente wurden in ein Trainings- und Testset unterteilt (90/10). Kategorien mit wenigen Beispieldokumenten wurden von der Klassifikation ausgeschlossen.

Wir vergleichen *fastText Supervised* mit einer klassischen, linearen *Support Vector Machine (SVM)*. Für den SVM-Algorithmus wird die Implementierung von *liblinear* [Fa08] verwendet. Zusätzlich wird der für Anwendungsfälle dieser Größenordnung ausgelegte lineare Lernalgorithmus von *Vowpal Wabbit* [La07] evaluiert. Bei der Evaluation wird auch die Trainingszeit bewertet, da Performance für die Umsetzung der Patentklassifikation ein maßgeblicher Faktor ist.

3.1.1 Klassifizierung auf Unterklassen-Ebene

Ziel der Klassifikation ist es, jeder Schrift die jeweils korrekte Unterklasse aus **584** Unterklassen zuzuweisen.

Verfahren	Accuracy @1	Accuracy @3	Trainingszeit	Laufzeit
SVM	70,0	89,5	1h 5m	18s
Vowpal Wabbit	69,0	86,8	25m	5m
<i>fastText Supervised</i>	68,5	87,1	17m	38s

Tab. 4: Ergebnisse der Klassifikation auf Unterklassen-Ebene

Der Vergleich zeigt, dass *fastText Supervised* und *Vowpal Wabbit* etwas schlechter abschneiden als die SVM. Die Unterschiede im Ressourcenbedarf sind jedoch signifikant. Im Gegensatz zu *fastText Supervised* sind Trainingszeit und Speicherauslastung bei der SVM wesentlich höher.

3.1.2 Klassifizierung mit dynamischer Verteilung

Eine Klassifikation bis zur Unterklassen-Ebene der IPC reicht für die Vorklassifikation beim DPMA nicht aus. Je tiefer in die IPC-Hierarchie abgestiegen wird, desto besser kann eine Anmeldung dem richtigen Prüfer zugeordnet werden. Wünschenswert wäre daher eine feinere Klassifizierung nach Haupt- oder Untergruppe. Es erfolgt eine Ausbalancierung der Kategoriezuteilung im Trainingsset, dies führt zu einer Unterscheidung von **4.448** Kategorien (Unterklassen, Haupt- und Untergruppen).

Verfahren	Accuracy @1	Accuracy @3	Trainingszeit	Laufzeit
SVM	-	-	-	-
Vowpal Wabbit	41,4	58,6	6h 31m	24m
fastText Supervised	68,2	83,8	17m	1m 10s

Tab. 5: Ergebnisse der Klassifikation mit dynamischer Verteilung

Mit steigender Anzahl von Kategorien kommen die Vorteile von *fastText Supervised* deutlich zum Tragen. Das Hinzunehmen von Klassen wirkt sich auf Grund von Negative Sampling nicht nachteilig auf Speicher- und Zeitkomplexität aus. Die Genauigkeit fällt nur leicht ab. Für die *liblinear*-Implementierung der SVM ist die Komplexität des Problems nicht mehr beherrschbar. *Vowpal Wabbit* benötigt im Vergleich zur Klassifikation von Unterklassen mehr Zeit, um zu konvergieren und fällt in der Genauigkeit deutlich ab.

Der *fastText Supervised*-Algorithmus skaliert hervorragend und eröffnet dadurch die Möglichkeit, mit vielen Kategorien und auf großen Datenbeständen schnell zu trainieren.

4 Ausblick

Es wurde gezeigt, dass kognitive Verfahren mächtige Instrumente liefern, um die Patentrecherche und die Patentklassifikation effektiver und effizienter zu gestalten.

Naturgemäß geht bei unscharfen, assoziativen Verfahren die Exaktheit verloren. So findet zum Beispiel eine Suche nach *Fahrrad* auch Dokumente zu *Kleinfahrzeugen* oder *Motorrädern*. Je nach Anwendungsfall kann dieser Effekt gewollt oder ungewollt sein. Im hochdimensionalen Raum sind viele verschiedene syntaktische und semantische Aspekte kodiert, die Grenzen zwischen ähnlich und unähnlich fließend. In diesem Zusammenhang muss über alternative Darstellungsformen und neue Ausdrucksmöglichkeiten für den Nutzer, zum Beispiel durch interaktive graphische Benutzeroberflächen, nachgedacht werden. Durch eine Interaktion mit dem Nutzer wird seine Rechercheintention deutlich und er kann mit seiner Fachkompetenz die Vorschläge des Systems optimieren. Hierin liegt der Schwerpunkt künftiger Weiterentwicklungen.

Literatur

- [ALM17] Arora, S.; Liang, Y.; Ma, T.: A simple but tough-to-beat baseline for sentence embeddings, 2017, URL: <https://openreview.net/pdf?id=SyK00v5xx>, Stand: 17. 10. 2017.
- [Ba13] Babbar, R.; Partalas, I.; Gaussier, E.; Amini, M. R.: On Flat versus Hierarchical Classification in Large-Scale Taxonomies. In (Burgess, C. J. C.; Bottou, L.; Welling, M.; Ghahramani, Z.; Weinberger, K. Q., Hrsg.): Advances in Neural Information Processing Systems 26. Curran Associates, Inc., S. 1824–1832, 2013, URL: <http://papers.nips.cc/paper/5082-on-flat-versus-hierarchical-classification-in-large-scale-taxonomies.pdf>.

- [BG11] Benzineb, K.; Guyot, J.: Automated Patent Classification. In: Current Challenges in Patent Information Retrieval. Springer-Verlag, Berlin Heidelberg, 2011.
- [Co17] Conneau, A.; Kiela, D.; Schwenk, H.; Barrault, L.; Bordes, A.: Supervised Learning of Universal Sentence Representations from Natural Language Inference Data, 2017, URL: <https://arxiv.org/abs/1705.02364v5>, Stand: 08. 07. 2018.
- [Co18] Conneau, A.; Lample, G.; Ranzato, M.; Denoyer, L.; Jégou, H.: Word Translation Without Parallel Data, 2018, URL: <https://arxiv.org/abs/1710.04087v3>.
- [DP18] DPMA: Internationale Patentklassifikation, Handbuch zur IPC Ausgabe 2018. Deutsches Patent- und Markenamt, München, 2018.
- [Fa08] Fan, R.-E.; Chang, K.-W.; Hsieh, C.-J.; Wang, X.-R.; Lin, C.-J.: LIBLINEAR: A Library for Large Linear Classification. J. Mach. Learn. Res. 9/, S. 1871–1874, Juni 2008, ISSN: 1532-4435, URL: <http://dl.acm.org/citation.cfm?id=1390681.1442794>.
- [HCK16] Hill, F.; Cho, K.; Korhonen, A.: Learning Distributed Representations of Sentences from Unlabelled Data, 2016, URL: <https://arxiv.org/abs/1602.03483v1>, Stand: 10. 02. 2016.
- [Jo16] Joulin, A.; Grave, E.; Bojanowski, P.; Mikolov, T.: Bag of Tricks for Efficient Text Classification, 2016, URL: <https://arxiv.org/abs/1607.01759v3>, Stand: 09. 08. 2016.
- [Ki15] Kiros, R.; Zhu, Y.; Salakhutdinov, R.; Zemel, R. S.; Torralba, A.; Urtasun, R.; Fidler, S.: Skip-Thought Vectors, 2015, URL: <https://arxiv.org/abs/1506.06726v1>, Stand: 22. 06. 2015.
- [Ko16] Korger, C.: Clustering of Distributed Word Representations and its Applicability for Enterprise Search, Diploma Thesis, Dresden University of Technology, 2016.
- [La07] Langford, J. et al.: Vowpal Wabbit, 2007, URL: <http://hunch.net/?p=309>.
- [Mi13] Mikolov, T.; Chen, K.; Corrado, G.; Dean, J.: Efficient Estimation of Word Representations in Vector Space, 2013, URL: <https://arxiv.org/abs/1301.3781v3>, Stand: 07. 09. 2013.
- [PGJ17] Pagliardini, M.; Gupta, P.; Jaggi, M.: Unsupervised Learning of Sentence Embeddings using Compositional n-Gram Features, 2017, URL: <https://arxiv.org/abs/1703.02507v2>, Stand: 17. 10. 2017.

Database Systems in the Cloud

IBM Cloud Databases: Turning Open Source Databases Into Cloud Services

Tim Waizenegger¹ Thomas Lumpp²

1 Abstract

Databases in all their forms are the backbone of most applications, running in the Cloud or on-premise. This creates a large demand for hosted, as-a-service database systems that are used either by Cloud applications, or even by on-premise applications. Through this demand, two types of offerings were created: new Cloud-native multi-tenant database systems and hosted instances of existing database systems. The first of those types, new cloud-native systems, may even have relational and ACID properties. On our IBM Cloud platform, we offer both types: “SQL Query” and “Cloudant” are our Cloud-native multi-tenant database systems. To cover the second type from above, “IBM Cloud Databases” offers popular Open Source databases like PostgreSQL, MongoDB or Redis as a Cloud service.

The IBM Cloud Databases offering includes database provisioning, maintenance and operations, backup and restore, scaling, and other features. As a newly built service, it has unique properties that make it stand out. IBM Cloud Databases is built as a native Kubernetes application that runs containerized versions of Open Source databases. Databases are provisioned as clusters or master/slave configurations, depending on the database, to offer high availability. To further decrease downtime for customers, we implemented a zero-downtime scaling feature for vertical memory scaling. This allows customers to save cost on memory when demand is low, but quickly scale-up without restarting the database when demand increases.

In this presentation, we give an overview of the system architecture and show how we integrated the different database systems. We discuss the advantages and disadvantages of running such a service on Kubernetes and provide insight into how we operate this service.

¹ IBM Deutschland Research & Development GmbH, Böblingen, tim.waizenegger3@ibm.com

² IBM Deutschland Research & Development GmbH, Böblingen, thomas.lumpp@de.ibm.com

Fighting Spam in Dating Apps

Uwe Jugel¹, Juan De Dios Santos², Evelyn Trautmann³, Diogo Behrens⁴

Abstract: Online dating allows for interactions between users with a high degree of connectivity. This digital form of interaction attracts spammers and scammers, who try to trick users to visit low-class competitors' websites or steal the users' money. Fortunately, each attacker leaves a footprint of its actions in the network. It is the task of an Anti-Spam system to detect these and punish the culprit.

In this paper, we demonstrate how LOVOO fights such illegal activities using a system of modular, scalable, and fault-tolerant Anti-Spam components. We describe our stream-processing architecture and how it ensures flexibility and facilitates resource-efficient processing of the millions of events produced by hundreds of thousands of users.

1 Introduction

Dating requires interaction and online dating allows interactions from one to many users and vice versa. On a dating platform, the participants belong to a very specific focus group: people looking for love. After a dating platform is established and has created a large user base, it becomes the perfect target for spammers and scammers to trick innocent users.

Our mobile dating app **LOVOO** allows our users to send *matches*, *likes*, *visits*, *smiles*, and *chat messages* from and to hundreds of thousands of users; 24 hours per day. Consequently, the LOVOO backend is a data-intensive application with near real-time requirements and so is our Anti-Spam system. It needs to oversee large volumes of various kinds of user interactions and detect and punish spammers, scammers, fraudsters, and misbehaving users in real-time.

For this task, the Anti-Spam system needs to be *modular*, *scalable*, and *fault-tolerant*. The individual Anti-Spam components need to collect user state and user history of various kinds to setup and improve machine learning models used to decide if an observed user behavior is regarded as spam or not. When using trained models for spam prediction, the components need live access to the acquired user state and history.

Moreover, training and adjustment of machine learning models must be quick and easy. All training data should be stored and be instantly accessible in a central data warehouse

¹ LOVOO GmbH, uwe.jugel@lovoo.com

² LOVOO GmbH, juan.dediossantos@lovoo.com

³ LOVOO GmbH, evelyn.trautmann@lovoo.com

⁴ LOVOO GmbH, Github: [@db7](https://github.com/db7)

to facilitate automatic training in particular and data analysis and research experiments in general.

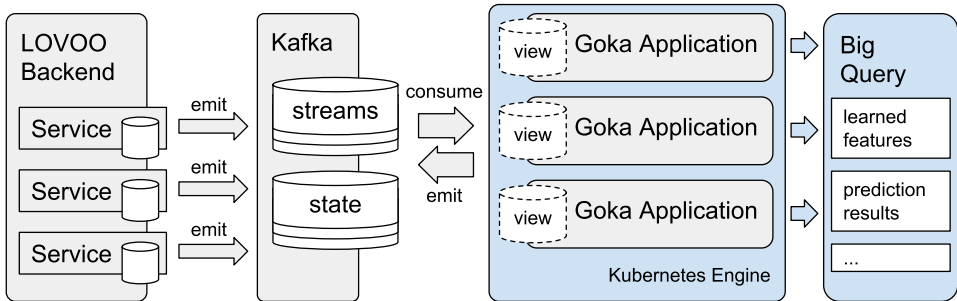


Fig. 1: Architecture Overview: Backend services publish interaction data to Kafka. Independent Goka processors and services asynchronously consume data in tables and emit state changes directly to Kafka. Machine learning features and predictions results are stored in BigQuery.

Paper Overview For fulfilling the aforementioned requirements and implementing the listed features, we combine a variety of cloud products and state-of-the-art technologies as depicted in Figure 1. All our components run in the Google Cloud using a variety of managed products such as Kubernetes Engine and BigQuery (cf. Section 2). We use Apache Kafka as our stream-processing foundation. Since Go is our core programming language for the backend systems, we have built Goka, an open-source library written in Go on top of Kafka that radically simplified our application development (cf. Section 3). We use these technologies to process text, images, user profiles, and user behavior to detect spam in our various Anti-Spam components (cf. Section 4). Eventually, we also evaluate our machine learning models and the quality of the spam-detection in general (cf. Section 5).

2 System Architecture

Our Anti-Spam components need to oversee any form of user interaction in near real-time and without slowing down the core services of the LOVOO backend. Therefore, all communication to and inside the Anti-Spam system is event-driven and thus asynchronous. Similarly, to achieve modularity, the individual Anti-Spam components should not block each other and work asynchronously. Figure 2 shows our system architecture.

Our Anti-Spam system consumes events from Apache Kafka topics. We refer to event topics as *streams*. Kafka is also used to store any kind of spam-related state about the users and devices connected to our platform. State is stored in log-compacted topics in Kafka, which we refer to as *tables*.

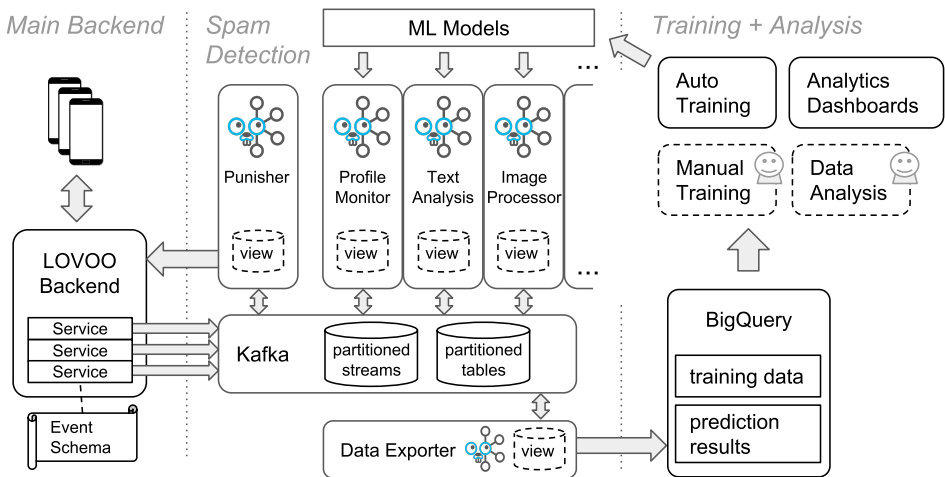


Fig. 2: LOOVO Anti-Spam Architecture.

The schema of events and state is defined with **Protocol Buffers** and shared in source code repositories. Any kind of service supporting Protocol Buffers and Kafka drivers can be used to publish or consume events and state updates from or to Kafka.

The majority of Anti-Spam components employ **Goka** to consume and update streams and tables. Goka is a stream-processing library written in Go that helps us building highly scalable applications with Kafka (cf. Section 3). Each component processes incoming events using rule sets, manually trained models, automatically trained models, or additional cloud services to learn about and detect spam in texts, images, user profiles, and user behavior. Learned data and prediction results are exported to **BigQuery**, i.e., our data warehouse in the Google Cloud. The warehouse data is used for data analysis and for training our machine learning models. The new models are tested against the current models in the individual spam-detection components to eventually replace the old models after a successful evaluation.

The Anti-Spam components (cf. Sections 4) are completely independent of each other and communicate exclusively via Kafka. For instance, our Image Processor extracts text from images and emits the obtained text back to Kafka, from where our Text Analyzers consume and analyze the text. If they detect spam in the text, they emit their findings back to Kafka, from where our Punisher component picks them up and applies some final safety measures before telling the LOOVO Backend to actually punish a user or device.

All time-critical components are written in Go using our Goka library that handles issues of scalability, local persistence, fault-tolerance, monitoring, data parallelism, and data consistency in Kafka.

For training our machine learning systems, we use [scikit-learn](#), [Keras](#) and [TensorFlow](#). The trained models are transferred to the time-critical components by dumping the learned coefficients and intercept value from scikit-learn and recreating the model in a Go data structure. The models trained on Keras and TensorFlow are hosted on Cloud ML Engine.

3 Goka: Go Stream Processing with Kafka

Our first version of Anti-Spam was built using a combination of event queues, off-the-shelf databases, and a caching layer. The system was hard to scale, caches needed to be carefully invalidated, and a lot of different technologies needed to be known and touched during each development cycle.

In the search for a cleaner and more scalable architecture, we redesigned the Anti-Spam system as a set of stateful stream processors, following the rationale of „I heart logs“ [Kr14] by Jay Kreps and „Making sense of stream processing“ [Kl16] by Martin Kleppmann.

To support this redesign, we developed and open-sourced Goka, a stream processing library written in Go that exploits several features of Apache Kafka, radically simplifying the implementation of applications. In fact, the new architecture turned out to be so flexible and performant that a number of our systems have been re-implemented with Goka. From user search to machine learning, Goka now powers several applications that handle large volume of data and have near real-time requirements.

The main features of Goka are the following.

Stream Processing Model. Goka allows us to easily define Kafka stream processors in our Go code, providing abstractions for setting up Kafka consumer groups, processor inputs, outputs, and state. Goka internally automates any handling of Kafka partitions.

Local Persistence. Goka provides a local storage model for caching compacted topics locally in [LevelDB](#), for near real-time lookups.

Monitoring. Goka provides built-in metrics to monitor the processing state of various kinds of data consumers.

We now briefly describe the Goka concepts required to understand the remainder of this paper. For more details, please see our article on the LOVOO engineering tech blog [Be17].

Tables and Processors. Most of our applications employ one or more key-value *tables* to store the application state. The goal of Goka is to allow for manipulation of such tables in a composable, scalable, and fault-tolerant manner.

Each table is owned by one single component, we call it *processor*. The processor consumes input streams and produces state changes. State changes can only be applied to a table by the processor that owns the table. A single table owner is essential to achieve key-wise sequential updates, eliminating the need for complex and error-prone synchronization of multiple writers.

Each table is stored in a log-compacted Kafka topic as well as in a local LevelDB storage. The combination allows for fast reads and writes. Reads access the local storage, not requiring any network hop. Writes are applied locally and remotely, but remote writes occur asynchronously to the computation.

Keeping the table stored in Kafka provides not only fault tolerance by default, but also allows for easy processor migration by recovering the local storage from the Kafka table.

Views. A table stored in Kafka has a single owner processor that writes to it, but other processors may also read the table using views. A *view* is a local storage that continuously consumes state updates from a table in Kafka. Views are eventually consistent and allow processors to join input streams with tables to perform updates to their own tables or to emit new events into output streams.

Messages and Partitions. Messages are key-valued. Every topic in Kafka is partitioned, and messages are routed to partitions by hashing the message key. If input streams and tables are co-partitioned, i.e., they have the same number of partitions and the keys have the same domain, then the processors can be split into multiple instances, each owning one or more partitions of the table.

The Kafka consumer-group protocol assigns partitions to consumers. Goka exploits the same mechanism to assign related stream and table partitions to processor instances. For instance, consider a processor owning a table T and consuming two input streams A and B . The streams and the table each have, e.g., two partitions. When starting two processor instances, partition 0 of A , B , and T is assigned to one processor instance, whereas partition 1 is assigned to the other. Since keys are routed to partitions using a global hash function, partition 0 of A , B , and T contain the same key subdomain, which is disjunct from the other key subdomain of partition 1.

Scaling a Goka processor consists in simply starting multiple instances. When a new instance connects to Kafka, Kafka reassigns the partitions and distributes the plan to the instances. Once the reassigned table partitions are fetched from Kafka and stored locally in LevelDB, the instance starts processing the input streams from where they stopped before the reassignment.

4 Anti-Spam Processors

In Sections 2 and 3 we described how to set up a modular and scalable Anti-Spam architecture using Kafka as its backbone (also cf. Figure 2).

However, independent of our use of Kafka and Goka for stream processing, we designed our Anti-Spam system as a lambda architecture that supports batch and streaming processing alike. The individual Anti-Spam components can make use of any kind of external service required to effectively perform their job. In particular, we combine Goka-based stream processing with batch processing using, e.g., Google BigQuery and **Cloud Vision**.

This design has proven to be highly beneficial for all the different problems that Anti-Spam seeks to solve when detecting and stopping the proliferation of fake or disrupting profiles.

Each Anti-Spam component employs an algorithm that is particular to its domain and that detects spammers in a way distinct from the others. In the following, we describe our core spam-detection components and how they leverage Kafka and Goka, and make efficient use of external service in the Google Cloud. These components are the following.

<i>Spotter</i>	detects text in images.
<i>Photographer</i>	extracts text from images.
<i>Terminator</i>	classifies text based on rules.
<i>Rosetta</i>	classifies text using a machine learning model.
<i>Sheriff</i>	oversees user behavior.
<i>Tracer</i>	detects sequential interaction patterns.
<i>Entity Reputation</i>	detects spam based on the reputation of specific entities.

4.1 Spotter and Photographer

One of the means spammers use for distributing spam on our platform are images. The spammer writes a message, e.g., an advertisement for a pornographic or low-class competitor website on top of the image and uploads the image to LOVOO. Detecting this text requires a combination of computer vision and machine learning to detect and extract the text and subsequently analyze it to determine the adequacy of the content.

The text detection component is called Spotter and is written in Python. It consumes `picture_uploaded` messages from Kafka that are sent every time a picture is uploaded on LOVOO. The message contains the location of the user image in Cloud Storage and metadata about the image and the upload event. For every incoming message, Spotter retrieves the image from Cloud Storage, and, using **OpenCV**, it will apply a technique known as Class-specific Extremal Region [NM12] to detect the regions of the image that contain text. If Spotter detects a text region, a message is emitted to another Kafka topic that is read by the Photographer.

The text extraction component is called Photographer and is a Goka processor. It consumes messages from the aforementioned text detection topic populated by Spotter. Each message again contains the storage location of a now suspicious image. The image is retrieved a second time from Cloud Storage and sent to the Google [Cloud Vision API](#) to extract the text. The extracted text is emitted as another message to Kafka and picked up by the Terminator that verifies if the text violates any rules and thresholds.

Splitting up the task of image processing into a low-cost but less versatile text detection (OpenCV) and a more versatile but also more expensive text extraction step (Cloud Vision) saved us a lot of costs. Google charges for every image uploaded and less than 1% of our images actually contain text.

4.2 Sheriff

In comparison to most spammers, our regular users behave in particular ways. For example, a female user who is 32 years old, usually interacts with the app in a different way than a 21 years old male user. Moreover, spammers usually try to reach a broad audience very quickly and thus use the app in very uncommon ways. We track and oversee this user behavior in our Sheriff, where we employ machine learning to distinguish regular user behavior from the malign behavior of spammers.

The Sheriff is a Goka processor that keeps track of the frequency of the actions executed by our users, as well as individual user characteristics, such as gender, age, and their search settings, i.e., the gender and age of the users they are looking for at LOVOO.

The Sheriff uses a machine learning model trained using logistic regression, an algorithm that uses a logistic function for modeling the relationship between a dependent variable and a set of independent or predictors variables.

The model training is done in Python using scikit-learn. The training dataset contains over 1 million observations of over 30 features. These features comprise the user characteristics mentioned above and the relative execution ratios of particular user actions. For instance, given the three types of user actions `message_sent`, `like_created`, and `message_received`, and absolute executions of 3 sent messages, 7 likes, 2 received messages, the features values for the user actions are 0.25, 0.58, and 0.17 respectively. To train the model, we use k -fold cross-validation with $k = 5$, and to find an optimal set of hyperparameters, we fit the model using grid search [BB12] in the following search space:

```
alpha:      0.0, 10-2, 10-3, 10-4, 10-5, 10-6
penalty:    L1, L2, elastic net [ZH05]
max_iter:   500, 1000
```


4.3 Tracer

The Tracer is similar to the Sheriff, as it also consumes user actions to analyze user behavior. But instead of tracking execution ratios, it collects and evaluates the order in which the user executed the actions.

The Tracer processor consumes the most important user action topics from Kafka and encodes the topic name and the time span between subsequent events using a sequence of characters, e.g., as follows.

- 0: Marks the start of the sequence.
- S: User has started the app and is logged in.
- 4: User is idle for 4 seconds.
- L: User created a like on a another users' profile.

The sequence not only includes the user's active actions, but also passive events, i.e., other users' actions that affect the current user. For instance, for every created like there is a user who received the like.

In Goka, processing passive events may require reading and writing data in other partitions. This is done using Goka's loopback API. A loopback is an operation that emits a message back to the current processor, but with a different key. The message is then picked up by the correct processor instance to update the correct partition of the corresponding table.

In the Tracer, the loopback is used to update another user's action sequence. When consuming a `like_created` message it has as key the current user and one of its values is the liked user's key. This key is used for the loopback to send a message to the correct processor instance and update the correct sequence.

The Tracer model is an recurrent neural network (RNN) [Ho82] based on long short-term memory cells (LSTM) [HS97] trained on Keras and TensorFlow. Our Tracer network consists of three layers. The first one is an embedding layer, with an input dimension of 10, output dimension of 32 and maximum length of 1000. The second layer is an LSTM layer of 100 units. In the last layer, we classify spammers vs. non-spammers, using a dense layer with a single unit and a sigmoid activation function. The model also uses binary cross entropy loss and an Adam optimizer [KB14].

4.4 Rosetta

Rosetta is the component of Anti-Spam that deals with text-related features. In the LOVOO app, there are two text fields that the user can fill with freetext, the username and the "about me" section. Spammers use these parts of the app as another medium to advertise and spread their content. Thus, we trained various logistic regression models in scikit-learn

using usernames and "about me" data to classify them into spam content and non-spam content.

For the username model, we used around 13000 usernames, of which 60% were labeled as non-spammer and the remainder 40% as spam. For the training, we process all text as lowercase bigrams on character level. Then we produced our feature vectors by using a bag of words approach. Like the Sheriff model, we found the best set of hyperparameters using grid search and trained using k -fold cross-validation with $k = 5$. The scoring metric was the $F1$ score.

Because LOVOO supports a diverse number of languages, for the "about me" problem, we fit several models, each for a particular language. The size of the training datasets and pipelines are similar across all languages. As training data we used around 40000 records with a spam to non-spam message ratio of 1 : 1.5. The training pipeline is similar to the one used for the usernames, except that in this case, we remove stop words from the text, before creating the bag of words. Afterwards, we fit a logistic regression model using grid search and 5-fold cross-validation; the scoring metric used is the accuracy.

4.5 Entity Reputation

The Entity Reputation component determines scores for the reputation of various users characteristics. These so-called entities are, e.g., the user's IP address, email domain, and uploaded images. If many users share an entity, e.g., the same image, and some of the users have been already flagged as spammers, the reputation score of that entity is decreased.

The component is a Goka processor that maintains the scores for such entities in a table. It consumes all topics from Kafka that contain updates for these entities.

The spam detection itself is a threshold-based geometrical separation between spammers and non-spammers in terms of user counts and share of received reports and other detected spammers associated with this entity. The scorer exploits that spammers in contrast to individual users share more entities just by being not an individual person. The ensemble of entities describes a spam probability.

To achieve a clear separation between scores of spammers and non-spammers, we model the spam-related features, such as the number of user-received spam reports and number of flagged spammers for that entity, as a sigmoid function per entity.

$$\frac{1}{1 + \exp(-\beta(x - \lambda))} \in [0, 1]$$

The sigmoid functions range between 0 and 1 with a scaling factor β and a separation threshold λ . The separation threshold is obtained from our historical data. The ensemble of single entity scores contributes to a total reputation score given to every user. If a predefined score is reached, the user has been identified as a spammer.

Scoring entities with sigmoid scores resemble a logistic regression model. However, since the various entities have very different significance and reliability, we cannot score them with the same model. Instead, we fit the sigmoid function to every single scalar x per entity.

5 Evaluation and Results

We publish the progress and results of our spam-fighting activities at tech.lovoo.com in the form of regular transparency reports. In the following, we summarize the results published in our June 2018 edition of the report.

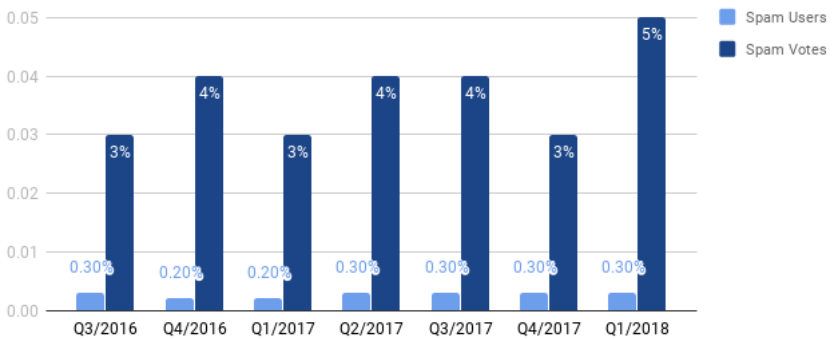


Fig. 3: Spam activity at LOVVO

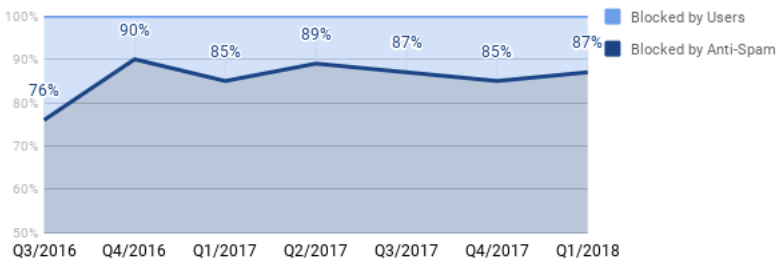


Fig. 4: Automatically detected vs. user-reported spam at LOVVO.

As shown in Figure 3, we currently (as of June 2018) have about 0.3% registered LOVVO profiles that are spammers and these spammers are responsible for 5% of the likes and dislikes, i.e., the votes on our platform, which is one of the main forms of digital interaction provided by the app. Figure 4 shows that of those spammers and scammers, over 87% are automatically detected and blocked by our Anti-Spam system. The other 13% are reported by other LOVVO users.

We are aware that these numbers alone are not the perfect measure of the quality of our spam detection. However, they show that our Anti-Spam system was able to maintain its

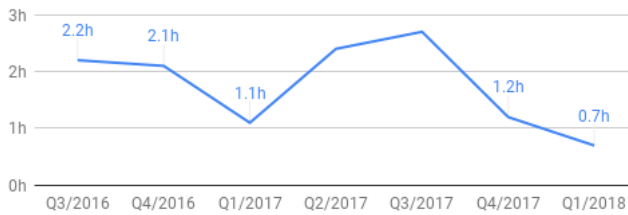


Fig. 5: Average hours before a spammer is blocked.

good spam-detection ratios over the last years, even though spammers and scammers are always coming up with new ways to work around our system and to trick our users.

Another indicator of the quality of a spam detection system is the duration before a spammer is blocked, as shown in Figure 5. Currently, we are able to block spammers in less than 42 minutes after they become active on the platform, with a significant improvement over previous quarters.

While the above numbers provide us with a very general metric how our Anti-Spam system performs, we also need to look at the details to see if our specific machine learning models are working as expected.

This validation of new detection models is one of our biggest challenges. Simple cross validation with historical data will always struggle with newly detected spam patterns, since the positive findings of the new models are not recognized by the former models and thus cannot contribute to the datasets used for validation.

We actually want to know two key metrics about the new models.

1. Does the new model detect spammers earlier?
2. Does the new model detect new spam patterns?

While the first question can be answered by cross validation with historical data, the second requires different, i.e., external metrics. Some of these external metrics are, e.g., the number of user-reported spam profiles, the number of requested user verifications, or the number and type of unblocks, i.e., when automatically or manually marking a blocked user as regular users again and no longer as spammer. Figure 6 depicts the trend for these unblocks on our platform, showing the number of unblocks over time for five different (undisclosed) block categories. An increase or decrease in a specific category tells us how well specific parts of the Anti-Spam system, and thus the new detection models, are performing.

In total, the number of unblocks of falsely detected users is decreasing, which suggests that our models and rules are actually blocking the correct profiles and less of our regular users.

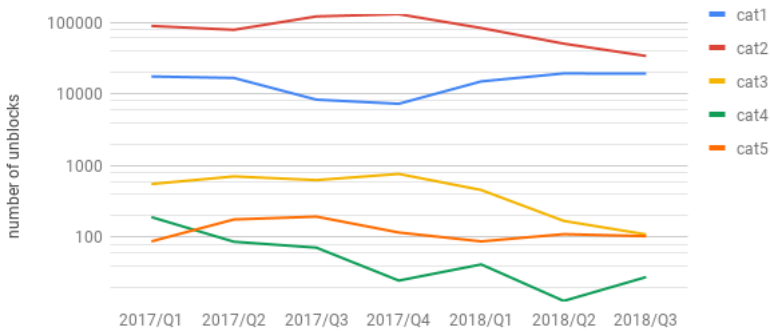


Fig. 6: Number of unblocks of falsely reported or detected user profiles in different (undisclosed) spam categories.

6 Conclusion

In this paper, we described LOVOO's Anti-Spam architecture and how we use stream-processing, batch-processing, and machine learning in production and in the cloud to fight spammers and scammers on our platform. In particular, we described how our open-source Goka framework helped us setting up a stream-processing solution on top of Kafka to create a scalable, fault-tolerant, and modular architecture.

For our main Anti-Spam components, we described how they interact asynchronously via Kafka, which kind of rules and machine learning they apply in which domain, and also how they interact with external services. In this regard, we also provided a real-world example of combining simple and sophisticated machine learning systems to save costs in the cloud.

Spammers are good at adapting their techniques to our systems and are constantly finding new approaches to circumvent our filters. In this regard, malicious content on user images is one of our biggest challenges, which we are trying to win using new self-developed detection models, but also pretrained models provided by machine learning services in the cloud.

References

- [Kr14] Krepes, Jay: I Heart Logs: Event Data, Stream Processing, and Data Integration. O'Reilly Media, 2014.
- [Kl16] Kleppmann, Martin: Making Sense of Stream Processing: The Philosophy Behind Apache Kafka and Scalable Stream Data Platforms". Report. O'Reilly Media, 2016.
- [Be17] Behrens, Diogo: Goka: Go stream processing with Kafka. LOVOO Engineering, LOVOO GmbH, 2017, (online: <https://tech.lovoo.com/2017/05/23/goka>).

- [NM12] Neumann, Lukáš; Matas, Jiří. Real-time scene text localization and recognition. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 281–305, IEEE, 2012.
- [BB12] Bergstra, James; Bengio, Yoshua: Random Search for Hyper-parameter Optimization. *The Journal of Machine Learning Research* 13(1), pp. 281–305, ACM, 2012.
- [ZH05] Zou, Hui; Hastie, Trevor: Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 67(2), pp. 301–320, Wiley Online Library, 2005.
- [HS97] Hochreiter, Sepp; Schmidhuber, Jürgen: Long short-term memory. *Neural Computation* 9(8), pp. 1735–1780, MIT Press, 1997.
- [KB14] Kingma, Diederik P. ; Ba, Jimmy L.: Adam: A Method for Stochastic Optimization. arXiv preprint for ICLR 2015, arXiv:1412.6980, 2014.
- [Ho82] Hopfield, John J.: Neural networks and physical systems with emergent collective computational abilities. In: *Proceedings of the National Academy of Sciences* 79(8), pp. 2554–2558, National Acad Sciences, 1982.

Graphs

Understanding Trolls with Efficient Analytics of Large Graphs in Neo4j

David Allen¹, Amy E. Hodler², Michael Hunger³, Martin Knobloch⁴, William Lyon⁵, Mark Needham⁶, Hannes Voigt⁷

Abstract: Analytics of large graph data set has become an important means of understanding and influencing the world. The use of graph database technology in the International Consortium of Investigative Journalists' (ICIJ) investigation of the Panama Papers and Paradise Papers or in cancer research illustrates how analysing graph-structured data helps to uncover important but hidden relationships. A very current example in that regards shows how graph analytics can help shed light on the operations of social media troll-networks, e.g. on Twitter. In similar fashion, graph analytics can help enterprises to unearth hidden patterns and structures within connected data, to make more accurate predictions and faster decisions. All this requires efficient graph analytics well-integrated with management of graph data.

The Neo4j Graph Platform provides such an environment. It provides transactional processing and analytical processing of graph data including data management and analytics tooling. A central element for graph analytics in the Graph Platform are the Neo4j graph algorithms. Neo4j graph algorithms provide efficiently implemented, parallel versions of common graph algorithms, integrated and optimized for the Neo4j transactional database. In this paper, we will describe the design and integration Neo4j Graph Algorithms, demonstrate its utility of our approach with a Twitter Troll analysis, and show case its performance with a few experiments on large graphs.

Keywords: Graph databases, graph analytics, graph algorithms, property graphs

1 Introduction

While we perceive ourselves as individuals, it is undeniable, that we are also highly-connected to everything around us. This contrast of individuals and connections give rise to two approaches of perceiving and understanding the world. The *entity-centered* approach focuses on the individual, it describes the world by primarily characterizing and specifying individual entities. A classical example of entity-centered is Aristotle's descriptions of

¹ Neo4j, San Mateo, US david.allen@neo4j.com

² Neo4j, San Mateo, US amy.hodler@neo4j.com

³ Neo4j, Dresden, Germany michael.hunger@neo4j.com

⁴ Avantgarde Labs, Dresden, Germany mknobloch@avantgarde-labs.de

⁵ Neo4j, San Mateo, US william.lyon@neo4j.com

⁶ Neo4j, London, UK mark.needham@neo4j.com

⁷ Neo4j, Leipzig, Germany hannes.voigt@neo4j.com

species by their parts such as live birth, number of legs, etc. during his stay on the Island of Lesbos. The *relationship-centered* approach, in contrast, focuses on relationships, it describes the world by primarily characterizing and specifying the relationships between individual entities. For instance, the world of the living can be further understood by looking at relationships among living such as prey–predator and evolutionary relationships.

We are connected to many other individuals, events and things that populate the world around us, near and far. Connectedness manifests in many different ways, facets, values, scopes, and scales. To understand reality or parts of it, we have to understand the connectedness that pervades it, the connectedness that forms its fabric. Because of the multifacet, multiscale nature of connectedness, the precise connections that help our understanding are not necessarily apparent, though. Often the interesting connections are intricately hidden in between many other, different layers of connectedness, or only apparent in transitive paths spanning many intermediate steps. A classical example of this is a conflict-of-interest connection of a person with a process the person has a decision power over. A conflict-of-interest can easily be behind an intricate path of affiliation, proximity, relationship, ownership, coercion, bribery, etc. connections. Other examples are understanding drug-target efficacy in drug design and environmental impact of genetically modified organisms.

Today’s ubiquity of information technology makes increasing amounts of connected data from many different sources accessible within an organization and outside of it. Such data becomes available in various data models. Conceptually the simplest data model of expressing and describing connected information is the property graph model [RN10]. A property graph is relationship-centered, it has relationships as first-class citizens. Entities carry, per instance, arbitrary properties and optional labels to describe their roles. Next to all the entities of consideration, a property graph explicitly supports the connections these entities have and – what sets it apart from other common graph data models – describes connections as rich relationships, with a type and arbitrary properties. A single property graph can capture whole variety of the connectedness existing in the world and makes it easy for humans to reason about very big complicated phenomena and their combined interactions.

Pulling multiple different data source of connections together in a property graph and analysing patterns of connection by means of graph theory and network analysis often reveals hidden relationships and structures, characterizes and quantifies them, and provides new understanding. Two main ingredients to make such an analysis happening are capabilities for managing graph data and run graph analysis algorithms efficiently even on very large graphs. Neo4j’s graph platform provides these capabilities, among others. In particular, (1) the graph platform offers ETL and data integration functionality to extract data connections from various data formats and moving them into a single graph database. (2) The graph platform provides high-performance graph algorithms for analysis, that are well integrated with management capabilities of graph database.

In this paper we provide an overview of the graph platform (Sect. 2) and give a detailed

presentation of the capabilities, design, and architecture of the graph algorithms (Sect. 3). We demonstrate its utility of our approach with a Twitter Troll analysis (Sect. 4), and show case its performance with a few experiments on large graphs (Sect. 5). Finally, we discuss related approaches (Sect. 6) and conclude the paper (Sect. 7).

2 Graph Platform

The graph platform is a well integrated set of tools, surfaces and infrastructure for storing, managing and querying connected data. At the frontend, the graph platform offers a variety of skill-specific, user-facing tools and APIs for solution developers, data scientists, and business users. Each user group has different needs to extract data from various sources, execute statements, explore query results, update information, perform analytical steps, and visualize connections. The comprehensive set of well-integrated tools on top of a common protocol, API, and query language provides efficient means for data scientists and solutions teams to move through the stages of discovery and design.

To make graph data equally accessible to tabular information, a query language that is able to express the richness of graph patterns in an comprehensive way is a powerful tool for developer and end-users alike. With Neo4j's declarative query language Cypher [Fr18], which uses graph pattern represented in ascii-art-symbols for pattern matching, creation, predicates and more, even complex conceptual connections can be expressed. With additional full support for list and map data structures and means processing those, many round trips of traditional query languages can be avoided. Built in data-flow paradigms allow users to clearly express more involved processing pipelines.

The graph platform is extensible with user defined procedures and functions, which are able to access the full scope of the underlying API and machine infrastructure, while being exposed as Cypher clauses or expressions. This extension mechanism was also used in our subsequently described work.

At the backend, the graph platform provides for both analytic and transactional operations. All operations leverage the efficient traversal of connections provided by the graph store and embedded in a scalable architecture. The graph platform can scale up to 32 TB of memory space enabling in-memory and near-memory graph processing on an ultra-large scale in a single machine. The graph platform can also scale out to multiple machines with Raft protocol-based [OO14] causal clustering architecture [HA90, Ah95] that supports ultra-large clusters and a wider range of cluster topologies for distributed data centers and cloud.

3 Graph Algorithms

Neo4j graph algorithms is a library of user-defined procedures that can be executed on a Neo4j database. The analysis procedures provide efficient parallel implementation of

common graph analysis algorithms. The library covers algorithms for path finding (e.g. all pair shortest path and minimum weight spanning tree), centrality analysis (e.g. pagerank and betweenness centrality), and community detection (e.g. label propagation and louvain modularity), as of now. The procedures can be called directly using Cypher in the Neo4j Browser, from the cypher shell, from Jupyter notebooks or any other client code. For running these algorithms a (projected) (sub-)graph of data is concurrently extracted from Neo4j into a separate, in-memory graph-storage to provide isolation and high read-only operations performance. Then the appropriate algorithm is executed concurrently on that graph storage, taking graph structure and additional information such as node- or relationship-weights into account. After the computation finished, the results can either be streamed to the client or optionally written back efficiently in a concurrent manner, e.g. to node-properties or relationships.

The general call syntax to call a analysis procedure is:

```
CALL algo.<name>(<nodeSelector>, <relSelector>, {<config>})
YIELD <columnList>
```

Here, <name> specifies the procedure to call, <nodeSelector> and <relSelector> specify the graph of interest, <config> provide additional parameters of the procedure, and <columnList> specifies, which column of output of the procedures should be returned. Most algorithms provide graph processing statistics and time measurements as well as statistical summaries of the computed graph metrics (e.g. min, max, avg, stdev, and percentiles of a centrality). An alternative variant of each algorithm is available as `algo.<name>.stream` which instead streams back the algorithm results in multiple columns of data. That volume would be equivalent to the size of the graph of interest, potentially billions of rows.

Process: All analysis procedure follow the same three-step process. (1) Load the graph of interest in parallel from the database into a succinct in-memory data structures. (2) Run graph algorithm in parallel. (3) Consume results. If multiple procedures should be executed, the graph of interest can be pre-loaded once and then referred to by name.

3.1 Graph Loading

Typically, the property graph managed in the database contains much more information than relevant for a certain analysis step. In such case, the (sub)graph of interest is a projection of the managed property graph. The projected graph is either a directed graph or undirected graph with the possibility of node and relationship weights. The projected graph is not multigraph (like a property graph) and allows only a single edge between a pair of nodes in each direction.

Projection: The graph algorithms library provides two kinds of projection: (1) label-based projection and (2) Cypher-based projection. Label-based projection extracts all nodes with a label given as <nodeSelector> and all relationships of a type given as <relSelector>. For instance,

```
CALL algo.pageRank('Page', 'LINKS')
```

calls the procedure `pageRank` on nodes with label `Page` and relationships of type `LINKS`. If either or both are left off then the algorithm will run on all entities of the respective, connected type.

In contrast, Cypher-based projection extracts a subgraph based on two Cypher queries. The first query given as `<nodeSelector>` specifies the nodes of interest as a node-id-list and the second query given as `<relSelector>` specifies the relationship of interest as a list of start-end-node-ids. For instance,

```
CALL algo.pageRank(
  "MATCH (u:User) WHERE exists( (u)-[:FRIENDS]-() ) RETURN id(u) AS id",
  "MATCH (u1:User)-[:FRIENDS]-(u2:User)
  RETURN id(u1) AS source, id(u2) AS target",
  {graph:"cypher"}
)
```

calls the procedure `pageRank` on a graph containing (1) nodes that match the pattern `(u:User)` and have at least one `FRIENDS` relationship and (2) relationships of type `FRIENDS` between two nodes with label `User`. Note that relationships that do not have both source and target nodes described by the `<nodeSelector>` will be ignored. This projection is presenting a *graph view* over the existing data and can be used to contract or aggregate the graph or collapse intermediate paths. For instance, it can be used to render a user-to-user graph from a social network via intermediate mentions, jointly used tags or replies on messages. Or it can provide category or product similarity graphs based on joint reviews or purchases by users. With the Cypher-based projection, any n-partite graph can be projected to a mono-partite graph for the means of running graph analysis.

Efficient Loading: To quickly load the relevant sub-graphs from Neo4j into the dedicated data structures, the library uses low level APIs of the graph database to avoid memory churn and preferably only use primitive numeric types. During the loading the id-space of the original graph is remapped to a consecutive id-space in the algorithm space to allow for gap free operations and iteration. The load operations are executed in parallel with the given concurrency, each thread operating on a batch of graph records at a time. For graph projections via Cypher we utilize the compiled runtime that turns Cypher queries into efficient Java bytecode. There we use Cypher's pagination capabilities to distribute the loading across concurrent threads.

Graph representation: The graph algorithm library uses a dedicated in-memory representation of the projected graph data. This allows for isolation from the transactional graph, for holding arbitrary graph projections and for high-performance random access. These data structures are presented with a thin, numeric-id based API to each algorithm, which allows for different implementations. Different aspects of the graph API are separated and can be passed individually to an algorithm according to its needs. The projected, potentially huge graph is stored in-memory in a succinct way. Particularly, all nodes are mapped to a

compressed, dense integer domain. The mapping is stored (1) in a dense array used to map the dense integer domain back to the original node ids and (2) in a sparse array to map the original node ids forward to the dense integer domain.

Relationships without weights are stored in a CSR-like structure [Bu09]. A source node id-indexed, paged array holds the position of each node's adjacency in a second array. Target node ids are delta and variable length encoded per adjacency. For relationships with weights the adjacency of a node is represented with two arrays, one for the target nodes and one for the weights. Depending on the algorithm's needs, the only incoming edge are presented or the adjacency is additionally replicated as outgoing edges. The representation is designed to scale to very large graphs of billions of edges. The loader allocates only the structures needed, e.g. weight entries are not created for default values or not at all if node or relationship weights are not needed.

After an algorithm is finished using the graph structure it is able to release that memory to make it available for further processing.

3.2 Algorithm Execution

Tab. 1 lists all algorithms currently available as procedure in Neo4j graph algorithms. There are algorithms for centrality measuring, path finding, and community detection. For each algorithm, we reference work on which its implementation is based on.

Path Finding	All-Pair Shortest Path	[Di59, KS91]
	K Shortest Path	[Ye70, Ye71]
	Single-Source Shortest Path	[MS03, Ma07]
	K Spanning Tree	[Pr57]
	Minimum Spanning Tree	[NMN01]
	A* Shortest Path	[HNR68]
Centrality Measuring	Betweenness Centrality	[Fr77, Br01, BP07, Ko13]
	Closeness Centrality	[Ba50]
	Dangalchev Centrality	[Da06]
	PageRank	[MR05, GZB04]
Community Detection	Label Propagation	[RAK07, FI14]
	Louvain Modularity	[BI08, Wi14]
	Triangle Counting and Clustering Coefficient	[Ts08, CC11]
	Strongly Connected Components	[Ta72, MNS17, SRM14]
	Balanced Triads	[He58, Fl63]

Tab. 1: Graph analytical algorithms implemented in Neo4j graph algorithms.

Each algorithm utilizes the provided graph representation to optimally batch and execute partial operations concurrently. The partial operations are orchestrated by a concurrency infrastructure that handles utilization and work redistribution. This allows threads that finish earlier to pick up work from threads computing metrics for nodes with a higher degree. For

the implementation of the algorithms we evaluated and implemented the latest state of the art research with dedicated focus on parallelization and scaling.

Each stage of loading, algorithm execution and result production tracks and reports memory consumption to allow configuration adjustments for future invocations.

3.3 Result Consumptions

The library offers three kinds of result consumptions: (1) *write back results*, (2) *tabular aggregated results*, and (3) *tabular streamed results*.

Write back results: The non-streaming procedures, write back results to the property graph in the database as node properties or relationships. This write back uses again low level transactional APIs and batches updates of larger chunks of the graph (e.g. 100 000 updates per transaction). By utilizing Neo4j's transaction-co-commit and joint-flush mechanics, it can achieve write performance of at least 1 M record updates per second, depending on available compute resources and I/O bandwidth.

The non-streaming procedures are indicated by the procedure name not having the suffix `.stream`. The parameter `write` indicates if only statistics should be computed or actual write back executed. For instance,

```
CALL algo.pageRank('Page', 'LINKS',
    { iterations: 20, dampingFactor: 0.85,
      write: true, writeProperty: "pagerank" })
```

calls the procedure `pageRank` with four config parameters. Two, `iterations: 20` and `dampingFactor: 0.85` are PageRank algorithm-specific, while `write: true` triggers a write back result and `writeProperty: "pagerank"` specifies the name of the node property to which the result shall be written back. In this case, the PageRank scores calculated for each node is written to the property `pagerank` of that node. Those written back graph metrics are used to enrich the original graph data and can be used afterwards by regular, OLTP query processing, e.g. for recommendations, ranking, or other decision making.

Tabular aggregated results: Procedures report the various statistics for the computed metrics and for operations (projected graph size, timings, memory usage), which the user can utilize for further processing and monitoring.

Tabular streamed results: Most algorithms can return tabular results also as a tuples stream. The procedure implementing the stream variant of an algorithm is suffixed with the name `.stream`, e.g. `algo.pageRank.stream(...)`. Tabular results are return as a driving table for further processing to the query that contains the `CALL` statement and potentially to the client. In this mode for each node entry in the projected graph one or more computed metrics are returned. For instance centrality measures, shortest paths, triangle counts,

and specific triangle triples. This computed information is proportional to the size of the projected graph, i.e. it can reach billions of result rows produced.

All columns that should be returned must be explicitly selected in the **YIELD** clause after the **CALL**. Which columns are provided is documented for every procedure and can also be inspected interactively. The driving table can be further processed within Cypher or returned to the user with a **RETURN** clause. For instance, the query

```
CALL algo.pageRank.stream('Page', 'LINKS',  
                          { iterations: 20, dampingFactor: 0.85 })
```

```
YIELD nodeId, score
```

```
RETURN nodeId, score ORDER BY score DESC LIMIT 5
```

calls the procedure `algo.pageRank.stream` and postfilters its result to return only the ids and scores of the top five page-ranked nodes.

Summary: The Neo4j graph algorithms procedure library provides a comprehensive set of graph analysis algorithms. The algorithms are implemented for parallel execution and operate on succinct optimized in-memory data structures that scale to very large graphs of interest. The procedures take care of data extraction and result routing. They blend in very well with Cypher. Cypher queries can call the procedures, specify the graph of interest, and consume the result for further processing in a seamless, composable fashion.

4 Analyzing Troll Behavior

To demonstrate usefulness and practicality we show how Twitter troll behavior can be analyzed and better understood with the help of Neo4j graph algorithms. As part of the House Intelligence Committee investigation into how Russia may have influenced the 2016 US Election, Twitter published the screen names of almost 3000 Twitter accounts believed to be connected to Russia's Internet Research Agency, a company known for operating social media troll accounts. Twitter immediately suspended these accounts, deleting their data from Twitter.com and the Twitter API. A team at NBC News including Ben Popken and EJ Fox was able to reconstruct a dataset consisting of a subset of the deleted data for their investigation and using Neo4j, were able to show how these troll accounts went on attack during key election moments. NBC News open-sourced the reconstructed dataset and released it as a Neo4j database. In this section we show how this analysis was conducted using Neo4j graph algorithms.

4.1 Preparation

Graph data schema: The Neo4j database was constructed from several anonymous sources who had been collecting election related tweets leading up to the 2016 US Election. These

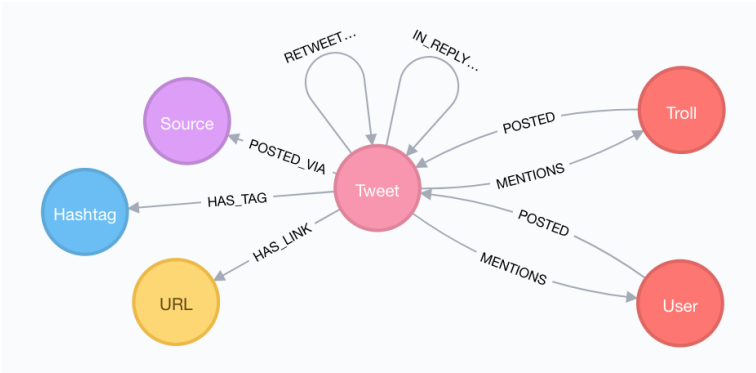


Fig. 1: The property graph schema used to represent the Russian Twitter Trolls dataset in Neo4j.

datasets were combined and imported into Neo4j so that the data is structured as shown in Fig. 1. The nodes are **Tweets**, **Users** (which can also be **Trolls**), **Hashtags**, **Sources** (the application used to post a **Tweet**), and **URLs** mentioned in **Tweets**. Relationships connect these nodes, showing for example, users mentioned in tweets and hashtags used. **User** nodes representing known troll accounts are additionally labeled **Troll**.

Exploratory data analysis with Cypher: Once the data was modeled and imported into Neo4j, Cypher queries permit exploration. Find tweets with the hashtag #thanksobama:

```

MATCH (u:User)-[:POSTED]->(t:Tweet)
  -[:HAS_TAG]->(ht:Hashtag {tag: "thanksobama"})
RETURN * LIMIT 50

```

Hashtags were used by the trolls to insert themselves into conversations and gain visibility. We can query the most common hashtags used by the trolls:

```

MATCH (ht:Hashtag)<-[:HAS_TAG]-(tw:Tweet)<-[:POSTED]-(:Troll)
WITH ht, count(tw) AS num ORDER BY num DESC
RETURN ht.tag AS hashtag, num LIMIT 10

```

One of the findings reported by the NBC team was that troll tweet volume spiked during key election related events. We can see that much of the tweet volume occurs leading up to and immediately following the 2016 US Election. Here we query for tweet volume by day:

```

MATCH (:Troll)-[:POSTED]->(t:Tweet)
WHERE t.created_str > "2016-10-01"
RETURN substring(t.created_str,0,10) AS day, count(t) AS num
ORDER BY day LIMIT 60

```

Projected graph of interest: Much of the analysis (including the application of graph algorithms) was done on a projected monopartite **User-to-User** graph from the broader domain graph. For example, we consider a **Troll** account to *amplify* another **Troll** account when it posts a **Tweet** that is a retweet of a **Tweet** posted by another **Troll** account as

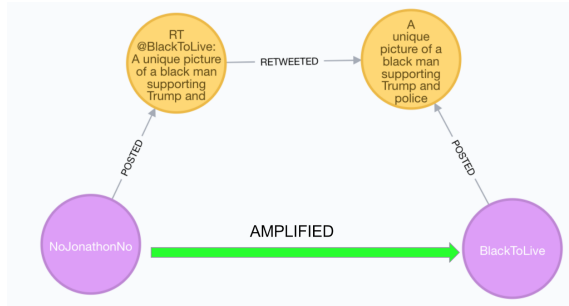


Fig. 2: Illustration of the projected graph used to run graph algorithms on the Russian Twitter Trolls dataset.

illustrated in Fig. 2. We construct an projected graph of this *retweet network* for analysis. This particular analysis is based on the insight that retweets on Twitter generally indicate agreement of a message, or at least desire for broader visibility of that message, and that they also permit a network of authors to limit the amount of original content that needs to be written in order to push key themes and messages. This projected graph can be expressed using Cypher:

```
MATCH p=(r1:Troll)-[:POSTED]->(:Tweet)
      <-[:RETWEETED]-(:Tweet)<-[:POSTED]->(r2:Troll)
RETURN p LIMIT 1
```

This is however only one of the possible projections, others could be based on mentions, replies, or jointly used hashtags for topic analysis.

4.2 Analysis

For the analysis, we apply two graph algorithms. Centralities measures helps us detect who are the most influential troll accounts. Subsequently community detection helps to answer which Troll accounts are strongly connected through Retweets.

Centrality: Centrality algorithms determine the most important nodes in the network. In the context of our retweet graph, centralities will allow us to determine the most influential accounts.

We can run PageRank on the projected graph:

```
CALL algo.pageRank(
  "MATCH (t:Troll) RETURN id(t) AS id",
  "MATCH (r1:Troll)-[:POSTED]->(:Tweet)
    <-[:RETWEETED]-(:Tweet)<-[:POSTED]->(r2:Troll)
  RETURN id(r2) as source, id(r1) as target",
```

```
{graph:"cypher", write: true, writeProperty: "pagerank"}
)
```

The above Cypher statement will write a `pagerank` property to the `Troll` nodes. We can use Cypher to find the trolls with the highest `pagerank` score, and are thus the most influential in the network by this measure:

```
MATCH (t:Troll) WHERE exists(t.pagerank)
RETURN t.screen_name AS troll, t.pagerank AS pagerank
ORDER BY pagerank DESC LIMIT 25
```

With a slight modification of this query, we can assign (`SET`) each of the most influential trolls a random seeding community number:

```
MATCH (t:Troll) WHERE exists(t.pagerank)
WITH t ORDER BY t.pagerank DESC LIMIT 25
SET t.community = toInteger(round(rand()*1000))
RETURN t
```

Community detection: Community detection algorithms are applied to this projected network to partition the graph. In the context of the retweet graph community detection will illustrate which accounts are frequently amplifying others. We want to see if there are clusters of troll accounts that are promoting certain types of content. For example, are certain accounts focused on pro-Trump content while others are focused on anti-Clinton content?

Here we partition the graph into communities using the Label Propagation algorithm. Label propagation is a particularly fast algorithm for finding communities in a graph. As we are particularly interested in communities around the most influential trolls, we use the algorithm in a semi-supervised way and seed it with the `community` property we have assigned to these trolls. Then run, the algorithm adds to a `community` property to all remaining nodes and updates the property values so that they eventually specify the communities the nodes have been assigned to by the algorithm. The value of the `community` property does not have inherent meaning apart from categorical separation of the detected communities.

```
CALL algo.labelPropagation(
  "MATCH (t:Troll) RETURN id(t) AS id",
  "MATCH (r1:Troll)-[:POSTED]->(t:Tweet)
    <-[:RETWEETED]-(:Tweet)<-[:POSTED]- (r2:Troll)
  RETURN id(r2) AS source,
    id(r1) AS target, count(t) AS weight",
  "OUTGOING",
  {graph:"cypher", write: true,
  partitionProperty: "community", iterations: 200}
)
```

We can then see which Trolls were assigned to each community:

```
MATCH (t:Troll) WHERE exists(t.community)
```

```
RETURN collect(t.screen_name) AS members, t.community
ORDER BY size(members) DESC LIMIT 10
```

Finally, we can see if there are certain themes that each community was focused on, by inspecting the top-10 common hashtags used by the top-10 largest communities:

```
MATCH (t:Troll) WHERE exists(t.partition)
WITH collect(t) AS members, t.community
ORDER BY size(members) DESC LIMIT 10
UNWIND members AS t
MATCH (t)-[:POSTED]->(tw:Tweet)-[:HAS_TAG]->(ht:Hashtag)
WITH community, ht.tag AS tag, count(tw) AS num
ORDER BY community, num DESC
RETURN community, collect(tag)[..10] AS toptags
```

Upon aggregating the hashtags that each community was using, we can see that the group around top influential troll @TEN_GOP was tweeting mainly about right-wing politics (#VoterFraud, #TrumpTrain); the group around @DanaGeezus was more left leaning, but not necessarily positively (#ObamasWishlist, #RejectedDebateTopics); and the group around @gloed_up covered topics in the Black Lives Matter community (#BlackLivesMatter, #Racism, #BLM). Each of these three clusters tended to have a small number of original content generators, with the bulk of the community amplifying the message. For example, one account @TheFoundingSon sent more than 3200 original tweets, averaging about 7 tweets per day. On the other hand, accounts like @AmelieBaldwin authored only 21 original tweets out of more than 9000 sent.

4.3 Visualization

Including the results of graph algorithms in visualization allows us to interpret the results of graph algorithms that otherwise might be difficult to make sense of. Fig. 3 visualizes the three distinct communities identified by our community detection algorithm and the most influential troll accounts within each community, as determined by PageRank. The communities are shown in different colors, node sizes are styled proportionally to their PageRank score, and relationship thickness is styled proportionally to relationship weights, in this case the number of times a troll retweeted another.

5 Evaluation

To demonstrate the performance of Neo4j graph algorithm procedure library we conducted a series of experiments. We ran PageRank, Union Find (connected components), Label Propagation, and Strongly-Connected Components on a number of standard graph datasets available on SNAP [LK14].

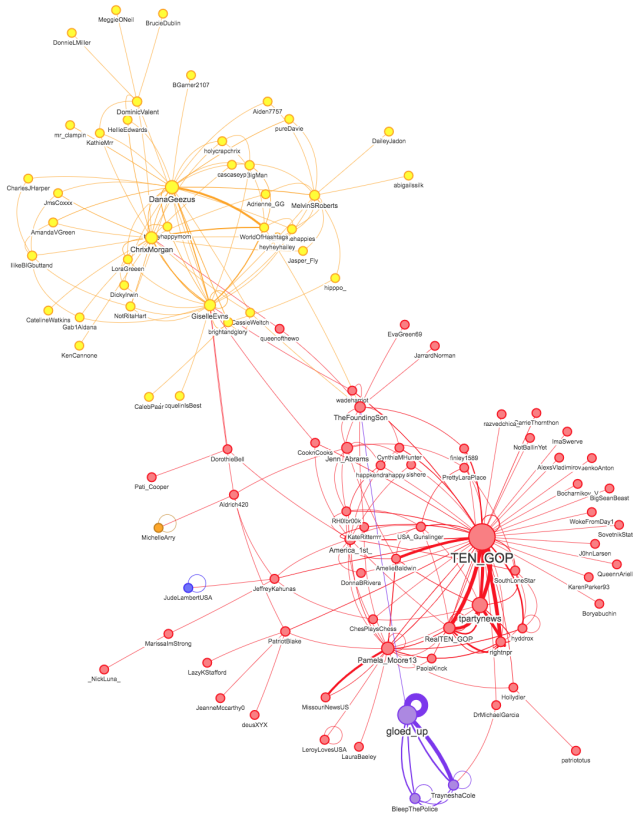


Fig. 3: Visualization of the three distinct communities and the most influential troll accounts within each community.

Setup: We used a machine with four Intel Xeon E7-8870 v3 CPUs, each featuring eighteen cores at 2.1 GHz clock speed, hyper threading, and 45 MB of cache. Additionally, the machine was equipped with 1 TB of DDR4 memory and 1.8 TB of PCIE-SSD storage, in which the Neo4j database files resided. The software stack consisted of Ubuntu Server 16.04 with kernel version 4.4.0, Java(TM) SE 1.8.0_121-b13 for compiling, Java HotSpot(TM) 64-Bit Server VM 25.121-b13 for running, Neo4j Enterprise 3.1.4. Neo4j was configured to a page cache size of 5 GB and the Java VM was allowed to allocate 32 GB of heap while executing Neo4j.

Graph dataset: We have not used the Twitter troll dataset for our experiments, since it is not very large in size. Instead, we used in total eight datasets of various sizes as shown in Tab. 2. Note that the disk size shows the size of the graphs in the Neo4j database, which resides on SSD and gets page cached in-memory. When loaded as graph of interest into

the in-memory representation, each graph fitted into the memory the JVM was allowed to allocate.

Graph		#Nodes [M]	#Relationships [M]	Avg. out degree	Disk size [GB]
Pokec	(PK)	1.63	30.62	18.75	0.99
cit-patents	(CP)	3.77	16.52	4.38	0.58
Graphs500-23	(G5)	4.61	129.33	28.05	4.17
soc-LifeJournal1	(LJ)	4.85	68.99	14.23	2.27
DBPedia	(DP)	11.47	116.60	10.16	3.87
Twitter-2010	(TW)	41.65	1468.37	35.25	47.60
Friendster	(FR)	65.61	1806.07	27.53	58.94

Tab. 2: Graph datasets used in measurements.

Algorithms: We ran four algorithms, precisely: PageRank, Union Find, Label Propagation, and Strongly-Connected Components (SCC) on each of the eight graph datasets. The corresponding procedure calls are listed in Tab. 3. As can be seen, we did not set parameters for graph projection, so that the algorithms ran on the complete graph dataset. All procedures were set to write back their results into the graph dataset (`write:true`).

Algorithm	Execution
Pagerank	<code>CALL algo.pageRank(null, null, {write:true, iterations:20});</code>
Union Find	<code>CALL algo.unionFind(null, null, {write:true});</code>
Label Propagation	<code>CALL algo.labelPropagation(null, null, 'OUTGOING', {write:true});</code>
SCC	<code>CALL algo.scc.iterative(null, null, {write:true});</code>

Tab. 3: Procedure calls used in measurements.

Results: We measured the total runtime of the complete procedure call, which included loading the graph, computing the analysis, and writing back the result. Fig. 4 shows the runtime for each procedure call on each graph dataset. All four algorithms completed within a few seconds for graphs upto 30 M relationships (PK, and CP) and within half a minute for graph upto 120 M relationships (G5, LJ, DP). On the very large graphs with more than 1 G relationships (TW and FR), all procedures finished within a few minutes.

To take a closer look, we also measured the individual runtimes of loading the graph (load), computing the analysis (compute), and writing back the result (write). Fig. 5 reports for each procedure call on each graph dataset the proportion of runtime spent on load, compute, and write. As can be seen most procedure call spent the majority of their runtime in loading the graph into to the representation used by the algorithms. This is as expected, since the whole graph of interest has to be read during load. Although it takes most of the runtime, the loading is still very efficient. Looking at the very large graphs TW and FR, the procedure accomplished to load the graph of interest at a rate of put to 20 000 relationships/s.

The benefit of loading is revealed during the compute phase. The succinct in-memory representation of the graph of interest facilitates very efficient graph algorithm computation. For instance, PageRank requires to read over all relationships ones per iteration. In

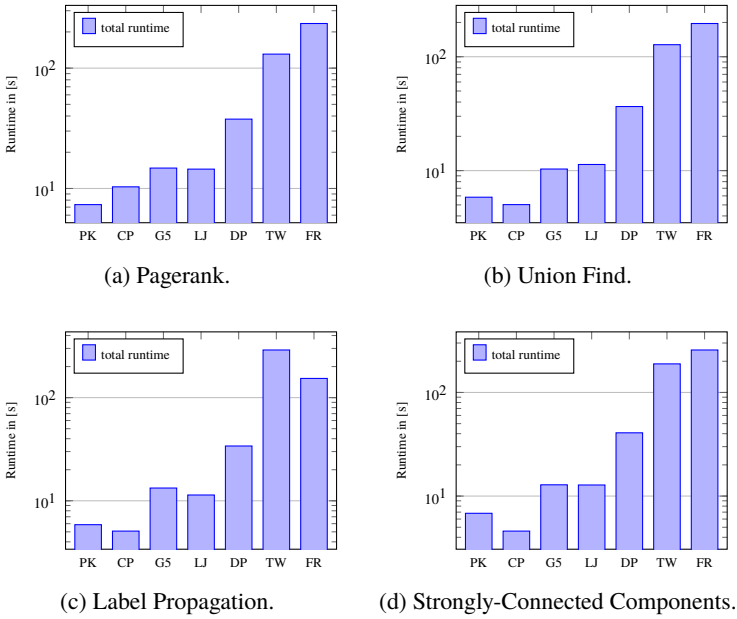


Fig. 4: Total runtimes.

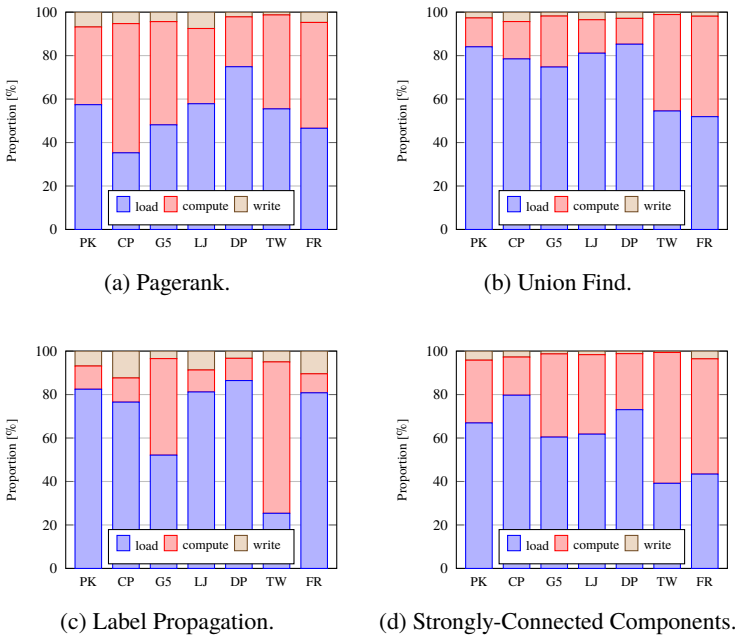


Fig. 5: Proportion of runtime spent on loading, computing, and writing back.

our experiments PageRank was computed with an average over all graph datasets of 275 000 relationships/s and on the Twitter graph with upto 519 000 relationships/s.

6 Related Work

Any system for graph analytics provides an abstract construct for a user to execute analytical graph algorithms without caring about the technical details of an efficient parallel execution. So far, one of the most prominent and common abstractions for graph analytics has been the *vertex-centric* programming model [Lo10, Ma10, SBC10]. A vertex-centric program requires users to provide a compute function that defines the actual analytical computation. A declarative variant worth mentioning is the GSQL [De18].

Since many graph algorithms are based on traversals, the traversal itself provides another common abstraction for graph analytics. *Traversal-based* languages provide native support for graph traversals as part of a domain-specific language, examples are Gremlin [Ro15], GreenMarl [Ho14], GEM [Ru13], and GraphScript [Pa17].

Instead of an dedicated abstraction, declarative graph query language [ARV18, Fr18, Re16, PPvR18] can be extended for analytical purposes by means of composition, grouping and aggregation [An18, Vo17]. Under certain constraints aggregation can be even allowed within recursion [SGL15].

All these approaches provide a means to implement or express algorithms as opposed to the specific graph algorithms themselves. While this is very useful for users developing their own custom algorithms, it unnecessarily complicates matters for users that simply want to deploy well established and understood algorithms. Such users are better served with algorithm libraries.

A very prominent graph algorithms library are the Boost Graph Library⁸ [SLL02] and Parallel Boost Graph Library⁹. Both libraries still require code from users in order to extract and feed their graph into the chosen algorithm or input results of one algorithm into another. While Neo4j graph algorithms provide a similar set of algorithms, they are, in addition, well-integration with the graph platform. Users can directly call graph algorithms on a graph database and have their result written back to the graph database, without any additional imperative programming needed. User can also directly leverage the Cypher query capabilities to declaratively select the graph of interest. This is a relevant feature, since the graph database contains in practice significantly more data or data in different shape than the graph of interest, as illustrated in Section 4.2.

For a more comprehensive overview on systems for big graph analytics we refer the reader to [PV17].

⁸ https://www.boost.org/doc/libs/1_66_0/libs/graph/doc/

⁹ https://www.boost.org/doc/libs/1_66_0/libs/graph_parallel/doc/html/

7 Conclusion

With a relationship-centered approach one does not look at individuals in isolation, builds understanding of the world by looking at how person, events, and things are connected. For many aspects, the relationships of interest are hidden behind layers of many other connections. For uncovering such hidden relationship, graph data and graph analytics becomes increasingly instrumental. The Neo4j graph platform provides a comprehensive software stack to perform graph analytics, from ingesting the data all the way to visualizing the results. A central piece provided by the graph platform and used in such a graph analytics are the graph algorithms, which we presented in this paper. The graph algorithms provide a comprehensive library of user-defined procedures offering graph analytical algorithm well-integrated with the other components of the graph platform. The library covers algorithms for centrality measuring, path finding, and community detection. All procedures can be called within a Cypher query. Cypher can be also used to project out the graph of interest for called procedure. The procedures can write back the result to the base data or stream it to the calling Cypher query for post processing. The graph algorithms are designed to be executed on large to very large graphs and were tested with several billion nodes and tens of billions of relationships, exhibiting linear performance gain and scalability on large multi-core machines.

In the future, we will continue to apply new research from graph analytics and machine learning. For instance, we are already in the processes of adding performant means for similarity computations including a number of different common similarity measure, such as Jaccard distance. We also look into the generation of k-nearest-neighbour networks. Another considered extension of the library will provide an infrastructure for user-defined algorithms that allows for efficient execution. We also intend to explore new execution models like distributed processing or GPU based approaches.

References

- [Ah95] Ahamad, Mustaque; Neiger, Gil; Burns, James E.; Kohli, Prince; Hutto, Phillip W.: Causal Memory: Definitions, Implementation, and Programming. *Distributed Computing*, 9(1):37–49, March 1995.
- [An18] Angles, Renzo; Arenas, Marcelo; Barceló, Pablo; Boncz, Peter A.; Fletcher, George H. L.; Gutierrez, Claudio; Lindaaker, Tobias; Paradies, Marcus; Plantikow, Stefan; Sequeda, Juan; van Rest, Oskar; Voigt, Hannes: G-CORE: A Core for Future Graph Query Languages. In: *SIGMOD. ACM*, 2018.
- [ARV18] Angles, Renzo; Reutter, Juan; Voigt, Hannes: Graph Query Languages. In: *Encyclopedia of Big Data Technologies*. Springer, 2018.
- [Ba50] Bavelas, Alex: Communication Patterns in Task-Oriented Groups. *The Journal of the Acoustical Society of America*, 22(6):725–730, November 1950.

- [BI08] Blondel, Vincent D.; Guillaume, Jean-Loup; Lambiotte, Renaud; Lefebvre, Etienne: Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008+12, October 2008.
- [BP07] Brandes, Ulrik; Pich, Christian: Centrality Estimation in Large Networks. *International Journal of Bifurcation and Chaos*, 17(7):2303–2318, 2007.
- [Br01] Brandes, Ulrik: A Faster Algorithm for Betweenness Centrality. *Journal of Mathematical Sociology*, 25(2):163–177, 2001.
- [Bu09] Buluç, Aydin; Fineman, Jeremy T.; Frigo, Matteo; Gilbert, John R.; Leiserson, Charles E.: Parallel Sparse Matrix-Vector and Matrix-Transpose-Vector Multiplication Using Compressed Sparse Blocks. In: SPAA. ACM, pp. 233–244, 2009.
- [CC11] Chu, Shumo; Cheng, James: Triangle Listing in Massive Networks and Its Applications. In: KDD. ACM, pp. 672–680, 2011.
- [Da06] Dangalchev, Chavdar: Residual closeness in networks. *Physica A: Statistical Mechanics and its Applications*, 365(2):556–564, June 2006.
- [De18] Deutsch, Alin: Querying Graph Databases with the GSQL Query Language. In: SBDD. SBC, p. 313, 2018.
- [Di59] Dijkstra, Edsger W.: A Note on Two Problems in Connexion with Graphs. *Numerische mathematik*, 1(1):269–271, 1959.
- [FI14] Fujiwara, Yasuhiro; Irie, Go: Efficient Label Propagation. In: ICML. JMLR.org, pp. 784–792, 2014.
- [Fl63] Flament, Claude: Application of Graph Theory to Group Structure. Prentice-Hall, chapter 3: Balancing Processes, 1963.
- [Fr77] Freeman, Linton C.: A Set of Measures of Centrality Based on Betweenness. *Sociometry*, 40(1), March 1977.
- [Fr18] Francis, Nadime; Green, Alastair; Guagliardo, Paolo; Libkin, Leonid; Lindaaker, Tobias; Marsault, Victor; Plantikow, Stefan; Rydberg, Mats; Selmer, Petra; Taylor, Andrés: Cypher: An Evolving Query Language for Property Graphs. In: SIGMOD. ACM, 2018.
- [GZB04] Gleich, David F.; Zhukov, Leonid; Berkhin, Pavel: Fast Parallel PageRank: A Linear System Approach. Technical report, Yahoo, 2004.
- [HA90] Hutto, Phillip W.; Ahamad, Mustaque: Slow Memory: Weakening Consistency to Enhance Concurrency in Distributed Shared Memories. In: ICDCS. IEEE Computer Society, pp. 302–309, 1990.
- [He58] Heider, Fritz: The Psychology of Interpersonal Relations. John Wiley & Sons, 1958.
- [HNR68] Hart, Peter E.; Nilsson, Nils J.; Raphael, Bertram: A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Trans. Systems Science and Cybernetics*, 4(2):100–107, July 1968.
- [Ho14] Hong, Sungpack; Salihoglu, Semih; Widom, Jennifer; Olukotun, Kunle: Simplifying Scalable Graph Processing with a Domain-Specific Language. In: CGO. ACM, pp. 208–218, 2014.

- [Ko13] Kourtellis, Nicolas; Alahakoon, Tharaka; Simha, Ramanuja; Iamnitchi, Adriana; Tripathi, Rahul: Identifying High Betweenness Centrality Nodes in Large Social Network. *Journal of Social Network Analysis & Mining*, 3(4):899–914, December 2013.
- [KS91] Kumar, Vipin; Singh, Vineet: Scalability of Parallel Algorithms for the All-Pairs Shortest-Path Problem. *Journal of Parallel and Distributed Computing*, 13(2):124–138, October 1991.
- [LK14] Leskovec, Jure; Krevl, Andrej: SNAP Datasets: Stanford Large Network Dataset Collection, June 2014.
- [Lo10] Low, Yucheng; Gonzalez, Joseph; Kyrola, Aapo; Bickson, Danny; Guestrin, Carlos; Hellerstein, Joseph M.: GraphLab: A New Framework For Parallel Machine Learning. In: *UAI*. pp. 340–349, 2010.
- [Ma07] Madduri, Kamesh; Bader, David A.; Berry, Jonathan W.; Crobak, Joseph R.: An Experimental Study of A Parallel Shortest Path Algorithm for Solving Large-Scale Graph Instances. In: *ALENEX*. SIAM, 2007.
- [Ma10] Malewicz, Grzegorz; Austern, Matthew H.; Bik, Aart J. C.; Dehnert, James C.; Horn, Ian; Leiser, Naty; Czajkowski, Grzegorz; Pregel: A System for Large-Scale Graph Processing. In: *SIGMOD*. ACM, pp. 135–146, 2010.
- [ML00] Marchiori, Massimo; Latora, Vito: Harmony in the small-world. *Physica A: Statistical Mechanics and its Applications*, 285(2–3):539–546, October 2000.
- [MNS17] Mehlhorn, Kurt; Näher, Stefan; Sanders, Peter: Engineering DFS-Based Graph Algorithms. *The Computing Research Repository*, abs/1703.10023, March 2017.
- [MR05] Manaskasemsak, Bundit; Rungsawang, Arnon: An Efficient Partition-Based Parallel PageRank Algorithm. In: *ICPADS*. IEEE Computer Society, pp. 257–263, 2005.
- [MS03] Meyer, Ulrich; Sanders, Peter: Δ -stepping: a parallelizable shortest path algorithm. *Journal of Algorithms*, 49(1):114–152, October 2003.
- [NMN01] Nesetril, Jaroslav; Milková, Eva; Nesetrilová, Helena: Otakar Boruvka on minimum spanning tree problem Translation of both the 1926 papers, comments, history. *Discrete Mathematics*, 233(1-3):3–36, April 2001.
- [OO14] Ongaro, Diego; Ousterhout, John K.: In Search of an Understandable Consensus Algorithm. In: *ATC*. USENIX Association, pp. 305–319, 2014.
- [Pa17] Paradies, Marcus; Kinder, Cornelia; Bross, Jan; Fischer, Thomas; Kasperovics, Romans; Gildhoff, Hinnerk: GraphScript: implementing complex graph algorithms in SAP HANA. In: *Proceedings of The 16th International Symposium on Database Programming Languages, DBPL 2017, Munich, Germany, September 1, 2017*. ACM, pp. 13:1–13:4, 2017.
- [PPvR18] Paradies, Marcus; Plantikow, Stefan; van Rest, Oskar: Graph Data Management Systems. In: *Encyclopedia of Big Data Technologies*. Springer, 2018.
- [Pr57] Prim, Robert C.: Shortest Connection Networks And Some Generalizations. *Bell System Technical Journal*, 36(6):1389–1401, November 1957.

- [PV17] Paradies, Marcus; Voigt, Hannes: Big Graph Data Analytics on Single Machines – An Overview. *Datenbank-Spektrum*, 17(2), July 2017.
- [RAK07] Raghavan, Usha Nandini; Albert, Réka; Kumara, Soundar: Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E*, 76(3):036106–1–11, September 2007.
- [Re16] van Rest, Oskar; Hong, Sungpack; Kim, Jinha; Meng, Xuming; Chafi, Hassan: PGQL: a property graph query language. In: *GRADES*. ACM, p. 7, 2016.
- [RN10] Rodriguez, Marko A.; Neubauer, Peter: Constructions from Dots and Lines. *Bulletin of the American Society for Information Science and Technology*, 36(6):35–41, August 2010.
- [Ro15] Rodriguez, Marko A.: The Gremlin Graph Traversal Machine and Language. In: *DBPL*. ACM, 2015.
- [Ru13] Rudolf, Michael; Paradies, Marcus; Bornhövd, Christof; Lehner, Wolfgang: The Graph Story of the SAP HANA Database. In: *BTW*. volume 214. GI, pp. 403–420, 2013.
- [SBC10] Stutz, Philip; Bernstein, Abraham; Cohen, William W.: Signal/Collect: Graph Algorithms for the (Semantic) Web. In: *ISWC*. Springer, pp. 764–780, 2010.
- [SGL15] Seo, Jiwon; Guo, Stephen; Lam, Monica S.: SocialLite: An Efficient Graph Query Language Based on Datalog. *IEEE Transactions on Knowledge and Data Engineering*, 27(7):1824–1837, 2015.
- [SLL02] Siek, Jeremy G.; Lee, Lie-Quan; Lumsdaine, Andrew: *The Boost Graph Library - User Guide and Reference Manual*. Pearson / Prentice Hall, 2002.
- [SRM14] Slota, George M.; Rajamanickam, Sivasankaran; Madduri, Kamesh: BFS and Coloring-Based Parallel Algorithms for Strongly Connected Components and Related Problems. In: *IPDPS*. IEEE Computer Society, pp. 550–559, 2014.
- [Ta72] Tarjan, Robert Endre: Depth-First Search and Linear Graph Algorithms. *SIAM Journal on Computing*, 1(2):146–160, June 1972.
- [Ts08] Tsourakakis, Charalamos E.: Fast Counting of Triangles in Large Real Networks without Counting: Algorithms and Laws. In: *ICDM*. IEEE Computer Society, pp. 608–617, 2008.
- [Vo17] Voigt, Hannes: *Declarative Multidimensional Graph Queries*. volume 280. Springer, pp. 1–37, 2017.
- [Wi14] Wickramaarachchi, Charith; Frincu, Marc; Small, Patrick; Prasanna, Viktor K.: Fast parallel algorithm for unfolding of communities in large graphs. In: *HPEC*. IEEE, pp. 1–6, 2014.
- [Ye70] Yen, Jin Y.: An algorithm for finding shortest routes from all source nodes to a given destination in general networks. *Quarterly of Applied Mathematics*, 27(4):526–530, January 1970.
- [Ye71] Yen, Jin Y.: Finding the K shortest loopless paths in a network. *Management Science*, 17(11):712–716, 1971.

Query Processing and Optimization

Konzept und Implementierung eines echtzeitfähigen Model Management Systems

am Beispiel zur Überwachung von Lastprognosen für den Intraday Stromhandel

Yvonne Hegenbarth,¹ Gerald H. Ristow²

Abstract: Zur Gewährleistung der Stromnetzstabilität in Deutschland müssen Verteilernetzbetreiber darauf achten, dass zu jedem Zeitpunkt Energie-Erzeugung und -Verbrauch in ihrem Zuständigkeitsbereich in Einklang stehen. Dafür werden Vorhersagemodelle benötigt, um den zu erwartenden Überschuß oder zusätzlichen Bedarf an Energie für den Folgetag der Strombörse für den sogenannten *Day-Ahead* Handel zu melden. Neben dem Stromhandel für den Folgetag können Marktteilnehmer beim kontinuierlichen *Intraday* Strommengen bis zu fünf Minuten vor der tatsächlichen Auslieferung kaufen oder verkaufen. Bei Fehlprognosen und demnach Fehleinkäufen könnte mit einer Früherkennung und Modellanpassung diese im Intraday ausgeglichen werden. Dazu wird in dieser Arbeit ein System beschrieben, das automatisiert Fehlprognosen frühzeitig erkennt und eine Modelländerung durchführt. Das Modell wird dabei an den aktuellen Sachverhalt der Verbrauchszeitreihe angepasst. Durch diese Modellanpassung wird die Vorhersage verbessert, sodass der Intraday Handel besser betrieben werden kann und Fehleinkäufe ausgeglichen werden.

Keywords: data stream • model management • data mining • time series • forecast • concept drift • concept evolution

1 Motivation

Damit die Sicherheit des Stromnetzes in Deutschland gewährleistet wird, muss jeder Stromproduzent und Stromabnehmer seine Strommenge prognostizieren, die er ins Stromnetz einspeist oder entnimmt. Zur Prognose werden sogenannte *vorhersagende Modelle* entworfen.

In der Regel wird bei dem Prozess zur Berechnung von vorhersagenden Modellen mehrere zehn bis hundert Modelle erstellt, bis eines den Anforderungen (z.B. Vorhersagegenauigkeit) der Data Scientists genügt und eingesetzt wird. Zur Verwaltung und zum Reproduzieren von vorhersagenden Modellen muss ein *Model Management System* aufgesetzt werden. Die erzeugten Modelle werden persistiert und stehen für die Wiederverwendung zur Verfügung [Va16]. Durch sogenannte „Human-in-the-Loop Workflows“ werden die Benutzer mit Hilfe von Visualisierungstools in den Prozess zur Erstellung von vorhersagenden Modellen

¹ Software AG, Research, Uhlandstraße 12, 64297 Darmstadt, Yvonne.Hegenbarth@softwareag.com

² Software AG, Research, Uhlandstraße 12, 64297 Darmstadt, Gerald.Ristow@softwareag.com

integriert [LWG14]. Dadurch wird ermöglicht, dass der Mensch als Überwacher agiert und aktiv die Parameter des Modells anpasst, falls die Vorhersagen zu stark von der Realität abweichen [Se16].

Im Hinblick auf den Wandel des Day-Ahead Energiemarktes und dem kontinuierlichen Intraday Handel, werden schnelle, detaillierte und flexible Planungsmöglichkeiten gefordert [Da15]. Eine Modellanpassung der Stromverbrauchsprognose des Folgetages ist bei einer zu hohen Abweichung sinnvoll. Basierend auf den Berechnungen des neuen Modells kann entsprechend im Intraday überflüssiger Strom verkauft oder auf Grund mangelnder Reserven Strom gekauft werden.

Im weiteren Verlauf wird ein *echtzeitfähiges Model Management System* (eMMS) vorgestellt, das nicht den Persistenz-Ansatz von Liu u. a. verfolgt und den Menschen als Überwacher entlastet. Hier soll von der Hypothese ausgegangen werden, dass mit Hilfe von Data Mining Techniken und einer anwendungsspezifischen *Concept Drift* Änderungserkennung zur Laufzeit automatisiert eine Modellanpassung erfolgt.

Der Bericht fokussiert sich nicht auf die Erstellung eines vorhersagenden Modells mit minimalem Vorhersagefehler. In diesem Bericht soll vielmehr aufgezeigt werden, wie ein auf *Complex Event Processing* (CEP) basiertes Model Management aufgesetzt und betrieben werden kann.

2 Stand der Technik

Die Erstellung eines Vorhersagemodells ist ad hoc, das heißt für einen bestimmten Augenblick gemacht. Daher besteht das Bedürfnis bei Data Scientists die im Laufe der Zeit erstellten Modelle zu persistieren. In [Va16] sind diverse Problemen beschrieben, die beim iterativen Modellierungsprozess auftreten. Zusammenfassend beschreibt der Autor, dass unter anderem das Reproduzieren von Modellen und Ergebnissen übermäßig zeitaufwendig oder auf Grund mangelnder Dokumentation nicht durchführbar ist. Die Data Scientists müssen sich an Ergebnisse und Parameter früherer Versionen eines Modells „erinnern“ und haben keine Möglichkeit verschiedene Ausführungen zu kennzeichnen. Die genannte Herausforderung hebt ein wichtiges Problem für Machine Learning Tools hervor: das *Model Management*. Model Management bezeichnet die Angelegenheit des Verfolgens, Speicherns und Indexierens einer großen Anzahl von vorhersagenden Modellen, die anschließend reproduziert, geteilt, abgefragt und analysiert werden können.

Viele Model Management Systeme (MMS) sind Lösungen basierend auf einem Datenbankmanagementsystem (DBMS), das eine strukturierte Umgebung zum Speichern, Manipulieren und Abrufen von vorhersagenden Modellen bereitstellt [Be03; DK84; Va16].

3 Vorgehensweise

Zur Erläuterung der Vorgehensweise zeigt Abb. 1 den Datenfluss des von uns vorgeschlagenen echtzeitfähigen Model Management Systems (eMMS). Das Konzept baut auf einer *Offline-* und *Online-Analyse* auf und wird in Abschnitt 5 detailliert vorgestellt.

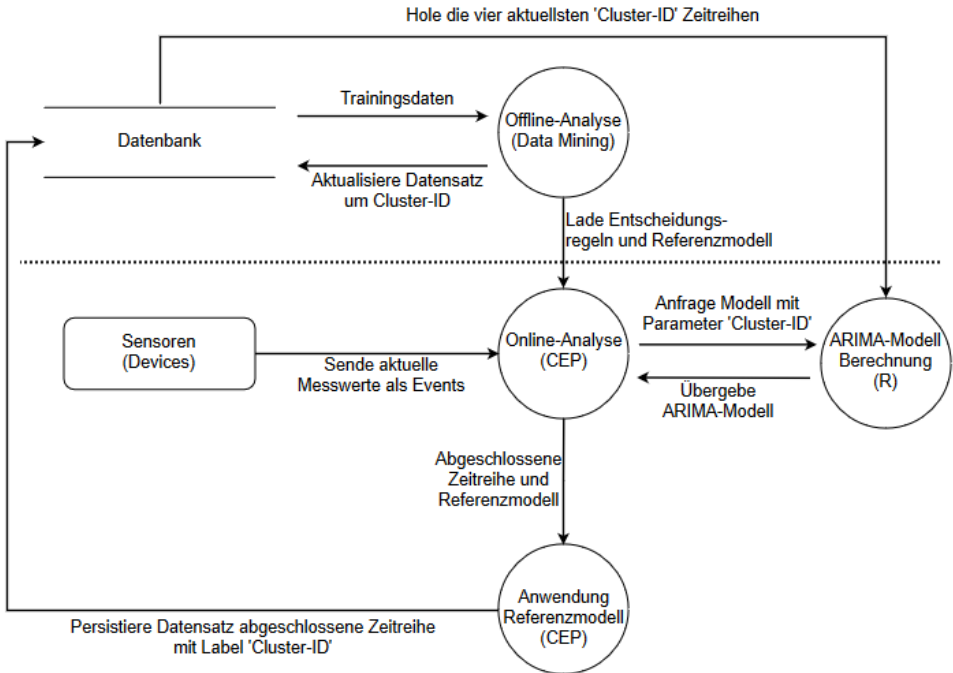


Abb. 1: Architektur eines echtzeitfähigen Model Management Systems

Das in diesem Bericht beschriebene eMMS nutzt ein DBMS lediglich zur Verwaltung von Messdaten in Form von *Zeitreihen*. Eine *Complex Event Processing* (CEP)-Engine ist für die Berechnung und Wartung von vorhersagenden Modellen zur Laufzeit verantwortlich (Abb. 1, Online-Analyse). CEP ist eine Technologie, die basierend auf Anfrageformulierungen Muster im Datenstrom in Form von Ereignissen (*Events*) erkennt [BK09]. Durch Aggregation und Zusammensetzung von auftretenden Ereignissen können neue komplexe Ereignisse (*Complex Events*) generiert und für weitere Analysen genutzt werden. Unter anderem senden verschiedene Sensoren (z.B. Smart Meters) ihre Messwerte in Form von Events an die CEP-Engine.

Als statistisches Modell zur Analyse und Prognose von *Zeitreihen*, wird ein *Auto Regressive Integrated Moving Average* (ARIMA)-Modell erstellt [BJ70]. Zu der Berechnung eines ARIMA-Modells wird die in [Da15] definierte *heuristische* Strategie verwendet, um automatisch eine Reihe vielversprechender Vorhersagemodelle für die vorliegenden Daten zu identifizieren und um den Menschen als Akteur zu entlasten. In [HK08] wird ein in R

implementierter, heuristischer Algorithmus zur ARIMA-Modellauswahl vorgestellt und in dem Bericht eingesetzt. Informationskriterien wie *Akaike Information Criterion* (AIC) [Ak74] und *Bayesian Information Criterion* (BIC) [Sc78] werten mehrere Modelle aus, indem schrittweise die Modellparameter erhöht werden.

Zur Wartung der Modelle in CEP wird eine Kombination der Strategien *periodisch* und *schwellwertbasiert* aus [Da15] angewendet, die bereits erfolgreich untersucht wurde [Da11]. Zum einen findet periodisch (z.B. täglich) eine Modellberechnung statt. Weiterhin wird ebenfalls erkannt, wenn eine Vorhersage sich von den tatsächlichen Messwerten entfernt. Dafür wird in [Tr13] zum einen eine *fensterbasierte* (manuell) und eine *clusterbasierte* (automatisiert) Änderungserkennung vorgestellt. Die fensterbasierte Methode basiert auf der Untersuchung eines gleitenden Fensters (*Sliding Window*), wodurch der Datenstrom segmentweise untersucht wird. Die clusterbasierte Methode hingegen nutzt ein Clustering-Modell und versucht jedes neu eingetroffene Ereignis einem Cluster zuzuweisen. Unter Berücksichtigung des berechneten Vorhersagefehlers in CEP und der in dem Bericht eingesetzten fensterbasierten Änderungserkennung, wird segmentweise überprüft, ob ein vordefinierter Schwellwert überschritten wurde und ob ein neues Modell berechnet werden muss. Ein Ausblick zur Anwendung einer clusterbasierten Änderungserkennung wird in Abschnitt 7 vorgestellt.

Unterstützend zur Berechnung neuer vorhersagender Modelle werden vorab die historischen Daten mit Hilfe von Data Mining Techniken untersucht (Abb. 1, Offline-Analyse). Als Ergebnis wird ein Clustering-Modell generiert, das im Folgenden als *Referenzmodell* bezeichnet wird. Ein Referenzmodell ist die Zusammenfassung einer Menge von *formähnlichen Zeitreihen*. Das Referenzmodell stellt die repräsentativsten Zeitverläufe (*Referenzzeitreihe*) einer endlichen Menge von Zeitreihendaten dar. Während das Referenzmodell basierend auf den Messwerten erstellt wird, soll zusätzlich ein Klassifikator die Referenzzeitreihen mit Hilfe von z.B. kalendarischen Merkmalen beschreiben. Als Klassifikator wird ein Entscheidungsbaum gewählt, zur Erstellung von bedingten Anweisungen (*Entscheidungsregeln*). Für das eMMS soll zu Beginn eine Entscheidungsregel dabei helfen, die Referenzzeitreihe (basierend auf der zuvor berechneten Clusteranalyse) für den folgenden Tag zu prognostizieren. Basierend auf der Entscheidungsregel werden die entsprechenden formähnlichen Trainingsdaten zur Zeitreihenvorhersage ausgewählt und ein initiales ARIMA-Modell erstellt.

Sobald eine Zeitreihe abschließt, wird das Referenzmodell auf dieser angewendet und mit einem entsprechenden Cluster-Label in der Datenbank persistiert. Dadurch wird sichergestellt, dass stets aktuelle Zeitreihen zur Berechnung von ARIMA-Modellen herangezogen werden. Außerdem können die historischen Daten für nachfolgende Datenanalyseprozesse verwendet werden.

4 Auswertung der historischen Daten mit Data Mining Techniken

Zur Auswertung des entwickelten eMMS wurden drei anonymisierte Stromverbrauchsdaten von der EWE AG³ zur Verfügung gestellt. Tabelle Tab. 1 beschreibt die Menge an aufgezeichneten Messwerten pro Datensatz und die Partitionierung der Daten in *historische* (Analyse mithilfe von Data Mining Techniken) und *Simulationsdaten* (zur Simulation neuer Messwerte und Auswertung des eMMS).

Datensatz	Partitionierung	Menge an Daten
Verbraucher I	Historisch: 01.01.2014 - 31.12.2014	365 Tage
	Simulation: 01.01.2015 - 02.09.2015	245 Tage
Verbraucher II	Historisch: 01.05.2015 - 30.04.2016	365 Tage
	Simulation: 01.05.2016 - 31.12.2016	245 Tage
Verbraucher III	Historisch: 01.01.2015 - 31.12.2015	365 Tage
	Simulation: 01.01.2016 - 01.09.2016	245 Tage

Tab. 1: Übersicht der zur Verfügung stehenden Datensätze

Für die bevorstehende Datenanalyse wird für jeden Verbraucher ein historischer Datensatz im Zeitraum von einem Jahr gewählt (365 Tage). Zur Simulation und Auswertung des entwickelten eMMS steht ein zeitlich aktuellerer Datensatz von 245 Tagen zur Verfügung. Die Messwerte wurden in einem Intervall von fünfzehn Minuten aufgezeichnet. Daraus folgt, dass pro Tag 96 Messwerte für jeden Verbraucher aufgezeichnet wurden.

Im Folgenden werden die Analyseschritte zur Erstellung von *Referenzmodellen* und *Entscheidungsregeln* beschrieben. Die Ergebnisse der Datenanalyse werden im Anschluss in das eMMS übertragen.

4.1 Clusteranalyse

Im Dezenniumbericht [ASW15] beschreiben die Autoren eine spezielle Art von Clustering – das *Zeitreihen-Clustering*. Das Clustering von Zeitreihen wird hauptsächlich zur Entdeckung interessanter Muster in den Zeitreihendatensätzen verwendet. Zum einen dient das Zeitreihen-Clustering zum Auffinden von Mustern, die häufig im Datensatz vorkommen. Zum anderen können ebenfalls Muster erkannt werden, die in den Datensätzen „überraschend“ auftreten. Das Auffinden von Mustern, die „ungewöhnlich“ sind und/oder „überraschend“ auftreten, wird als *Anomalie-Erkennung* bezeichnet.

³ Die EWE AG (ehemals Energieversorgung Weser-Ems) ist ein Versorgungsunternehmen im Bereich Strom, Erdgas, Telekommunikation, Informationstechnologie und Umwelt. Die Software AG und EWE AG sind Forschungspartner des Projektes *enera*, zur Erstellung einer neuen Modellregion für die Energiewende. Siehe <http://energie-vernetzen.de/>

Das Zeitreihen-Clustering dient zur Erstellung von sogenannten Referenzmodellen, die für das eMMS benötigt werden.

Definition 4.1 (Referenzmodell)

Ein Referenzmodell ist die Zusammenfassung einer Menge von formähnlichen Zeitreihen (als Resultat eines Clusterings). Das Referenzmodell stellt die repräsentativsten Zeitverläufe einer endlichen Menge von Zeitreihendaten dar. Jede der Zeitreihen aus der Grundgesamtheit der Daten wird einem der Cluster des Referenzmodells zugeordnet.

Definition 4.2 (Referenzzeitreihe)

Eine Referenzzeitreihe repräsentiert den Zeitverlauf einer Menge von formähnlichen Zeitreihen. Eine Referenzzeitreihe beschreibt das Zentrum eines Clusters, resultierend aus einem Clustering.

Zum Clustern von Zeitreihen und der Generierung von Referenzmodellen wird ein Trainingsdatensatz von einem Jahr gewählt (Tab. 1, historischer Datensatz). Zum Clustern der Messwerte (in Form von Zeitreihen) können sowohl partitionierende (zum Beispiel *k-Means*) als auch hierarchische Verfahren angewendet werden [ASW15, S. 26]. Für die vorliegende Arbeit wird der *k-Means* Algorithmus angewendet, der bereits in anderen Publikationen großen Zuspruch für Zeitreihen-Clustering genießt [FRB98; Ma67] und sehr leistungsstark ist [BFR98].

Aus dem *k-Means* Clustering resultiert ein Referenzmodell mit *k* Referenzzeitreihen. Um die geeignete Anzahl von *k* Clustern im Datensatz zu ermitteln, wird zur Orientierung die *Ellbogen-Methode* angewendet [HKP11, S. 486].

Nach der Untersuchung einer geeigneten Clustergröße *k* zeigt Abb. 2 folgende Referenzmodelle für Verbraucher I (links), II (mitte) und III (rechts).

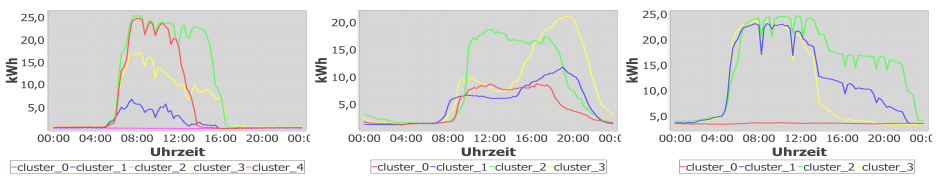


Abb. 2: Referenzmodelle basierend auf historischen Daten

Aus dem Referenzmodell von Verbraucher I und III ist anhand der durchgängigen Linie nahe Null zu erkennen, dass an manchen Tagen kein Strom bezogen wird. Lediglich Verbraucher II bezieht regelmäßig Strom.

Abb. 3 zeigt ebenfalls die Mengenverteilung der jeweiligen Cluster als Kuchendiagramm. Anhand der Clustergröße lässt sich herleiten, dass die größeren Cluster vermutlich einen normalen Zustand beschreiben und die kleineren Cluster auf eine mögliche Anomalie hinweisen. Demnach können für Verbraucher I die Referenzzeitreihen *cluster_1* und *cluster_3*

auf eine Anomalie des Verbraucherverhaltens hinweisen, bei Verbraucher II eventuell *cluster_2* und bei Verbraucher III schließlich die Referenzzeitreihe *cluster_3*.

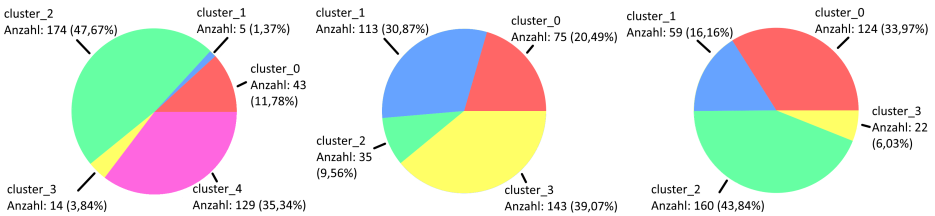


Abb. 3: Kuchendiagramme zur Darstellung der Clustergröße im Referenzmodell

Weiterhin wird mithilfe eines Klassifikators untersucht, welche Merkmale zur Beschreibung und Vorhersage einer Referenzzeitreihe geeignet sind. Die zuvor aufgestellte Hypothese der Anomalie-Cluster wird hierbei unter anderem bestätigt.

4.2 Klassifikation

Für das eMMS soll zu Beginn eine Entscheidungsregel (Klassifikator) dabei helfen, die Referenzzeitreihe (basierend auf der zuvor berechneten Clusteranalyse) für den folgenden Tag zu prognostizieren. Basierend auf der Entscheidungsregel werden die entsprechenden formähnlichen Trainingsdaten zur Zeitreihenvorhersage ausgewählt.

Zur Erstellung eines Klassifikators werden zunächst Merkmale (*Features*) generiert. Im vorliegenden Bericht werden folgende kalendarische Merkmale herangezogen:

Monat, Quartal, Jahreszeit, Wochentag, Arbeitstag und Feiertag.

Mit einer Filterauswahlmethode (z.B. *Forward Selection* oder *Backward Selection*) werden für jeden Verbraucher die bedeutsamsten Merkmale zur Beschreibung und Vorhersage der Referenzzeitreihe ausgewählt [JBB15].

Schließlich wird ein Entscheidungsbaum-Algorithmus, wie z.B. *C4.5* [Qu93] oder *SPRINT* [SAM96], zur Generierung von Entscheidungsregeln eingesetzt. Zur Erstellung und Auswertung der Regeln wird mit der sogenannten *Holdout-Methode* der historische Datensatz in Trainings- und Testdaten partitioniert (75:25) [HKP11, S. 370].

Bei der Klassifikation von Verbraucher I und III konnten für manche Referenzzeitreihen keine Regeln definiert werden und sind in Tab. 2 als „/“ gekennzeichnet. Die Referenzzeitreihen ohne Entscheidungsregeln wurden in Abb. 3 unter Berücksichtigung ihrer Mengenverteilung im Referenzmodell bereits zuvor als mögliche Anomalien in der Zeitreihe erkannt. Zellen mit „-“ wiederum kennzeichnen, dass die entsprechende Cluster-ID für den Datensatz nicht existiert.

Nachdem basierend auf einer Trainingsdatenmenge ein Klassifikator berechnet wurde, wird zur Auswertung der Vorhersagegenauigkeit (*Accuracy*) das Modell auf einer Testmenge ausgeführt. Im Anschluss erfolgt die Ausführung des Modells auf der Gesamtmenge (Training und Test), um die Robustheit⁴ eines Modells zu bewerten. In Tab. 2 ist ebenfalls die Vorhersagegenauigkeit auf dem Testdatensatz sowie auf der Gesamtmenge aufgelistet⁵.

Die Gesamtgenauigkeit in allen drei Verbrauchsdatensätze entspricht nahezu die der Testdatenmenge. Daher können alle drei Klassifikatoren als relativ robust angesehen werden.

Klassifikatoren – Entscheidungsregeln			
Cluster-ID	Verbraucher I	Verbraucher II	Verbraucher III
cluster_0	Freitag	(Q2 OR Q3) AND NOT Arbeitstag	Mittwoch OR Donnerstag
cluster_1	/	(Q2 OR Q3) AND Arbeitstag	Dienstag
cluster_2	Montag OR Dienstag OR Mittwoch OR Donnerstag	(Q1 OR Q4) AND NOT Arbeitstag	Freitag OR Samstag OR Sonntag OR Montag
cluster_3	/	(Q1 OR Q4) AND Arbeitstag	/
cluster_4	Samstag OR Sonntag	–	–
Accuracy Testdaten	89,13%	88,04%	77,17%
Accuracy Gesamt	84,93%	86,3%	78,63%

Tab. 2: Entscheidungsregeln basierend auf der Ausgabe der Clusteranalyse und ihre Genauigkeit

5 Echtzeitfähiges Model Management System

Das eMMS ist in zwei voneinander losgelöste Prozesse unterteilt: *Offline-* und *Online-Analyse*. Die Offline-Analyse beschreibt den bereits in Abschnitt 4 vorgestellten Prozess zur Datenanalyse historischer Daten mit Data Mining Techniken. Die Online-Analyse wiederum umfasst sowohl die Berechnung von Modellen zur Zeitreihenvorhersage als auch deren Auswertung und Austausch in Echtzeit.

Im Folgenden wird zunächst die Architektur des eMMS vorgestellt und im Anschluss der Prozess zur Online-Analyse erläutert.

⁴ Die Robustheit eines Modells beschreibt, dass auch bei verrauschten Daten oder Daten mit fehlenden Werten korrekte Vorhersagen getroffen werden [HKP11, S. 369].

⁵ Bezeichnung 'Q' in Tab. 2 steht für 'Quartal'

5.1 Architektur

Die einzelnen Komponenten der Architektur werden in *Online* und *Offline* unterteilt und sind in Abb. 4 dargestellt.

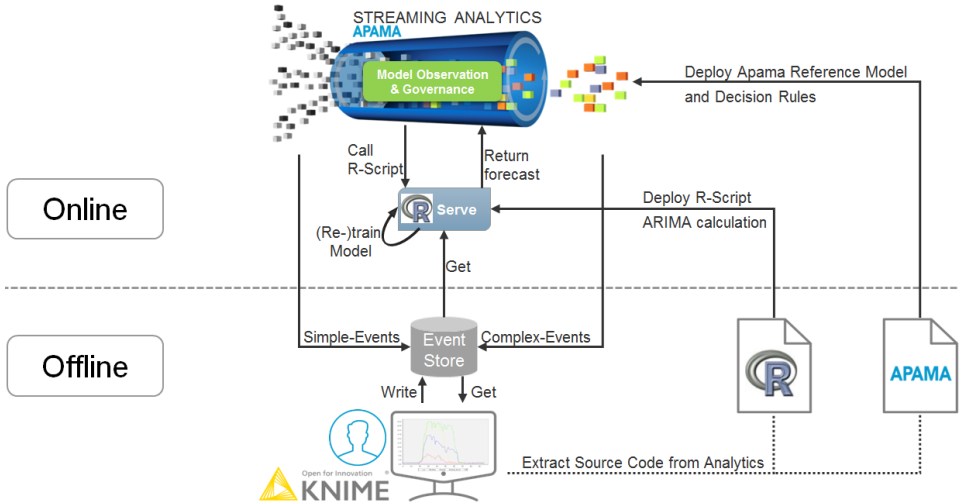


Abb. 4: Architektur eines echtzeitfähigen Model Management Systems

Zu den Online-Komponenten gehört eine CEP-Engine (hier *Apama*⁶) und *Rserve*⁷. *Rserve* ist ein TCP/IP-Server, mit dessen Hilfe andere Programme (hier CEP-Engine) statistische Berechnungen in der Programmiersprache *R* ausführen können [Ur03].

Die CEP-Engine hat die Aufgabe, die Flut an Datenströmen in Echtzeit zu überwachen und Berechnungen auf Basis von Events durchzuführen. Als wesentlichen Bestandteil des eMMS ist die CEP-Engine verantwortlich zur Auswertung von Zeitreihenprognosen und involviert im Prozess zur Berechnung neuer ARIMA-Modelle. Die Berechnungen eines ARIMA-Modelles laufen auf dem *Rserve*. Die Aufrufe des R-Skripts werden von der CEP-Engine via TCP realisiert. Das hinterlegte R-Skript umfasst die zuvor vorgestellte heuristische Methode nach [HK08]. Zur Modellgenerierung wird von der CEP die erwartete Referenzzeitreihe (Cluster-ID) übergeben. Basierend auf dieser Information werden zur Modellberechnung nur Trainingsdaten hinzugezogen, die der gleichen Referenzzeitreihe zugehören. Das Modell mit dem besten Informationskriterium wird daraufhin automatisiert ausgewählt und der CEP-Engine in Form einer prognostizierten Zeitreihe übergeben.

Das DBMS (hier *Event Store*) und eine Analytik-Plattform (hier *KNIME*⁸) gehören zu

⁶ Informationen und eine kostenfreie Version zum Download zur Streaming Analytic Platform Apama der Software AG, siehe <http://www.apamacommunity.com/>.

⁷ Information und Download, siehe <https://www.rforge.net/Rserve/>.

⁸ Konstanz Information Miner (KNIME) ist eine freie Software für die interaktive Datenanalyse. Weitere Informationen und Download, siehe <http://knime.org/>.

den Offline-Komponenten. Der Event Store beinhaltet sowohl die historischen Messwerte, die analysiert werden sollen, als auch Messwerte für die Simulation und Auswertung des eMMS. Sowohl der Rserve als auch die Analytik-Plattform beziehen ihre Daten aus dem Event Store. Die historisch hinterlegten Zeitreihen werden visuell aufbereitet und mit Hilfe von statistischen Verfahren aus Abschnitt 4 analysiert. Die Erkenntnisse aus der Datenanalyse werden zur Erstellung der R-Skripte und CEP-Anfragen übertragen und zur Laufzeit ausgeführt. Weiterhin können von der CEP-Engine aggregierte, verarbeitete Ereignisse (z.B. die Beschriftung einer neuen Zeitreihe) im Event-Store gespeichert werden.

5.2 Online-Analyse

Die Online-Analyse beschreibt den Wartungsprozess des ausführenden vorhersagenden Modells. Zur Wartung wird eine Kombination der Strategien *periodisch* und *schwelligwertbasiert* angewendet (vgl. Abschnitt 3). Neben der täglichen Modellberechnung wird ebenfalls unter Berücksichtigung des Vorhersagefehler ein neues Modell angefordert und gegebenenfalls neu berechnet. Der Vorhersagefehler (engl.: *Forecast Error* (FE)) e_i beschreibt allgemein die Differenz zwischen dem beobachteten und vorhergesagten Messwert. Entspricht y_i dem beobachteten und \hat{y}_i dem vorhergesagten Wert, dann gilt:

$$e_i = y_i - \hat{y}_i \quad (1)$$

In der Echtzeit-Modellvalidierung wird der FE von der CEP-Engine stets beobachtet. Um eine Modellabweichung zu detektieren, wird eine Concept Drift Anfrage formuliert.

Definition 5.1 (Concept Drift) *Im Bereich des maschinellen Lernens bezeichnet der Ausdruck Concept Drift das Phänomen, dass die Trainingsdaten nicht mit dem aktuellen Anwendungsfall übereinstimmen [Ts04; Zl10]. Das zugrundeliegende Konzept zur Vorhersage der Daten weicht von der Realität ab, wodurch falsche Vorhersagen von ausführenden Modellen getroffen werden. Daher muss das vorhersagende Modell regelmäßig an das neueste Konzept angepasst werden [Ha16].*

Durch die Berechnung des Vorhersagefehlers FE wird von der CEP-Engine ein neuer Datenstrom erzeugt. Dieser wird nach einer fensterbasierten Änderungserkennung untersucht (vgl. Abschnitt 3). Ein gleitendes Fenster mit einer vordefinierten Fenstergröße und einem Grenzwert für FE beschreiben eine Concept Drift Erkennung, wann die Vorhersage nicht mehr den Anforderungen genügen und eine Modelländerung womöglich erforderlich ist.

Nachdem ein Concept Drift erkannt wurde, wird daraufhin validiert, ob eine Modelländerung angebracht ist. Hierfür wird die Vorhersagegenauigkeit *Mean Absolute Error* (MAE) der gesamten, bisherigen Zeitreihe mit den Referenzzeitreihen des entsprechenden Referenzmodells berechnet. Der MAE beschreibt den durchschnittlichen, absoluten Fehler und wird wie folgt definiert:

$$MAE = \frac{1}{n} \sum_{i=0}^n |e_i| \quad (2)$$

Die Referenzzeitreihe mit dem kleinsten Fehler wird als geeignetes Modell ausgewählt. Entspricht die Beschreibung des ausführenden Modells nicht der Referenzzeitreihe mit dem kleinsten Fehler, dann wird ein neues ARIMA-Modell berechnet. Andernfalls würde ein unnötiger Rechenaufwand entstehen, der damit vermieden wird. Zur Neuberechnung eines ARIMA-Modells wird nun ein Trainingsdatensatz der vier aktuellsten Zeitreihen⁹ aus der Gruppe der Referenzzeitreihe mit dem kleinsten Fehler gewählt. Das neue Modell wird in das CEP-System geladen und weiterhin nach einem möglichen Concept Drift untersucht.

Folgend veranschaulicht Abb. 5 das Konzept der Modelländerung nach einem Concept Drift Alarm und das Heranziehen einer Referenzzeitreihe zur Unterstützung der Neuberechnung eines ARIMA-Modells.

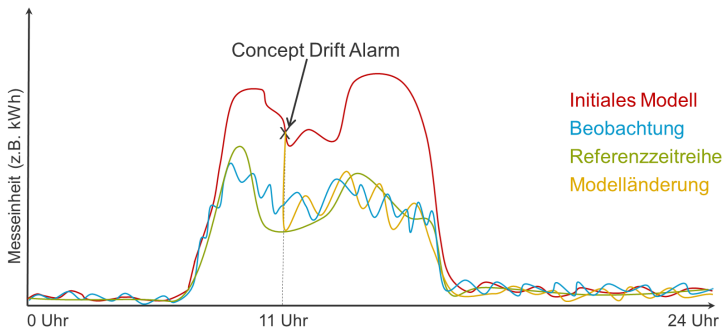


Abb. 5: Illustration einer Modelländerung nach einem Concept Drift

Die Illustration zeigt das initiale Modell mit der Zeitreihenvorhersage des gesamten Tages (rote, obere Kurve). Im Verlauf des Tages trifft der tatsächlich beobachtete Verbrauch ein (blaue, stark schwankende Kurve) und berechnet die Differenz von der Vorhersage und der Beobachtung. Der Concept Drift Alarm wird ausgelöst, wenn die Differenz zu hoch ist und eine vorgegebene Zeitspanne überdauert. Aus dem Referenzmodell wird eine Referenzzeitreihe ausgewählt, die zum Zeitpunkt des Alarms (hier 11 Uhr) den momentanen Verbrauch am besten beschreibt (grüne Kurve). Basierend auf der vorzeitigen Clustering-Anwendung um 11 Uhr, wird ein neuer Trainingsdatensatz zur Neuberechnung eines Modells herangezogen, das den zu erwartenden Verbrauch ab sofort besser beschreiben soll (orangefarbene Kurve). Während die Illustration zeigt, dass das initiale Modell sich im Laufe des Tages immer mehr vom tatsächlichen Verbrauch entfernt (engl.: *drift*), weist das neu generierte Modell ab 11 Uhr tatsächlich eine höhere Übereinstimmung mit dem aktuellen Anwendungsfall auf.

⁹ Das Auswahlkriterium „vier aktuellste Zeitreihen“ basiert auf den Erfahrungswerten von Experten der EWE AG.

Damit die neue abgeschlossene Zeitreihe ebenfalls im DBMS eine Clusterbezeichnung erhält, wird das Referenzmodell angewendet. Dafür wird die Vorhersagegenauigkeit MAE der Referenzzeitreihen auf die abgeschlossene Zeitreihe berechnet. Die Referenzzeitreihe mit dem kleinsten Fehler wird für das Clustering ausgewählt und die entsprechende Cluster-ID im DBMS eingetragen. Dadurch wird gewährleistet, dass für die nächste ARIMA-Modellberechnung stets die aktuellsten Zeitreihen als Trainingsdatensatz ausgewählt werden.

6 Auswertung

Zur Auswertung des eMMS wird im vorliegenden Bericht die Genauigkeit der Zeitreihenvorhersage des Folgetages über den gesamten Simulationsdatensatz berechnet. In [He18] sind die Ergebnisse zusätzlich gruppiert nach der Cluster-ID aufgelistet. Weiterhin wird die Vorhersage der auf historischen Daten basierenden Entscheidungsregeln auf dem Simulationsdatensatz validiert.

Das eMMS, das zur Laufzeit auf Änderungserkennungen mit einer Modelländerung (im Folgenden „*Change-Modell*“ genannt) reagieren kann, wird neben dem initialen ARIMA-Modell (im Folgenden „*Day-Ahead-Modell*“ genannt) und dem erzeugten „*Referenzmodell*“ mit zwei weiteren Modellen verglichen: „*Mean*“ und „*Naiv-Modell*“.

Im Buch [HA18] werden diese Verfahren als Benchmark zum Vergleich von Vorhersagemodelle verwendet. Die untersuchten Modelle werden wie folgt zusammengefasst:

Referenzmodell Ergebnis des Zeitreihen-Clustering aus Abschnitt 4.1.

Day-Ahead Initiales ARIMA-Modell zur Vorhersage des Stromverbrauchs für den Folgetag, berechnet nach dem Verfahren aus [HK08].

Change Modellanpassung durch Beobachtung des Vorhersagefehlers und Concept Drift Erkennung.

Mean Die Vorhersage aller zu prognostizierenden Werte gleicht dem Durchschnitt der historischen (Trainings-) Daten. Wenn $\{y_1 + \dots + y_T\}$ die historischen Daten bezeichnet, dann wird die Vorhersage wie folgt definiert:

$$y_{T+h} = \bar{y} = (y_1 + \dots + y_T)/T \quad (3)$$

Die Prognose wird als y_{T+h} bezeichnet und basiert auf den Daten $\{y_1 + \dots + y_T\}$.

Naiv Die gesamte Vorhersage gleicht der letzten Beobachtung. Demnach gilt:

$$\hat{y}_{T+h} = y_T. \quad (4)$$

Tab. 3 zeigt die Auswertung des eMMS mit Modelländerung (*Change*) im Vergleich mit dem initialen Modell (*Day-Ahead*). Die zwei einfachen Modelle (*Mean* und *Naiv*) und das Referenzmodell dienen als Vergleichsgröße zur Orientierung der Auswertung. In Abschnitt 4.2 wurden bereits die Entscheidungsregeln auf ihre Robustheit geprüft. Zur Bestätigung und finalen Untersuchung des Klassifikators wird ein dritter, eigenständiger Datensatz zur Validierung empfohlen¹⁰ [Bi95; RH96]. Zu diesem Zweck wird die Vorhersagegenauigkeit des Klassifikators aus Abschnitt 4.2 auf dem Simulationsdatensatz validiert und ebenfalls in Tab. 3 präsentiert.

Durchschnitt MAE in kWh & Accuracy der Entscheidungsregeln						
Datensatz	Day-Ahead	Change	Referenzmodell	Mean	Naiv	Accuracy
Verbraucher I	2,41	1,91	1,76	7,82	6,27	85,71%
Verbraucher II	1,81	1,71	1,29	2,9	3,17	54,69%
Verbraucher III	6,34	2,95	4,14	7,62	5,66	11,43%

Tab. 3: Vorhersagegenauigkeit der Modelle und Entscheidungsregeln auf Simulationsdatensatz

Während die Validierung des Klassifikators von Verbraucher I positiv ausfällt (> 80%, ähnlich zu dem Ergebnis aus Abschnitt 4.2), fallen hingegen die Klassifikatoren von Verbraucher II und III deutlich schlechter aus. Wie in Abschnitt 3 beschrieben, haben die Vorhersagen des Klassifikators einen Einfluss auf die Berechnung der Zeitreihenvorhersagen des initialen ARIMA-Modells durch die Auswahl des Trainingsdatensatzes. Eine ARIMA-Modelländerung wird hierbei vermutlich dringend benötigt.

Bei allen drei Datensätzen konnte eine bessere Vorhersagegenauigkeit mit der Modelländerung erzielt werden. Die Auswertungen bei zwei der Verbrauchsdaten ergaben eine geringe Verbesserung, wohingegen beim dritten Datensatz eine erhebliche Besserung nachgewiesen wurde (Vorhersagefehler halbiert).

Aufgrund des Nachweises einer Verschlechterung der Genauigkeit der Entscheidungsregeln für Verbraucher II und III, zeigen weiterführende Untersuchungen in [He18] Erkenntnisse einer Entwicklung des Datenstroms (*Concept Evolution*).

Definition 6.1 (Concept Evolution) *Concept Evolution tritt als Resultat vollkommen neuer Klassenobjekte auf, die sich im Lauf der Zeit im Datenstrom entwickelt haben [Mo10]. Durch die Entwicklung des Konzepts im Datenstrom sind die Anzahl der vorhandenen Klassenobjekte dynamisch. Bei einer unentdeckten Concept Evolution werden bei einer Klassifikation die Instanzen der neuen Klasse fälschlicherweise bestehenden Klassen zugewiesen, wodurch der Klassifikationsfehler zunimmt [Ha16].*

¹⁰ In der Literatur zu maschinellem Lernen wird die Terminologie zu „Validierung“ und „Test“ gegebenenfalls umgekehrt beschrieben.

Bei einer Concept Drift Erkennung müssen *vorhersagende Modelle* (prädiktiv), wie zum Beispiel ARIMA, an das neue Konzepte im Datenstrom angepasst werden. Davon abweichend wird bei einer Concept Evolution Erkennung eine Anpassung von *beschreibenden Modellen* (deskriptiv), wie zum Beispiel dem Referenzmodell oder die Entscheidungsregeln¹¹, benötigt. Sowohl bei der Erkennung eines Concept Drift als auch Concept Evolution muss das ausführende Modell neu trainiert werden.

Zur Untersuchung einer Concept Evolution wird in [He18] auf dem Simulationsdatensatz erneut eine Clusteranalyse und Klassifikation durchgeführt und mit den Ergebnissen basierend auf den historischen Datensatz verglichen. Die Idee liegt hierbei, dass im Zuge einer Concept Evolution Beobachtung geprüft wird, inwiefern sich die Daten im Zeitraum von knapp zwei Jahren (610 Tagen) verändert haben.

Durch den Vergleich stellte sich heraus, dass bei allen drei Datensätzen eine Verschiebung der Clusterzentren festgestellt wurde und dass neu entstandene Klassenobjekte fälschlicherweise den bestehenden Klassen zugeteilt wurden. Weiterhin zeigte sich bei der Untersuchung in [He18], dass nicht die Vorhersagegenauigkeit des Klassifikators entscheidend ist, sondern ob und wie stark der Klassifikator im Zuge einer Concept Evolution sich verändert. Trotz einer Verschiebung der Clusterzentren hat sich der Klassifikator von Verbraucher I und II kaum verändert. Daher sind die Ergebnisse der Zeitreihenvorhersage für Verbraucher II weiterhin positiv, trotz schlechter Accuracy des Klassifikators (54, 69% und nur 1, 81 kWh MAE). Die neuen Klassenobjekte werden zwar einer falschen Klasse zugeordnet, jedoch mit der richtigen Regel zur Modellberechnung abgefragt. Anders waren die Ergebnisse bei Verbraucher III. Hier haben sich auch die Entscheidungsregeln im Zuge der Concept Evolution grundlegend verändert. Dies hatte Auswirkungen auf die Findung des initialen *Day-Ahead* Modells, das zu einem mittleren absoluten Fehler von 6,43 kWh führte. Mit dem im Bericht vorgestellten Verfahren konnte hier der Vorhersagefehler halbiert werden.

7 Zusammenfassung und Ausblick

Zusammenfassend ist das in der Arbeit implementierte eMMS im Stande drei wesentliche Aspekte zu behandeln:

1. Zum einen werden unvorhergesehene Zeitreihen, für die keine Regeln bei der Klassifikation gefunden wurden (Anomalien), frühzeitig mit der Concept Drift Erkennung erfasst. Ein neues vorhersagendes Modell zur Zeitreihenprognosen wird mit Hilfe eines Referenzmodells berechnet.
2. Zum anderen ist das System in dem untersuchten Beispiel bei der Verschiebung von Clusterzentren im Referenzmodell robust, solange die Entscheidungsregeln

¹¹ Der Klassifikator wird im Rahmen des Berichts als beschreibendes Modell bezeichnet, weil das Modell Merkmale extrahiert, die signifikant das Konzept in den Daten beschreiben. Diese Erkenntnisse werden zur anschließenden Zeitreihenvorhersage genutzt.

zur initialen ARIMA-Modellberechnung sich nicht verändert haben. Durch die zur Laufzeit durchgeführte Anwendung des Referenzmodells werden neue Zeitreihen einer bestehenden Referenzzeitreihe zugeordnet und im DBMS vermerkt. Die Auswahl der vier aktuellsten Zeitreihen einer Referenzzeitreihe bei gleichbleibender Regel führt daher zu einer Relativierung des Vorhersagefehlers spätestens nach der vierten Prognose.

3. Schließlich zeigte sich in [He18], dass das System auch im Zuge einer Concept Evolution mit sich ändernden Entscheidungsregeln angemessene Vorhersagen treffen kann.

Der Bericht zeigt, dass die Verwendung des eMMS die Vorhersagequalität des Stromverbrauchs für den Folgetag von allen drei untersuchten Verbrauchern verbessert. Insbesondere wenn die Verbrauchswerte von Tag zu Tag stark schwanken oder eine Concept Evolution in den Daten vorliegt, ist dieses Konzept von Vorteil.

Zum Abschluss soll auf die clusterbasierte Änderungserkennung im Datenstrom aufmerksam gemacht werden. Die in dem Bericht verwendete fensterbasierte Änderungserkennung benötigt Expertenwissen zur Definition der Fenstergröße und des Schwellwertes. Um auch hier den Menschen im eMMS Prozess zu entlasten, empfiehlt sich eine automatisierte Änderungserkennung durchzuführen (vgl. Abschnitt 3). Zum einen muss kein Schwellwert des Vorhersagefehlers vordefiniert werden, sondern ein Concept Drift wird automatisiert erkannt. Zum anderen ließe sich das allgemeine Datenstromverhalten untersuchen und eine Concept Evolution Erkennung implementieren. Des Weiteren könnte eine Änderungsrate für einen Modellwechsel mit hinzugezogen werden, zur Abschätzung der Parametereinstellungen der Concept Drift Erkennung (Fenstergröße und ggf. Schwellwert bei fensterbasierter Strategie). Diese Aspekte werden in dem hier vorgestellten eMMS nicht berücksichtigt und könnten für die Weiterentwicklung aufgenommen werden.

Literatur

- [Ak74] Akaike, H.: A new look at the statistical model identification. *Automatic Control, IEEE Transactions on* 19/6, S. 716–723, 1974.
- [ASW15] Aghabozorgi, S.; Shirkhorshidi, A. S.; Wah, T. Y.: Time-series clustering – A decade review. *Information Systems* 53/, S. 16–38, 2015, ISSN: 0306-4379.
- [Be03] Bernstein, P. A.: Applying Model Management to Classical Meta Data Problems. *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*, 2003.
- [BFR98] Bradley, P. S.; Fayyad, U.; Reina, C.: Scaling Clustering Algorithms to Large Databases. In: *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining. KDD'98*, AAAI Press, New York, NY, S. 9–15, 1998.
- [Bi95] Bishop, C. M.: *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., New York, NY, USA, 1995, ISBN: 0198538642.

- [BJ70] Box, G. E.; Jenkins, G. M.: *Time Series Analysis: Forecasting and Control*. Holden-Day, Inc., San Francisco, CA, USA, 1970.
- [BK09] Buchmann, A. P.; Koldehofe, B.: *Complex Event Processing. it - Information Technology 51/*, S. 241–242, 2009.
- [Da11] Dannecker, L.; Böhm, M.; Lehner, W.; Hackenbroich, G.: *Forecasting Evolving Time Series of Energy Demand and Supply*. In: *Advances in Databases and Information Systems*. Springer Berlin Heidelberg, Berlin, Heidelberg, S. 302–315, 2011, ISBN: 978-3-642-23737-9.
- [Da15] Dannecker, L.: *Energy Time Series Forecasting – Efficient and Accurate Forecasting of Evolving Time Series from the Energy Domain*. Springer Fachmedien Wiesbaden, 2015.
- [DK84] Dolk, D. R.; Konsynski, B. R.: *Knowledge Representation for Model Management Systems*. *IEEE Transactions on Software Engineering SE-10/6*, S. 619–628, Nov. 1984.
- [FRB98] Fayyad, U.; Reina, C.; Bradley, P. S.: *Initialization of Iterative Refinement Clustering Algorithms*. In: *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining. KDD'98*, AAAI Press, New York, NY, S. 194–198, 1998.
- [Ha16] Haque, A.; Khan, L.; Baron, M.; Thuraisingham, B.; Aggarwal, C.: *Efficient handling of concept drift and concept evolution over Stream Data*. In: *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. S. 481–492, Mai 2016.
- [HA18] Hyndman, R. J.; Athanasopoulos, G.: *Forecasting: Principles and Practice*. OTexts, 2018, ISBN: 9780987507112.
- [He18] Hegenbarth, Y.: *Konzept und Implementierung eines echtzeitfähigen Model Management Systems*, Masterarbeit, Hochschule Darmstadt h_da, Germany, 2018.
- [HK08] Hyndman, R.; Khandakar, Y.: *Automatic Time Series Forecasting: The forecast Package for R*. *Journal of Statistical Software, Articles 27/3*, S. 1–22, 2008, ISSN: 1548-7660.
- [HKP11] Han, J.; Kamber, M.; Pei, J.: *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2011, ISBN: 0123814790, 9780123814791.
- [JBB15] Jović, A.; Brkić, K.; Bogunović, N.: *A review of feature selection methods with applications*. In: *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. S. 1200–1205, 2015.
- [LWG14] Liu, J.; Wilson, A.; Gunning, D.: *Workflow-based Human-in-the-Loop Data Analytics*. In: *Proceedings of the 2014 Workshop on Human Centered Big Data Research. HCBDR '14*, ACM, Raleigh, NC, USA, 49:49–49:52, 2014, ISBN: 978-1-4503-2938-5.
- [Ma67] Macqueen, J.: *Some methods for classification and analysis of multivariate observations*. In: *In 5-th Berkeley Symposium on Mathematical Statistics and Probability*. S. 281–297, 1967.
- [Mo10] Mohammad, M. M.; Chen, Q.; Khan, L.; Aggarwal, C.; Gao, J.; Han, J.; Thuraisingham, B.: *Addressing Concept-Evolution in Concept-Drifting Data Streams*. In: *Proceedings of the 2010 IEEE International Conference on Data Mining. ICDM '10*, IEEE Computer Society, Washington, DC, USA, S. 929–934, 2010, ISBN: 978-0-7695-4256-0.
- [Qu93] Quinlan, J. R.: *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993, ISBN: 1-55860-238-0.
- [RH96] Ripley, B.; Hjort, N.: *Pattern Recognition and Neural Networks*. Cambridge University Press, 1996, ISBN: 9780521460866.

- [SAM96] Shafer, J. C.; Agrawal, R.; Mehta, M.: SPRINT: A Scalable Parallel Classifier for Data Mining. In: Proceedings of the 22th International Conference on Very Large Data Bases. VLDB '96, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, S. 544–555, 1996, ISBN: 1-55860-382-4.
- [Sc78] Schwarz, G.: Estimating the Dimension of a Model. *The Annals of Statistics* 6/2, S. 461–464, 1978, URL: <https://projecteuclid.org/euclid.aos/1176344136>.
- [Se16] Self, J. Z.; Vinayagam, R. K.; Fry, J. T.; North, C.: Bridging the Gap Between User Intention and Model Parameters for Human-in-the-loop Data Analytics. In: Proceedings of the Workshop on Human-In-the-Loop Data Analytics. HILDA '16, ACM, San Francisco, California, 3:1–3:6, 2016, ISBN: 978-1-4503-4207-0.
- [Tr13] Tran, D.: Change detection in streaming data, Diss., Technische Universität Ilmenau, Germany, 2013.
- [Ts04] Tsymbal, A.: The Problem of Concept Drift: Definitions and Related Work. In: Department of Computer Science Trinity College Dublin. Ireland, Mai 2004.
- [Ur03] Urbanek, S.: Rserve – A Fast Way to Provide R Functionality to Applications. In: Proceedings of the 3rd International Workshop on Distributed Statistical Computing (DSC 2003). 2003.
- [Va16] Vartak, M.; Subramanyam, H.; Lee, W.-E.; Viswanathan, S.; Husnoo, S.; Madden, S.; Zaharia, M.: ModelDB: a system for machine learning model management. In. HILDA '16 – Proceedings of the Workshop on Human-In-the-Loop Data Analytics, San Francisco, California, USA, S. 10–12, Juni 2016.
- [Zl10] Zliobaite, I.: Learning under Concept Drift: an Overview. CoRR abs/1010.4784/, 2010.

Machine Learning

Machine Learning Applied to the Clerical Task Management Problem in Master Data Management Systems

Lars Bremer¹, Mariya Chkalova², Martin Oberhofer³

Abstract: Clerical tasks are created if a duplicate detection algorithm detects some similarity of records but not enough to allow an auto-merge operation. Data stewards review clerical tasks and make a final non-match or match decision. In this paper we evaluate different machine learning algorithms regarding their accuracy to predict the correct action for a clerical task and execute that action automatically if the prediction has sufficient confidence. This approach reduces the amount of work for data stewards by factors of magnitude.

Keywords: IBM⁴ Master Data Management, MDM, Machine Learning, Random Forest, XGBoosting, Sorted Neighborhood Method, Data Fusion, Matching, Clerical Task Processing, Duplicate Detection

1 Introduction

Enterprises across all industries have a need to manage master data such as customer, supplier, employee, patient, product or contract information. Master Data Management (MDM) is a discipline comprised of people, processes and technology to manage that master data. Data stewards use automated processes to manage data quality for master data to achieve repeatable, consistent data quality outcomes. One of the key data quality requirements for any MDM solution is the ability to detect and resolve duplicate master data records using matching (matching is also known as duplicate detection). Duplicated master data records exist for at least three reasons. The first one is that before companies introduce an MDM system, master data has been created and managed redundantly and inconsistently in many different application systems. When an MDM system is introduced, master data records from these different sources need to be harmonised and deduplicated to create the 360° view (also referred to as golden record) for a master data entity. The second reason is that in certain industries the sources are outside of the company's control. For example, a healthcare insurance company gets patient data from doctors and hospitals where the sourcing IT systems are outside of the company. Consequently, each time new patient information is received, the matching needs to detect duplicates again. The third reason

¹ IBM Research and Development, Schoenaicherstrasse 220, 71032, Boeblingen, Germany, lbremer@de.ibm.com

² IBM Research and Development, Schoenaicherstrasse 220, 71032, Boeblingen, Germany, mariya.chkalova1@ibm.com

³ IBM Research and Development, Schoenaicherstrasse 220, 71032, Boeblingen, Germany, martino@de.ibm.com

⁴ IBM is a trademark of IBM in USA and/or other countries.

is that at the time a new master data record is added to the MDM system, the information might not be complete enough to detect at time of creation that this master data record is a duplicate of some other record and should not be created. Only when a subsequent update adds additional attributes matching might be able to detect that the more complete master data record is now indeed a duplicate to at least one other record and should be merged (also referred in literature as data fusion [BN09]). A typical use case for this is the customer lifecycle. A customer record for a lead is sparsely populated, often times not much more than a name and contact method. As the lead becomes a customer prospect and eventually an active customer additional information like an address, identifiers, etc. are added. Based on this more complete information a match might be found.

For matching, there are two fundamentally different techniques known as deterministic matching and probabilistic matching. Figure 1 shows the basic problem. A base record is compared with pairwise comparison to two other records.

Base Record Indicator	First Name, Middle Name Last Name	Weight	Street, House Number, City, ZIP, Country	Weight	DOB	Weight	Gender	Weight	Identifier	Weight	Total Score
Y	Robert Fisher		Main Street 150 New York 92123, USA		24-10-1975		M		333-22-4444		
N	Bobbie Fisher	3	Main Str. 150 New York 92123, USA	3	24-10-1975	5	F	-0.5	333-22-4444	5	15.5
N	Robert M Fisher	3.3	Main Str. 150 Los Angeles, USA	2.2	24-10-1974	4.5	M	0.5	999-77-8888	-5	5.5



Lower Threshold = 5

Upper Threshold = 17

Fig. 1: Matching Example

Deterministic matching uses a set of static rules to decide whether or not two or more records are similar enough and should be merged into a single record. Probabilistic matching typically uses fuzzy techniques comparing the attributes (or set of attributes together, e.g. like the attributes of an address as shown in Figure 1) of two or more records. Fuzzy techniques include for example edit distance, phonetic similarity, nickname resolution, etc. For each attribute (or set of attributes) a weight is assigned how much it contributes for the total similarity score of a record compared to other records to make a non-match or match decision. For example, a gender attribute which has only two values typically contributes with less weight to the overall decision of the matching in comparison to maybe an identifier field as shown in Figure 1. Weights can be positive or negative depending on the similarity found as shown in the Figure 1 with the values in the identifier column. In addition, probabilistic matching considers frequency distribution. For example, in a typical data set of last names in Germany, the last name Cheng is substantially less frequent than Müller. That means if probabilistic matching finds a similarity in the last name field, the base weight for the last name increases in case Cheng is found whereas in the case of Müller it

decreases to reflect the significance of the value frequency. In a given data set with n records, comparing each record with every other record is not efficient for real-time decision making. Techniques like Sorted Neighbourhood Method (SNM) were introduced in [HS95] and various optimisations have been studied in [HS98] and [BN09]. An evaluation of different techniques like traditional blocking, SNM, adaptive SNM, Q-gram, threshold-based canopy clustering, suffix-array based indexing and more has been done in [Ch12] showing variations in performance, completeness and quality. The basic idea is to create a structure acting as a means to quickly find records which share some similarity so that the matching only needs to compare a small subset of records in a small bucket regarding similarity. A well-configured matching algorithm in an MDM system in production aims for buckets of 200 to 500 records at a maximum to enable match decisions with a response time in milliseconds range.

Once the matching algorithm has compared two records the computed weights in probabilistic matching are summarized to a total score which is compared against two thresholds as shown in Figure 1. If the total score is below the lower threshold, the result is considered a non-match and the records are kept separate. If the total score is above the upper threshold, the records are deemed similar enough that based on a set of predefined survivorship rules the records are automatically merged to create a new golden record. If the total score is between the lower and upper threshold a clerical task is created for a human data steward. For the two records shown in Figure 1 which are compared to the base record, the total scores are within the clerical range. A data steward claiming a clerical task is reviewing the records to make a final decision if they should be merged or kept separate.

In two scenarios this approach with clerical processing to manage data quality does not work well. The first scenario is the initial load of the MDM system and the second scenario is the onboarding of additional sources in sub-sequent deployment phases. In an initial load scenario of medium size with 20,000,000 records with just 1% duplicates in the clerical range 200,000 tasks are created for the data stewards. If a data steward processes 50-200 per day a single data steward would need to work between 1000 to 4000 days until all of them are processed. During that time new clerical tasks are created while the MDM system is used in ongoing mode and other sources are onboarded in a batch load in subsequent deployment phases keeping the task list growing further.

Even if a company hires a data stewardship team to process all these clericals at least several months will pass. This comes at a certain labor cost and not all companies want to invest that much in data quality. We have seen cases where the clerical task list in an MDM system has more than one million pending tasks which is factors of magnitude too large for the data stewardship team to ever catch up and process. In addition to the labor cost consideration, there is also the motivation of the data stewards to be considered. Practice has shown that over time the clerical task processing becomes somewhat repetitive and boring for the data stewards causing frustration with negative impact on performance and hence data quality. If a data steward or a data stewardship team works for a while, a resolution history of clerical tasks is created which contains the matching details and the decision of the data stewards

showing which records have been merged and which ones have been kept separate based on the pairwise comparisons by matching.

The obvious question is why the matching configuration and particularly the thresholds are not adjusted from the beginning so that there are not that many tasks in the clerical range. Companies in certain industries like banking, insurance and healthcare are very reluctant when MDM is introduced to allow a lot of auto-collapse results in fear that there are too many false positives which could cause problems. In banking for example, a false positive caused by an auto-collapse could mean that the related financial accounts are linked to the golden record and someone sees someones else banking details which should not happen. So the upper threshold is set high enough so that not a lot of auto-collapse cases occur. On the other end of the spectrum, enterprises want to avoid issues with too many false negatives which could become problematic in case compliance with anti-money laundering regulations (e.g. like the 4th Anti-Money Laundering Directive [EUAML15]) are required and hence set the lower threshold often times too low. As a result, many match results end up in the clerical range causing the previously outlined issues.

In this paper we want to study how machine learning can be used to reduce the clerical task processing problem.

2 Approach

In this section we describe the approach we have taken to explore how machine learning can be applied to the matching and clerical task processing problem. We applied machine learning to the resolution history of clerical tasks. The basic idea is to train a machine learning model on the resolution history, so that if a new clerical task comes in, the trained model is used to predict the appropriate data steward action. If the trained machine learning model is above a configurable confidence threshold, the predicted action (non-match or auto-collapse) is taken automatically which means the clerical task is not routed to a data steward anymore reducing the workload for a data steward (or the data stewardship team).

Since this was a new area for us, we decided to do an empirical study of different machine learning algorithms to compare them in terms of quality of results as well as runtime characteristics such as performance, memory and CPU consumption. We tested with machine learning algorithms such as Random Forest [Xu17] and [SBP17], XGBoosting, KNeighbors [SBP17], Gradient Boosting, AdaBoost [FS95], Multi-layer Perceptron, Decision Tree, Logistic Regression, Quadrant Discrete Analysis, Gaussian Naive Bayes, Logistic Regression and Support Vector Machines [SBP17].

3 Architecture

The evaluation of machine learning for the use case was not only about finding out if machine learning yields good results - the evaluation also needed to show if a machine

learning component could be added to an MDM system for production use. Figure 2 shows a simplified view of the architecture for an MDM system limited to the areas relevant for this discussion (for a full discussion of all aspects of an MDM system see [Dr08] and [Ob14]). Consumption of an MDM system is through the services layer. The batch and delta-batch processing for initial and delta-loads is typically just a multi-threaded wrapper around the services layer to ensure that for real-time and batch consumption for data consistency reasons the same code is used. The services providing create, update, delete, read and search functionality are used to read and write data to the persistency layer. As part of the create and update services, probabilistic matching is triggered (in the create case that happens all the time, in update cases only if matching relevant attributes are changed). As outcome of matching clerical tasks might be created which data stewards can process through a data stewardship user interface (UI). Every time a clerical task is finished, a new entry to the resolution history is written.

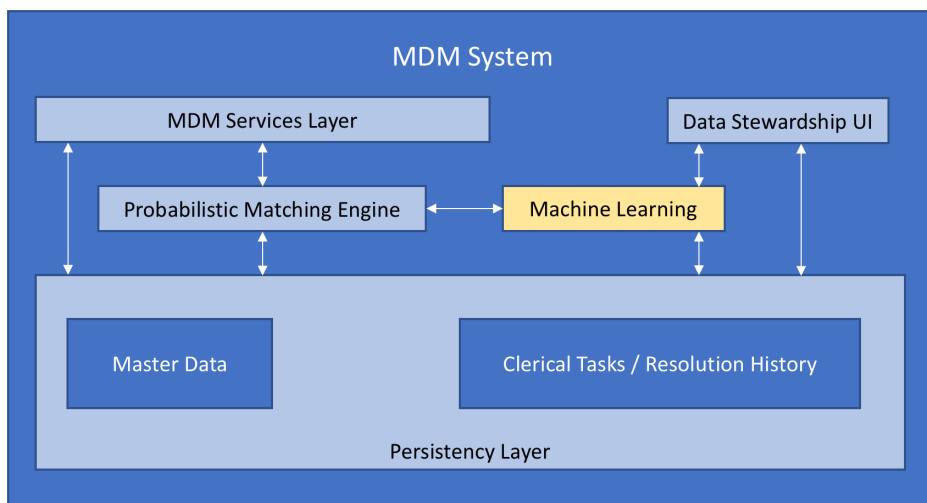


Fig. 2: Target Solution Architecture

Our goal was thus not only to find out if a particular machine learning method is useful for one or both use cases. We wanted to analyze if a trained model can score fast enough to be useful for the first use case of predicting the right action for a new clerical task. MDM systems in production typically have service level agreements against the services layer where the entire service call needs to finish in double-digit milliseconds at most including matching. As a result, the objective for us was if the trained model used for scoring clerical tasks can perform that action in a single to very low double-digit millisecond range at most. Only if that performance target is accomplished the scoring for the prediction can be part of an end-to-end synchronous processing model, otherwise the scoring for the prediction with the auto-resolution need to be asynchronously to the service call. In addition to performance, the scoring needs to scale. In the most demanding operational environments for an MDM

system, the services layer needs to deal with thousands of concurrent calls per second while not compromising on performance. Deploying a trained model for scoring can be done on different technology stacks and we explored some of them to find an answer for the scalability question as well.

4 Implementation

This section describes the implementation and development environment we used to implement different data pre-processing steps as well as to evaluate different machine learning techniques.

We used the Jupyter notebooks in IBM Watson⁵ Studio for data pre-processing, training, and evaluation. This allowed a collaborative approach to work on machine learning with multiple developers. As runtime we used Python with the scikit-learn within Watson Studio.

4.1 Training Data

The data we used for training is structured as shown in Table 1. Each line contains one clerical task which is the result of a comparison of two person records by the matching engine using a probabilistic matching algorithm. Each of these clerical tasks contains the numerical results of different attribute comparisons. The training data contains n of these comparison results. The actual number depends on the configuration used by a customer. Each of these values is one feature f_i we are using for machine learning. The higher the value, the higher the likelihood of the compared attributes being the same of the two person records. The label l is the decision ('same' or 'different') the steward took to resolve the clerical task.

l	f_0	f_1	...	f_n
S	2.3	12.9	...	1.2
D	3.1	0.0	...	1.1

Tab. 1: Original Training Data

4.2 Data Pre-Processing

Our training data is retrieved from a clerical task resolution history which contains a very high data quality. There is no need to clean the data before training can be started. However, the clerical task resolution history we are using as training data is skewed. Of all

⁵ IBM Watson is a trademark of IBM in USA and/or other countries. IBM Watson Studio documentation can be found here: <https://dataplatform.cloud.ibm.com/docs/content/getting-started/overview-ws.html>.

decisions around 75% of the tasks were resolved by data stewards with the decision that the two compared user records are the same. Only in the remaining 25% they are considered different.

We evaluated different sampling methods to balance the data. Among these methods were a simple random oversampler as well as four Synthetic Minority Oversampling (SMOTE) methods as described in [Ch02] and [HWM05]. We used different classifiers for the evaluation. The average F1 score and the area under the ROC curve for the sampling methods is shown in Figure 3.

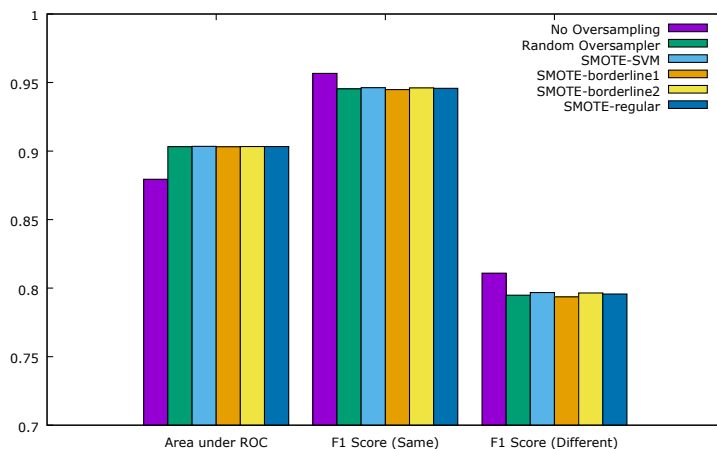


Fig. 3: Sampling Comparison

All sampling methods showed very similar results. They all reduced the F1 score for the minority class. The major reason for this is a reduction of about 10% on average of the precision of the minority class when using oversampling. However, we identified the area under the ROC curve as well as the precision for the majority class (same) increased. This is important for our use case as it is expensive to split user records that were previously merged into one. For that reason, we decided to use random oversampling in the remainder of the evaluations in this paper.

4.3 Performance

For real-time use cases we require high scalability as well as fast response times for predictions. Therefore, we evaluated Spark, Python with Scikit-learn, and MLeap⁶ for prediction performance on Random Forest. MLeap allows to serialize Spark, Tensorflow, and Scikit-learn pipelines. The serialized pipelines can then be used for predictions. MLeap

⁶ <http://mleap-docs.combust.ml/>

removes the need for a heavy engine like Spark. The performance of MLeap for predictions and memory consumption reflected this in our tests.

In contrast, Spark showed much higher response times and memory consumption. We reduced this by moving away from running predictions on a Spark cluster but instead in a Spark local instance that is running in the same Java Virtual Machine as our product code. This reduced the response times significantly down to a minimum response time of less than 20ms and a mean of about 40ms. These responses don't change much if executing just one prediction or passing in 1,000 predictions in one batch.

Predictions with Python using Scikit-learn libraries were even faster. A single prediction was returned in about 2ms in our environment. When batching predictions, however, the response time increased using Python to about 15ms for 1000 predictions which is close to the performance we experienced with Spark.

With Python we were able to reach a throughput of 1,500 requests per second each returning a single prediction. These performance evaluations were executed on a single virtual machine with eight 2.4GHz cores and 16GB of memory.

5 Evaluation

This section summarizes the outcome of our comparisons of different machine learning classifiers with pre-processed data as described in Section 4.2. Our comparison includes 10 different classifiers. We compared these classifiers with regards to the area under the ROC curve, accuracy, precision, recall and F1 score.

We started our evaluations with a comparison of ten different classifiers for machine learning to get an overview of their prediction quality. The results of this comparison are depicted in Figure 4. The figure shows the comparison of all ten classifiers with regards to the area under the ROC curve and the F1 score for both labels. In all of these metrics, XGBoosting showed the best results, followed only by a small margin by Random Forest. The difference between XGBoosting and Random Forest is 0.4% for the area under ROC curve and 0.06% and 0.3% for the F1 for the two classes 'same' and 'different' respectively. Other classifiers clearly fall short in comparison.

For a more detailed analysis we moved forward focussing on XGBoosting, Random Forest, and Logistic Regression. We decided on XGBoosting and Random Forest as the best performers in our initial tests. We included Logistic Regression as a reference point as being one of the most widely used classification algorithms.

The probabilistic score we are using for training the models reaches from a negative rational number to a positive. Zero, however, describes the situation when one of the compared entities does not contain the specific attribute that is compared. For example, compare the social security number of two persons. If at least one of the two person records is missing

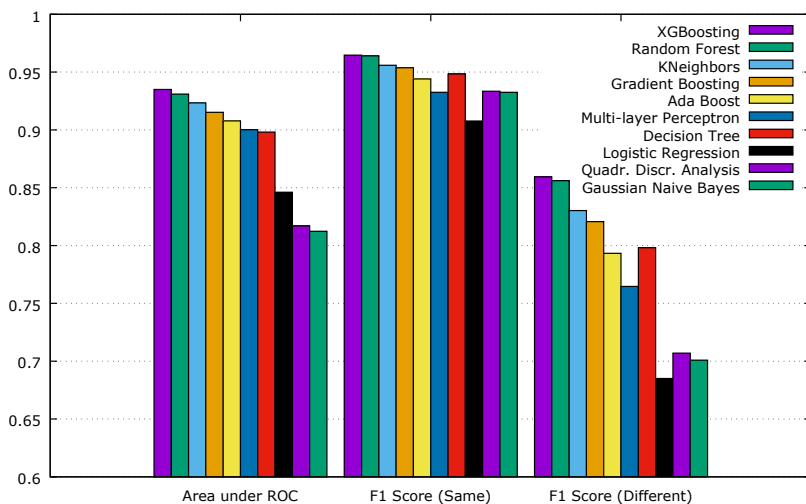


Fig. 4: Classifier Comparison

this number, the comparison value is zero. This means a value of zero does not describe it is more likely than a negative score to be a match or less likely than a positive. Instead, zero values describe that there is insufficient data to define a score for this comparison. To allow different classifiers to compensate for this we introduced a new column f'_n for each feature f_n . This column contains two classes: '0' for when the corresponding matching value equals zero and '1' for non-zero matching values as illustrated in Tables 2 and 3.

f_0	f_1
2.3	1.2
3.1	0.0

Tab. 2: Original Training Data

f_0	f'_0	f_1	f'_1
2.3	1	1.2	1
3.1	1	0.0	0

Tab. 3: Training Data with Artificial Features

Some classifiers trained with these artificial features improved the quality of predictions significantly whereas others showed only very minor improvements as illustrated in Figure 5. The figure compares the F1 score for XGBoosting, Random Forest and Logistic Regression. The quality of predictions of the tree-based classifiers Random Forest and XGBoosting did not change. However, logistic regression improved about 6% for the the majority class 'same' and 11% for the minority class 'different'.

The tree-based classifiers split the branches in a way that they identify the zero values as being different already. This is not possible with logistic regression. For that reason, Logistic Regression benefits from the additional columns. In the remainder of this document all

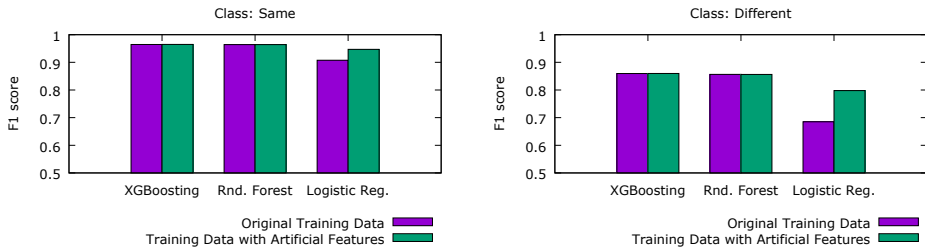


Fig. 5: Evaluation of Artificial Feature Column

results are from classifiers trained with the artificial features. However, for Random Forest and XGBoosting our findings indicate that this is an optional step.

While the quality of predictions is an important metric to compare machine learning classifiers another important aspect is runtime performance.

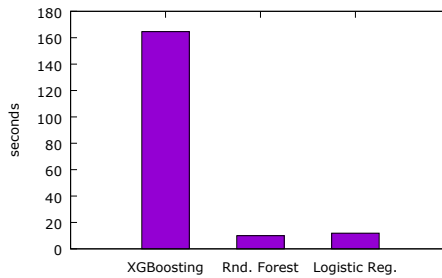


Fig. 6: Training Duration

In the matching use case with classifiers tuned for best prediction quality we observed very different training speeds as shown in Figure 6. Random Forest and Logistic Regression trained a model with about 1 million data records within about 10 seconds using Scikit-Learn libraries. XGBoosting performed much slower. Training the same amount of data takes more than 150 seconds on a single virtual machine with eight 2.4GHz cores and 16GB of memory.

The full comparison of XGBoosting, Random Forest, and Logistic Regression can be found in Figure 7. This shows how similar XGBoosting and Random Forest are for the use case described in this paper.

6 Conclusion

We have shown how historic matching data can be used to make predictions on a steward's matching decisions. We found that with the right machine learning classifier and good input

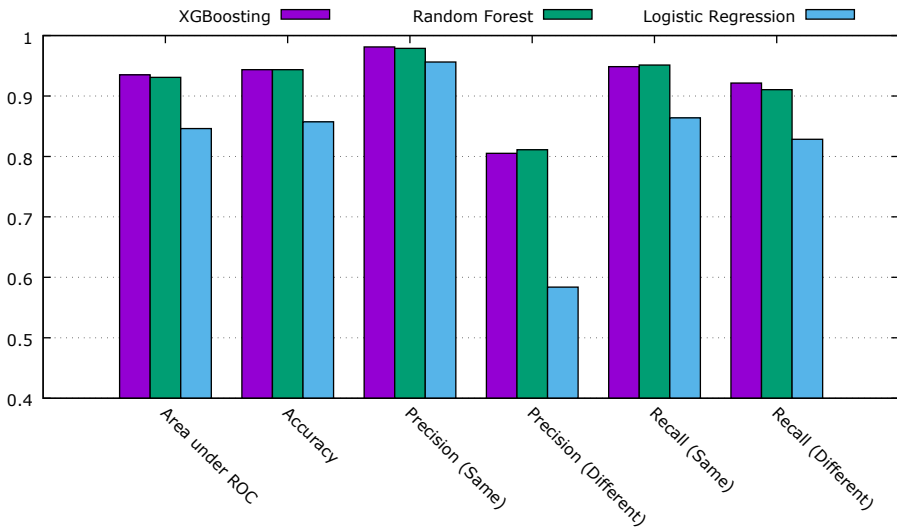


Fig. 7: Full Comparison

data it is possible to predict the steward's decision with an accuracy of over 94%. The precision for identifying two records as being the same can reach up to over 98%. This is particularly important as these predictions lead to a merge of two records. In case of an error it is very costly to divide these records again. Almost all used metrics show results above 90% with the exception precision in the minority class. The precision in this case is only a bit above 80%. Interestingly, this is the only metric in which Random Forest outperforms XGBoosting.

These results were possible after evaluating ten different classifiers and understanding the input data. We realized that artificial columns for zero valued features can be helpful as well as decided to use random oversampling to improve the quality of the predictions.

Finally, we identified Random Forest and XGBoosting as best fitted for the given use case. The prediction results of both classifiers are very similar. However, Random Forest outperformed XGBoosting related to training speed and appears to be the best option for the given scenario.

7 Outlook

One field of further investigations is understanding how the tree-based algorithms prioritize features with regards to their complexity. The score for a name comparison for instance can contain many different individual values because names can be different on many levels

(e.g., spelling, phonetic distance). A birth date, however, can only be different by a certain number of days. We will investigate further how these different distributions of values impact the machine learning classifiers.

A second field for further investigations is to understand how a neural network compares to the classifiers used in this work.

A third field which requires further research is if a probabilistic matching algorithm configuration related to its weights and thresholds can be optimized through machine learning approach such as logistic regression.

References

- [BN09] Bleiholder, J.; Naumann, F.: Data Fusion. In: ACM Computing Surveys (CSUR) Surveys Homepage table of contents archive Volume 41 Issue 1. ACM New York, New York, NY, USA, 2009, URL: <http://dx.doi.org/10.1145/1456650.1456651>.
- [Ch02] Chawla, N. V.; Bowyer, K. W.; Hall, L. O.; Kegelmeyer, W. P.: SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research* 16/, pp. 321–357, 2002.
- [Ch12] Christen, P.: A Survey of Indexing Techniques for Scalable Record Linkage and Deduplication. In: *IEEE Transactions on Knowledge and Data Engineering* Volume 24 Issue 9. IEEE, pp. 1537–1555, 2012, URL: <http://dx.doi.org/10.1109/TKDE.2011.127>.
- [Dr08] Dreibelbis, A.; Hechler, E.; Milman, I.; Oberhofer, M.; Run, P. v.; Wolfson, D.: *Enterprise Master Data Management (Paperback): An SOA Approach to Managing Core Information*. IBM Press, 2008, ISBN: 978-0132366250.
- [EUAML15] DIRECTIVE (EU) 2015/849 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL, 2015, URL: eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=OJ:L_2015_141_R_0003&from=ES, visited on: 09/30/2018.
- [FS95] Freund, Y.; Schapire, R. E.: A decision-theoretic generalization of on-line learning and an application to boosting. In: *Computational Learning Theory. EuroCOLT 1995. Lecture Notes in Computer Science (Lecture Notes in Artificial Intelligence)*, vol 904. Springer, Berlin, Heidelberg, Germany, pp. 20–23, 1995, URL: https://doi.org/10.1007/3-540-59119-2_166.
- [HS95] Hernández, M.; Stolfo, S.: The merge/purge problem for large databases. In: *Proceedings of the 1995 ACM SIGMOD International Conference on Management of data*. Pp. 127–138, 1995.
- [HS98] Hernández, M.; Stolfo, S.: Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem. 2/, pp. 9–37, Jan. 1998.
- [HWM05] Han, H.; Wang, W.-Y.; Mao, B.-H.: Borderline-SMOTE: A New Over-sampling Method in Imbalanced Data Sets Learning. In: *Proceedings of the 2005 International Conference on Advances in Intelligent Computing - Volume Part I. ICIC'05*, Springer-Verlag, Hefei, China, pp. 878–887, 2005, URL: http://dx.doi.org/10.1007/11538059_91.

- [Ob14] Oberhofer, M.; Hechler, E.; Milman, I.; Schumacher, S.; Wolfson, D.: Beyond Big Data: Using Social MDM to Drive Deep Customer Insight. IBM Press, 2014, ISBN: 978-0133509809.
- [SBP17] Sasikala, B. S.; Biju, V. G.; Prashanth, C. M.: Kappa and accuracy evaluations of machine learning classifiers. In: 2017 2nd IEEE International Conference on Recent Trends in Electronics, Information and Communication Technology (RTEICT). Pp. 20–23, 2017.
- [Xu17] Xu, Y.: Research and implementation of improved random forest algorithm based on Spark. In: 2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA). Pp. 499–503, 2017.

Challenges in Data Processing

Ganzheitliches Metadatenmanagement im Data Lake: Anforderungen, IT-Werkzeuge und Herausforderungen in der Praxis

Christoph Gröger,¹ Eva Hoos²

Abstract: Data Lakes haben sich in der industriellen Praxis als Plattformen für die Speicherung und Analyse aller Arten von (Roh-)daten etabliert. Erweiterte Anforderungen hinsichtlich Governance und Self-Service machen das Metadatenmanagement im Data Lake zum kritischen Erfolgsfaktor. Bisher gibt es dazu jedoch nur wenige wissenschaftliche Arbeiten, es mangelt insbesondere an einer ganzheitlichen Betrachtung zur Konzeption und Realisierung des Metadatenmanagements im Data Lake. Diese Arbeit adressiert das Thema und basiert auf praktischen Erfahrungen aus einem Industriekonzern beim Aufbau eines unternehmensweiten Data Lake. Es werden praktische Anforderungen und Anwendungsbeispiele für das Metadatenmanagement im Data Lake diskutiert und die unterschiedlichen Arten von Metadaten anhand des Praxisbeispiels analysiert. Zur Umsetzung des Metadatenmanagements werden anschließend unterschiedliche IT-Werkzeuge anhand definierter Kriterien analysiert. Das Analyseergebnis zeigt, dass Datenkataloge grundsätzlich die geeignete Werkzeugart darstellen, wobei noch technische Unzulänglichkeiten existieren. Abschließend werden die in der Praxis bestehenden Herausforderungen für ein ganzheitliches Metadatenmanagement im Data Lake zusammengefasst und zukünftige Forschungsbedarfe aufgezeigt.

Keywords: Metadaten, Meta Data, Data Lake, Datenkatalog, Data Catalog, Governance, Self-Service

1 Einleitung

Die digitale Transformation, Big Data und Advanced Analytics verändern die Datenlandschaft in Unternehmen erheblich [Lv17]. Die großen Mengen heterogener Daten sowie die Vielzahl unterschiedlicher analytischer Anwendungsfälle erfordern neue Konzepte und Plattformen für das Datenmanagement [OL13]. In diesem Zuge hat sich der Data Lake in den letzten Jahren als ein neuer Typ von Datenplattform für die Speicherung, Integration und Analyse aller Arten von (Roh-)daten etabliert [Ma17]. Data Lakes erfreuen sich in der industriellen Praxis zunehmender Beliebtheit und werden in unterschiedlichen Branchen eingesetzt. Ein prominentes Anwendungsfeld für Data Lakes stellt die Fertigungsindustrie dar, da im Zuge der Industrie 4.0 [Ba14], also der Digitalisierung der industriellen Wertschöpfungskette, enorme Datenmengen generiert und ausgewertet werden [GCA15]. Es geht beispielsweise um die Mustererkennung in Maschinendaten zur Optimierung von

¹ Robert Bosch GmbH, 70469 Stuttgart, christoph.groeger@de.bosch.com

² Robert Bosch GmbH, 70469 Stuttgart, eva.hoos@de.bosch.com

Fertigungsprozessen oder die Ad-hoc-Exploration von Produktentwicklungs- und Felddaten zur Verbesserung des Produktdesigns [Gr18].

Mit dem zunehmenden Einsatz von Data Lakes ergeben sich in der industriellen Praxis erweiterte Anforderungen, insbesondere hinsichtlich der Sicherstellung von Transparenz, Qualität und Compliance der Daten im Data Lake sowie der Unterstützung von Self-Service-Szenarien für Fachanwender. Diese Anforderungen machen Metadatenmanagement³ im Data Lake zum kritischen Erfolgsfaktor [QHV16, HGQ16]. Es soll damit verhindert werden, dass aus dem Data Lake ein Datensumpf (engl. data swamp) [HGQ16] entsteht, also eine Datenplattform mit nicht mehr sinnvoll nutzbaren Daten.

Bisher gibt es jedoch nur wenige wissenschaftliche Arbeiten zum Metadatenmanagement im Data Lake. Existierende Arbeiten fokussieren auf einzelne Teilaspekte, insbesondere die allgemeine Bedeutung von Metadaten im Data Lake [Te15, MT16, Sh18] sowie die Erfassung und Anreicherung von Metadaten im Data Lake mittels semantischer Technologien [HGQ16, QHV16, Al16]. Es mangelt an einer ganzheitlichen Betrachtung zur Konzeption und Realisierung des Metadatenmanagements im Data Lake, insbesondere hinsichtlich in der Praxis relevanter Anforderungen, zu adressierender Arten von Metadaten sowie passender IT-Werkzeuge zur Umsetzung.

Die vorliegende Arbeit adressiert diese Aspekte und basiert auf praktischen Erfahrungen aus einem weltweit operierenden Industriekonzern beim Aufbau eines unternehmensweiten Data Lake mit ganzheitlichem Metadatenmanagement. Zuerst werden in Kapitel 2 das Unternehmen sowie die Data-Lake-Architektur vorgestellt, die technisch auf einem Cross-Plattform-Ansatz basiert. Zusätzlich wird das ganzheitliche Metadatenmanagement im Data Lake als Ziel dargestellt. Auf dieser Grundlage werden in Kapitel 3 Kernanforderungen und praktische Anwendungsbeispiele für das Metadatenmanagement im Data Lake abgeleitet und analysiert. Davon ausgehend werden in Kapitel 4 die unterschiedlichen Arten relevanter Metadaten im Kontext von Data Lakes anhand des Praxisbeispiels identifiziert und diskutiert. Anschließend geht es in Kapitel 5 um die praktische Umsetzung des Metadatenmanagements mittels unterschiedlicher IT-Werkzeuge. Für die Auswahl geeigneter IT-Werkzeuge werden zuerst produktunabhängige Werkzeugarten definiert und anhand von aus den Anforderungen abgeleiteten Bewertungskriterien analysiert. Das Ergebnis zeigt, dass Datenkataloge grundsätzlich die geeignete Werkzeugart für das ganzheitliche Metadatenmanagement im Data Lake darstellen, wobei noch technische Unzulänglichkeiten existieren. Abschließend werden in Kapitel 6 die in der Praxis bestehenden technischen und organisatorischen Herausforderungen für ein ganzheitliches Metadatenmanagement im Data Lake zusammengefasst und zukünftige Forschungsbedarfe aufgezeigt. Die Arbeit schließt mit einem Fazit in Kapitel 7.

³ Unter dem Begriff Metadaten werden im Kontext des Datenmanagements allgemein Daten zur Verwaltung und Nutzung von Daten verstanden [VVS00]. Metadatenmanagement bezieht sich auf die systematische Verwaltung und Bereitstellung von Metadaten [He17].

2 Praxisbeispiel: Unternehmensweiter Data Lake und ganzheitliches Metadatenmanagement

Die vorliegende Arbeit basiert auf Erfahrungen aus einem realen Praxisbeispiel aus der Fertigungsindustrie, das die Grundlage für die Ableitung der Anforderungen sowie die Analyse relevanter Arten von Metadaten im weiteren Verlauf der Arbeit darstellt. Im Folgenden wird zuerst das zugrundeliegende Unternehmen kurz vorgestellt, um anschließend auf den Ansatz des unternehmensweiten Data Lake mit ganzheitlichem Metadatenmanagement einzugehen.

Das Praxisbeispiel bezieht sich auf einen *global tätigen Industriekonzern* mit mehreren hunderttausend Mitarbeitern und einer weltweit verteilten Fabriklandschaft. Der Industriekonzern gliedert sich in mehrere Geschäftsbereiche, die ein breites Spektrum an Produkten entwickeln und fertigen, speziell in den Domänen Industrie- und Gebäudetechnik, Mobilität und Konsumgüter. Dementsprechend vielfältig gestaltet sich die Prozesslandschaft des Konzerns, von der Massenfertigung hoch standardisierter Produkte bis zur Einzelfertigung von Spezialprodukten nach Kundenanforderung.

Ein wesentliches strategisches Ziel des Konzerns ist die digitale Transformation zum datengetriebenen Unternehmen als Teil der Industrie 4.0. Unterschiedliche Digitalisierungsinitiativen des Industriekonzerns haben in den letzten Jahren zu enormen Mengen heterogener Daten entlang der Wertschöpfungskette geführt. Diese Daten sollen insbesondere genutzt werden, um Produkte und Prozesse ganzheitlich zu optimieren. Es geht beispielsweise um Produktlebenszyklus-Analysen [Ka15], um Produktentwicklungs-, Fertigungs-, Feld- und Kundendaten umfassend zu analysieren. Ein weiteres Anwendungsfeld stellt die Ende-zu-Ende-Analyse von Geschäftsprozessen im Rahmen von Process Mining [Aa16] dar, um Engpässe und Wartezeiten zu eliminieren und die Prozessqualität zu erhöhen. Diese Anwendungsfelder erfordern die unternehmensweite Integration von klassischen transaktionalen Unternehmensdaten, speziell aus Enterprise Resource Planning (ERP) Systemen, mit Maschinen-, Sensor- und Kundendaten, z.B. aus Manufacturing Execution Systemen (MES) und dem Web (siehe [GSMb14, GSMA14] zur Datenintegration in Industrieunternehmen sowie [GP14] für eine Beschreibung der IT-Systeme in Industrieunternehmen).

Zu diesem Zweck wird ein *unternehmensweiter Data Lake* aufgebaut, der die unternehmensweite Speicherung, Integration und Analyse aller Arten von (Roh-)daten unterstützt. Eine vereinfachte Darstellung der konzeptionellen Architektur des Data Lake zeigt Abb. 1. Die Architektur basiert auf dem Lambda-Architektur-Paradigma [MW15] und umfasst Komponenten zur Batch- und Streaming-Datenverarbeitung zur Umsetzung unterschiedlicher Analyseanwendungen, von Reporting über Exploration bis zu Data Mining (siehe [HKP12, KBM10] für eine Beschreibung der Analyseanwendungen). Die Quelldaten umfassen sämtliche betrieblich relevanten strukturierten und unstrukturierten Daten, die in vier Kategorien unterteilt werden: klassische transaktionale Unternehmensdaten, benutzer-generierte Daten, Maschinen- und Sensordaten sowie Webdaten.

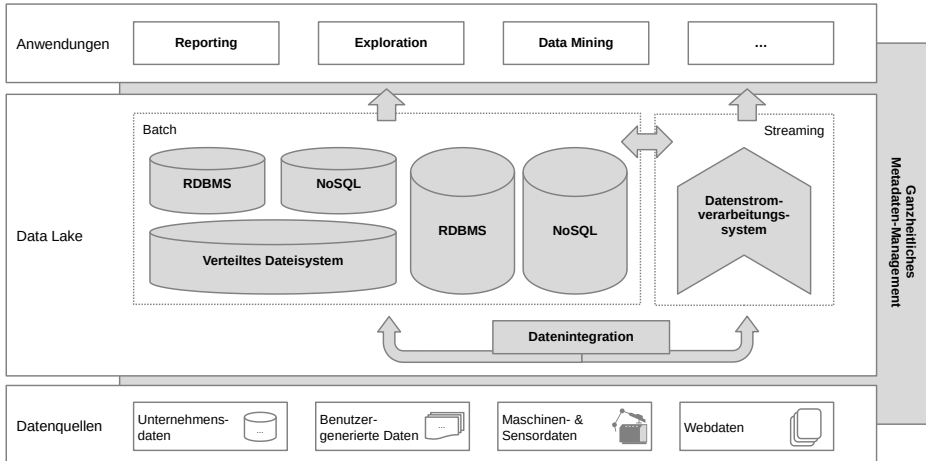


Abb. 1: Konzeptionelle Data-Lake-Architektur des Industriekonzerns (vereinfacht)

Eine wesentliche Besonderheit des Data Lake besteht darin, dass ein umfassender Cross-Plattform-Ansatz verfolgt wird. Dies bezieht sich auf folgende Merkmale:

- *Kombination von Batch- und Streaming-Komponenten*, um sowohl Stapelverarbeitung als auch echtzeitnahe Datenverarbeitung zu unterstützen
- *Kombination unterschiedlicher Datenhaltungstechnologien*, insbesondere von relationalen Datenbanken, spalten- und dokumentenorientierten NoSQL-Datenbanken sowie verteilten Dateisystemen, um eine polyglotte Persistenz [GR15] umzusetzen und die für den jeweiligen Anwendungsfall passende Datenhaltungstechnologie zu nutzen
- *Kombination unterschiedlicher Bereitstellungsvarianten der Komponenten*, um sowohl On-Premise- als auch Cloud-Bereitstellungen zu nutzen

Eine zentrale Komponente des Data Lake, die sich über sämtliche Schichten und Komponenten erstreckt, ist das Metadatenmanagement. Das Ziel ist, ein *ganzheitliches Metadatenmanagement* für den Data Lake zu realisieren, das sämtliche Kernanforderungen (siehe Kapitel 3), sämtliche relevanten Arten von Metadaten (siehe Kapitel 4) über sämtliche Datenhaltungssysteme des Data Lake mit geeigneten IT-Werkzeugen (siehe Kapitel 5) unterstützt. Diese Aspekte stehen in den folgenden Kapiteln der Arbeit im Vordergrund.

3 Kernanforderungen und Anwendungsbeispiele für Metadatenmanagement im Data Lake

Auf der Basis der praktischen Erfahrungen im Industriekonzern lassen sich die *Unterstützung von Self-Service* (siehe Kapitel 3.1) sowie die *Unterstützung von Governance* (siehe Kapitel 3.2) im Data Lake als Kernanforderungen für das Metadatenmanagement im Data Lake identifizieren. Diese werden im Folgenden beschrieben und anhand praktischer Anwendungsbeispiele aus dem Industriekonzern konkretisiert (siehe Abb. 2).

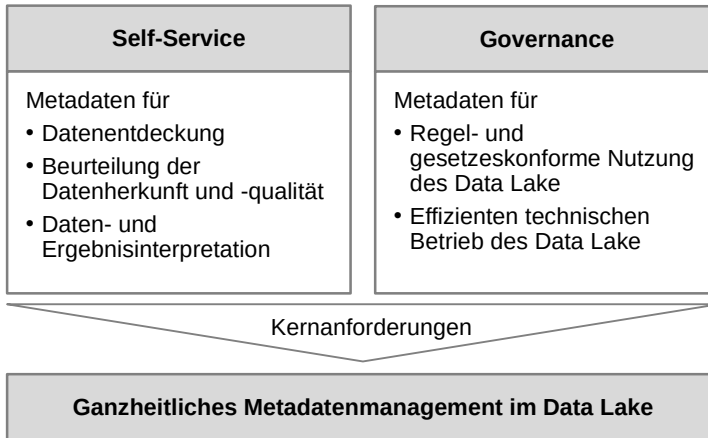


Abb. 2: Kernanforderungen für das Metadatenmanagement im Data Lake

3.1 Unterstützung von Self-Service

Ein wesentliches Ziel des unternehmensweiten Data Lake des Industriekonzerns besteht darin, eine zentrale Datenplattform für sämtliche Arten von Endbenutzern zu realisieren, um eine Demokratisierung, d.h. eine möglichst umfassende Durchdringung sämtlicher Prozesse und Entscheidungsebenen des Konzerns mit datengetriebenen Methoden und Anwendungsfällen, zu erreichen. Es geht folglich darum, nicht nur Data Scientists und IT-Experten als kleinere Gruppen spezialisierter Data-Lake-Nutzer zu adressieren, sondern auch Fachanwendern, z.B. Prozessingenieuren und Marketing-Spezialisten, die Nutzung des Data Lake zu ermöglichen. In diesem Rahmen strebt der Industriekonzern auch an, die Rolle des Citizen Data Scientist [Gr18] im Sinne von Fachanwendern mit vertieften Kenntnissen in Data Science zu etablieren. Ein typisches Anwendungsbeispiel umfasst einen Prozessingenieur, der als Citizen Data Scientist unterschiedliche Analysen der Fertigungsqualität durchführt.

Für diese Demokratisierung der Data-Lake-Nutzung sind insbesondere Konzepte und IT-Werkzeuge für Self-Service-Szenarien relevant, die unter den Schlagwörtern Self-Service-Business-Intelligence [AS16] und Self-Service-Analytics [Di16] diskutiert werden.

Das Ziel besteht allgemein darin, Fachanwendern die Datenaufbereitung und die Datenanalyse mit einfach nutzbaren IT-Werkzeugen zu ermöglichen, ohne dass eine umfassende Unterstützung durch IT-Experten erforderlich ist. Metadaten spielen eine zentrale Rolle bei der Umsetzung von Self-Service-Szenarien [Te15]. Ausgehend von den praktischen Erfahrungen im Industriekonern geht es hierbei insbesondere um die Unterstützung der Datenentdeckung, der Beurteilung der Datenherkunft und Datenqualität sowie der Daten- und Ergebnisinterpretation durch Fachanwender.

Bei der *Datenentdeckung* (engl. data discovery) steht die Identifikation relevanter Datenbereiche im Data Lake im Vordergrund, um sämtliche für eine spezifische Fragestellung relevanten Daten zu selektieren. Der Prozessingenieur sucht beispielsweise sämtliche Qualitätsdaten zu einem bestimmten Produkt. Es geht einerseits um MES-Daten zu Qualitätsprüfungen, die in einer relationalen Datenbank des Data Lake gespeichert sind. Andererseits geht es um Sensordaten aus Fertigungsmaschinen, die in einem verteilten Dateisystem des Data Lake abgelegt sind. Metadaten ermöglichen nun eine einfache Identifikation und Wiederauffindung dieser Daten im Data Lake, indem z.B. sämtliche Qualitätsdaten im Data Lake einheitlich gekennzeichnet sind. Der Prozessingenieur erhält anhand einer metadatenbasierten Suche beispielsweise eine Liste von Tabellen sowie Dateien im Data Lake, die Qualitätsdaten enthalten, und kann diese nun explorieren.

Zur *Beurteilung der Datenherkunft* (engl. data provenance) und der *Datenqualität* sind diverse Metadaten erforderlich. Es geht beispielsweise um Details zu den Quellsystemen der Daten, zu erfolgten Transformationsprozessen sowie um Datenprofile, z.B. Werteverteilungen und Statistiken. Metadaten zur Herkunft und Qualität der Daten sind ein kritischer Erfolgsfaktor, um eine zuverlässige und vertrauenswürdige Nutzung der Daten im Data Lake zu ermöglichen. Der Prozessingenieur untersucht beispielsweise anhand von Metadaten, ob ein Teil der MES-Daten aus einem produktiven MES oder einem Testsystem stammt. Zudem prüft er, welche Transformationen der Daten im Data Lake vorgenommen werden, da die MES-Daten bereits als vorverarbeitete aggregierte Tabellen zur Verfügung gestellt werden. Darüber hinaus beurteilt der Prozessingenieur die Sensordaten hinsichtlich ihrer Datenqualität anhand des Anteils fehlender Sensorwerte.

Zur *Daten- und Ergebnisinterpretation* sind insbesondere Details zur fachlichen Bedeutung der Daten relevant. Es geht beispielsweise um die Bedeutung betriebswirtschaftlicher Kennzahlen, die Abgrenzung von Fachbegriffen im Rahmen von Glossaren und die Struktur von Aggregationshierarchien. All diese Details stellen zentrale Metadaten zur Dateninterpretation dar und können sich auch auf Analyseergebnisse, wie z.B. Dashboards und Berichte beziehen. Zur Auswertung der MES-Daten benötigt der Prozessingenieur beispielsweise Angaben zur fachlichen Bedeutung einzelner Attribute der Tabellen, da diese keine sprechenden Namen, sondern nur technische Kürzel enthalten. Außerdem fügt der Prozessingenieur neue Tabellen in der relationalen Datenbank des Data Lake hinzu, um die Sensordaten mit den MES-Daten zu integrieren und zu bereinigen. Diese Tabellen enthalten auch berechnete Kennzahlen auf der Basis von MES- und Sensordaten, die in einem Dashboard grafisch veranschaulicht werden. Die fachliche Bedeutung dieser Kennzahlen dokumentiert der

Prozessingenieur anhand von Metadaten und kennzeichnet die neu erstellten Tabellen und das Dashboard mittels Metadaten, um die Ergebnisse wiederauffindbar zu machen und zukünftige Qualitätsanalysen damit zu unterstützen.

Alles in allem geht es beim Metadatenmanagement zur Unterstützung von Self-Service-Szenarien sowohl um eine ganzheitliche Verwaltung und Bereitstellung der Metadaten für Fachanwender entlang des gesamten Datenaufbereitungs- und Analyseprozesses als auch darum, zusätzliche Metadaten durch Fachanwender zu erfassen und wiederverwendbar zu machen.

3.2 Unterstützung von Governance

Mit zunehmender Nutzung des Data Lake gewinnen organisatorische und technische Regelungenanforderungen, d.h. Governance-Anforderungen, im Industriekonzern an Bedeutung. Diese werden allgemein auch unter dem Stichwort Data Governance [KB10] diskutiert.

Zum einen geht es darum, die *regel- und gesetzeskonforme Nutzung des Data Lake* (engl. compliance) sicherzustellen. Ein wesentlicher Faktor sind *Anforderungen aus der EU-Datenschutz-Grundverordnung (DSGVO)* [Eu16], die eine transparente und zweckgebundene Verarbeitung personenbezogener Daten vorschreibt und nachweispflichtig macht. Dies erfordert eine umfassende und ständig aktuell gehaltene Metadatendokumentation über sämtliche Batch- und Streaming-Komponenten im Data Lake. Es geht insbesondere darum, sämtliche Datenbereiche mit personenbezogene Daten im Data Lake zu inventarisieren, also beispielsweise alle relevanten Schemata und Tabellen, Verzeichnisstrukturen und Message Queues. Ein praktisches Anwendungsbeispiel sind Anforderungen von Kunden zur Löschung ihrer personenbezogenen Daten, z.B. Adressdaten oder Produktnutzungsdaten, die mittels entsprechender Metadaten im Data Lake effektiv umgesetzt werden können.

Ein weiterer Faktor sind *Regelungsanforderungen, die sich aus der digitalen Transformation* des Industriekonzerns ergeben. Daten sind ein zentraler Wertgegenstand (engl. asset) im datengetriebenen Unternehmen und erfordern ein systematisches Datenqualitätsmanagement sowie organisatorische Verantwortlichkeiten und Prozesse. Es geht insbesondere um unternehmensinterne Regelungen für Dateneigentum und -nutzung, um beispielsweise zu dokumentieren, welche Geschäftsdivisionen des Industriekonzerns welche Daten im unternehmensweiten Data Lake bereitstellen und verantworten. Hierbei sind auch umfassende Details zur Datenherkunft relevant. Sämtliche Informationen rund um die Herkunft, Qualität und um die organisatorischen Verantwortlichkeiten der Daten sind Metadaten im Data Lake, die systematisch verwaltet werden müssen.

Zum anderen geht es bei Governance-Anforderungen darum, einen *effizienten technischen Betrieb des Data Lake* zu gewährleisten. Der im Industriekonzern verfolgte Cross-Plattform-Ansatz mit seiner Vielzahl an Datenhaltungssystemen bedingt *Datenredundanzen im Data Lake, die systematisch verwaltet und transparent gemacht werden müssen*, um Inkonsistenzen

zu vermeiden und die Speicherkosten im Data Lake zu optimieren. Datenredundanzen entstehen beispielsweise bei der kombinierten Analyse von Stammdaten zu Maschinen und Produkten mit Sensordaten. Die Stammdaten werden im Data Lake des Industriekonzerns meist in relationalen Datenbanken gehalten, wohingegen die Sensordaten in dokumenten- oder spaltenorientierten Datenbanken abgelegt sind. Zur kombinierten Analyse werden die Daten in einer Datenbank zusammengeführt, um darauf z.B. Data-Mining-Verfahren anzuwenden. Die daraus resultierenden Redundanzen sind zu dokumentieren. Darüber hinaus sind für einen effizienten Betrieb des Data Lake *effiziente Änderungsprozesse bei Änderungen der Quellsysteme* wesentlich. Änderungen bei den Quellsystemen des Data Lake ergeben sich beispielsweise durch Systemkonsolidierungen oder Softwareaktualisierungen. Diese Änderungen müssen im Data Lake nachvollzogen werden, um z.B. Transformationsprozesse anzupassen und die Datenbereitstellung im Data Lake aufrecht zu erhalten. Die aufgeführte Transparenz über Datenredundanzen sowie über Auswirkungen von Quellsystemänderungen basiert auf entsprechenden Metadaten zu Datenbereichen sowie Quellsystemen des Data Lake, die es zu verwalten gilt.

Summa summarum stellt ein ganzheitliches Metadatenmanagement im Data Lake ein zentrales Instrument zur Umsetzung verschiedenster Governance-Anforderungen dar und ist deswegen in der industriellen Praxis meist Teil umfangreicher Data-Governance-Projekte.

4 Heterogene Metadaten im Kontext von Data Lakes

In der Literatur werden unterschiedliche allgemeine Klassifikationen für Metadaten in analytischen IT-Systemen beschrieben, insbesondere aktive und passive Metadaten sowie technische und betriebswirtschaftliche Metadaten [KBM10]. Diese Klassifikationen erfassen allerdings nur einen Teil der im Praxisbeispiel relevanten Metadaten. Auf der Basis der im Industriekonzern durchgeführten Workshops zum Metadatenmanagement lassen sich die in Abb. 3 dargestellten Arten von Metadaten im Data-Lake-Kontext ableiten, die nachfolgend beschrieben und diskutiert werden.

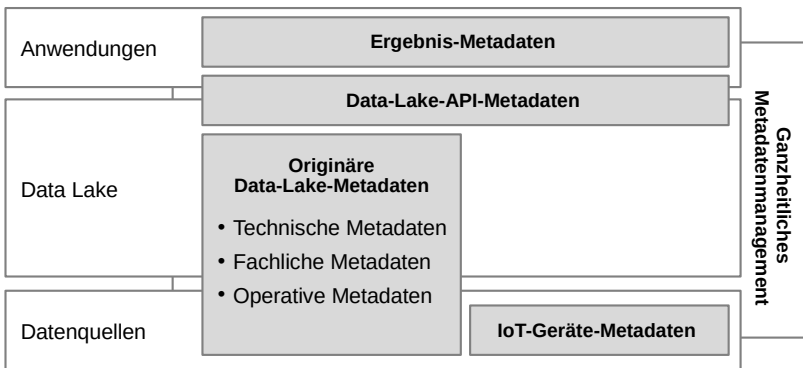


Abb. 3: Metadaten im Kontext von Data Lakes

- *Originäre Data-Lake-Metadaten* sind Metadaten über die im Data Lake gespeicherten Daten, z.B. Daten in relationalen Datenbanken und Message-Queues des Data Lake. Für originäre Data-Lake-Metadaten hat sich in der industriellen Praxis die Unterscheidung in technische, fachliche und operative Metadaten etabliert [He17]:
 - *Technische Metadaten* beziehen sich auf sämtliche technisch-strukturellen Aspekte der Daten sowie der zugrundeliegenden Datenhaltungssysteme im Data Lake, wie z.B. Tabellenstrukturen, Attributnamen, Wertelisten und Zugriffsrechte.
 - *Fachliche Metadaten* beschreiben inhaltliche Bedeutungen und Zusammenhänge der Daten. Dazu gehören beispielsweise Begriffsabgrenzungen, Kennzahldefinitionen, organisatorische Verantwortlichkeiten und konzeptionelle Datenmodelle.
 - *Operative Metadaten* umfassen technische Details zu Datentransformationen und Datenzugriffen, z.B. Informationen über ETL-Jobs, Quellsysteme und Zugriffsmuster.
- *Ergebnis-Metadaten* stellen Metadaten zu erzeugten Analyseergebnissen, wie z.B. Kennzahlenberichte, Dashboards und Data-Mining-Modelle, dar. Es geht sowohl um Angaben zum Erstellungsprozess der Analyseergebnisse, z.B. verwendete Parameterwerte von Data-Mining-Algorithmen, als auch um Interpretationen der Analyseergebnisse, z.B. die Kennzeichnung bestimmter Knoten in einem Entscheidungsbaum.
- *IoT-Geräte-Metadaten* sind Metadaten über Geräte im Internet der Dinge, wie z.B. Smart-Home-Geräte oder intelligente Fertigungsmaschinen, die Datenquellen für den Data Lake darstellen. IoT-Geräte-Metadaten umfassen Daten zur Einbindung, Verwaltung und Steuerung von IoT-Geräten, beispielsweise ID, Typ, Status und Hersteller eines IoT-Geräts. IoT-Geräte-Metadaten sind folglich von Metadaten über die von IoT-Geräten bereitgestellten Sensordaten, z.B. Datentypen von Sensorwerten, zu unterscheiden. Metadaten über Sensordaten von IoT-Geräten, die im Data Lake verarbeitet werden, werden als originäre Data-Lake-Metadaten betrachtet.
- *Data-Lake-API-Metadaten* sind Metadaten zu den vom Data Lake bereitgestellten Programmierschnittstellen (engl. application programming interface (API)). Diese ermöglichen einen programmatischen Zugriff auf Daten im Data Lake, typischerweise mittels Representational State Transfer (REST). Data-Lake-API-Metadaten umfassen beispielsweise Angaben zu Methoden und Parametern einer API.

Den Kern für das Metadatenmanagement im Data Lake bilden originäre Data-Lake-Metadaten sowie Ergebnis-Metadaten. Ergebnis-Metadaten werden häufig in separaten Analysewerkzeugen, z.B. Data-Mining-Werkzeugen oder Business-Intelligence-Werkzeugen, erzeugt und verwaltet. Wenn Analyseergebnisse wiederum als Daten im Data Lake gespeichert werden, indem beispielsweise Data-Mining-Modelle als Dateien im Predictive-Model-Markup-Language-Format (PMML-Format) im Data Lake abgelegt werden, verschwimmen

die Grenzen zwischen originären Data-Lake-Metadaten und Ergebnis-Metadaten. Eine integrierte Betrachtung dieser Metadaten ist folglich sinnvoll.

Da Data-Lake-APIs den Zugriff auf Daten im Data Lake ermöglichen, ist eine Verknüpfung von Data-Lake-API-Metadaten mit originären Data-Lake-Metadaten sowie Ergebnis-Metadaten vielversprechend. Die Rückgabewerte einer API-Methode können beispielsweise direkt auf fachliche Metadaten der zurückgelieferten Daten verweisen, um die gezielte Verwendung der Daten zu unterstützen.

IoT-Geräte stellen zwar nur eine von mehreren Arten von Quellsystemen des Data Lake dar. Der Unterschied zu traditionellen Quellsystemen wie relationalen Datenbanksystemen besteht aus Data-Lake-Sicht jedoch darin, dass zur effektiven Analyse von Sensordaten von IoT-Geräten häufig IoT-Geräte-Metadaten erforderlich sind. Beispielsweise sind zur Analyse von Sensordaten einer Messstation einer Fertigungslinie Angaben zur Positionierung der Messstation an der Fertigungslinie sowie zum aktuellen Software-Stand der Messstation erforderlich.

Summa summarum sind Metadaten im Data-Lake-Kontext sehr heterogen und stark unterschiedlich hinsichtlich Struktur, erzeugenden Systemen und Verwendungszwecken. Hinzukommt, dass Metadaten auch analytisch gewonnen werden und selbst wiederum Quelldaten für weitere Analysen darstellen. Ein typischer Anwendungsfall im betrachteten Industriekonzern ist beispielsweise die analytische Gewinnung von Metadaten zu Videos aus autonomen Fahrzeugen. Diese Metadaten stellen wiederum die Grundlage für Dashboards und Berichte zur Auswertung des Fahrverhaltens dar.

5 IT-Werkzeuge zur praktischen Umsetzung des Metadatenmanagements im Data Lake

Die Umsetzung eines ganzheitlichen Metadatenmanagements im Data Lake erfordert eine integrierte Betrachtung sämtlicher in Kapitel 4 aufgeführter Arten von Metadaten zur Adressierung der in Kapitel 3 beschriebenen Kernanforderungen um Self-Service und Governance. Das Metadatenmanagement im Data Lake ist damit deutlich umfangreicher und komplexer als das Metadatenmanagement im klassischen Data Warehouse, das hauptsächlich Metadaten zu strukturierten transaktionalen Unternehmensdaten umfasst. Diese Komplexität spiegelt sich auch im Markt für IT-Werkzeuge für das Metadatenmanagement im Data Lake wieder. Der Markt ist sehr heterogen und entwickelt sich sehr dynamisch, da verschiedene Hersteller unterschiedlich ausgerichtete Produkte anbieten und meist ähnliche Funktionalitäten unter verschiedenen, marketinglastigen Begrifflichkeiten referenziert werden.

Als Ausgangspunkt für die praktische Umsetzung im Industriekonzern wurde deswegen eine umfangreiche Softwarestudie durchgeführt. In diesem Rahmen werden produktunabhängige Arten von Werkzeugen für das Metadatenmanagement im Data Lake definiert, die als Strukturierungselemente für die Marktanalyse dienen. Darüber hinaus werden

Bewertungskriterien entwickelt, die sich aus den dargestellten Kernanforderungen für das Metadatenmanagement sowie den relevanten Arten an Metadaten ableiten. Anhand dieser Kriterien wird anschließend die Eignung der Werkzeugarten für das Metadatenmanagement im Data Lake analysiert. Im Folgenden werden die im Rahmen der Studie identifizierten Arten an IT-Werkzeugen für das Metadatenmanagement im Data Lake vorgestellt (siehe Kapitel 5.1), um anschließend deren Eignung zu diskutieren (siehe Kapitel 5.2). Anzumerken ist, dass aus Vertraulichkeitsgründen nicht näher auf einzelne Produkte bestimmter Hersteller eingegangen werden kann.

5.1 Werkzeugarten: Datenverzeichnisse, Datenkataloge und Data-Lake-Management-Plattformen

Im Rahmen der Studie werden ausschließlich produktiv einsetzbare Werkzeuge, sowohl auf Open-Source- als auch auf Closed-Source-Basis, analysiert. Forschungsprototypen werden nicht einbezogen, da sie sich nicht für den Aufbau einer praxistauglichen Lösung eignen. Die Analyse ergibt drei Arten von Werkzeugen für das Metadatenmanagement im Data Lake, die den Markt repräsentieren: systemintegrierte Datenverzeichnisse, Datenkataloge sowie Data-Lake-Management-Plattformen (siehe Abb. 4).

Systemintegrierte Datenverzeichnisse	Datenkataloge	Data-Lake-Management-Plattformen
<ul style="list-style-type: none"> • Erfassung, Speicherung und Bereitstellung von Metadaten innerhalb eines Systems • Schwerpunkt auf technischen und operativen Metadaten 	<ul style="list-style-type: none"> • Eigenständige Spezialwerkzeuge für das Metadatenmanagement • Erfassung, Speicherung, Bereitstellung und Analyse von originären Data-Lake-Metadaten sowie von Ergebnismetadaten 	<ul style="list-style-type: none"> • Werkzeug-Suiten aus Datenkatalog und weiteren Datenmanagement-Funktionalitäten • Zusätzliche Verwaltung von Data-Lake-API-Metadaten

Abb. 4: Arten von IT-Werkzeugen für das Metadatenmanagement im Data Lake

Systemintegrierte Datenverzeichnisse (engl. data dictionary) [KE06] sind in einem Datenhaltungssystem integriert und dienen der Erfassung, Speicherung und Bereitstellung von Metadaten innerhalb dieses Systems. Der Schwerpunkt liegt auf technischen und operativen Metadaten. Typische Beispiele sind Datenverzeichnisse in relationalen Datenbanksystemen, die z.B. Details zu Schemata, Tabellen und Attributen enthalten. Im Hadoop- und NoSQL-Kontext haben sich nach diesem Vorbild ähnliche Ansätze etabliert, wie z.B. der Hive Metastore als Datenverzeichnis für Apache Hive [Ap18a].

Datenkataloge (engl. data catalog) [Sh18] stellen eigenständige Spezialwerkzeuge für das Metadatenmanagement unabhängig von einem spezifischen Datenhaltungssystem dar. Sie ermöglichen die Erfassung, Speicherung, Bereitstellung sowie die Analyse von technischen, fachlichen und operativen Metadaten sowie von Ergebnismetadaten. Dabei unterstützen sie meist eine Vielzahl an Datenhaltungssystemen zur Metadaten-Erfassung, von relationalen

Datenbanken über NoSQL-Datenbanken bis zu Datenhaltungssystemen aus dem Hadoop-Kontext. Datenkataloge basieren auf einem Metadaten-Repository und bieten typischerweise Funktionen zur systemübergreifenden Analyse des Datenverlaufs (engl. data lineage) und der Datenqualität sowie zur fachlichen Klassifikation von Metadaten, beispielsweise mittels Tagging und Glossaren. Es werden darüber hinaus auch Kollaborationsfunktionen zur Einbindung der Endbenutzer bei der Anreicherung und Nutzung von Metadaten angeboten, z.B. durch Bewertungs- und Kommentarfunktionen. Beispielhafte Werkzeuge dieser Art sind Waterline Smart Data Catalog [Wa18], Informatica Enterprise Data Catalog [In18] und Collibra Catalog [Co18]. Auch Datenkataloge wie Apache Atlas [Ap18b], die auf unterschiedliche Datenhaltungssysteme im Hadoop-Kontext fokussieren, gehören zu dieser Art.

Data-Lake-Management-Plattformen stellen Werkzeugen-Suiten dar, die auf der Basis eines Datenkatalogs weitere Funktionalitäten für das Datenmanagement im Data Lake integrieren. Typischerweise geht es um ergänzende Funktionalitäten für ETL, Self-Service-Data-Preparation und Datenföderation (engl. data federation), die eng mit dem Datenkatalog integriert sind. Beispielsweise werden Metadaten zu Datentransformationen in ETL-Jobs direkt im Datenkatalog abgelegt. Data-Lake-Management-Plattformen unterstützen häufig auch die Definition von APIs zum Zugriff auf Daten im Data Lake und ermöglichen damit zusätzlich die Verwaltung von Data-Lake-API-Metadaten. Beispielhafte Data-Lake-Management-Plattformen sind Kylo [Te18] und Iguazio [Ig18].

5.2 Bewertung

Um die Eignung der Werkzeugarten für ein ganzheitliches Metadatenmanagement im Data Lake zu bewerten, werden die folgenden grundlegenden Kriterien definiert, die aus den Kernanforderungen (siehe Kapitel 3) sowie den unterschiedlichen Arten von Metadaten (siehe Kapitel 4) abgeleitet werden:

- Die *systemübergreifende Metadaten-Verwaltung* bezieht sich darauf, dass Metadaten über unterschiedliche Datenhaltungssysteme im Data Lake hinweg gemäß des Cross-Plattform-Ansatzes (siehe Kapitel 2) zu verwalten sind.
- Der *offene Metadaten-Austausch* bezieht sich auf standardisierte technische Schnittstellen zum Zugriff auf sowie zum Import und Export von Metadaten, um eine Metadaten-Nutzung in anderen Systemen zu ermöglichen.
- Die *unterstützten Metadaten-Arten* umfassen sämtliche Metadaten für ein ganzheitliches Metadatenmanagement, d.h. *originäre Data-Lake-Metadaten*, *Ergebnis-Metadaten*, *IoT-Geräte-Metadaten* und *Data-Lake-API-Metadaten* (siehe Kapitel 4).
- Der *Funktionsumfang zur Metadaten-Verwaltung* bezieht sich auf grundlegende Funktionalitäten zur Verarbeitung von Metadaten. Hierbei werden die *Speicherung*

und Bereitstellung von Metadaten, Datenverlaufsanalysen sowie metadatenbasierte Mehrwertfunktionen, z.B. zur automatisierten Erkennung personenbezogener Daten für DSGVO-Szenarien, unterschieden.

Wie in Tab. 1 dargestellt, fokussieren *systemintegrierte Datenverzeichnisse* auf die datenhaltungsinterne Speicherung und Bereitstellung von originären Data-Lake-Metadaten und bieten typischerweise keine standardisierten Schnittstellen für den offenen Metadaten-Austausch. Dementsprechend dienen systemintegrierte Datenverzeichnisse der im Data Lake verwendeten Datenhaltungssysteme zwar als Quellen für Metadaten, eignen sich aber nicht als eigentliche Werkzeuge zur Realisierung eines ganzheitlichen Metadatenmanagements im Data Lake.

<i>Kriterien / Werkzeugarten</i>	Systemintegrierte Datenverzeichnisse	Datenkataloge	Data-Lake-Management-Plattformen
Systemübergreifende Metadaten-Verwaltung	-	+	+
Offener Metadaten-Austausch	-	+	+
Unterstützte Metadaten-Arten			
Originäre Data-Lake-Metadaten	+	+	+
Ergebnis-Metadaten	-	+	+
IoT-Geräte-Metadaten	-	-	-
Data-Lake-API-Metadaten	-	-	+
Funktionsumfang zur Metadaten-Verwaltung			
Metadaten-Speicherung/-Bereitstellung	+	+	+
Datenverlaufsanalyse	-	+	+
Mehrwertfunktionen (DSGVO, ...)	-	+	-

Tab. 1: Bewertung der Werkzeugarten für ein ganzheitliches Metadatenmanagement

Datenkataloge ermöglichen als eigenständige Werkzeuge eine systemübergreifende Metadaten-Verwaltung und unterstützen mit standardisierten Schnittstellen, meist APIs, einen offenen Metadaten-Austausch. Der Funktionsumfang zur Metadaten-Verwaltung ist im Vergleich zu den anderen Werkzeugarten sehr ausgeprägt. Sie bieten Kollaborationsfunktionen, um den Endbenutzer im Rahmen von Self-Service-Szenarien einzubinden, und adressieren Governance-Anforderungen, um z.B. mittels Workflowfunktionen eine hohe, kontrollierte Qualität der Metadaten sicherzustellen. Darüber hinaus werden häufig metadatenbasierte Mehrwertfunktionen zur Erkennung personenbezogener Daten angeboten.

Data-Lake-Management-Plattformen stellen eine relativ neue Werkzeugart am Markt dar. Der Funktionsumfang zur Metadaten-Verwaltung der Data-Lake-Management-Plattformen orientiert sich an dem von Datenkatalogen, weist aber aktuell nicht denselben Reifegrad und

dieselbe Breite auf, insbesondere was metadatenbasierte Mehrwertfunktionen angeht. Ein Vorteil im Vergleich zu Datenkatalogen besteht jedoch in der integrierten Verwaltung von Data-Lake-API-Metadaten. Diese müssen bei der Verwendung von Datenkatalogen über entsprechende Schnittstellen für die Metadaten-Bereitstellung manuell gepflegt werden.

Summa summarum stellen Datenkataloge grundsätzlich die geeignete Werkzeugart zur praktischen Umsetzung eines ganzheitlichen Metadatenmanagements im Data Lake dar. Die Studienergebnisse weisen jedoch auf funktionale Unzulänglichkeiten und Erweiterungspotentiale von Datenkatalogen hin. Diese werden als Herausforderungen im folgenden Kapitel zusammengefasst.

6 Praktische Herausforderungen und Forschungsbedarfe

Ausgehend von den bisherigen Erfahrungen zum Aufbau eines ganzheitlichen Metadatenmanagements für den unternehmensweiten Data Lake des Industriekonzerns lassen sich IT-technische und fachlich-organisatorische Herausforderungen ableiten und zugehörige Forschungsbedarfe identifizieren (siehe Abb. 5).

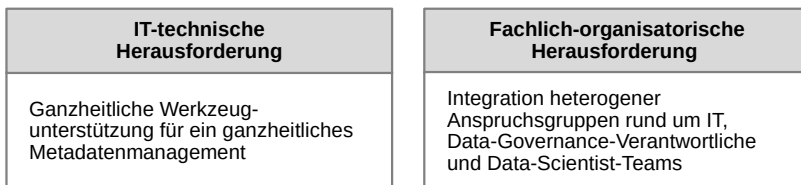


Abb. 5: Herausforderungen für ein ganzheitliches Metadatenmanagement im Data Lake

Die wesentliche *IT-technische Herausforderung* besteht darin, eine *ganzheitliche Werkzeugunterstützung für ein ganzheitliches Metadatenmanagement* zu realisieren. Hierbei geht es insbesondere darum, Unzulänglichkeiten existierender Datenkataloge zu adressieren, wobei folgende Aspekte eine zentrale Rolle spielen:

- *Datengetriebene Analyse des Datenverlaufs*: Die Analyse des Datenverlaufs, d.h. von Zusammenhängen zwischen Metadaten, erfolgt in existierenden Datenkatalogen größtenteils modellgetrieben, d.h. durch die Nutzung gegebener Modelle zu Metadaten und deren Zusammenhängen. Dies wird typischerweise durch das Auslesen von Metadaten aus ETL-Jobs realisiert. Im Rahmen von Self-Service-Szenarien stehen jedoch meist keine umfassenden Modelle zum Datenverlauf aus ETL-Werkzeugen zur Verfügung, da sich der Datenverlauf aus manuellen, iterativen Datentransformationen durch den Endbenutzer ergibt. Es geht folglich darum, Metadaten-Zusammenhänge datengetrieben zu erfassen und zu analysieren. Denkbar wäre beispielsweise, Data-Mining-Verfahren auf relationale Tabellenstrukturen anzuwenden, um mögliche Beziehungen und Ähnlichkeiten zwischen den Tabellen zu erkennen. In existierenden Datenkatalogen ist die datengetriebene Analyse von Metadaten-Zusammenhängen

noch sehr rudimentär ausgeprägt und basiert häufig auf Zeitstempeln und existierenden relationalen Beziehungen.

- *Metadaten-Erfassung von Streaming-Komponenten:* Der Fokus existierender Datenkataloge liegt auf der Erfassung von Metadaten aus Batch-Datenhaltungssystemen, z.B. durch die Anbindung von systemintegrierten Datenverzeichnissen relationaler Datenbanken. Die Unterstützung der Metadaten-Erfassung von Streaming-Komponenten, wie z.B. Apache Kafka, ist bei aktuell verfügbaren Datenkatalogen wenig ausgeprägt. Für einen spezifischen Datenkatalog muss typischerweise ein eigener Adapter für jede Streaming-Komponente entwickelt werden. Streaming-Komponenten sind in Data Lakes für die echtzeitnahe Datenverarbeitung jedoch von großer Bedeutung [Ma17]. Für ein ganzheitliches Metadatenmanagement ist eine integrierte Verwaltung von Metadaten über Batch- und Streaming-Komponenten erforderlich, d.h. der (Batch-)Datenkatalog ist um einen Ereigniskatalog mit Metadaten zu Ereignisströmen (engl. event stream) zu ergänzen. Damit können beispielsweise ganzheitliche Analysen des Datenverlaufs über Batch- und Streaming-Komponenten, z.B. für DSGVO-Nachweispflichten, durchgeführt werden.
- *Integration von IoT-Geräte-Metadaten und Data-Lake-API-Metadaten:* IoT-Geräte-Metadaten sowie Data-Lake-API-Metadaten werden von existierenden Datenkatalogen typischerweise nicht betrachtet. Meist werden IoT-Geräte-Metadaten in dedizierten IoT-Gerätemanagement-Anwendungen verwaltet. Ein ganzheitliches Metadaten-Management erfordert dementsprechend eine integrierte Verwaltung von IoT-Geräte-Metadaten und Data-Lake-API-Metadaten gemeinsam mit originären Data-Lake-Metadaten und Ergebnis-Metadaten.

Die wesentliche *fachlich-organisatorische Herausforderung* für ein ganzheitliches Metadatenmanagement besteht darin, die *heterogenen Anspruchsgruppen* (engl. stakeholder) *rund um IT, Data-Governance-Verantwortliche und Data-Scientist-Teams zu integrieren*, um eine erfolgreiche Umsetzung des Metadatenmanagements als Gesamtprojekt sicherzustellen. Data-Governance-Verantwortliche sind in der industriellen Praxis ein wesentlicher Initiator für das Metadatenmanagement, sowohl in analytischen als auch in operativen IT-Systemen, und definieren zentrale Anforderungen für das Metadatenmanagement. Es werden beispielsweise Rollen, Prozesse und Verantwortlichkeiten für Dateneigentum und Datennutzung im Rahmen von Data-Governance-Projekten festgelegt, die im Datenkatalog abzubilden sind. Data Scientists sind in der Praxis meist als unternehmensübergreifende Expertenteams organisiert, die stark self-service-orientiert arbeiten, um Anwendungsfälle schnell umzusetzen [Gr18]. Sie stellen damit eine wesentliche Anspruchs- und Endbenutzergruppe für den Datenkatalog dar. In der Praxis liegt meist eine organisatorische Trennung der IT, die für die Konfiguration und den Betrieb des Datenkatalogs verantwortlich ist, Data-Governance-Verantwortlichen in Fachabteilungen sowie Data Scientist-Teams vor, was zu unterschiedlichen funktionalen Zielsetzungen führt. Die IT fokussiert auf den effizienten und sicheren Betrieb des Datenkatalogs, Data Scientists sind an möglichst weitreichenden kollaborationsgetriebenen Self-Service-Funktionalitäten interessiert, wohingegen Data-

Governance-Verantwortliche meist Transparenz und regelgetriebene Strukturen bevorzugen. Zur zielgerichteten Umsetzung des Metadatenmanagements ist folglich eine kontinuierliche Einbeziehung und Harmonisierung dieser Anspruchsgruppen, d.h. ein systematisches Stakeholder-Management, erforderlich.

7 Fazit

Die in dieser Arbeit diskutierten Praxiserfahrungen machen deutlich, dass ein ganzheitliches Metadatenmanagement im Data Lake einen zentralen Erfolgsfaktor für die Umsetzung von Governance-Anforderungen und Self-Service-Szenarien im Data Lake darstellt. Gleichzeitig wird deutlich, dass die Anforderungen an ein ganzheitliches Metadatenmanagement sowohl IT-technisch als auch fachlich-organisatorisch sehr komplex sind. Heterogene Metadaten sind über heterogene Datenhaltungssysteme im Data Lake systematisch zu verwalten und organisatorisch getrennte Anspruchsgruppen mit unterschiedlichen Zielsetzungen – IT, Data Scientists und Data-Governance-Verantwortliche – sind für eine erfolgreiche Umsetzung konstruktiv einzubeziehen. Zukünftige Forschungsbedarfe ergeben sich insbesondere aus einer ganzheitlichen Werkzeugunterstützung für das Metadatenmanagement im Data Lake, bei der die integrierte Verwaltung von Metadaten aus Batch- und Streaming-Komponenten sowie die datengetriebene Gewinnung von Metadaten über den Datenverlauf im Vordergrund stehen.

Danksagungen

Die Autoren danken Dieter Neumann, Thomas Müller und Arnold Lutsch für die kontinuierliche Unterstützung und die konstruktiven Diskussionen zu dieser Arbeit.

Literaturverzeichnis

- [Aa16] Aalst, W. v.d.: Process Mining. Springer, Heidelberg, 2016.
- [Al16] Alserafi, A. et al.: Towards Information Profiling: Data Lake Content Metadata Management: Proceedings of the IEEE International Conference on Data Mining Workshops (ICDMW) 2016. IEEE, S. 178–185, 2016.
- [Ap18a] Apache Software Foundation: Apache Hive. <http://hive.apache.org>, Stand: 12.09.18.
- [Ap18b] Apache Software Foundation: Apache Atlas. <http://atlas.apache.org>, Stand: 12.09.18.
- [AS16] Alpar, P.; Schulz, M.: Self-Service Business Intelligence. In Business & Information Systems Engineering, 58(2), S. 151–155, 2016.
- [Ba14] Bauernhansl, T.: Die Vierte Industrielle Revolution – Der Weg in ein wertschaffendes Produktionsparadigma. In (Bauernhansl, T.; Hompel, M. t.; Vogel-Heuser, B. Hrsg.): Industrie 4.0 in Produktion, Automatisierung und Logistik. Anwendung, Technologien, Migration. Springer Vieweg, Wiesbaden, S. 5–35, 2014.

- [Co18] Collibra: Collibra Catalog. <http://www.collibra.com/data-governance-solutions/data-catalog>, Stand: 12.09.18.
- [Di16] Dinsmore, T. W.: *Disruptive Analytics*. Apress, Berkeley, 2016.
- [Eu16] Europäische Union: EU Verordnung 2016/679 (Datenschutz-Grundverordnung), 2016.
- [GCA15] Gölzer, P.; Cato, P.; Amberg, M.: Data processing requirements of industry 4.0 – use cases for big data applications: Proceedings of the European Conference on Information Systems (ECIS) 2015, paper 61, 2015.
- [GP14] Gausemeier, J.; Plass, C.: *Zukunftsorientierte Unternehmensgestaltung. Strategien, Geschäftsprozesse und IT-Systeme für die Produktion von morgen*. Hanser, München, 2014.
- [GR15] Gessert, F.; Ritter, N.: Polyglot Persistence. In *Datenbank-Spektrum*, 15(3), S. 229–233, 2015.
- [Gr18] Gröger, C.: Building an Industry 4.0 Analytics Platform. In *Datenbank-Spektrum*, 18(1), S. 5–14, 2018.
- [GSMa14] Gröger, C.; Schwarz, H.; Mitschang, B.: The Manufacturing Knowledge Repository. Consolidating Knowledge to Enable Holistic Process Knowledge Management in Manufacturing: Proceedings of the International Conference on Enterprise Information Systems (ICEIS) 2014. SciTePress, S. 39–51, 2014.
- [GSMb14] Gröger, C.; Schwarz, H.; Mitschang, B.: The Deep Data Warehouse. Link-based Integration and Enrichment of Warehouse Data and Unstructured Content: Proceedings of the IEEE International Enterprise Distributed Object Computing Conference (EDOC) 2014. IEEE, Los Alamitos, S. 210–217, 2014.
- [He17] Henderson, D. et al.: *DAMA-DMBOK. Data management body of knowledge*. Technics Publications, New Jersey, 2017.
- [HGQ16] Hai, R.; Geisler, S.; Quix, C.: Constance: An Intelligent Data Lake System: Proceedings of the International Conference on Management of Data (SIGMOD) 2016. ACM Press, New York, S. 2097–2100, 2016.
- [HKP12] Han, J.; Kamber, M.; Pei, J.: *Data mining. Concepts and techniques*. Elsevier/Morgan Kaufmann, Amsterdam, 2012.
- [Ig18] Iguazio: Iguazio Data Platform. <http://www.iguazio.com>, Stand: 12.09.18.
- [In18] Informatica: Informatica Enterprise Data Catalog. <http://www.informatica.com/de/products/big-data/enterprise-data-catalog.html>, Stand: 12.09.18.
- [Ka15] Kassner, L. et al.: Product Life Cycle Analytics - Next Generation Data Analytics on Structured and Unstructured Data. In *Procedia CIRP*, 33, S. 35–40, 2015.
- [KB10] Khatri, V.; Brown, C. V.: Designing data governance. In *Communications of the ACM*, 53(1), S. 148–152, 2010.
- [KBM10] Kemper, H.-G.; Baars, H.; Mehanna, W.: *Business Intelligence – Grundlagen und praktische Anwendungen*. Vieweg+Teubner, Wiesbaden, 2010.
- [KE06] Kemper, A.; Eickler, A.: *Datenbanksysteme*. Oldenbourg, München, 2006.

- [Lv17] Lv, Z. et al.: Next-Generation Big Data Analytics. State of the Art, Challenges, and Future Research Topics. In *IEEE Transactions on Industrial Informatics*, 13(4), S. 1891–1899, 2017.
- [Ma17] Mathis, C.: Data Lakes. In *Datenbank-Spektrum*, 17(3), S. 289–293, 2017.
- [MT16] Miloslavskaya, N.; Tolstoy, A.: Big Data, Fast Data and Data Lake Concepts. In *Procedia Computer Science*, 88, S. 300–305, 2016.
- [MW15] Marz, N.; Warren, J.: *Big data. Principles and best practices of scalable real-time data systems*. Manning, Shelter Island, 2015.
- [OL13] OLeary, D.: Artificial intelligence and big data. In *IEEE Intelligent Systems*, 28(2), S. 96–99, 2013.
- [QHV16] Quix, C.; Hai, R.; Vatov, I.: Metadata Extraction and Management in Data Lakes With GEMMS. In *Complex Systems Informatics and Modeling Quarterly*, (9), S. 67–83, 2016.
- [Sh18] Sharma, B.: *Architecting data lakes*. O’Reilly, Sebastopol, 2018.
- [Te15] Terrizzano et al.: *Data Wrangling: The Challenging Journey from the Wild to the Lake: Proceedings of the Biennial Conference on Innovative Data Systems Research (CIDR) 2015*, 2015.
- [Te18] Terradata: Kylo. www.kylo.io, Stand: 12.09.18.
- [VVS00] Vetterli, T.; Vaduva, A.; Staudt, M.: Metadata standards for data warehousing. In *ACM SIGMOD Record*, 29(3), S. 68–75, 2000.
- [Wa18] Waterline Data: Waterline Smart Data Catalog. <https://www.waterlinedata.com/product-overview>, Stand: 12.09.18.

Partial Reload of Incrementally Updated Tables in Analytic Database Accelerators

Knut Stolze,¹ Felix Beier,¹ Jens Müller¹

Keywords: accelerator, data synchronization, replication, mvcc

Abstract:

The IBM Db2 Analytics Accelerator (IDAA) is a state-of-the-art hybrid database system that seamlessly extends the strong transactional capabilities of Db2 for z/OS (Db2z) with very fast column-store processing in Db2 Database for Linux, Unix, and Windows. IDAA maintains a copy of the data from Db2z in its backend database. The data can be synchronized in batch with a granularity of table partitions, or incrementally using replication technology for individual rows.

In this paper we present the enablement of combining the batch loading of a true subset of a table's partitions for replicated tables. The primary goal for such an integration is to ensure data consistency. A specific challenge is that no duplicated rows stemming from the two data transfer paths come into existence. We present a robust and yet simple approach that is based on IDAA's implementation of multi-version concurrency control.

1 Introduction

The IBM[®] Db2[®] Analytics Accelerator for z/OS (IDAA, cf. Fig. 1) [Pa15] is an extension for IBM's[®] Db2[®] for z/OS[®] database system (Db2z) [IB18]. Its primary objective is the extremely fast execution of complex, analytical queries on a snapshot of the data copied from Db2z. Many customer installations have proven that the combination of Db2z with a seamlessly integrated IDAA delivers an environment where both, transactional workload and analytical queries, are supported without negatively impacting existing and new applications. In fact, many new use cases and applications have been developed specifically because of the existence of IDAA and the performance it delivers on core business data. The achieved query acceleration for analytical workloads is at least an order of magnitude, often even exceeding that.

After the initial version of IDAA became available in 2011, functional enhancements were continuously added to expand the product's scope. Fine-granular data synchronization mechanisms, like *partial reload* and *incremental update*, were among the first enhancements.

¹ IBM Germany Research & Development GmbH, {*stolze, febe, jens.mueller*}@de.ibm.com

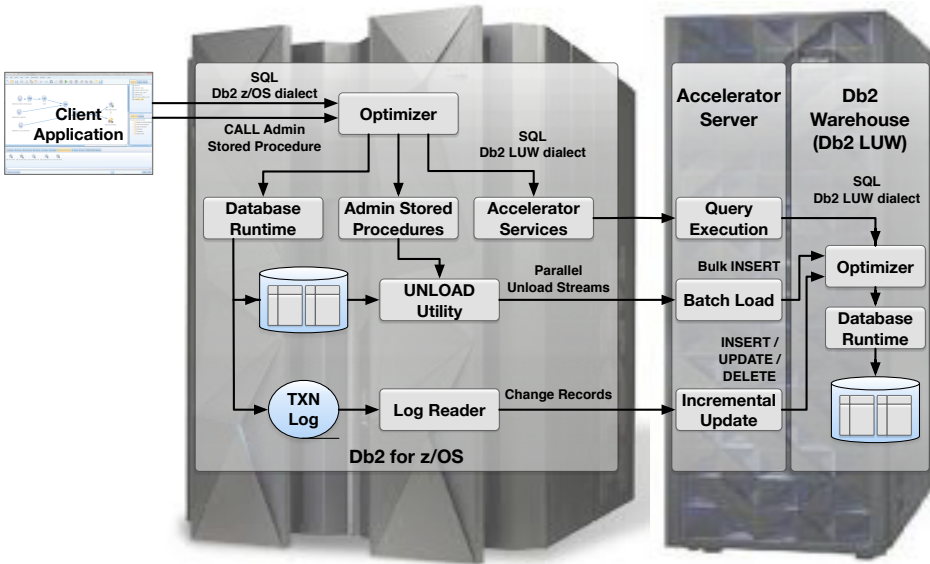


Fig. 1: System Overview of the IBM Db2 Analytics Accelerator for z/OS (IDAA)

Partial reload allows batch reloading of a subset of the table on a granularity of horizontal partitions. A table can be partitioned either by-range or by-growth. Contrary to that, incremental update operates on individual rows. It reads the transaction log of Db2 to detect data changes and replicates those to IDAA’s backend database system.

Starting with IDAA Version 6, a strategic business decision heavily influenced IDAA and basically led to a new product: the backend database system changed to IBM Db2 Database² (for Linux, UNIX, Windows platforms) [IB16]. Db2 LUW does not provide multi-version concurrency control (MVCC) and snapshot semantics as Netezza [Fr11] did. In this paper, we present our work to implement MVCC on top of Db2 LUW. We developed a mechanism to allow Incremental Update to operate most efficiently, while allowing partial reload on such tables as well. The interaction of both data synchronization processes posed several challenges, and the solution to them is explained.

The remainder of the paper is structured as follows. The architecture of the IBM Db2 Analytics Accelerator is briefly touched on in section 2, including the simulation of MVCC in that environment. IBM’s Change Data Capture (CDC) [Be12] is being employed as replication technology to detect and extract data changes in Db2z and apply them to IDAA’s backend database. Section 3 outlines how CDC deals with IDAA’s MVCC setup. How we bring CDC and partial reload together is explained in section 4. Finally, the paper concludes with a summary in section 5.

² IBM Db2 Database for Linux, UNIX, and Windows is called Db2 LUW henceforth.

2 IBM Db2 Analytics Accelerator

The IBM Db2 Analytics Accelerator (IDAA) [Pa15] is a hybrid system, which uses Db2[®] for z/OS[®] [IB18] for transactional workload with excellent performance characteristics, proven time and again in many customer scenarios. A copy of the Db2z data resides in the accelerator, which deals with analytical workload in an extremely high-performing way.

The benefits of this system are a reduced complexity of the necessary IT infrastructure on customer sites for executing both types of workloads and its tight integration into the existing Db2z system, which results in overall cost reductions for customers. A single system is used and not a whole zoo of heterogenous platforms needs to be operated and maintained.³ Aside from the system management aspects, existing investments into a business' applications is protected, which is crucial for companies of a certain size. With IDAA, existing applications can continue to use Db2z unchanged. At the same time, the analytics capabilities of IDAA can be exploited without any (or just with minuscule) changes.

IDAA provides the data storage and SQL processing capabilities for large amounts of data. Fig. 1 already showed the high-level architecture. Key is the seamless integration of all components with Db2z to leverage the appliance as analytical backend. Db2z is the central entry point for queries and administrative functions. Its optimizer decides whether to execute queries locally in Db2z – or to route them to the accelerator. In the second case, a new SQL statement in the dialect of Db2 LUW will be generated by Db2z and sent to the Accelerator Server, which passes it on to Db2 LUW for execution. Db2 LUW itself optimizes the statement and executes it on its own copy of the data. The copy is created (and maintained) by the admin stored procedures (running in Db2z) that are provided as part of the IDAA solution. Thus, the interface for customers to work with IDAA is Db2z only.

2.1 Data Replication Strategies

IDAA offers three options for refreshing the data that has been shadow-copied from Db2z. Entire tables or individual table partitions can be refreshed in the accelerator batch-wise. The latter is called *partial reload*. Fig. 2 and 3 illustrate both batch-loading scenarios conceptually. The Db2z table is on the left side, and the IDAA table on the right. The sketch above the arrow shows which pieces of the Db2z table are copied to the IDAA table.

For tables having a higher update frequency and where a high data concurrency is desired, the Incremental Update feature (cf. Fig. 4) is suitable. This feature uses IBM[®] InfoSphere[®] Change Data Capture (CDC) [IB13a, Be12], which reads Db2z transaction logs and extracts all changes to accelerated Db2z tables. Unlike the batch-load strategies, CDC replicates only changed rows while unchanged data is not affected. A draw-back is the dependency

³ Other systems like [KN11] may also provide an integrated solution. However, customers have made significant investments over the past decades into Db2z and applications on top of it. That's why IDAA has been developed as an enhancement of that platform right from the start.

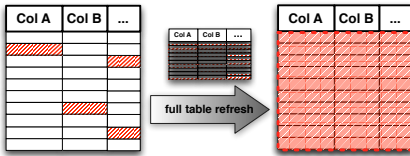


Fig. 2: Full Table Reload

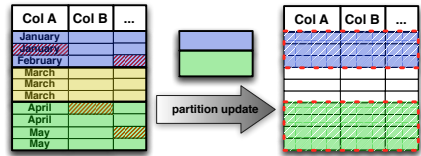


Fig. 3: Partial Reload

on the transaction log: any data maintenance operation that bypasses the log cannot be replicated. For example, batch-loading data into Db2z is often done using the LOAD utility, which does not log.

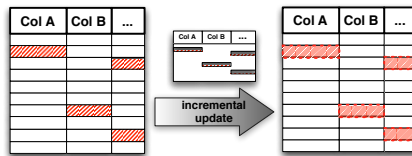


Fig. 4: Refreshing Table Data with Incremental Update

The batch-load strategies into IDAA offer great throughputs but require to copy large data volumes and, hence, are high-latency operations. The incremental update strategy replicates lower data volumes with larger overheads per tuple, but with low latency. It is the responsibility of the database user/administrator to trigger the refresh of the data in each accelerator or to set up incremental update where appropriate [IB13b]. Since many customers have tens of thousands of tables in Db2z and also in IDAA, it is important to understand query access patterns to the individual tables and analyzing the accelerated workload with the help of monitoring tools.

2.2 Simulating Multi-Version Concurrency Control using Views

Multi-version concurrency control (MVCC) [WV02] is a well-known means in database systems to enable highly parallel data access for read-only and read/write operations. The basic idea is to not apply data modifications in-place but rather to create a new version of the row on every such modification. The old row versions are still available for concurrently running transactions. The following Fig. 5 illustrates the physical storage of such rows. The arrows indicate the chain of versions that is created for a sample Row 1.

Each INSERT operation produces a new row, of course. But also each UPDATE effectively produces a new version of the row. The prior existing row remains, but it is marked as logically deleted. Likewise, DELETE operations only mark a row as logically deleted, but the row remains in existence.

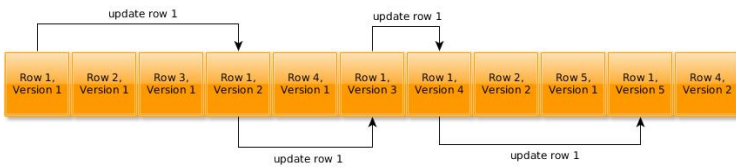


Fig. 5: Example for MVCC storage of rows

A key element of this approach is to determine which versions of the rows shall be visible to data access operations to provide consistent snapshots. For example, a query may access the table with the (physical) rows shown in Fig. 5. If the query (or rather its transaction) was started before any of the updates took place, it should see Row 1 in Version 1 – but not any other version of that row, which may have been created by concurrently running data modifications. Likewise, if the query starts after the 2nd update of Row 1, it should see that row in Version 3, but not any of the prior existing versions.

IDAA’s Db2 LUW backend database system does not provide MVCC out of the box. Using SQL transactions to apply all data changes atomically is also not an option, due to limitations on the available transaction log space and lock contention that can occur. (Those are traditional issues with long-running transactions in database systems.) IDAA uses views to define data visibility and, thus, simulates MVCC. That permits small, arbitrary transactions to populate data in the backend and even committing the changes in between. All data ingested into the target table in IDAA via LOAD processing will be versioned along the way. Data visibility is not handled on individual rows but rather per *partition*. An artificial column (named *mapped partition ID* or *backend partition ID*) is added to the table in IDAA’s backend to store the *mapped* value for a Db2z partition.

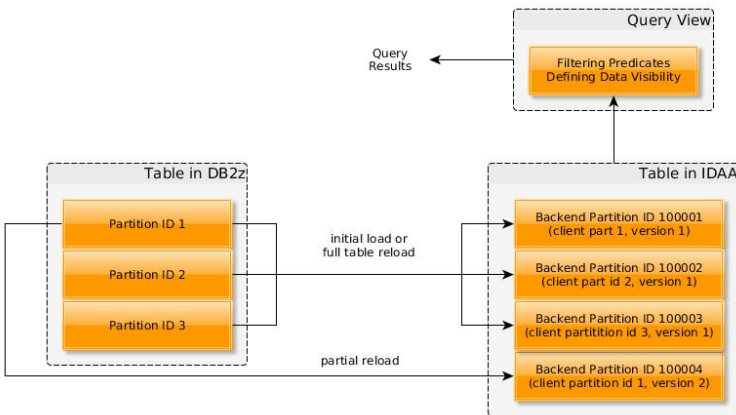


Fig. 6: Mapping of Db2z-side partitions to IDAA table

The view definition will use a simple predicate to ensure correct data visibility. The generalized pattern for the predicate is the following. The NOT IN predicate is only added if there are actually any invisible partitions.

```
mappedPartitionId <= maximum-mapped-id-of-all-visible-partitions AND
mappedPartitionId NOT IN ( list-of-mapped-ids-of-invisible-partitions )
```

For our specific example in Fig. 6, the predicate becomes:

```
mappedPartitionId <= 100004 AND mappedPartitionId NOT IN ( 100001 )
```

All LOAD operations are serialized on a table, which means that no concurrent LOADs can happen and potentially interfere. The values used for the mappedPartitionId column in the target table start at 100,000 and are monotonously growing. Thus, any rows that a later LOAD operation inserts into the table will automatically be invisible, whether those rows are committed already in the backend database or not. Once the LOAD operation finishes, it changes the predicates in the view definition, sets the new value for the maximum mapped partition ID, and updates the list of invisible partitions IDs. Fig. 7 visualizes this. Subsequent SQL statements, like queries, will pick up the new view definition while currently running SQL statements continue to use the view definition as it was in effect when the SQL statement started its execution.

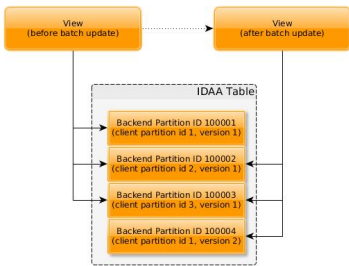


Fig. 7: View Update for Partial Reload

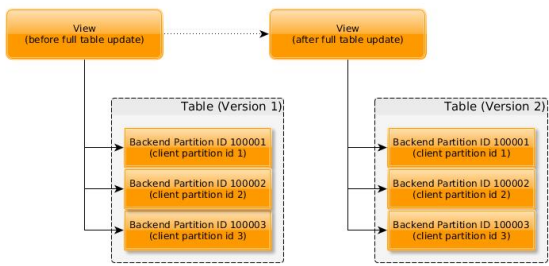


Fig. 8: View Update for Full Reload

Changing the definition of a view is an atomic and fast operation. Furthermore, view resolution takes place during the compilation of SQL statements. The compiled SQL statement remains unchanged during its execution – even if the view definition changes concurrently. Thus, stable and consistent results for the SQL statement are guaranteed.

Db2 LUW keeps track of which currently running SQL statements are using which version of the view definition. Due to that, it is possible to determine whether a specific version of the view definition is no longer in use. Once an old version of a view definition is no longer used, those invisible rows will not be accessed henceforth and can be physically purged from the table. In IDAA, this purging is implemented in asynchronous reorganization jobs that periodically check view usage and drop outdated data using a DELETE statement.

A special case occurs if all partitions of a table are to be reloaded. In such a *full table reload*, it is possible to create a new target table into which all rows are copied, and the view definition is changed to refer to that new table (cf. Fig. 8). This optimization drastically simplifies the physical cleanup of old data, because a `DROP TABLE` statement is sufficient.

3 Replication

CDC, the replication technology for IDAA's Incremental Update feature, is not aware of the mapping of the Db2z-side partition ID as described in Sec. 2.2. It would have been possible to extend CDC to perform that mapping, but that raises 2 major concerns: synchronization and performance.

Whenever a `LOAD` operation in IDAA happens, this mapping has to be updated. CDC has to be aware of the new mapping, and it has to be aware of it at the right point in time. Replication runs asynchronously, but changing the mapping has to happen synchronously. While this can be solved with well-known techniques, implementing them reliably and robustly requires a significant effort.

Such a synchronization of the partition ID mapping indirectly causes heavy performance impacts for the replication itself. A lookup via that map would have to occur for each row being replicated. Achieving throughput rates that can keep up with transactional workload occurring on Db2z-side is virtually impossible with such a bottleneck.

IDAA's Incremental Update takes a different route: No mapping of the partition ID occurs. Instead, all replicated rows get the Db2z-side partition ID assigned. Those partition IDs are in the range of 0 to 4096, which is lower than the starting value for mapped partition IDs. Due to the filtering predicate in the view definition (`mappedPartitionID <= x`) new replicated rows will be immediately visible once the Db2 LUW update transaction commits, which also physically deletes any old rows right away. Extending the example from Fig. 6 to include rows produced by replication in the target table leads to the situation in Fig. 9.

4 Handling Replicated Rows during Partial Reload

A partial reload on a replicated table runs into the issue that each loaded row carries a mapped partition ID greater than 100000. Any existing version of that row with a mapped partition ID assigned by IDAA during a prior full table load or partial reload does not pose a problem. Such a row will become invisible once the view definition is changed at the end of the `LOAD` operation. However, that very row may exist already in the target table with the Db2z-side partition ID – which is what the replication tool would generate.

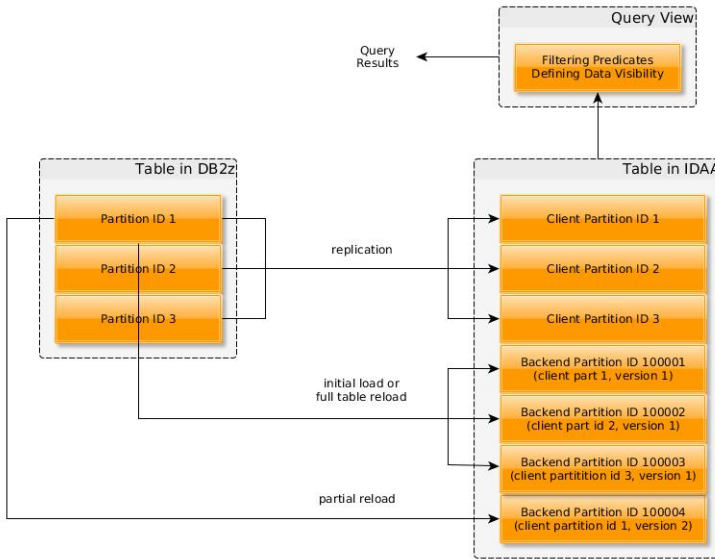


Fig. 9: Example for MVCC storage of rows

The filtering predicate in the view definition will always treat all rows with a Db2z-side partition ID as visible. That is, the Db2z-side partition ID must not be included in the NOT IN list and it must not be filtered out by any other means. The reason is that data changes occur in Db2z after the partial reload, and then those changes are replicated to the accelerator, using the Db2z-side partition ID. Propagating those changes again via a partial reload must not create duplicates, which implies that formerly replicated rows must be made invisible as part of the partial reload. Fig. 10 shows the desired result.

Several ideas came to our mind to solve this problem of hiding rows replicated before the partial reload but still allowing rows replicated after the partial reload to be visible. The three most promising are discussed in the following.

4.1 Deleting Replication Rows

The first idea to get rid of the old replicated rows is to delete them by a SQL statement like `DELETE FROM ... WHERE mappedPartitionId = ...`. That statement could be executed during the partial reload. However, that is not a viable option. Concurrently running queries may still need to access those rows. In fact, such queries may run longer than the partial reload itself. Physically deleting the rows would lead to incorrect query results.

4.2 Using Separate Tables for Partial Reload

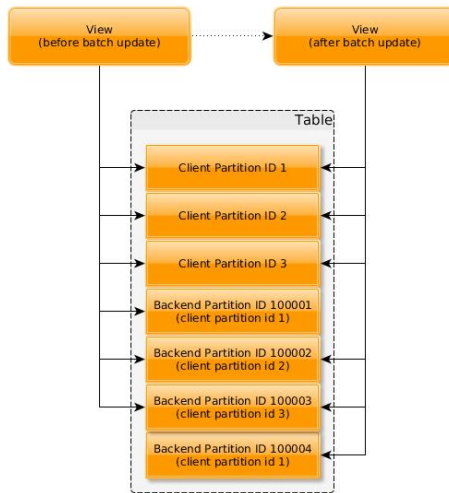


Fig. 10: View Update on Partial Reload with Replicated Data

Similarly to the optimization for a full table reload, partial reload could use separate tables for the newly loaded data. The view definition could be used to pick the data for each partition from the correct table and combine all of them with a `UNION ALL` clause. Correctness is not jeopardized, but this approach increases the number of base tables by several orders of magnitude. It also impacts SQL statements accessing the view and increases the pressure on the Db2 LUW optimizer to come up with a good access plan. Since IDAA is used by our customers to drive their business, maintainability is at least as important as correctness and performance. This idea was yet not further pursued since the next one has proven to meet our expectations with much less overhead and was easy to implement.

4.3 Moving Replicated Rows into Loaded Partitions

As part of the partial reload, all previously replicated rows of the affected partition(s) are updated. The value in the `mappedPartitionId` column are changed from the Db2z-side partition ID to the IDAA-defined partition ID – as it was before the update. Referring to Fig. 10, all rows of partition 1 (mapped to 100001) are being reloaded (now mapped to 100004). All rows that have partition ID 1 originate from the same Db2z-side partition. So the following SQL statement moves those rows to the old mapped partition ID 100001.

```
UPDATE ...
SET    mappedPartitionId = 100001
WHERE  mappedPartitionId = 1
```

Visibility of the rows being updated is not impacted – the rows are visible before the filtering predicates in the view are changed, regardless of whether the partition ID being 1 or 100001. After the filtering predicates are changed at the end of the batch update, those updated rows become invisible, exactly as intended. All the way, concurrently running queries can access the rows.

This UPDATE statement is executing concurrently to the insertion of the rows being reloaded with mapped partition ID 100004. The statement is actually slightly more complex to limit the number of rows being updated by one statement execution in order to inject intermediate COMMITs. Using a sequence of small-sized intermediate transactions significantly reduces the amount of logging and locking that has to be done on the target table. In particular, lock escalations are avoided. For example, the following statement, which is executed in a loop until no more rows are found and modified, limits updates the first 100,000 rows only:

```
UPDATE ( SELECT ROW_NUMBER() OVER () as rowNumber, mappedPartitionId
          FROM ...
          WHERE mappedPartitionId = 1 ) AS table
SET    mappedPartitionId = 100001
WHERE  rowNumber BETWEEN < 100000
```

Evaluation We found that the UPDATE statement has about the same run time as an INSERT statement to process the same number of rows. Since the INSERT and the UPDATE can run concurrently, there is no measurable impact. Only when the system reaches its limits in terms of CPU and I/O resources, the INSERTs need to be throttled [SBM17].

5 Summary

In this paper we have presented recent enhancements of the IBM Db2 Analytics Accelerator to support reloading a selected subset of a table's partitions, which is also synchronized between Db2 for z/OS and the accelerator using the Incremental Update feature. Data consistency has to be ensured to avoid that duplicate rows become visible on accelerator-side due to the two different data synchronization paths. The solution turned out to be really simple but very effective at the same time: an UPDATE statement moves the replicated rows into the old mapped partition, which is subsequently made invisible by changing the filtering predicate of the view definition. Since this can be parallelized with the data transfer and insertion that has to take place as part of the partial reload anyway, no performance impact on a used system could be observed.

The next step for our work is the further investigation of reloading partition data into separate tables (see section 4.2). We strive to evaluate how well the Db2 LUW optimizer handles this on all possible cases. Functionality-wise, it is not yet clear to us if replicating the deletion

of a row would try to scan all legs of the UNION ALL or if the Db2 LUW optimizer can directly determine which of the base tables need to be processed and which ones can be skipped. A similar question arises for insertion of new (replicated) rows through the view and whether they will be applied to the correct target base table. It may become necessary to employ INSTEAD OF triggers on the view to ensure the correct row distribution, whose impact on replication performance and throughput needs to be evaluated.

Trademarks

IBM, DB2, and z/OS are trademarks of International Business Machines Corporation in USA and/or other countries. Other company, product or service names may be trademarks, or service marks of others. All trademarks are copyright of their respective owners.

References

- [Be12] Beaton, A.; Noor, A.; Parkes, J.; Shubin, B.; Ballard, C.; Ketchie, M.; Ketelaars, F.; Rangarao, D.; Tichelen, W.V.: . Smarter Business: Dynamic Information with IBM InfoSphere Data Replication CDC. IBM Redbooks, 2012.
- [Fr11] Francisco, P.: . The Netezza Data Appliance Architecture: A Platform for High Performance Data Warehousing and Analytics. IBM Redbooks, 2011.
- [IB13a] IBM: . IBM InfoSphere Data Replication V 10.2.1 documentation, 2013.
- [IB13b] IBM: Synchronizing Data in IBM DB2 Analytics Accelerator for z/OS. Technical report, IBM, 2013. <http://www-01.ibm.com/support/docview.wss?uid=swg27038501>.
- [IB16] IBM: . IBM Db2 Database for Linux, UNIX, and Windows Version 11.1, 2016. https://www.ibm.com/support/knowledgecenter/en/SSEPGG_11.1.0/com.ibm.db2.luw.welcome.doc/doc/welcome.html.
- [IB18] IBM: . Db2 12 for z/OS documentation, 2018. https://www.ibm.com/support/knowledgecenter/en/SSEPEK_12.0.0/home/src/tpc/db2z_12_prodhome.html.
- [KN11] Kemper, Alfons; Neumann, Thomas: HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots. In: Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany. pp. 195–206, 2011.
- [Pa15] Parziale, Lydia; Benke, Oliver; Favero, Willie; Kumar, Ravi; LaFalce, Steven; Madera, Cedrine; Muszytowski, Sebastian: . Enabling Real-time Analytics on IBM z Systems Platform. IBM Redbooks, 2015.
- [SBM17] Stolze, Knut; Beier, Felix; Müller, Jens: Autonomous Data Ingestion Tuning in Data Warehouse Accelerators. Datenbanksysteme für Business, Technologie und Web (BTW) 2017, March 2017.
- [WV02] Weikum, Gerhard; Vossen, Gottfried: Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

Dissertationspreise

Data Profiling – Effiziente Entdeckung Struktureller Abhängigkeiten

Thorsten Papenbrock¹

Abstract: Daten sind nicht nur in der Informatik, sondern auch in vielen anderen wissenschaftlichen Disziplinen ein unverzichtbares Wirtschaftsgut. Sie dienen dem Austausch, der Verknüpfung und der Speicherung von Wissen und sind daher unverzichtbar in Forschung und Wirtschaft. Leider sind Daten häufig nicht ausreichend dokumentiert um sie direkt nutzen zu können – es fehlen Metadaten, welche die Struktur und damit Zugriffsmuster der digitalen Informationen beschreiben. Informatiker und Experten anderer Disziplinen verbringen daher viel Zeit damit, Daten strukturell zu analysieren und aufzubereiten. Da die Suche nach Metadaten jedoch eine hoch komplexe Aufgabe ist, scheitern viele algorithmische Ansätze schon an kleinen Datenmengen.

In der Dissertation, die dieser Zusammenfassung zugrunde liegt, stellen wir drei neuartige Entdeckungsalgorithmen für wichtige und zugleich schwierig zu findende Typen von Metadaten vor: Eindeutige Spaltenkombinationen, funktionale Abhängigkeiten und Inklusionsabhängigkeiten. Die vorgeschlagenen Algorithmen übertreffen deutlich den bisherigen Stand der Technik in Laufzeit und Ressourcenverbrauch und ermöglichen so die Nutzbarmachung von erheblich größeren Datensätzen. Da die Anwendung solcher Algorithmen für fachfremde Nutzer nicht einfach ist, entwickeln wir zusätzlich das Programm METANOME zur intuitiven Datenanalyse. METANOME bietet dabei nicht nur die in dieser Arbeit vorgeschlagenen Algorithmen an, sondern auch Entdeckungsalgorithmen für andere Typen von Metadaten. Am Anwendungsfall der Schema-Normalisierung demonstrieren wir schließlich, wie die gefundenen Metadaten effektiv genutzt werden können.²

1 Extraktion struktureller Metadaten

Data Profiling bezeichnet eine Disziplin der Informatik, bei der es darum geht Metadaten in verschiedenen Datensätzen zu bestimmen. Die verschiedenen Typen von Metadaten reichen von einfachen Statistiken wie Tupelzahlen, Spaltenaggregationen und Wertverteilungen bis hin zu weit komplexeren Strukturen, insbesondere Inklusionsabhängigkeiten (INDs), eindeutige Spaltenkombinationen (UCCs) und funktionale Abhängigkeiten (FDs). Sofern vorhanden dienen diese Statistiken und Strukturen dazu, die Daten zu verstehen, sie effizient zu speichern, zu lesen und zu ändern. Da die meisten Datensätze ihre Metadaten aber nicht explizit als beschreibendes Regelwerk zur Verfügung stellen, sind Informatiker häufig gezwungen diese strukturellen Regeln mittels Data Profiling zu bestimmen.

¹ Universität Potsdam, Hasso-Plattner-Institut, Fachgebiet Informationssysteme, Prof.-Dr.-Helmert-Str. 2-3,
D-14482 Potsdam, Deutschland, thorsten.papenbrock@hpi.de

² Dieser Artikel ist ebenfalls erschienen im LNI Sammelband "Ausgezeichnete Informatikdissertationen 2017".

Inclusion Dependencies (INDs) <small>Pokemon.Location \sqsubseteq Location.Name</small>				Functional Dependencies (FDs) <small>Type \rightarrow Weak</small>						
ID	Name	Evolution	Location	Sex	Weight	Size	Type	Weak	Strong	Special
25	Pikachu	Raichu	Viridian Forest	m/w	6.0	0.4	electric	ground	water	false
27	Sandshrew	Sandslash	Route 4	m/w	12.0	0.6	ground	gras	electric	false
29	Nidoran	Nidorino	Safari Zone	m	9.0	0.5	poison	ground	gras	false
32	Nidoran	Nidorina	Safari Zone	w	7.0	0.4	poison	ground	gras	false
37	Vulpix	Ninetails	Route 7	m/w	9.9	0.6	fire	water	ice	false
38	Ninetails	null	null	m/w	19.9	1.1	fire	water	ice	true
63	Abra	Kadabra	Route 24	m/w	19.5	0.9	psychic	ghost	fighting	false
64	Kadabra	Alakazam	Cerulean Cave	m/w	56.5	1.3	psychic	ghost	fighting	false
130	Gyarados	null	Fuchsia City	m/w	235.0	6.5	water	electric	fire	false
150	Mewtwo	null	Cerulean Cave	m/w	122.0	2.0	psychic	ghost	fighting	true

(Name, Sex)

Unique Column Combinations (UCCs)

Abb. 1: Eine Relation über Pokémon, die drei ausgewählte Schlüsselbeziehungen zeigt: ein potentieller Primärschlüssel (UCC), ein Fremdschlüssel (IND) und eine innere Schlüsselabhängigkeit (FD).

Während einfache Statistiken noch relativ schnell zu berechnen sind, stellen die komplexeren Strukturen schwere, zumeist NP-vollständige Entdeckungsaufgaben dar. In der Regel ist es also auch mit gutem Domänenwissen nicht möglich, sie händisch zu bestimmen. Es wurden daher bereits verschiedenste Profiling Algorithmen entwickelt, um die Entdeckung zu automatisieren. Keiner dieser Algorithmen kann allerdings Datensätze von heutzutage typischer Größe verarbeiten, weil entweder der Ressourcenverbrauch oder die Rechenzeit effektive Grenzen überschreitet.

In dieser Arbeit stellen wir neuartige Profiling Algorithmen vor, die automatisch die drei populärsten Typen komplexer Metadaten entdecken, nämlich UCCs, FDs und INDs. Die Popularität dieser drei Strukturen begründet sich in der Tatsache, dass mit ihrer Hilfe die wichtigsten Formen von Schlüssel-Abhängigkeiten beschrieben werden: UCCs beschreiben Schlüssel *für* eine relationale Tabelle, FDs beschreiben Schlüssel *innerhalb* einer relationalen Tabelle und INDs beschreiben Fremdschlüsselbeziehungen *zwischen* relationalen Tabellen. Sie dienen damit nicht nur der Identifikation von Entitäten in einem Datensatz, sondern auch der Verknüpfung, Bereinigung, Anfrage, Integration und logischen Formatierung von Daten. Abbildung 1 zeigt eine Beispielrelation über Pokémon Daten – kleine “Taschenmonster” für Kinder. Von den ebenfalls dargestellten Schlüsselabhängigkeiten lernen wir, dass Pokémon über ihren Namen und ihr Geschlecht eindeutig identifiziert werden (UCC), der Typ eines Pokémon auch dessen Schwäche bestimmt (FD) und zusätzliche Informationen über die Herkunft eines Pokémon in einer anderen Tabelle gefunden werden können (IND).

Die Aufgabe eines Entdeckungsalgorithmus ist es alle gültigen Vorkommen einer speziellen Abhängigkeit aus einer gegebenen relationalen Instanz zu extrahieren – ein induktives Such- und Prüfverfahren also, welches die Daten systematisch auf Abhängigkeiten untersucht. Die von uns entwickelten Algorithmen nutzen dazu sowohl bewährte Entdeckungstechniken aus verwandten Arbeiten des Data Profiling, als auch für das Data Profiling neuartige Techniken, wie beispielsweise Teile-und-Herrsche Verfahren, hybrides Suchen, Progres-

sivität, Speichersensibilität, Parallelisierung und innovative Streichungsregeln. Über eine Reihe systematischer Experimente zeigen wir, dass die vorgeschlagenen Algorithmen nicht nur um Größenordnungen schneller sind als alle verwandten Algorithmen, sie heben auch einige der aktuellen Beschränkungen auf und sind so in der Lage Datensätze von häufig vorkommender Größe, d.h. mehrerer Gigabyte Größe, mit akzeptablem Speicher- und Zeitverbrauch zu verarbeiten. Um die entwickelten Algorithmen der Forschungs- und Entwicklungsgemeinschaft, sowie Informatik-Laien zugänglich zu machen, haben wir alle Verfahren in das praktische Profiling Werkzeug METANOME³ integriert, welches frei und quelloffen verfügbar ist. Zusammengefasst leistet diese Arbeit daher die folgenden Beiträge:

1. **HyFD**: Ein effizienter Algorithmus zur Entdeckung funktionaler Abhängigkeiten, der ein hybrides Suchverfahren zur Identifikation valider FDs im Suchraum einsetzt [PN16].
2. **HyUCC**: Ein effizienter Algorithmus zur Entdeckung eindeutiger Spaltenkombinationen, der ebenfalls auf eine hybride Suche setzt, um Schlüsselkandidaten zu finden [PN17b].
3. **BINDER**: Ein effizienter Algorithmus zur Entdeckung von Inklusionsabhängigkeiten, der mittels Datenpartitionierung die Prüfung von IND-Kandidaten beschleunigt [Pa15c].
4. **METANOME**: Ein leicht erweiterbares Data Profiling Werkzeug, das verschiedene Entdeckungsalgorithmen praktisch nutzbar macht [Pa15a].
5. **NORMALIZE**: Ein Algorithmus zur Schema-Normalisierung, der entdeckte Schlüssel-Abhängigkeiten automatisiert bewertet und zur Schema-Reorganisation einsetzt [PN17a].

Im Folgenden erläutern wir zunächst die theoretischen Grundlagen für diese Arbeit und fassen anschließend die einzelnen Beiträge der Dissertation kapitelweise zusammen. Die vollständige Fassung der Dissertation ist in englischer Sprache erschienen [Pa17].

2 Schlüsselabhängigkeiten

Das relationale Datenmodell stellt Daten in tabellarischer Form mittels eines festen Schemas dar und ist damit das zur Zeit am weitesten verbreitete Modell. Jede Spalte hat eine eindeutige Bezeichnung, und jede Zeile beschreibt eine in der Tabelle gespeicherte Entität. Spalten und Zeilen werden häufig auch als Attribute und Einträge bezeichnet. Wir definieren nun unsere drei Schlüsselabhängigkeiten als Beziehungen zwischen verschiedenen Spalten:

Funktionale Abhängigkeiten (FDs) werden geschrieben als $X \rightarrow A$ und drücken damit aus, dass alle Paare von Einträgen mit gleichem Wert in der Attribut-Menge X auch den gleichen Wert im Attribut A haben – die X -Werte bestimmen funktional die A -Werte. Wenn wir die relationale Instanz mit r und ihr Schema mit R bezeichnen, dann definiert sich dieser Zusammenhang formal als $\forall t_i, t_j \in r : t_i[X] = t_j[X] \Rightarrow t_i[A] = t_j[A]$, wobei $X \subseteq R$ und $A \in R$ ist. Um alle funktionalen Abhängigkeiten eines Datensatzes zu bestimmen reicht es aus, nur minimale Abhängigkeiten aufzuzählen, bei denen aus der Attributmenge X kein Attribut entnommen werden kann, ohne die FD zu verletzen. Da das Hinzufügen

³ www.metanome.de

beliebiger Attribute auf der linken Seite der FD die Abhängigkeit nicht verletzt, lassen sich alle nicht-minimalen FDs aus der vollständigen Menge aller minimalen FDs leicht ableiten.

Eindeutige Spaltenkombinationen (UCCs) sind Mengen von Attributen X , in denen kein Wert (bzw. keine Wertekombination) doppelt vorkommt. Jeder einzelne Wert identifiziert daher eindeutig einen bestimmten Eintrag in der Tabelle. Formal definieren wir eindeutige Spaltenkombinationen X mit $X \subseteq R$ als $\forall t_i, t_j \in r, i \neq j : t_i[X] \neq t_j[X]$. Analog zu FDs reicht es zur Entdeckung aller UCCs aus nur solche aufzuzählen, die minimal sind. Eine minimale eindeutige Spaltenkombination ist jene, die bei Entfernung eines beliebigen Attributes aus X ungültig wird. Aus den minimalen UCCs lassen sich dann ebenfalls alle anderen UCCs ableiten, indem diese durch weitere Attribute ergänzt werden.

Inklusionsabhängigkeiten (INDs) werden geschrieben als $R_i[X] \subseteq R_j[Y]$ und besagen, dass alle Werte (bzw. Wertekombinationen) der Attribute X auch in der Menge der Werte von Y vorkommen – sie sind also darin enthalten. Nützlich ist dieser Zusammenhang, weil über Join-Operationen die Einträge, die an diesen Werten hängen, miteinander verbunden werden können. Eine formale Definition dieses Zusammenhangs ist $\forall t_i[X] \in r_i, \exists t_j[Y] \in r_j : t_i[X] = t_j[Y]$, wobei r_i und r_j relationale Instanzen der Schemata R_i und R_j darstellen. Im Gegensatz zu den meisten anderen Arten von Datenabhängigkeiten suchen wir bei INDs nach maximalen Ausdrücken, da beim Entfernen von Attributen auf linker und rechter Seite der Beziehung die Gültigkeit immer erhalten bleibt, die Gültigkeit aber verloren gehen kann, wenn Attribute hinzugefügt werden.

Die Mengen aller FD, UCC und IND Kandidaten wird bestimmt durch die Potenzmengen ihrer linken Seiten. Der Suchraum wird daher häufig als Gitter (engl. lattice) von Kandidaten modelliert: Beginnend mit Beziehungen zwischen einzelnen Attributen werden sukzessiv mehr Attribute zu diesen Kandidaten hinzugefügt. Abbildung 2 visualisiert wie diese Gitter, also die Suchraumgrößen, für die einzelnen Abhängigkeiten mit der Anzahl an Attributen im Datensatz immer weiter wachsen. Bezüglich der Anzahl von Attributen m liegen die Komplexitäten der automatischen Entdeckung dieser drei Arten von Metadaten daher in $O(2^m)$ für UCCs, $O(2^m \cdot (\frac{m}{2})^2)$ für FDs und $O(2^m \cdot m!)$ für INDs (wobei $m!$ eine Vereinfachung ist) [Li12].

3 Entdeckung von funktionalen Abhängigkeiten

Um effizient alle minimalen funktionalen Abhängigkeiten zu finden haben verwandte Arbeiten bereits zwei unterschiedliche Ansätze vorgeschlagen: Der erste Ansatz testet die FD Kandidaten im Suchraumgitter systematisch von unten nach oben und streicht dabei als ungültig erkennbare Kandidaten von der weiteren Suche [Hu99]; der zweite Ansatz vergleicht alle Paare von Einträgen in der Tabelle, berechnet aus diesen Vergleichen alle Verletzungen der FDs (entsprechend ihrer Definition) und leitet am Ende aus der Menge aller Verletzungen die Menge aller gültigen minimalen FDs ab [FS99]. In einer evaluierenden Arbeit [Pa15b] haben wir festgestellt, dass der erste Ansatz auf breiten Datensätzen (ca. 40

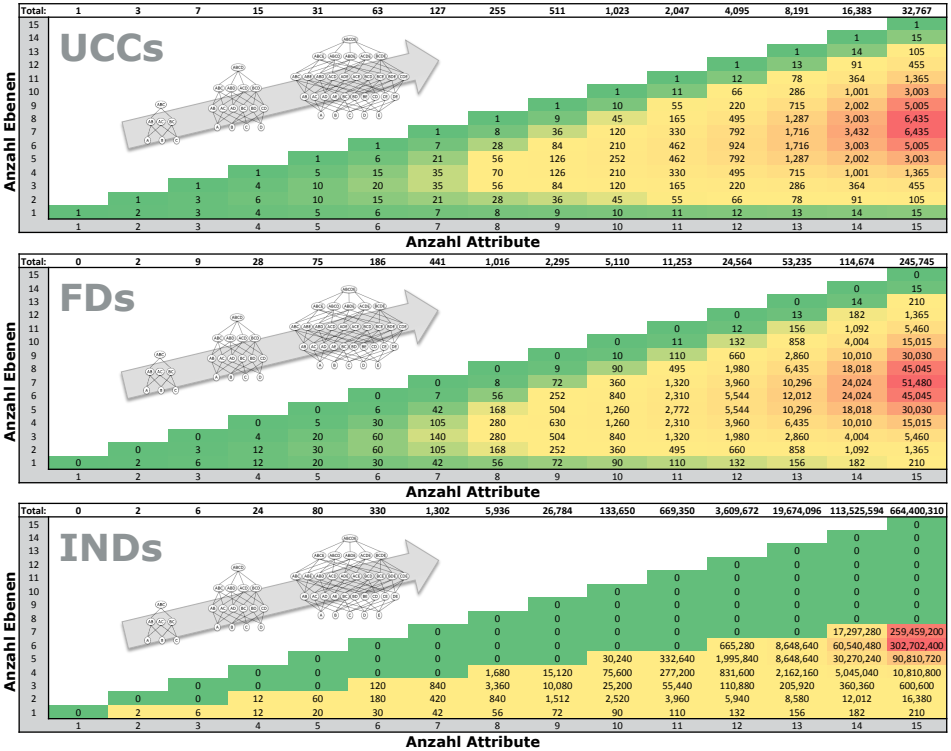


Abb. 2: Wachstum der Suchräume für UCCs, FDs und INDs mit der Anzahl Attribute.

Attribute und aufwärts) ineffizient wird und der zweite Ansatz bei langen Datensätzen (ca. 100.000 Einträge und aufwärts) Schwächen aufweist, viele Datensätze aber sowohl breit als auch lang sind. Wir schlagen daher den hybriden Algorithmus HyFD vor, der beide Strategien so kombiniert, dass sie sich gegenseitig unterstützen, um sowohl auf breiten als auch langen Datensätzen effizienter zu sein als je eine Strategie für sich allein.

Abbildung 3 veranschaulicht die hybride Suchstrategie von HyFD: In der Preprocessor Komponente wird der Datensatz zunächst in kompakte Indexstrukturen übersetzt: Positionlisten-Index (PLI) und komprimierter Entitäten-Index (PLI-RECORD). Anschließend beginnt die Sampler Komponente bestimmte Zeilen in der Tabelle zu vergleichen und daraus Verletzungen der FDs zu berechnen. Dies ist eine der beiden Suchstrategien, die allerdings abgebrochen wird, sobald eine überwiegende Mehrheit an Zeilenvergleichen keine neuen Verletzungen mehr geliefert hat – die Strategie ist ineffizient geworden. Daraufhin leitet der Inductor des HyFD Algorithmus all jene FD Kandidaten ab, die unter Berücksichtigung der bisher gefundenen Verletzungen noch minimal und gültig sein können. Das Ergebnis formt nun ein Kandidatenset, welches die Validator Komponente systematisch mit dem Gitter-Verfahren zu überprüfen beginnt. Auch hier gilt nun, dass HyFD die Validierung

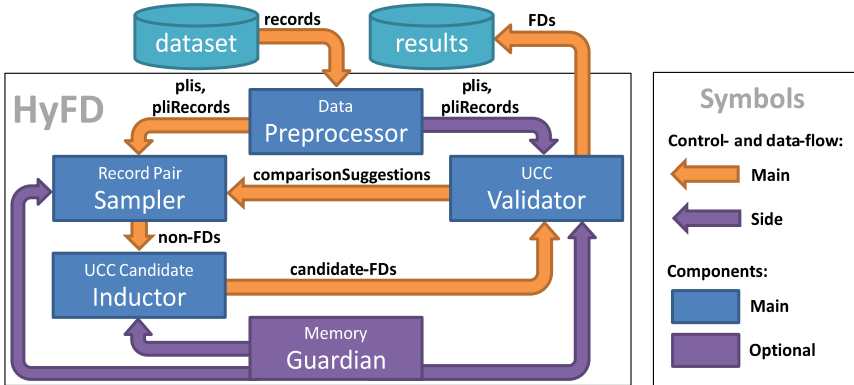


Abb. 3: Übersicht über den HyFD Algorithmus.

dynamisch abbricht, sobald eine Überzahl an Überprüfungen ein negatives Ergebnis liefern und sich die Validierung als ineffizient herausstellt. Wir tauschen nun Vergleichsvorschläge mit der Sampler Komponente aus und wechseln zurück zur ersten Suchstrategie. Alle Ergebnisse, die vom Validator bisher als gültig identifiziert wurden, sind exakte, minimale funktionale Abhängigkeiten und werden ausgegeben. Der Algorithmus terminiert, sobald dem Validator keine weiteren Kandidaten mehr vorliegen.

Tabelle 1 zeigt die Laufzeiten von HyFD für verschiedene Echtwelt-Datensätze. Die aufgeführten Datensätze bereiten den bisherigen Entdeckungsalgorithmen große Schwierigkeiten, da diese Ansätze entweder ein Speichervolumen von mehr als 100 GB überschreiten oder auch innerhalb mehrerer Tage kein vollständiges Ergebnis berechnen können. Der hybride Ansatz konnte jede Analyse auf einem Rechner mit 16 Prozessorkernen und 128 GB RAM (wovon tatsächlich aber nur ein Bruchteil genutzt wird) problemlos durchführen. Vor allem aber zeigt sich in diesen (und weiteren) Experimenten, dass die Laufzeit mehr von der zu findenden Ergebnisgröße abhängt als von der Größe des Datensatzes – eine ideale Laufzeiteigenschaft für Profiling Algorithmen.

Datensatz	Spalten [#]	Zeilen [#]	Größe [MB]	FDs [#]	HyFD [s/m/h/d]
TPC-H.lineitem	16	6 m	1,051	4 k	4 m
PDB.ATOM_SITE	31	27 m	5,042	10 k	64 m
SAP_R3.ILOA	48	45 m	8,731	16 k	8 h
SAP_R3.CE4HI01	65	2 m	649	2 k	10 m
NCVoter.statewide	71	1 m	561	5 m	31 h
UCI.flight	109	1 k	1	982 k	54 s

Tab. 1: Laufzeiten des HyFD Algorithmus auf verschiedenen Datensätzen.

4 Entdeckung von eindeutigen Spaltenkombinationen

Zur Entdeckung eindeutiger Spaltenkombinationen nutzt der HYUCC Algorithmus ein sehr ähnliches Verfahren wie der HYFD Algorithmus: Nach der Komprimierung des Datensatzes in Indexstrukturen wechseln sich eine Suchstrategie, die auf dem Vergleich von Einträgen basiert, und eine Suchstrategie, die Kandidaten im Suchgitter vergleicht, immer wieder gegenseitig ab. Die Strategien tauschen Zwischenergebnisse untereinander aus und implementieren zudem Optimierungen wie beispielsweise das frühzeitige Abbrechen und Parallelisieren von Kandidaten-Checks. So ist auch HYUCC allen bisherigen Ansätzen der UCC Entdeckung überlegen, da diese immer auf nur einer Suchstrategie beruhen.

Mit HYUCC konnten wir zeigen, dass hybrides Suchen nicht nur für funktionale Abhängigkeiten, sondern auch für weitere Arten komplexer Metadaten eine vielversprechender Ansatz ist. Mittlerweile haben sich weitere Entdeckungsalgorithmen mit demselben Suchprinzip bewährt, nämlich `AID-FD` [B116b] für approximative funktionale Abhängigkeiten (approximate FDs), `MVDDETECTOR` [Dr16] für Multivalued Dependencies (MVDs), `HYDRA` [B116a] für Denial Constraints (DCs) und `HYMD` [Dr18] für Matching Dependencies.

5 Entdeckung von Inklusionsabhängigkeiten

Die größte Herausforderung bei der Suche nach Inklusionsabhängigkeiten in einem relationalen Datensatz ist die effiziente Ausführung einer extrem großen Anzahl von Kandidatenüberprüfungen. Jede einzelne Überprüfung eines IND Kandidaten ist dabei aufwendiger als die Überprüfung eines FD oder UCC Kandidaten, weil nicht nur die Positionen gleicher Werte in einer Spalte relevant sind, sondern das exakte Übereinstimmen von Werten in verschiedenen Spalten. Eine Komprimierung des Datensatzes in Positionslisten ist daher nicht möglich. Um die Kandidaten möglichst effizient zu prüfen setzen einige IND Entdeckungsalgorithmen – wie auch unser `BINDER` Algorithmus – auf das simultane Testen mehrerer Kandidaten. Das von `BINDER` vorgeschlagene Testverfahren ähnelt einem großen Hash-Join aller Attribute, wohingegen Konkurrenzalgorithmen wie `SPIDER` [BLN06] auf Sort-Merge-Joins setzen.

Solch simultane Validierungsverfahren für INDs sind recht effizient, allerdings ist auch der benötigte Speicherbedarf hoch. Damit der Algorithmus nicht – wie einige verwandte Verfahren – an Speichermangel scheitert, schlägt `BINDER` einen Teile-und-Herrsche Ansatz für die Datenhandhabung vor: Wie in Abbildung 4 veranschaulicht wird der Datensatz zunächst Attribut-weise per Hash-Verfahren in Körbe aufgeteilt. Der Algorithmus entfernt dabei doppelte Werte in den Körben, um diese möglichst klein zu halten. Alle Körbe landen schließlich auf der Festplatte und werden dann zum Validieren der IND Kandidaten schrittweise wieder eingelesen. Wurde ein Attribut von allen IND Kandidaten entfernt, so müssen dessen Körbe in kommenden Validierungsschritten nicht mehr mitgelesen werden. Alle Kandidaten, die alle Validierungsschritte überstehen, sind gültige INDs und werden von `BINDER` ausgegeben.

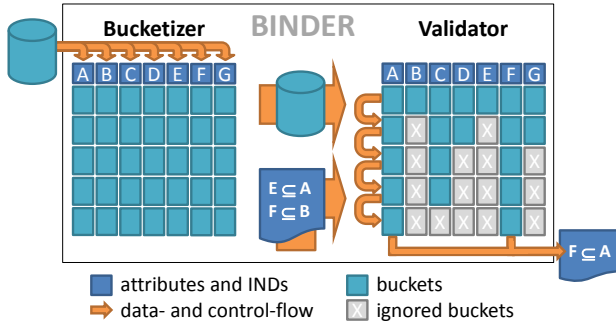


Abb. 4: Bucketierung und Validierung im BINDER Algorithmus.

Unsere Experimente zum BINDER Algorithmus zeigen, dass dieser meist effizienter ist als seine Konkurrenz. Der wichtigste Beitrag besteht aber darin, dass der Algorithmus an keiner der bisher bekannten Grenzen scheitert: Er setzt nicht das Vorhandensein einer Datenbank voraus, er scheitert nicht an unzureichend großem Hauptspeicher und er erschöpft nicht das Maximum offener Dateizeiger (engl. file handles) eines Betriebssystems.

6 Das Metanome Data Profiling Werkzeug

Aufgrund der praktischen Relevanz des Data Profilings hat die Industrie verschiedene Profiling Werkzeuge entwickelt, die Informatiker in ihrer Suche nach Metadaten unterstützen sollen. Diese Werkzeuge bieten zwar eine gute Unterstützung für die Berechnung einfacher Statistiken, und sie sind auch in der Lage einzelne Abhängigkeiten zu validieren. Allerdings mangelt es ihnen an Funktionen zur echten *Entdeckung* von Metadaten. Um diese Lücke zu schließen schlagen wir das Werkzeug METANOME vor, eine erweiterbare Profiling Plattform, die nicht nur unsere eigenen Algorithmen, sondern auch viele weitere Algorithmen anderer Forscher integriert. Derzeit bieten wir 21 Entdeckungsalgorithmen für 9 verschiedene Arten von Metadaten an und arbeiten kontinuierlich mit Wissenschaftlern anderer Institute, darunter Institute in Frankreich, Italien, Kanada, Neuseeland, Indien, China und den USA, an weiteren Verfahren. Wir haben METANOME ebenfalls in Kooperation mit den Unternehmen FlixBus, IBM und SAP erfolgreich eingesetzt.

Abbildung 5 zeigt die Nutzeroberfläche des Tools: Im linken Teil der Oberfläche werden die zu entdeckenden Metadaten über ihre Algorithmen ausgewählt, links daneben befindet sich der Import- und Auswahlbereich für Datensätze, und im unteren Teil können Profiling Prozesse konfiguriert und ausgeführt werden. Die Ergebnisse werden dann nach erfolgreicher Ausführung in einer weiteren Maske angezeigt. Auf diese Weise macht METANOME unsere Forschungsergebnisse nicht nur für Informatiker sondern auch für fachfremde Nutzer zugänglich. Neben der Metadaten-Entdeckung bietet die Plattform auch Unterstützung bei der Bewertung und Visualisierung gefundener Metadaten.

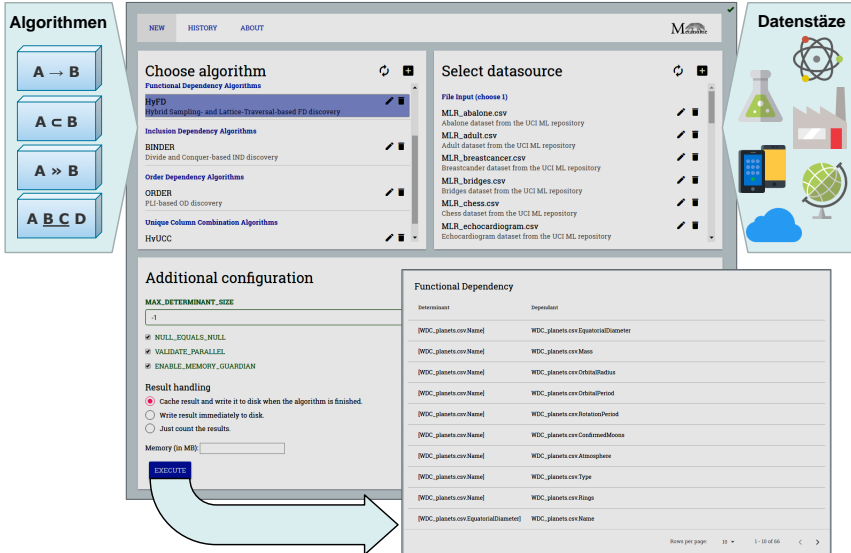


Abb. 5: Die Nutzersicht auf das METANOME Werkzeug mit dem Fenster zur Erstellung von Profiling-Läufen (Mitte) und einem Ergebnis-Fenster (Vorne).

7 Metadaten getriebene Schema-Normalisierung

Unsere neuen Profiling Algorithmen ermöglichen die effiziente Entdeckung aller syntaktisch korrekten Metadaten auf realistisch großen Datensätzen. Dies führt nun zur Folgeaufgabe, aus den gefundenen Abhängigkeiten nur die für einen gegebenen Anwendungsfall semantisch bedeutsamen Teile zu extrahieren. Das Extrahieren bedeutsamer Metadaten aus allen findbaren Abhängigkeiten ist eine Herausforderung, da zum einen die Mengen der gefundenen Metadaten überraschenderweise groß sind (oft größer als der untersuchte Datensatz selbst) und zum anderen die Entscheidung über die Anwendungsfall-spezifische semantische Relevanz einzelner Abhängigkeiten schwierig ist.

Um zu zeigen, dass die Vollständigkeit der Metadaten sehr wertvoll für ihre Nutzung ist, veranschaulichen wir die effiziente Verarbeitung und effektive Bewertung von Schlüssel-Abhängigkeiten am Anwendungsfall der Schema-Normalisierung: Das NORMALIZE Verfahren bewertet automatisch alle gefundenen Abhängigkeiten daraufhin, ob sie auch semantisch korrekte Schlüssel darstellen, und strukturiert die zugehörige Tabelle anschließend entsprechend um. Wir haben NORMALIZE getestet, indem wir zunächst wohlgeformte Schemata denormalisiert haben, dann alle Schlüsselabhängigkeiten mit METANOME finden ließen und anhand dieser Abhängigkeiten schließlich ein normalisiertes Schema abgeleitet haben. Da das abgeleitete Schema dem ursprünglichen sehr ähnlich ist, schließen wir auf eine hohe Effektivität für den NORMALIZE Algorithmus.

Literaturverzeichnis

- [B116a] Bleifuß, Tobias: Efficient Denial Constraint Discovery. Masterarbeit, Hasso-Plattner-Institute, Prof.-Dr.-Helmert-Str. 2-3, D-14482 Potsdam, 2016.
- [B116b] Bleifuß, Tobias; Bülow, Susanne; Frohnhofen, Johannes; Risch, Julian; Wiese, Georg; Kruse, Sebastian; Papenbrock, Thorsten; Naumann, Felix: Approximate Discovery of Functional Dependencies for Large Datasets. In: Proceedings of the International Conference on Information and Knowledge Management (CIKM). S. 1803–1812, 2016.
- [BLN06] Bauckmann, Jana; Leser, Ulf; Naumann, Felix: Efficiently Computing Inclusion Dependencies for Schema Discovery. In: ICDE Workshops. S. 2, 2006.
- [Dr16] Draeger, Tim: Multivalued Dependency Detection. Masterarbeit, Hasso-Plattner-Institute, Prof.-Dr.-Helmert-Str. 2-3, D-14482 Potsdam, 2016.
- [Dr18] Draeger, Tim: Efficient Discovery of Matching Dependencies. Masterarbeit, Hasso-Plattner-Institute, Prof.-Dr.-Helmert-Str. 2-3, D-14482 Potsdam, 2018.
- [FS99] Flach, Peter A; Savnik, Iztok: Database dependency discovery: a machine learning approach. *AI Communications*, 12(3):139–160, 1999.
- [Hu99] Huhtala, Ykä; Kärkkäinen, Juha; Porkka, Pasi; Toivonen, Hannu: TANE: An efficient algorithm for discovering functional and approximate dependencies. *The Computer Journal*, 42(2):100–111, 1999.
- [Li12] Liu, Jixue; Li, Jiuyong; Liu, Chengfei; Chen, Yongfeng: Discover Dependencies from Data – A Review. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 24(2):251–264, 2012.
- [Pa15a] Papenbrock, Thorsten; Bergmann, Tanja; Finke, Moritz; Zwiener, Jakob; Naumann, Felix: Data Profiling with Metanome. *Proceedings of the VLDB Endowment*, 8(12):1860–1863, 2015.
- [Pa15b] Papenbrock, Thorsten; Ehrlich, Jens; Marten, Jannik; Neubert, Tommy; Rudolph, Jan-Peer; Schönberg, Martin; Zwiener, Jakob; Naumann, Felix: Functional Dependency Discovery: An Experimental Evaluation of Seven Algorithms. *Proceedings of the VLDB Endowment*, 8(10):1082–1093, 2015.
- [Pa15c] Papenbrock, Thorsten; Kruse, Sebastian; Quiané-Ruiz, Jorge-Arnulfo; Naumann, Felix: Divide & Conquer-based Inclusion Dependency Discovery. *Proceedings of the VLDB Endowment*, 8(7):774–785, 2015.
- [Pa17] Papenbrock, Thorsten: Data Profiling - Efficient Discovery of Dependencies. doctoralthesis, Universität Potsdam, 2017.
- [PN16] Papenbrock, Thorsten; Naumann, Felix: A Hybrid Approach to Functional Dependency Discovery. In: Proceedings of the International Conference on Management of Data (SIGMOD). S. 821–833, 2016.
- [PN17a] Papenbrock, Thorsten; Naumann, Felix: Data-driven Schema Normalization. In: Proceedings of the International Conference on Extending Database Technology (EDBT). S. 342–353, 2017.
- [PN17b] Papenbrock, Thorsten; Naumann, Felix: A Hybrid Approach for Efficient Unique Column Combination Discovery. In: Proceedings of the Conference Datenbanksysteme in Büro, Technik und Wissenschaft (BTW). S. 195–204, 2017.

Architectural Principles for Database Systems on Storage-Class Memory¹

Ismail Oukid²

Abstract: Storage-Class Memory (SCM) is a novel class of memory technologies that combine the byte addressability and low latency of DRAM with the density and non-volatility of traditional storage media. Hence, SCM can serve as *persistent main memory*, i.e., as main memory and storage at the same time. In this thesis, we dissect the challenges and pursue the opportunities brought by SCM to database systems. To solve the identified challenges, we devise necessary building blocks for enabling SCM-based database systems, namely memory management, data structures, transaction concurrency control, recovery techniques, and a testing framework against new failure scenarios stemming from SCM. Thereafter, we leverage these building blocks to build SOFORT, a novel hybrid SCM-DRAM transactional storage engine that places data, accesses it, and updates it directly in SCM, thereby doing away with traditional write-ahead logging and achieving near-instant recovery.

Keywords: Non-Volatile Memory, Storage-Class Memory, Indexing, Transaction Processing, Memory Management, Testing, Recovery.

1 Introduction

Database systems have long been optimized to hide the higher latency of storage media, yielding complex persistence mechanisms. With the advent of large DRAM capacities, it became possible to keep a full copy of the data in DRAM. Systems that leverage this possibility, such as main-memory databases, keep two copies of the data: one in main memory and the other one in storage. The two copies are kept synchronized using snapshotting and logging. This main-memory-centric architecture yields orders of magnitude faster analytical processing than traditional, disk-centric architectures. The rise of Big Data emphasized the importance of such systems with an ever-increasing need for more main memory. However, DRAM is hitting its scalability limits: It is intrinsically hard to further increase its density.

Storage-Class Memory (SCM) is a group of novel memory technologies that promise to alleviate DRAM's scalability limits. They combine the non-volatility, density, and economic characteristics of storage media with the byte-addressability and a latency close to that of DRAM. Therefore, SCM can serve as *persistent main memory*, thereby bridging the gap between volatile main memory and persistent storage. In this thesis, we explore the impact

¹ This paper is a summary of the author's PhD thesis of the same title.

² TU Dresden & SAP SE, Database Systems Group, Nöthnitzer Str. 46, D-01062 Dresden, ismail.oukid@sap.com

of SCM as persistent main memory on the design and architecture of database systems. Assuming a hybrid SCM-DRAM hardware setting, we propose a novel software architecture for database systems that places primary data in SCM and directly operates on it, eliminating the need for traditional I/O. This architecture yields many benefits: First, it obviates the need to reload data from storage to main memory during restart, as data is discovered and accessed directly in SCM. Second, it allows replacing the traditional logging infrastructure by fine-grained, cheap micro-logging techniques at data-structure level. Third, secondary data can be stored in DRAM and efficiently reconstructed during recovery. Fourth, system runtime information can be stored in SCM to improve recovery time. Finally, the system may retain and continue in-flight transactions in case of system failures.

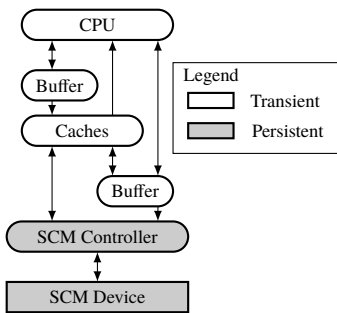


Fig. 1: Volatility chain in x86-like processors.

Unfortunately, SCM is no panacea as it raises unprecedented programming challenges. Given its byte-addressability and low latency, processors can access, read, modify, and persist data in SCM using load/store instructions at a CPU cache line granularity. The path from CPU registers to SCM is long and mostly volatile, as illustrated in Figure 1, including store buffers and CPU caches, leaving the programmer with little control over when data is persisted. Therefore, there is a need to enforce the order and durability of SCM writes using persistence primitives, such as cache line flushing instructions. This in turn creates new failure scenarios, such as missing or misplaced persistence primitives.

Within this thesis, we devise several building blocks to overcome these challenges [OL17]. First, we tackle memory management, as the first required building block to build a database system, by designing a highly scalable SCM allocator, named PAllocator [Ou17a], that fulfills the versatile needs of database systems (Section 2). Thereafter, we propose the FPTree [Ou16b], a highly scalable hybrid SCM-DRAM persistent B⁺-Tree that bridges the gap between the performance of transient and persistent B⁺-Trees (Section 3). Using these building blocks, we realize our envisioned database architecture in SOFORT [Ou14, Ou15], a hybrid SCM-DRAM columnar transactional engine. We propose an SCM-optimized MVCC scheme that eliminates write-ahead logging from the critical path of transactions (Section 4). Since SCM-resident data is near-instantly available upon recovery, the new recovery bottleneck is rebuilding DRAM-based data. To alleviate this bottleneck, we propose a novel recovery technique that achieves nearly instant responsiveness of the database by accepting queries right after recovering SCM-based data, while rebuilding DRAM-based data in the background [Ou17b] (Section 5). Additionally, SCM brings new failure scenarios that existing testing tools cannot detect. Hence, we propose an online testing framework that is able to automatically simulate power failures and detect missing or misplaced persistence primitives [Ou16a] (Section 6). In summary, our proposed building blocks can serve to devise more complex systems, paving the way for future database systems on SCM.

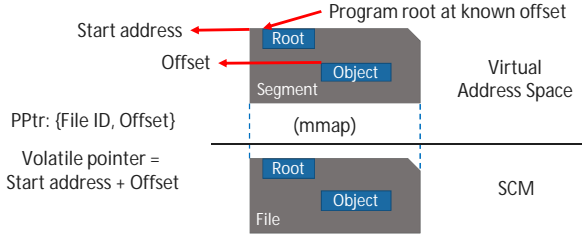


Fig. 2: Data recovery using persistent pointers consisting of a file ID and an offset within that file.

2 Persistent Memory Management

The rise of SCM might spur a major change in the architecture of database systems, as it invalidates long-standing architectural assumptions. In this incoming era of SCM-based database systems, everything is yet to be done, starting from persistent memory management as a fundamental building block. In general, SCM is handled using a file system. User space access to SCM is granted via memory mapping using *mmap()*. The mapping behaves like *mmap()* for traditional files, except that the persistent data is directly mapped to the virtual address space, instead of to a DRAM-cached copy. When a program crashes, its pointers become invalid since the program gets a new address space when it restarts. This implies that these pointers cannot be used to recover persistent data structures.

To solve this problem, we propose a new pointer type, denoted Persistent Pointer (PPtr), that remains valid across restarts. It consists of a base, which is a file ID, and an offset within that file that indicates the start of the block pointed to. Persistent pointers can easily be translated into regular pointers by adding the offset to the start address of the memory-mapped file, as illustrated in Figure 2. To perform recovery, we need to keep track of an entry point. One entry point is sufficient for the whole storage engine since every structure is encapsulated into a larger structure up to the full engine.

To manage SCM, we propose PAllocator [Ou17a], a highly scalable, fail-safe, and persistent allocator for SCM, specifically designed for databases that require very large main memory capacities. PAllocator uses internally two different allocators: SmallPAllocator, a small block persistent allocator that implements a segregated-fit strategy; and BigPAllocator, a big block persistent allocator that implements a best-fit strategy and uses hybrid SCM-DRAM trees to persist and index its metadata. The use of hybrid trees enables PAllocator to also offer a fast recovery mechanism. PAllocator uses big SCM files that are cut into smaller blocks for allocation. It maps these files to virtual memory to enable the conversion of PPtrs to regular pointers. At restart time, a new mapping to virtual memory is created, allowing to re-convert PPtrs to new valid regular pointers.

Moreover, PAllocator addresses fragmentation in persistent memory, which we argue is an important challenge, and implements an efficient defragmentation algorithm that is able to reclaim the memory of fragmented blocks by leveraging the hole punching feature of sparse

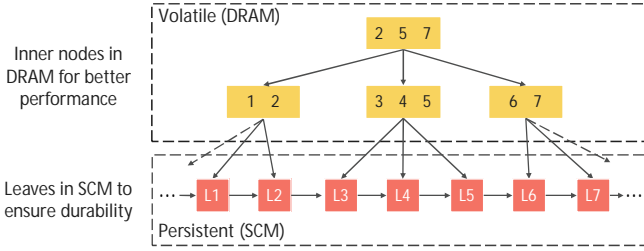


Fig. 3: Selective persistence for a B^+ -Tree: Inner nodes are in DRAM while leaf nodes are in SCM.

files. To the best of our knowledge, PAllocator is the first SCM allocator that proposes a transparent defragmentation algorithm as a core component for SCM-based database systems. Our evaluation shows that PAllocator improves on state-of-the-art persistent allocators by up to two orders of magnitude in operation throughput, and by up to three orders of magnitude in recovery time. Furthermore, we integrate PAllocator and a state-of-the-art persistent allocator in a persistent B^+ -Tree, and show that PAllocator enables up to 2.39x better operation throughput than its counterpart.

3 Persistent Data Structures

In this section we investigate the design of persistent index structures as one of the core database structures, motivated by the observation that traditional main memory B^+ -Tree implementations do not fulfill the consistency requirements needed for such a use case. Furthermore, while expected in the range of DRAM, SCM latencies are higher and asymmetric with writes noticeably slower than reads. We argue that these performance differences between SCM and DRAM imply that the design assumptions made for previous well-established main memory B^+ -Tree implementations might not hold anymore. Therefore, we see the need to design a novel, persistent B^+ -Tree that leverages the capabilities of SCM while exhibiting performance similar to a traditional transient B^+ -Tree.

To lift this shortcoming, we propose the *Fingerprinting Persistent Tree (FPtree)* [Oul6b] that is based on the following design principles to achieve near-DRAM performance:

1. **Fingerprinting.** Fingerprints are one-byte hashes of in-leaf keys, placed contiguously in the first cache-line-sized piece of the leaf. The FPtree uses unsorted leaves with in-leaf bitmaps – originally proposed in [CGN11] – such that a search iterates linearly over all valid keys in a leaf. By scanning the fingerprints first, we are able to limit the number of in-leaf probed keys to **one** in the average case, which leads to a significant performance improvement.
2. **Selective Persistence.** The idea is based on the well-known distinction between primary data, whose loss infers an irreversible loss of information, and non-primary data that can be rebuilt from the former. Selective persistence consists in storing

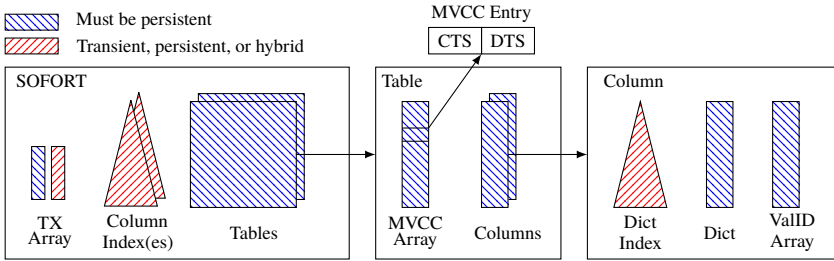


Fig. 4: Data layout overview in SOFORT.

primary data in SCM and non-primary data in DRAM. Applied to the FPTree (illustrated in Figure 3), this corresponds to storing the leaf nodes in SCM and the inner nodes in DRAM. Hence, only leaf accesses are more expensive during a tree traversal compared to a fully transient counterpart.

3. **Selective Concurrency.** This concept consists in using different concurrency schemes for the transient and persistent parts: The FPTree uses Hardware Transactional Memory (HTM) to handle the concurrency of inner nodes, and fine-grained locks to handle that of leaf nodes. Selective Concurrency elegantly solves the apparent incompatibility of HTM and persistence primitives required by SCM such as cache line flushing instructions which cause HTM transactions to abort.

We implemented the FPTree and two state-of-the-art persistent trees, namely the NV-Tree [Ya16] and the wBTree [CJ15]. Using microbenchmarks, we show that the FPTree outperforms the two competitors by up to 4.8 \times for an SCM latency of 90 ns (DRAM’s latency), and by up to 8.2 \times for an SCM latency of 650 ns. The FPTree achieves these results while keeping less than 3% of its data in DRAM. Additionally, we demonstrate how the FPTree scales on a machine with 88 logical cores. Moreover, we show that the FPTree recovery time is 76.96 \times and 29.62 \times faster than a full rebuild for SCM latencies of 90 ns and 650 ns, respectively.

4 SOFORT: A Hybrid SCM-DRAM Storage Engine

After having outlined the core building blocks, namely memory management and data structures for the design and implementation of SCM-based data-management systems, we demonstrate such a system and present SOFORT, a hybrid SCM-DRAM dictionary-encoded columnar transactional engine tailored for hybrid transactional and analytical workloads and fast data recovery [Ou14, Ou15]. SOFORT is a single-level store, i.e., the working copy of the data is the same as the durable copy of the data. To achieve this, SOFORT leverages the byte-addressability of SCM to persist data in small increments at cache-line granularity. Since the database state is always up-to-date, SOFORT does not need a redo log. SOFORT implements serializable multi-version concurrency control (MVCC) coupled with cooperative garbage collection.

SOFORT is architected as a twin-store columnar main-memory database, with a larger static immutable read-optimized store and a smaller dynamic read/write store. The dynamic store is periodically merged into the static store to do compaction. This keeps the size of the dynamic store small and therefore results in good performance for reads and writes. Andrei et al. [An17] showed that the static store can be entirely kept in SCM at a negligible cost for query execution, and instantly recovered after failure. Therefore, we focus only on the dynamic part in this thesis.

Figure 4 gives an overview of the data organization in SOFORT. Tables are stored as a collection of append-only columns. Each column consists of an unsorted dictionary stored as an array of the column’s unique data values, and an array of value IDs, where a value ID corresponds to a dictionary index (position). These two arrays are sufficient to provide data durability and constitute the *primary* data. SOFORT stores primary data, accesses it, and updates it directly in SCM. Other data structures are required to achieve reasonable performance including, for each column, a dictionary index that maps values to value IDs, and for each table, a set of multi-column inverted indexes that map sets of value IDs to the set of corresponding row IDs. We refer to these structures as *secondary* data since they can be reconstructed from the primary data. SOFORT can keep secondary data in DRAM or in SCM. Putting all indexes to DRAM gives the best performance but might stress DRAM resources, while placing indexes in SCM exposes them to its higher latency which compromises performance.

To track conflicts between transactions, MVCC keeps for every transaction, among other metadata, a write set that contains the row IDs of the tuples that the transaction inserted or deleted. This information is enough to undo a transaction in case it is aborted. We make the observation that the same information can be used to undo the effects of in-flight transactions during recovery. Therefore, to provide durability, SOFORT places the MVCC write set in SCM, which enables it to remove the traditional write-ahead log from the critical path of transactions. Furthermore, by persisting more MVCC metadata, SOFORT can allow the user to continue executing open transactions after a system failure.

SOFORT stores its columns contiguously in memory, which complicates memory reclamation of deleted tuples. Indeed, the latter would require a writer-blocking process which would replace the current columns with new, garbage-free ones. To remedy this issue, we propose to keep track of deleted rows and re-use them when inserting new tuples whenever possible instead of appending them to the table. Through an extensive experimental evaluation, we show that SOFORT exhibits competitive OLTP performance despite being a column-store.

5 SOFORT Recovery Techniques

SCM-enabled database systems such as SOFORT keep a single copy of the data that is stored, accessed, and modified directly in SCM. This eliminates the need to reload a consistent state from durable media to memory upon recovery, as primary data is accessed

directly in SCM. Hence, the new recovery bottleneck is rebuilding DRAM-based data structures. We address this bottleneck following two orthogonal dimensions: The first one pertains to secondary data placement in SCM, in DRAM, or in a hybrid SCM-DRAM format. We show that near-instant recovery is achievable if all secondary data is persisted in SCM. However, this comes at the cost of a decreased query performance by up to 51.1%. Nevertheless, near-instant recovery offers guarantees that are appealing to business applications where availability is critical. Hybrid SCM-DRAM data structures offer a good compromise by reducing recovery time by up to 5.9 \times while limiting the impact on query performance between 16.6% and 32.7%. The second dimension is optimizing the recovery of DRAM-based data independent of data placement.

After recovering its SCM-based data structures, SOFORT rebuilds DRAM-based secondary data structures, then starts accepting requests. If DRAM-based secondary data structures are large, restart times can still be unacceptably long. To address this, we propose an *Instant Recovery* strategy. It allows queries to be answered concurrently with the rebuilding of DRAM-based data structures, i.e., while recovery is still in progress. However, while the secondary data structures are being rebuilt, request throughput is reduced, partially because of the overhead of rebuilding, but more importantly because of the unavailability of the DRAM-based secondary data structures, resulting in sub-optimal access plans.

To remedy the shortcomings of instant recovery, we propose a novel recovery strategy, named *Adaptive Recovery* [Ou17b]. It is inspired by the observation that not all secondary data structures are equally important to a given workload. It improves on instant recovery in two ways. First, it prioritizes the rebuilding of DRAM-based secondary data structures based on their benefit to a workload (instant recovery simply uses an arbitrary order). Second, it releases most of the CPU resources dedicated to recovery once all of the important secondary data structures have been rebuilt (instant recovery statically splits CPU resources between recovery and query processing for the entire recovery period).

Through our experimental evaluation, we showed that SOFORT regains its peak performance up to 4.3 \times faster with adaptive recovery than with synchronous recovery, while allowing the system to be responsive near-instantly. We demonstrated that our benefit ranking adapts well to workload changes during recovery and allows to regain pre-failure throughput up to 2.1 \times faster than rankings that do not take into consideration the recovery workload.

6 Testing of SCM-Based Software

Consistency failure scenarios and recovery strategies of software that persists data depend on the underlying storage technology. In the traditional case of block-based devices, software has full control over when data is made persistent. Basically, software schedules I/O to persist modified data at a page granularity. The application has no direct access to the primary copy of the data and can only access copies of the data that are buffered in main memory. Hence, software errors can corrupt data only in main memory which can be reverted

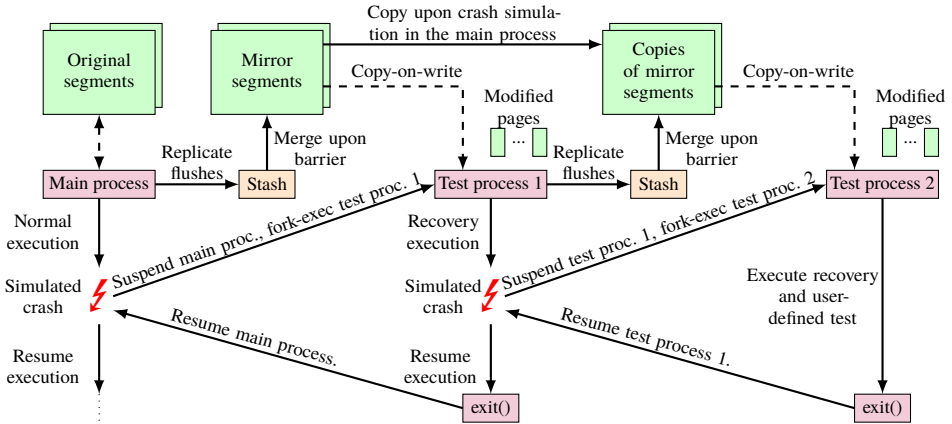


Fig. 5: Illustration of crash simulation in the testing framework.

as long as the corruption was not explicitly propagated to storage. In fact, crash-safety for block-based software highly depends on the correctness of the underlying file system. In contrast, SCM is byte-addressable and is accessed via volatile store buffers and CPU caches, over which software has little control. As a side effect, changes can be speculatively propagated from the CPU cache to SCM at any time, and compilers and out-of-order CPU execution can jeopardize consistency by reordering memory operations. Moreover, changes are made persistent at a cache line granularity which necessitates the use of CPU persistence primitives. This adds another level of complexity as enforcing the order in which changes are made persistent cannot be delayed like with block-based storage devices, and must be synchronous. Hence, storing the primary copy of the data in SCM and directly updating it in-place significantly aggravates the risk of data corruption.

Several proposals tackled these challenges following two main approaches. The first one focuses on providing global software-based solutions, mainly transactional-memory-like libraries, to make it easier for developers to write SCM-based software. The second and more mainstream approach is to rely solely on existing hardware persistence primitives, such as cache line flushing instructions and memory barriers to achieve consistency. Nevertheless, all approaches have in common that SCM-related errors may result in data corruption. In contrast to volatile RAM where data corruption can be cured with a restart of the program, data corruption in SCM might be irreversible as it is persistent. Therefore, we argue for the need of testing the correctness of SCM-based software against software crashes and power failures—which result in the loss of the content of the CPU cache.

We tackle this challenge by proposing a lightweight automated on-line testing framework, illustrated in Figure 5, that helps detecting and debugging a wide range of SCM-related bugs that can arise upon software or power failures [Ou16a]. We particularly focus on detecting missing cache line flushing instructions. Our testing framework employs a suspend-test-

resume approach and simulates power failures using data replication, similar to shadow memory testing approaches [NS07]. The testing framework creates a mirror segment for each segment that the program creates; the mirror segment contains only data that is explicitly flushed by the program. The testing framework triggers randomly simulated crashes in the path of persistence primitives, upon which a test process is forked and the main process is suspended. Then, the test process executes a user-defined program that recovers using the mirror segment with copy-on-write access semantics. Upon completion of the user-defined program, the test process is terminated and the changes to the mirror segments are discarded.

An important feature of our testing framework is its ability to avoid excessive duplicate testing by tracking the call stack information of already tested code paths, which leads to fast code coverage. Additionally, our testing framework is able to partially detect errors that might arise due to the compiler or the CPU speculatively reordering memory operations. It can further simulate crashes in the recovery procedure of the tested program, which we argue is important since hidden SCM-related errors in the recovery procedure may compromise the integrity of the data upon every restart. We show with an experimental evaluation on the FPTree and PAllocator that our testing framework achieves fast testing convergence, even in the case of nested crash simulations.

7 Conclusion

SCM is emerging as a disruptive hybrid memory and storage technology, requiring us to fundamentally rethink current database system architectures. In this thesis, we endeavored to explore this potential by building a hybrid transactional and analytical database system from the ground up that leverages SCM as persistent main memory. Our pathfinding work led us to identify the challenges and opportunities brought by SCM for database systems. As a result, we devised a set of building blocks, including an SCM allocator, a hybrid SCM-DRAM persistent and concurrent B⁺-Tree, an adaptation of MVCC for SCM, novel database recovery techniques, and a testing tool for SCM-based software. Armed with these building blocks, we designed and implemented SOFORT, a hybrid SCM-DRAM transactional engine that keeps primary data in SCM and directly operates on it. We showed how SOFORT's architecture enables near-instant recovery, allows to continue unfinished transactions after failure, removes traditional write-ahead logging from the critical path of transactions, and therefore achieves low transaction latencies. These building blocks can be used to build more complex systems, exemplified by SOFORT, paving the way for future database systems on SCM.

8 Acknowledgments

I would like to warmly thank Prof. Wolfgang Lehner for his mentorship and guidance which were essential to the success of this thesis. Special thanks also go to all current and past

members of the SAP HANA Campus, the SAP HANA development team, and the Intel onsite team at SAP for their help and support throughout my PhD thesis.

References

- [An17] Andrei, Mihnea; Lemke, Christian; Radestock, Günter; Schulze, Robert; Thiel, Carsten; Blanco, Rolando; Meghlan, Akanksha; Sharique, Muhammad; Seifert, Sebastian; Vishnoi, Surendra; Booss, Daniel; Peh, Thomas; Schreter, Ivan; Thesing, Werner; Wagle, Mehul; Willhalm, Thomas: SAP HANA Adoption of Non-volatile Memory. *Proc. VLDB Endow.*, 10(12):1754–1765, August 2017.
- [CGN11] Chen, Shimin; Gibbons, Phillip B; Nath, Suman: Rethinking Database Algorithms for Phase Change Memory. In: *CIDR*. 2011.
- [CJ15] Chen, Shimin; Jin, Qin: Persistent B+-trees in Non-volatile Main Memory. *Proc. VLDB Endow.*, 8(7):786–797, February 2015.
- [NS07] Nethercote, Nicholas; Seward, Julian: Valgrind: A Framework for Heavyweight Dynamic Binary Instrumentation. *SIGPLAN Not.*, 42(6):89–100, June 2007.
- [OL17] Oukid, Ismail; Lehner, Wolfgang: Data Structure Engineering For Byte-Addressable Non-Volatile Memory. In: *Proceedings of the 2017 ACM International Conference on Management of Data*. *SIGMOD '17*, ACM, New York, NY, USA, pp. 1759–1764, 2017.
- [Ou14] Oukid, Ismail; Booss, Daniel; Lehner, Wolfgang; Bumbulis, Peter; Willhalm, Thomas: SOFORT: A Hybrid SCM-DRAM Storage Engine for Fast Data Recovery. In: *Proceedings of the Tenth International Workshop on Data Management on New Hardware*. *DaMoN '14*, ACM, New York, NY, USA, pp. 8:1–8:7, 2014.
- [Ou15] Oukid, Ismail; Lehner, Wolfgang; Kissinger, Thomas; Willhalm, Thomas; Bumbulis, Peter: Instant Recovery for Main-Memory Databases. In: *CIDR*. 2015.
- [Ou16a] Oukid, Ismail; Booss, Daniel; Lespinasse, Adrien; Lehner, Wolfgang: On Testing Persistent-memory-based Software. In: *Proceedings of the 12th International Workshop on Data Management on New Hardware*. *DaMoN '16*, ACM, New York, NY, USA, pp. 5:1–5:7, 2016.
- [Ou16b] Oukid, Ismail; Lasperas, Johan; Nica, Anisoara; Willhalm, Thomas; Lehner, Wolfgang: FPTree: A Hybrid SCM-DRAM Persistent and Concurrent B-Tree for Storage Class Memory. In: *Proceedings of the 2016 International Conference on Management of Data*. *SIGMOD '16*, ACM, New York, NY, USA, pp. 371–386, 2016.
- [Ou17a] Oukid, Ismail; Booss, Daniel; Lespinasse, Adrien; Lehner, Wolfgang; Willhalm, Thomas; Gomes, Grégoire: Memory Management Techniques for Large-scale Persistent-main-memory Systems. *Proc. VLDB Endow.*, 10(11):1166–1177, August 2017.
- [Ou17b] Oukid, Ismail; Nica, Anisoara; Dos Santos Bossle, Daniel; Lehner, Wolfgang; Bumbulis, Peter; Willhalm, Thomas: Adaptive Recovery for SCM-Enabled Databases. In: *ADMS@VLDB*. 2017.
- [Ya16] Yang, J.; Wei, Q.; Wang, C.; Chen, C.; Yong, K. L.; He, B.: NV-Tree: A Consistent and Workload-Adaptive Tree Structure for Non-Volatile Memory. *IEEE Transactions on Computers*, 65(7):2169–2183, July 2016.

Modern techniques for transaction-oriented database recovery¹

Caetano Sauer²

Abstract:

Transaction-oriented database recovery has been a “solved problem” for at least 25 years since the introduction of the ARIES methods for logging and recovery. However, recent technological developments have urged the need for new software architectures that can better exploit the efficiency of modern hardware. In the context of recovery, new algorithms are required to effectively accommodate the exponential decrease in main-memory cost, the advent of flash memory, the rapid expansion into many-core CPUs, the ever-increasing capacity of magnetic disks, and, on the long term, the potential adoption of non-volatile memory. In our research, we evaluated a variety of new software techniques for efficient transaction-oriented database recovery, focusing on availability and architectural simplicity. The techniques presented here differ from most recent work in the field in which they aim to be *hardware-agnostic*, supporting different memory and storage configurations with the same software, as well as *fully functional* in comparison with traditional database systems, e.g., by supporting media recovery, index management, larger-than-memory datasets, and arbitrary access structures with structural modifications.

1 Motivation

Most existing approaches for database recovery based on write-ahead logging (WAL), including the widely used ARIES [Mo92] methods implemented in the vast majority of commercial database systems, suffer from two major problems. The first problem is that recovery is usually performed offline, meaning that the database and its applications only become available to new transactions after the full recovery process is completed. This process can take multiple hours to complete, depending on factors such as hardware characteristics, workload access patterns, and transaction volume. The second problem is that individual recovery actions, such as the replay of updates on a single data page or the rollback of a transaction, are not scheduled in a way that prioritizes the needs of applications after a failure. Aiming to solve these two problems, a new technique called *instant recovery* enables access to individual data pages (or contiguous sets thereof) before the complete recovery process is finished. These techniques cover all classes of database failures, most notably system and media failures. By performing recovery actions on demand, system

¹ This short article summarizes a doctoral dissertation with the same title [Sa17]

² Tableau Software, csauer@tableau.com (Dissertation completed at TU Kaiserslautern)

availability as observed by individual transactions can be effectively increased by up to two “nines” by means of simple and incremental software techniques.

Besides increasing availability by optimizing the recovery process, efficient *checkpointing* as well as *backup* techniques are required to maintain the persistent database in a fresh state and thus effectively reduce the amount of recovery work to be performed in case of a failure. Building upon a discussion of efficient checkpoint algorithms, as well as on ideas from instant recovery, this work presents a novel family of techniques called *decoupled persistence*. The main goal of this effort is to simplify checkpoint and recovery techniques by decoupling them from critical components of the database system such as the buffer pool and the transaction manager. The key technique employed is to rely solely on log information to perform checkpoints and propagate changes to the persistent database. This approach not only enhances the reusability of the database system’s internal components, but also potentially improves performance by eliminating the interference of checkpoint actions on critical system components.

Looking beyond traditional WAL architectures, these ideas are taken further with a novel design for database storage and recovery called *FineLine*. Its goal is to simplify the recovery and checkpointing processes by eliminating the persistent database, relying solely on the recovery log for data storage and retrieval. With *FineLine*, there is no duality between persisted database and log and, as a consequence, no algorithmic logic that is exclusive to recovery from failures; instead, recovery is embedded in the data access protocol, without distinction between normal and recovery processing modes. This results in a much simpler system architecture, which also decouples in-memory data structures from persistence concerns and reduces log volume, thus improving performance for memory-resident workloads. From a more general perspective, the goal of *FineLine* is to provide efficient, transparent, reliable, and highly-available persistence as a decoupled component, accommodating arbitrary implementations of in-memory access methods and concurrency control. As such, it blurs the lines between in-memory and disk-based database systems.

If a system is able to keep the amount of recovery work manageable and perform this recovery work not only while transactions are running but also prioritizing the needs of those transactions, then the challenge postulated by Jim Gray in his ACM 1998 Turing Award Lecture of a system “unavailable for less than one second per hundred years” [Gr03] gets one step closer to becoming a reality.

2 Key contributions

This section summarizes the contributions of the author’s dissertation in five parts. Further details are provided in the cited references as well as in the published dissertation [Sa17].

2.1 Instant restart

Instant restart is a technique for recovery from system failures that builds upon single-page repair [GK12] to provide incremental, page-oriented redo in addition to the traditional log-oriented redo phase of ARIES recovery. Furthermore, by collecting acquired locks during checkpoints and log analysis, the technique also enables incremental and on-demand undo actions by detecting lock conflicts between pre-failure (i.e., “loser”) and post-failure transactions. The resulting recovery algorithm allows the execution of new transactions immediately after the log analysis phase, and the access pattern of post-failure transactions actually guides the redo and undo actions required for recovery. This requires exploiting the independence of recovery among objects that is inherent to physiological logging [Mo92] and reorganizing the recovery process accordingly. Building upon this independence, fine-granular recovery actions can then be scheduled following the demands of new transactions started after a failure, which essentially reduces the observed mean time to repair by multiple orders of magnitude.

The instant restart algorithm uses the same building blocks and actually executes the exact same actions as the ARIES algorithm during recovery. In the redo phase, the same log records are replayed on the same set of pages as in ARIES restart; in the undo phase, the same transactions are rolled back, producing the same logical compensation actions. The key difference is that these actions can be performed concurrently with post-failure transactions, and their access pattern is actually used to guide recovery in an on-demand schedule.

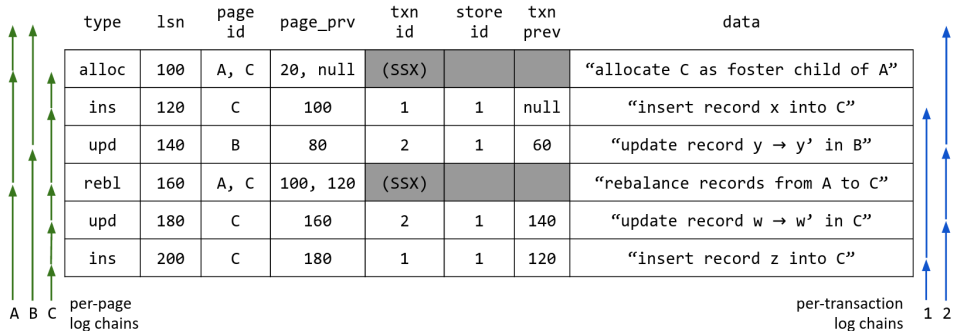


Fig. 1: Example of log records with per-page and per-transaction chains

Log-record chains are a fundamental technique to enable on-demand restart recovery. They are illustrated in Fig. 1, which shows an example of log snippet in which two transactions, 1 and 2, make modifications to three pages, A, B, and C. In this log, two independent system transactions, identified with SSX, are performing a split on page A of a B-tree data structure in two steps: first, a new page C is allocated, and then records are moved from A to C in a *rebalance* operation. On the right side of the diagram, two in-flight user transactions are shown with their per-transaction log chains; these are used for undo recovery and are also present in the original ARIES design [Mo92]. Note that user transaction 1 is allowed to

insert a record in page C while the page split is still happening—this illustrates the utility of system transactions as well as the fundamental distinction between database contents and their representation [Gr12].

The left side of the diagram shows per-page log chains for pages A, B, and C. Each log record points to the last log record that modified that same page. This chain can be easily maintained by saving the page LSN value into the log record before applying its corresponding update and setting the new page LSN to its own LSN value [GG14].

To understand instant restart by example, consider that a system failure occurred and the log of Figure 1 is all the log that is found after restart and that none of its updates has been propagated to the database. In that case, the log analysis phase determines that transactions 1 and 2 need to be undone and pages A, B, and C need to be redone. The exclusive locks held by transactions 1 and 2 (i.e., the loser transactions) before the failure are also reacquired during log analysis. This information can be kept in the lock manager and in the buffer manager, respectively, as special entries that also mark the head of each log chain. Once log analysis is finished, and, most importantly, *before any undo or redo actions take place*, the system is made available for new transactions.

Post-failure transactions have unrestricted access to the database: if they fix a page in the buffer pool that needs redo recovery, the old image is fetched and redo recovery is performed, on that page only, by following the per-page log chain until the page LSN value and reapplying the log records in reverse order (i.e., using a stack). Following the *repeating history* paradigm, updates of loser transactions are also redone [Mo92].

Once a page is fixed and all its updates redone, it may still contain updates of loser transactions, which have to be undone. This is achieved by aborting a loser transaction on demand as soon as a post-failure transaction requests a lock on a record or key value that is currently held by it (i.e., a lock reacquired during log analysis).

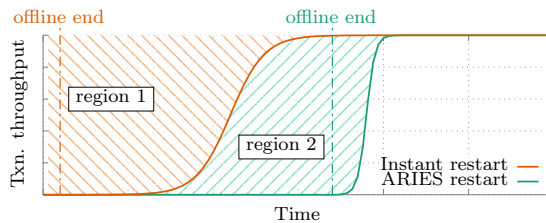


Fig. 2: Logistic function patterns of instant restart and ARIES restart

The performance of instant restart depends on factors such as storage hardware characteristics, transaction volume, and workload behavior. These were evaluated in detail in the dissertation [Sa17], but the general behavior expected during restart is that of a *logistic function* depicted in Fig. 2, where time is shown in the x-axis and transaction throughput in the y-axis. The chart shows the behavior of instant restart in comparison with ARIES restart as two logistic curves and their respective *offline phases*, i.e., the points for which transaction

throughput is zero. With instant restart, the offline phase finishes much earlier than ARIES, and recovery is performed as a side-effect of the workload access pattern; thus, throughput is increased more gradually as the buffer pool is warmed up. The net result is that the number of transactions missed due to a failure—the shaded area labeled “region 1”—is significantly lower. Once again, we refer to the dissertation for concrete numbers under different experiment configurations [Sa17].

2.2 Single-pass restore

The benefit of on-demand recovery can also be achieved for media failures, but since the amount of log that must be accessed during media recovery is likely to be many orders of magnitude larger than during restart recovery, we first address the issue of media recovery efficiency with a technique called *single-pass restore* [SGH15].

Single-pass restore maintains the log archive in a partially sorted organization, by introducing a run generation phase to the log archiving process. Within each run, log records are sorted primarily by page identifier rather than by LSN, so that a merge of log archive partitions and an old database backup is able to produce an up-to-date database using a single sequential pass, rather than a sequential pass of the log with random accesses on a database backup and its replacement device. The process is illustrated in Fig. 3, and more details are available in the original publication [SGH15].

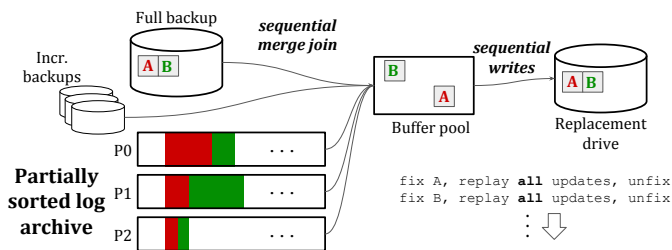


Fig. 3: Single-pass restore

In our measurements, run generation during log archiving adds less than 3% overhead to regular forward processing [SGH15], and the gained benefit is that full recovery of an up-to-date database takes the same amount of time as copying an old backup image in traditional methods. Single-pass restore also makes the management of backups much simpler, renders incremental backups obsolete, and enables system administrators to substantially reduce the frequency of full backups without compromising availability [GG14].

2.3 Instant restore

Instant restore [SGH17] builds upon the partially sorted log archive of single-pass restore by adding an index to each sorted partition. This enables the retrieval of log records of a given database page or—more appropriately to exploit sequential access speeds—a contiguous set of database pages, which we call a *segment*. Such an *indexed log archive* enables on-demand restoration of database segments in the same fashion that individual pages are redone in instant restart; the only difference is that log records are retrieved from an index rather than by following a chain of log records.

Despite sharing the same general principle of on-demand recovery, instant restart after a system failure and instant restore after a media failure are quite different in their causes, effects, and recovery measures. When a media failure is detected, the system process is not affected, and thus its main-memory state—including buffer pool, transaction and lock managers, etc.—is not lost. This is unlike a system failure, or *crash*, in which all volatile state is lost. This means that user transactions may continue execution after a media failure, provided that they only access data that is either in the buffer pool or on other (still healthy) persistent devices. A transaction that accesses data on a failed device is normally aborted, but with instant restore, it can actually trigger the restoration of the segments that contain the accessed pages on demand. Thus, a moderate delay is added to the transaction’s response time (in addition to the page miss that would have occurred even without the media failure) rather than aborting it.

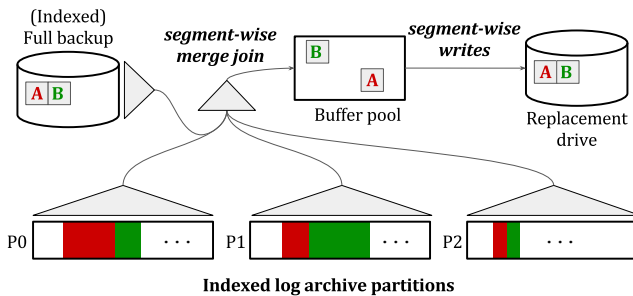


Fig. 4: Illustration of instant restore

Restoration of a failed device from a full backup and an indexed log archive is illustrated in Fig. 4, which shows an example segment consisting of just two pages, A and B. Thanks to the grouping of pages into segments and the partially-sorted order of log archive partitions, the access pattern is mostly sequential, and thus recovery is as efficient as single-pass restore. Thanks to the index added on top of the sorted log partitions, segments which are needed most immediately by applications are prioritized, while the remaining “cold” segments can be restored in the background.

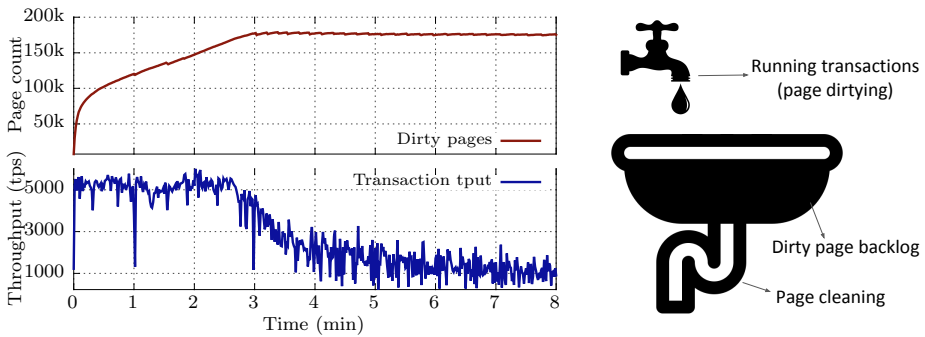


Fig. 5: Dirty page backlog and its implication on system performance

2.4 Decoupled persistence

Our work on propagation strategies [Sa16] emphasizes the problem of efficiently propagating updates from the buffer pool (i.e., cached dirty data in volatile storage) to persistent storage. It highlights the need for a well-balanced architecture in which the aggregate bandwidth of propagation matches the update rate of the transaction workload—in other words, in which the *cleaning speed* matches the *dirtying speed*. This problem is especially relevant for write-heavy workloads (such as TPC-C) running on memory-abundant systems. In such cases, inefficient propagation may lead to a disk bottleneck on transaction throughput, even though an application’s working set fits entirely in main memory, as illustrated in Fig. 5 (more details of this experiment in the original publication [Sa16]).

These observations lead to the insight that update propagation for memory-intensive and write-heavy applications must happen under control of a dedicated background service called the *page cleaner* [Sa17] rather than on a page-by-page basis and triggered exclusively by a page replacement strategy, which is the traditional approach. Such a cleaner service must decide, from a set of dirty candidate pages, which pages to write back to disk at a particular point in time. This choice depends on a model that takes into account the benefit of writing a particular page and the cost of doing so for a given storage hardware configuration. In addition to these observations, our work proposes a log-based model in which updates are propagated not by writing cached pages directly, but by replaying log records from the partitioned index presented earlier for instant restore [Sa16]. This idea, when taken to the extreme, leads to the system design presented in the next section.

2.5 FineLine

FineLine [SGH18] is a novel system design that stores all data in the log, thus departing from a traditional write-ahead logging architecture, in which persistent data lives both in the log and in a database file. Unlike a traditional, time-ordered log, the FineLine log is

actually a partitioned index on database page identifiers, as proposed for instant restore. Propagation of updates from volatile caches to persistent storage happens solely via logging, which contains only redo information thanks to its *no-steal* approach [HR83]. In order to fetch a page from persistent storage, its state is simply reconstructed from its pertaining log records, as illustrated in Fig. 6.

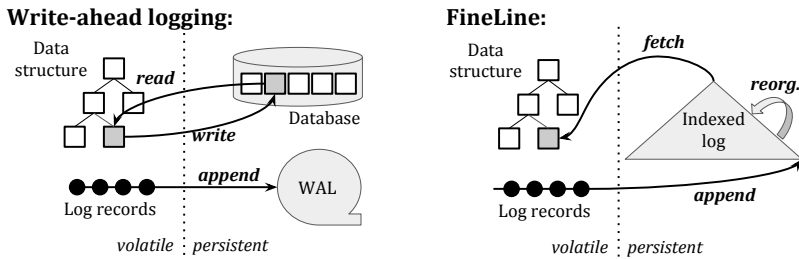


Fig. 6: Propagation of updates in WAL (left) vs. FineLine (right)

The partitions in the log are periodically merged in order to deliver acceptable read performance, similar to log-structured merge trees [ON96]. The key difference is that FineLine relies on physiological logging, thus indexing log records by their page identifier rather than by key values in the application domain. As such, it can be seen as a persistence module that supports transactional durability to arbitrary in-memory data structures. Unlike log-structured merge trees, the log index itself provides durability and there is no separate write-ahead log. By relying on physiological logging, FineLine also inherits all the benefits of ARIES and instant recovery, including on-demand system and media recovery, support for system transactions for space management and structural operations, secondary indexes, buffer management, and orthogonality to isolation mechanisms (i.e., two-phase locking, optimistic validation, multi-versioning, etc. are all supported). Lastly, compared to a Shore-MT-based prototype that implements ARIES-based logging and recovery, FineLine improves performance of a memory-resident workload by about 2 \times , requires less code, and is more modular [SGH18].

3 Lessons learned and outlook

In this dissertation, we attempted to improve the availability and efficiency of logging and recovery algorithms for transaction-oriented database systems. The following paragraphs summarize some guiding principles that we followed in this work and are likely also relevant for future work.

Database systems are effective and versatile tools that build the backbone of some of the most critical and useful applications in the world. Given this key position, database systems are also extremely performance-critical, and thus a large share of research efforts in the field are invested towards making databases as fast as possible. However, in many cases, optimizing for performance conflicts with the goals of versatility and data independence.

As in many endeavors in science and engineering, the challenge therefore lies in finding the correct compromise.

Over the last decade, the advent of cheap main memory and multi-core CPUs pushed database researchers to reconsider their architectures. Perhaps most influential in this effort was the work of Stonebraker et al. on H-Store [St07]. This system achieves orders-of-magnitude improvement in transactional performance by completely eliminating components such as logging and recovery, locking, and buffer management. While this work served as an important “wake-up call” that deeply influenced many of the systems designed in the last decade, the rather extreme approach of trimming off as much functionality as possible for the goal of performance is likely too restrictive for database systems in general. Since H-Store was proposed, research papers that followed gradually reintroduced functionality such as concurrency control, recovery, and buffer management. In a way, the “complete rewrite” was extremely useful as a reconsideration of software architectures in the light of modern hardware, but the end result (if there is one) might look much more similar to a traditional architecture than expected.

In the context of transaction recovery, our work advocates for the retainment of certain fundamental design principles, such as physiological logging and buffer management. These are crucial to provide functionality such as media recovery (by means other than replication and failover), native indexing support, access-path (i.e., data-structure) independence, system transactions, and independence of concurrency control mechanisms (e.g., fine-grained two-phase locking). Our benchmark has been to support all the functionality that ARIES [Mo92] supports—if our work can be as applicable as the most ubiquitous recovery mechanism in the history of database systems, then we have indeed succeeded across at least one dimension.

The “main-memory revolution“ of the last decade has been, in a perhaps quite ironic way, accompanied by a revolution in storage hardware architectures. In fact, this “storage revolution” is still going on and will likely continue until non-volatile memory (NVM) is established in the market and its development somehow stabilizes. Flash memory and fast solid-state drives already changed many assumptions of the storage hierarchy, but upcoming NVM storage has the potential to beget another “complete rewrite”. While preliminary proposals for NVM-based transactional systems are promising, many of them wage on NVM “taking over” the storage hierarchy, while, in reality, the storage landscape is likely to remain heterogeneous.

Given the uncertainty about how a future storage hierarchy might look like, the unlikelihood of it being NVM-only, and the lessons learned from the main-memory revolution about versatility and applicability, our work focused on hardware-agnostic software techniques, which have the potential to be optimized for NVM rather than being designed exclusively for it. We believe this is an important principle to follow if our research community should invent a “new ARIES” that will become standard in database systems to come.

Acknowledgments: My “doctoral father” Theo Härder took me under his wing as an undergraduate research assistant from Brazil, supported my work all the way through a PhD, and continues to do so as I moved into industry. I will be forever grateful for that. Goetz Graefe was not only an invaluable source of knowledge with his great contributions to the field but also co-advisor of my thesis and a mentor who was always available and inspired me with his approach to building complex systems using simple software techniques. Thomas Neumann kindly agreed to participate in my dissertation committee and is also a source of inspiration for his great work, thanks to which I now have an exciting and challenging job.

References

- [GGS14] Graefe, G.; Guy, W.; Sauer, C.: Instant Recovery with Write-Ahead Logging: Page Repair, System Restart, and Media Restore. Morgan & Claypool, 2014.
- [GK12] Graefe, G.; Kuno, H. A.: Definition, Detection, and Recovery of Single-Page Failures, a Fourth Class of Database Failures. *PVLDB* 5/7, pp. 646–655, 2012.
- [Gr03] Gray, J.: What next?: A dozen information-technology research goals. *J. ACM* 50/1, pp. 41–57, 2003.
- [Gr12] Graefe, G.: A survey of B-tree logging and recovery techniques. *ACM Trans. Database Syst.* 37/1, p. 1, 2012.
- [HR83] Härder, T.; Reuter, A.: Principles of transaction-oriented database recovery. *ACM Comput. Surv.* 15/4, pp. 287–317, 1983.
- [Mo92] Mohan, C.; Haderle, D.; Lindsay, B.; Pirahesh, H.; Schwarz, P.: ARIES: a transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. *ACM TODS* 17/1, pp. 94–162, 1992.
- [ON96] O’Neil, P. E.; Cheng, E.; Gawlick, D.; O’Neil, E. J.: The Log-Structured Merge-Tree (LSM-Tree). *Acta Inf.* 33/4, pp. 351–385, 1996.
- [Sa16] Sauer, C.; Lersch, L.; Härder, T.; Graefe, G.: Update Propagation Strategies for High-Performance OLTP. In: *Proc. ADBIS, LNCS 9809*, pp. 152–165. 2016.
- [Sa17] Sauer, C.: Modern techniques for transaction-oriented database recovery, PhD thesis, Dr. Hut Verlag, pp. 1–141: TU Kaiserslautern, Germany, 2017.
- [SGH15] Sauer, C.; Graefe, G.; Härder, T.: Single-pass restore after a media failure. In: *Proc. BTW, LNI 241*, pp. 217–236. 2015.
- [SGH17] Sauer, C.; Graefe, G.; Härder, T.: Instant Restore After a Media Failure. In: *Proc. ADBIS, LNCS 10509*, pp. 311–325. 2017.
- [SGH18] Sauer, C.; Graefe, G.; Härder, T.: FineLine: log-structured transactional storage and recovery. *PVLDB* 11/13, pp. 2249–2262, 2018.
- [St07] Stonebraker, M.; Madden, S.; Abadi, D. J.; Harizopoulos, S.; Hachem, N.; Helland, P.: The End of an Architectural Era (It’s Time for a Complete Rewrite). In: *Proc. VLDB*, pp. 1150–1160. 2007.

Demonstrationen

The Borda Social Choice Movie Recommender

Johannes Kastner,¹ Nemanja Ranitovic,² Markus Endres¹

Abstract: In this demo paper we present a recommender system, which exploits the *Borda social choice voting rule* for clustering recommendations in order to produce comprehensible results for a user. Considering existing clustering techniques like k-means, the overhead of normalizing and preparing the preferred user data is dropped. In our demo showcase we facilitate a comparison of our clustering approach to the well known k-means++ with traditional distance measures.

Keywords: Clustering, k-means, Borda, Social choice

1 Introduction

Recommendations are becoming more and more common, because the quantity of data, e.g., in online shopping platforms like Amazon, movie on-demand streaming services like Netflix and Amazon Prime Video, or music-streaming platforms, e.g., Spotify is growing continuously. In order to handle these large and confusing sets of objects easily, clustering is a very promising approach to encapsulate similar objects and to present only a few representatives of the sets to the user.

For example, consider the case where Bob wants to watch a movie. He favors *old-school movies of the late 70s to the early 90s*. He only wants to watch movies, which have a runtime *between 90 and 130 minutes*. Furthermore Bob prefers ambitious movies with a *user rating higher than 7* on a score from 0 to 10. The result of such a *preference query* (cp. [KEW11]) on a movie data set, e.g., the Internet Movie Database (IMDb) could produce large and confusing results (cp. Table 1). Now, Bob has to select one movie out of this quite confusing set of movies. This is in most cases a difficult decision, especially if the user preferences get more and more complex regarding constraints in several dimensions.

ID	movie	rating	running time	release year	genres
1	Star Wars	8.8	125	1977	Action, Sci-Fi
7	Reservoir Dogs	8.4	99	1992	Crime, Drama, Thriller
23	Indiana Jones II	7.6	118	1984	Action, Adventure, Fantasy
27	Die Hard 2	7.1	124	1990	Action, Thriller, Crime
...

Tab. 1: Sample result of Bob's 4-dim preference query.

¹ Institute for Computer Science, University of Augsburg, 86135 Augsburg, Germany
<firstname>.<lastname>@informatik.uni-augsburg.de

² nemanja.ranitovic89@gmail.com

Clustering approaches like k-means ensure that these large and confusing sets are encapsulated and presented in a clear manner to the user. However, if we consider the individual domains of each dimension, traditional distance measures, like the Euclidean distance, stretch to their limits. Since these dimensions have quite diverse domains, using traditional distance measures in k-means meet problems with this use case, because the domains are not set into an equal relation to each other. To apply k-means, one has to adjust the domains before the clustering process by normalization. However, this might be a very challenging task due to various and versatile domains.

In our work, we adapt the *Borda social choice voting rule for cluster allocation* in recommender systems. Each object is considered equally in each dimension and receives a “voting”, which yields to a competitive result compared to a cluster allocation using traditional distance measures. In order to present our novel decision criterion we created an online movie recommender system to facilitate a visual comparison of using different distance measures for k-means clustering. Furthermore we include quality measures like Silhouette and Davies-Bouldin for choosing the possibly best number of desired clusters [DB79, Ro87].

2 Background

In general, social choice deals with the aggregation of individual preferences for managing social assessments and ruling. *Borda* is a voting rule, which is omnipresent in political or other elections, e.g., the Eurovision Song Contest. As mentioned in [De92], Borda is a very appealing approach to consider each dimension in a multi-dimensional scenario in an equal manner. We adapt Borda for the allocation of objects in k-means to one and only one cluster and therefore more influence of smaller domains are allowed, because every candidate receives equal weighed votes from each voter.

Given k candidates C_i , and d voters V_j , where each voter votes for each candidate. Each voter V_j has to allocate the voting $v_{jm} \in \{0, \dots, k-1\}$, $m = 1, \dots, k$, where all v_{jm} are pairwise distinct. Afterwards, the votes for each candidate are summed up as $bordaSum_{C_i} = \sum_{l=1}^d v_{li}$, while the Borda winner is determined as $bordaWinner = \max\{bordaSum_{C_i} \mid i = 1, \dots, k\}$.

If we apply this approach to our clustering framework, the *candidates* correspond to the available *clusters* and the *voters* to the *dimensions of the d-dimensional object which should be allocated to a cluster*. Then, for each dimension votes are assigned for the distances between the object and the centroids of the clusters. While the closest distance receives a maximum vote of $k-1$, the second closest a vote of $k-2$, the largest distance obtains a vote of 0. After the voting the sum of all votes for each cluster and subsequently the winner is determined. We integrated this novel approach into the basic k-means++ clustering algorithm as it is defined in [AV07]. Using Borda, dimensions, which would not be equally considered because of a smaller domain, get equal weighted votes like the other dimensions and have a higher influence on the clustering process. Finally we avoid the overhead of normalization.

Table 2 shows an example based on the dataset in Table 1 for our Borda cluster allocation. We want 3 clusters and present the allocation for movie (27). The centroids of the initial clusters C_1, C_2, C_3 are the movies with the IDs (1), (7), (23). For each dimension the distances between movie (27) and the centroids are calculated. The Borda votes are depicted in parentheses, e.g., the dimension *rating* is closest to C_3 and therefore gets a vote of $k - 1 = 2$. The second closest centroid is C_2 with vote 1, and C_1 gets the vote 0. Finally, C_2 with *movie* (7) as initial centroid is determined as the *bordaWinner* with a *bordaSum* of 5. Compared to the Euclidean distance we obtain a more concise result for the cluster allocation, due to ranking the values in each dimensions according to their closeness. Note that we used the Jaccard coefficient³ for calculating the similarity of *genres* between the movie and the centroid and that a Jaccard coefficient of 1.0 is the best value.

Dimension	Movie (27): Die Hard		
	C_1	C_2	C_3
rating	1.70 (0)	1.30 (1)	0.50 (2)
running time	1.00 (2)	25.00 (0)	6.00 (1)
release year	13.00 (0)	2.00 (2)	6.00 (1)
genre	0.25 (1)	0.50 (2)	0.20 (0)
Σ	3	5	4

Tab. 2: Cluster allocation for movie (27).

3 Showcase Application

Our demonstration scenario showcases a web application based recommender system on the IMDb. The application assists the users in finding movies which satisfy their preferences. In order to evaluate our novel Borda clustering technique in a user study, we added an evaluation mode as well. The user interface of our web-based demo-application consists of a search bar where users can determine their preferences for movie search (cp. Fig. 1). On the one hand, favorite actors can be searched and genres selected in the drop down list. On the other hand, further features of movies can be chosen, such as the release year and the length as an interval and the IMDb movie rating.

In our demo application always two clustering setups can be compared side by side. For each setup the distance measure, the number of desired clusters and the initialization with k-means++ can be chosen. Furthermore we allow the user to choose the number of desired clusters k , based on clustering quality measures. To find the most promising k , we used the quality measures Silhouette and Davies-Bouldin from which the user can choose a value.

In an extensive user study with 165 participants we evaluated our novel Borda social choice based k-means++ clustering technique against k-means++ using Euclidean and Canberra distance in order to investigate the quality of our approach on different scenarios. In each scenario sets between 50 and 60 movies were evaluated. We used this demo to show that our Borda clustering approach reaches adequate results considering the internal

³ Jaccard: $J(A, B) = |A \cap B| / |A \cup B|$ for two sets A and B . $J_\delta(A, B) = 1 - J(A, B)$

The image shows a search interface for a movie recommender system. It includes several input fields and sliders:

- Actor:** A text input field with the placeholder "Enter the actor's name...".
- Genre:** A dropdown menu showing "Action, Adventure, Drama, Fantasy, Mystery".
- Movie Rating:** A star rating system with 5 stars, where the first 4 stars are filled.
- Release Year:** A range slider from 1980 to 1999.
- Length in Minutes:** A range slider from 90 to 165.
- Movie Rating (Range):** A range slider from 6 to 10.
- Search:** A dark grey button labeled "Search".

Fig. 1: Showcase recommender search menu.

clustering quality. Furthermore we reached satisfying results in our study, so our Borda approach constitutes a worthwhile alternative to the basic k -means++ with traditional distance measures with the benefit of avoiding normalization before the clustering process.

We also investigated runtime and the number of iterations until a stable clustering is reached in [EKR19]. In summary, our *Borda* approach works nearly as good as the classic k -means algorithm w.r.t. the runtime, though the complexity of the Borda voting rule is $O(nd \cdot k^2 \cdot \log(k))$ where n is the number of d -dimensional objects, which get clustered in k clusters. However, our approach needs only a fractional part of iterations until termination and therefore is an alternative to k -means++ clustering using traditional distance measures without the overhead of normalization. While the convergence of our approach depends on the initial seeding, especially in higher dimensions, k -means++ provides an auxiliary benefit for our alternative approach.

References

- [AV07] Arthur, D.; Vassilvitskii, S.: K-means++: The Advantages of Careful Seeding. In: 18th ACM-SIAM Symposium on Discrete Algorithms. SODA '07, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, pp. 1027–1035, 2007.
- [DB79] Davies, D. L.; Bouldin, D. W.: A Cluster Separation Measure. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1(2):224–227, April 1979.
- [De92] Debord, B.: An Axiomatic Characterization of Borda's k -choice Function. Social Choice and Welfare, 9(4):337–343, Oct 1992.
- [EKR19] Endres, M.; Kastner, J.; Rudenko, L.: Analyzing and Clustering Pareto-Optimal Objects in Data Streams. In (Sayed-Mouchaweh, Moamar, ed.): Learning from Data Streams in Evolving Environments: Methods and Applications. Springer, pp. 63–91, 2019.
- [KEW11] Kießling, W.; Endres, M.; Wenzel, F.: The Preference SQL System - An Overview. Bulletin of the Technical Committee on Data Engineering, 34(2):11–18, 2011.
- [Ro87] Rousseeuw, P.: Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis. Journal of Comp. and Applied Math., 20:53 – 65, 1987.

RelaX: A Webbased Execution and Learning Tool for Relational Algebra

Johannes Kessler¹, Michael Tschuggnall², Günther Specht³

Abstract: The relational model and especially the relational algebra is the fundament of each relational database system and thus content of almost every database lecture. Even though there exist a few tools allowing to experiment with relational algebra, a common way to learn it is still by formulating queries on paper, without the option of checking them for syntax or even executing them. To fill this gap and to support students in their learning process, we propose RelaX, a webbased tool which is capable of executing arbitrary relational algebra statements on arbitrary datasets. By drawing interactive operator trees corresponding to the queries, it is also possible to compute the final result in a step-by-step manner. Finally, RelaX is also equipped to execute SQL queries and to automatically translate them to relational algebra.

Keywords: relational algebra, database systems, learning tool

1 Motivation

The relational model and the relational algebra have been proposed in 1970 by E. F. Codd [Co70] and have evolved to be the theoretical fundament of almost every modern relational database system. Through its procedural structure it is very well suited to transmit an understanding of the basic operations of the relational model, and is thus often utilized for (academic) teaching purposes as a solid base for a subsequent introduction to SQL. While there exist numerous possibilities to learn the latter using various database systems, query tools and corresponding datasets, relational algebra is still mostly taught purely theoretically due to the lack of proper execution tools. The two fundamental differences to SQL in terms of learning support can be summarized as follows: First, while the SQL standard defines a clear syntax with minor variations in current database systems, there is no official, standardized syntax for relational algebra. This leads to the problem that notations often differ significantly from one text book or author to another, making the verification of correctness a cumbersome procedure for students as well as for tutors. Second, SQL can be executed and tested directly on different database systems. On the contrary, tools that allow the execution of relational algebra statements are rare (e.g., *radb* [Ya18], *IRA* [Mu18] or *Relational* [To18]) and come with certain limitations. These include restrictions

¹ Department of Computer Science, Universität Innsbruck

² Department of Computer Science, Universität Innsbruck, michael.tschuggnall@uibk.ac.at

³ Department of Computer Science, Universität Innsbruck, guenther.specht@uibk.ac.at

to predefined datasets (*IRA*) or general usability flaws in terms of tool installation/execution (*radb*, *Relational*), editing queries (*IRA*) or creating, importing and sharing datasets. In addition, advanced but yet important operators like `GROUP BY` (γ) are not supported by any of those tools. Finally, the execution of relational algebra statements is usually done by transforming queries to SQL and utilizing a relational database system in the background (*IRA*, *radb*, *Relational*), making results difficult to understand.

2 Relax

With *Relax*⁴ we created a webbased learning tool for relational algebra, which addresses the previously mentioned problems by allowing to execute arbitrary relational algebra statements on predefined or custom datasets, offering a rich set of operators.

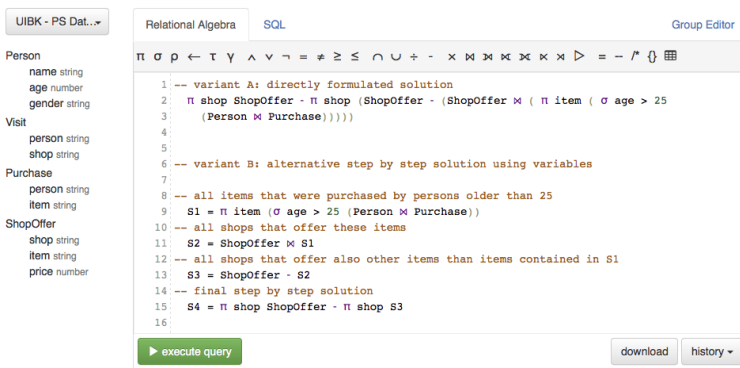


Fig. 1: The Relax editor.

Syntax and Editor

Analogously to SQL the statements are thereby treated as *source code*, which allows comfortable editing with syntax highlighting, auto completion and the possibility of stating comments. For query formulation, it is possible to directly use the mathematical notation as well as relying on corresponding key words. For example, when using a selection, both the use of σ as well as sigma is possible. With the proposed editor shown in Figure 1 it is thus possible to develop step-by-step solutions, while at the same time addressing and avoiding syntax errors directly during editing. For example, variant B in Figure 1 constructs the final solution S_4 using the three intermediate steps $S_1 - S_3$.

With respect to syntax we rely on the definitions given by Kemper and Eickler [KE15] as well as Garcia-Molina et al. [GMUW08]. During the process of creating *Relax* with the aim

⁴ Relax stands for Relational Algebra eXecutor and is available at <http://dbis-uibk.github.io/relax>

of supporting every possible relational algebra operator/formula, we came across several ambiguities as well as missing information in literature. For example, even for a simple operator like the natural join there exist several definitions, which either merge attributes with same names, inherit the corresponding attribute from the left or right table or use both attributes for the result. Among several other similar problems, the lack of definitions of proper scoping and precedence of operators had to be solved. With *RelaX*, we resolved these issues by following the semantic of SQL whenever possible.

Execution

For the execution, we laid the main focus on transparency and reproducibility, i.e., such that users can reconstruct the final result in a step-by-step manner. Consequently, statements are not transformed to SQL and executed in an external database system to retrieve results, but rather interpreted and executed directly in relational algebra. For this, a statement is transformed to and visualized as an interactive operator tree. For example, the statement of variant B from Figure 1 results in the operator tree depicted in Figure 2. Thereby, users can click on every node of the tree to not only inspect the intermediate result up to this node, but also to verify the corresponding schema and view other information like which attributes have been used for a natural join.

As we are interested in providing a transparent tool with respect to understandability, statements are not optimized at all and rather executed as a beginner would expect it. Therefore, all operations are implemented such that they preserve the order of tuples of the original relation. For joins, a nested loop join is implemented as it is the most intuitive variant. Finally, also grouping and aggregation is implemented in a comprehensible way.

3 Operators, Datasets and SQL support

RelaX supports all common unary operators like selection (σ), projection (π) or renaming (ρ , \leftarrow) and can handle several join variants (e.g., \bowtie , \ltimes , \bowtie_{\rightarrow}) as well as set operations (\cup , \cap , $-$) and division (\div). Besides a grouping operator (γ), also the sorting operator τ as proposed by Garcia-Molina et al. [GMUW08] has been implemented. To allow users to construct a step-by-step solution, we introduced variables as can be seen in variant B of Figure 1. A comprehensive description on all supported operators, their usage as well as syntax in general is provided in the help section of the tool.

With respect to datasets – where statements can be executed on – we provide several predefined, commonly used datasets that can be used directly (e.g., the university schema from Kemper and Eickler [KE15]). Moreover, it is possible to include custom datasets using Github-Gists⁵, SQL dumps or by editing them directly in an embedded spreadsheet editor.

⁵ <https://gist.github.com/>

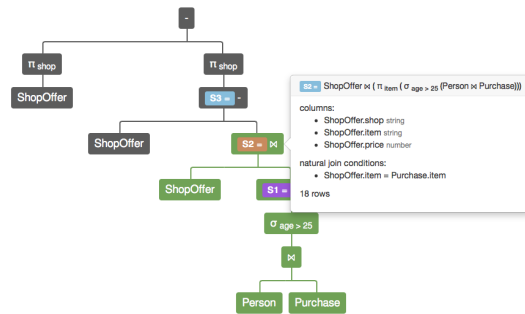


Fig. 2: Automatically computed operator tree, optionally showing intermediate results.

Finally, *RelaX* also supports the formulation of SQL statements⁶ that are automatically transformed to relational algebra, which then can be executed and inspected using the operator tree. With this functionality, students can more easily learn SQL if the concepts of relational algebra are understood, but also vice versa.

4 Conclusion

With *RelaX* we created a webbased learning tool for relational algebra which allows the execution of arbitrary statements on predefined or custom datasets. Conceptualized for the use in teaching we integrated several functions and means such as a comprehensive editor and operator tree visualization, all of which should help students understand relational algebra better.

References

- [Co70] Codd, E. F.: A Relational Model of Data for Large Shared Data Banks. *Commun. ACM*, 13(6):377–387, 1970.
- [GMUW08] Garcia-Molina, Hector; Ullman, Jeffrey D.; Widom, Jennifer: *Database Systems - The Complete Book*. Pearson Prentice Hall, New Jersey, 2nd edition edition, 2008.
- [KE15] Kemper, Alfons; Eickler, André: *Datenbanksysteme - Eine Einführung*, 10. Auflage. De Gruyter Oldenbourg, 2015.
- [Mu18] Muehe, Henrik: *IRA - Interaktive Relationale Algebra*. <http://db.in.tum.de/people/sites/muehe/ira/>, visited September 2018.
- [To18] Tomaselli, Salvo: *Educational Tool for Relational Algebra*. <https://github.com/ltworf/relational/>, visited September 2018.
- [Ya18] Yang, Jun: *IRA - Interaktive Relationale Algebra*. <https://users.cs.duke.edu/~junyang/radb/>, visited September 2018.

⁶ with the exception of correlated subqueries and recursive statements

MSDataStream — Connecting a Bruker Mass Spectrometer to the Internet

Roman Zoun,¹ Kay Schallert,² David Broneske,¹ Wolfram Fenske,¹ Marcus Pinnecke,¹
Robert Heyer,² Sven Brehmer,³ Dirk Benndorf,² Gunter Saake¹

Abstract: Metaproteomics is the biological research of proteins of whole communities comprised of thousands of species using tandem mass spectrometry. But still it follows a sequential non parallelizable workflow. Hence, researchers have to wait for hours or even days until the measurement data are available. In our demo, we show a way to decrease the smallest unit of the workflow to a minimum to realize a near real time stream processing system on a fast data architecture.

Keywords: Internet of Things; Bioinformatics; Mass Spectrometry; SMACK Stack; Streaming

1 Introduction

Metaproteomics is the biological research of proteins of whole communities with thousands of species (e.g. from ocean samples or the human gut) [Ma07]. Metaproteomics is very important for diagnosing diseases, optimizing biogas plants, etc. [Ma07, PF17]. This research relies on a mass spectrometer, that is figuratively speaking a huge scale for tiny particles [Ma07]. The mass spectrometer measures the mass of parts (peptides) of a protein complex. Proteomics and metaproteomics follow a similar workflow, as shown in Figure 1. The workflow encompasses the following steps: (1) The biological sample gets purified, in a way that only proteins are left in the sample. (2) The proteins are split into smaller parts, called peptides. (3) The peptides are measured in the mass spectrometer. (4) The mass spectrometer transforms all peptides into a digital signal, one peptide is represented as one spectrum (~2 hours) [Ma07]. (5) Conversion of the digital signal into a readable format, such as MGF (~1 hours) [Ki10]. (6) Compare the experimental spectra with real-world proteins to identify the experimental data (~2 hours) [Mi13]. (7) The identified spectra need a validation to remove false positive results (~2 hours) [El10]. (8) Biological researchers analyze the experimental results. All these steps are sequentially executed and it takes hours to complete. The workflow has not changed for years and newer devices increase the amount of data produced by a mass spectrometer (~20GB per experiment, ~45000 spectra,

¹ University of Magdeburg, Working Group Databases and Software Engineering, Germany, {first-name.lastname@ovgu.de}

² University of Magdeburg, Chair of Bioprocess Engineering, Germany, {firstname.lastname@ovgu.de}

³ Bruker Daltonik GmbH, 28359 Bremen, Germany, Sven.Brehmer@bruker.com

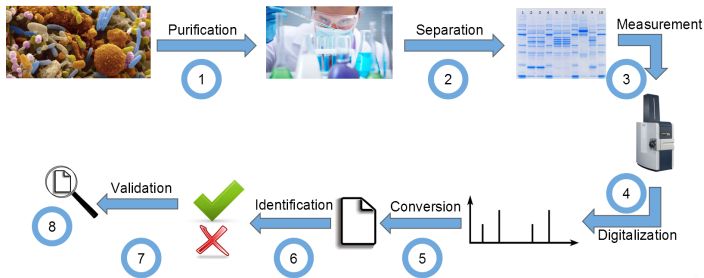


Fig. 1: The current (meta-)proteomics research workflow.

(~6 spectra/second) and the current file-based workflow is outdated [He17]. Especially in clinical use cases, the goal is to process the data in real-time, but this is infeasible with the current approach. The steps 1, 2 and 3 are realized in a laboratory, step 4 is done by a mass spectrometer, but the other steps are completely digital and can be optimized using IT. Currently, the smallest parallelizable unit is a whole experiment. Since the mass spectrometer measurement and digitalization duration (~2 hours) cannot be avoided, we shrink the smallest unit after the fourth step to a single spectrum instead of a whole experiment. That means, we connect a mass spectrometer to the cloud for outsourcing the calculation and overlap mass spectrometer processing (3-4) with data processing (5-8) by using a streaming-based architecture [Wa16]. We choose a Fast data architecture, since we need near real time processing of huge amounts of mass spectrometry data [Wa16]. Specifically, we deployed a SMACK Stack⁴. In this demo, we present a cornerstone of our architecture, our tool MSDDataStream. The tool is responsible for grabbing the single spectrum data from the mass spectrometer as it arrives, converting it into a readable format and streaming the data to the cloud for processing. In the cloud each spectrum needs a comparison to all peptides in our database (~27 millions) and validation of the matched results using machine learning classification.

2 System Architecture

For our development, we collaborate with Bruker Daltonik GmbH, a mass spectrometer company from Bremen, Germany. Each of their devices are connected via a digitizer to a computer. The digital signal is collected in a proprietary RAW file format. Additionally, the measurement software provides structure and meta data to index spectrum data that belongs together. Since each manufacturer uses their own RAW file format, Bruker provided us with a library (DLL file) to access the digital spectrum data. The index data is stored in an SQLite database and provides the location of spectrum data in the RAW file. The index structure consists of 16 tables that describe meta-information and the spectrum location

⁴ pipeline of Big Data technologies: Spark, Mesos, Akka, Cassandra, Kafka

for each single spectrum. MSDataStream reads the meta-information for a single spectrum from the SQLite index and extracts the spectrum data from the RAW file via the DLL. Afterwards, MSDataStream converts the spectrum data into a readable format. Then, several pre-processing methods increase the quality of the spectrum data. Finally, MSDataStream sends the data to the cloud via Kafka broker for further processing or to write the data into a file. Summarized, MSDataStream checks periodically (e.g. every 2 seconds) for new measured data, collects it and produces messages. The system architecture is shown in Figure 2. MSDataStream requires several parameters for the execution. We implemented

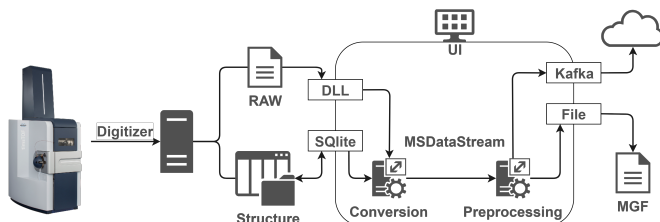


Fig. 2: The flow of the mass spectrometer data through the MSDataStream software

a JavaFX interface, which queries MSDataStream tasks and helps the user to configure the parameters. After sending the first data messages, the user waits until the results are generated from the fast data system and shown in a live updated visualization.

3 Streaming Workflow

The streaming data processes work for each spectrum separately (Figure 3). (1) Each spectrum, which arrives at the cloud system first gets prepared for the identification step with several filters on the data. (2) This step requires real-world protein data (sequence database), which is already in the system [Zo18a]. (3) The comparison of the experimental spectrum with the real world data produces peptide spectrum matches [Zo18b]. (4) Each of these matches needs validation, which is based on a machine learning classifier [Zo18c]. (5) Validated matches are stored as results in our database. The database stores all the experiment data such as validated results, spectra data and the proteins.

4 Demo Walkthrough

Our tool MSDataStream works with live measured data or with already finished measurements. For the demonstration, we will use a laggy version of the second method with time delays, which simulates live measurement. After a demo of the measurement process on a model of a mass spectrometer, the user can use the MSDataStream UI to configure and start a streaming process. During the process, the user will be shown the throughput of the system in the Apache Spark UI. Furthermore, the user can take a closer look at the components of the SMACK stack deployment in Marathon and mesos webUI. Afterwards, we show a live and continuously updated visualization of the experiment results, i.e., protein list.

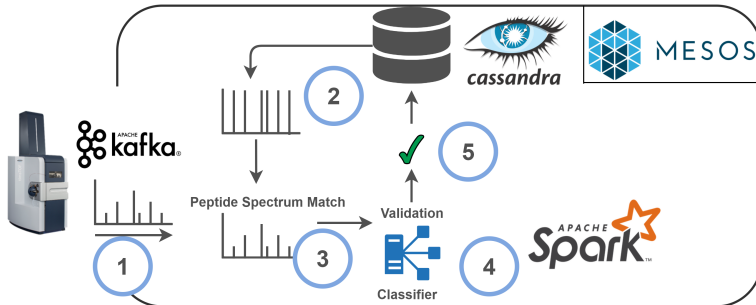


Fig. 3: The metaproteomics data analysis workflow on a SMACK stack.

Acknowledgments The authors sincerely thank Xiao Chen, Gabriel Campero Durand, Sebastian Krieter and Andreas Meister for their support and advice. This work is partly funded by the de.NBI Network (031L0103), the European Regional Development Fund (grant no.: 11.000sz00.00.0 17 114347 0), the DFG (grant no.: SA 465/50-1), by the German Federal Ministry of Food and Agriculture (grants no.: 22404015) and dedicated to the memory of Mikhail Zoun.

References

- [El10] Elias, Joshua et al: Target-Decoy Search Strategy for Mass Spectrometry-Based Proteomics. *Methods in Molecular Biology*, 604:55–71, 2010.
- [He17] Heyer, Robert et al: Challenges and perspectives of metaproteomic data analysis. *Journal of Biotechnology*, 261(Supplement C):24 – 36, 2017.
- [Ki10] Kirchner, Marc et al: MGFp: An Open Mascot Generic Format Parser Library Implementation. *Journal of Proteome Research*, 9(5):2762–2763, 2010. PMID: 20334363.
- [Ma07] Maron, Pierre-Alain et al: Metaproteomics: A New Approach for Studying Functional Microbial Ecology. *Microbial Ecology*, Volume 53:486—493, 2007.
- [Mi13] Millionsi, Renato et al: Pros and cons of peptide isoelectric focusing in shotgun proteomics. *Journal of chromatography. A*, 1293:1—9, June 2013.
- [PF17] Petriz, Bernardo A.; Franco, Octávio L.: Metaproteomics as a Complementary Approach to Gut Microbiota in Health and Disease. *Front Chem*, 2017.
- [Wa16] Wampler, Dean: *Fast Data Architectures for Streaming Applications*. O’Reilly Media, 1005 Gravenstein Highway North, Sebastopol, CA 95472., first edition, sep 2016.
- [Zo18a] Zoun, Roman: *Internet of Metaproteomics - Optimizing the Metaproteomics Workflow Using Fast Data on the SMACK Stack*. Phd Symposium, ICDE, 2018.
- [Zo18b] Zoun, Roman et al: Protein Identification as a Suitable Application for Fast Data Architecture. In: *BIOKDD workshop, DEXA*. Springer, Cham, pp. 168–178, 2018.
- [Zo18c] Zoun, Roman et al: Streaming FDR Calculation for Protein Identification. In: *New Trends in Databases and Information Systems*. Springer, Cham, pp. 80–87, 2018.

Database-Supported Video Game Engines: Data-Driven Map Generation

Daniel O’Grady¹

Abstract: Video game engines can benefit greatly from being tightly coupled with database systems. To make this point and exemplify the similarities in database and game engine technology, we demonstrate a data-driven approach to generate maps for video games, expressed purely in *SQL*. The demonstration will feature such a live database-supported game that is playable on-site.

1 A Marriage of Game and Database Engines

Early video games were conceived as little more than a pastime, but the video game industry has grown immensely since its beginnings in the mid-twentieth century [Ke01]. Video games have since seeped into many aspects of our lives, such as education, simulations, and serious gaming in our work life. Games further hold a significant economic impact on the world, grossing billions of dollars every year, and employing thousands of workers [Fa18]. Contrasting the humble beginnings of video games, today’s games are becoming ever more computationally demanding, with hundreds of players playing the game at once, huge persistent worlds to explore, and large numbers of objects to interact with.

To avoid implementing common components over and over, numerous *video game engines* have been developed throughout the years, which offer commonly required functionality to game developers. Possible game engine components include, but are not limited to: (1) Simulation of physics, (2) collision detection, (3) pathfinding, (4) control of non-player characters (NPCs or “AI”), (5) network communication, (6) video and audio output, (7) processing input from the player, (8) creating and managing game worlds and objects. Most of these components have to deal with large amounts of information that has to be processed rapidly – a true forte of database systems. Unfortunately, databases are predominantly used as little more than dumber-down persistent storage in the context of video games. This strongly contrasts with the database community’s creed to *move the computation closer to the data*. Indeed, in this paper we propose a stronger connection between video games and database systems, by moving parts of a game’s internals to the database system, to benefit from database system technology: (1) Selecting a subset of objects is a task where databases excel. This is interesting for finding objects in close vicinity to a player, visual clipping during rendering (“culling”), or collision detection.

¹ University of Tübingen, Department of Computer Science, daniel.ogrady@uni-tuebingen.de

- (2) Updating the state of many game objects in bulk, as proposed by Gehrke et al. [Wh07].
- (3) Guaranteeing consistent state through transactions.

As we are aware that implementing algorithms in *SQL* may seem foreign to many game developers, we aim to *couple the database tightly with the video game engine* while not exposing developers to the intricacies of database internals.

To exemplify our claim, this demo showcases *declarative map generation*, which is the automatic generation of a playing field for games. The generation either happens before the player dives into the game or when they reach the border of the field to create the illusion of an infinite world. This paper focuses on *real-time strategy* (RTS) games. In RTS games, players control multiple game figures (so-called *units*) on a playing field, the *map*. Units are instructed to collect resources, attack enemy units or just move to a new position. This calls for large open spaces to maneuver units properly, so maps can not just be completely random but need certain properties to be compelling to the player.

2 Data-Driven Iterative Map Generation

The maps we create are comprised of *tiles*, arranged in a two-dimensional grid. Each tile represents a certain type of terrain specific for the game the map is generated for. Such a set of tiles could be *walkable* □, *wall* ■, *coast* ∙, and *water* ≈. Our approach combines these into *modules*, which are predefined clusters of 3×3 tiles.² Fig. 1 shows an example of a module representing a piece of horizontal shoreline on the left. The map generation starts with a seed of one module and then *joins* modules to the outer edges repeatedly. The input for the algorithm is a tuple $(\mathcal{T}, \mathcal{M}, C, \mathcal{S})$ with

1. \mathcal{T} : a set of tiles,
2. \mathcal{M} : a relational representation of modules³, each consisting of 3×3 tiles $\in \mathcal{T}$,
3. a relation $C \subseteq \mathcal{T} \times \mathcal{T}$ describing the *compatibility* of tiles (explained below) and
4. $\mathcal{S} \in \mathcal{M}$: an initial seed module.

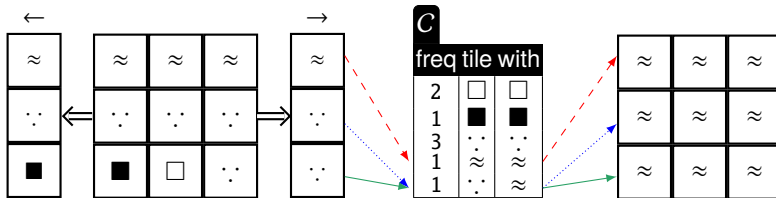


Fig. 1: Two compatible modules. Each pair of neighbouring tiles in the adjacent edges can be joined on the compatibility table C after extracting the edges (only shown for two edges of the left module).

² While those dimensions have turned out to be rather convenient for defining modules, any other rectangular dimensions work as well.

³ Many tabular module encodings are conceivable and we abstract from these here.

```

1  WITH RECURSIVE map(x,y,tile) AS (
2  (SELECT S)
3  UNION ALL
4  (SELECT ...
5   FROM C, map, M
6   WHERE
7    C.tile = edge(map) AND
8    C.with = edge(M) AND
9    NOT(<termination condition>)
10 ))

```

Fig. 2: Pseudocode for the map generation. It creates a table map of tiles tile positioned at coordinates x,y. edge(...) is a function that produces edges as shown in Fig. 1 on the left.

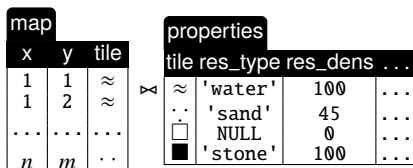


Fig. 3: Joining a map of size $n \times m$ on a table of properties. Here, each tile type is associated with a resource type, a resource density, and possibly other information.

The algorithm selects the *edges* of all modules in \mathcal{M} and of the outermost modules of the map together with the direction they are facing. An edge is comprised of the three adjacent tiles of either side on a module. Take Fig. 1, where two of the four edges of the left module are extracted (denoted by \Leftarrow and \Rightarrow). Each edge of the map is then joined with a compatible module in \mathcal{M} . Two modules are said to be *compatible* if their adjacent edges (the ones facing opposite directions) are compatible in all neighbouring tiles. Tiles, in turn, are compatible with each other if they can be joined on C , as is shown in Fig. 1. An additional value freq per compatibility rule denotes the frequency, which allows us to control how likely a rule is selected in ambiguous situations where more than one compatibility rule qualifies. A formulation of this iterative map expansion in pseudo *SQL* is shown in Fig. 2. Starting with a small seed S , the intermediate result is gradually expanded by recursively joining matching modules to it. Usually, the termination condition limits the map size, but can also be bound to other constraints. This type of iterative map generation is a perfect match for a recursive common table expression.

The data-driven nature of this algorithm enables game developers to easily control this process with their own modules and compatibilities. For example, adding a rule (5, ≈, ■) to C in Fig. 1 would allow water to be generated next to walls. In fact, this would occur more often than, say, water next to water, because of the higher frequency of 5 of the rule.

To finalise the map, it is joined with a tile property table to attach additional semantics that are required for a particular game. This is shown in Fig. 3, where the result of the recursive CTE map is joined with a table properties which contains resource information for each tile type. The map is subsequently converted into a format that is understood by the game engine. Then, actual gameplay commences.

3 Demonstration Setup

The on-site demonstration will showcase the discussed map generation algorithm by tapping into the OpenRA⁴ engine, an engine specifically tailored to RTS games. OpenRA is written

⁴ <https://www.openra.net>

in C#, and is still under active development since 2007. The map generation is implemented in pure *SQL* and runs on a *PostgreSQL 10* database system. The process of generating a map is triggered from within a game implemented on top of OpenRA by sending a query to the database and receiving a stream of tuples. Those are then used to create the native data structure provided by the engine, just as if the map was read from disk.⁵

To demonstrate the capabilities of our approach, we will bring canned tilesets with us. These tilesets will have realistic size in terms of modules and compatibilities and will be used to produce maps of typical dimensions for OpenRA, between 40×40 and 200×200 tiles. But the audience is invited to propose changes to the input data to explore how it affects the generated maps. The generation process takes about 60 milliseconds,⁶ which allows for rapid re-generation of maps to compare them with each other.⁷ We will also bring the source code of the modified OpenRA engine for interested audience members to inspect the details of our implementation and convince themselves that the fingerprint we left on the engine is rather small.

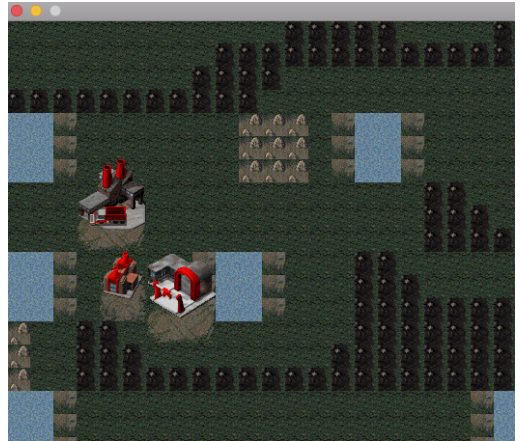


Fig. 4: Screenshot of a generated map with pools of water with shores and enclosed walkable territory. The game will be playable on site.

References

- [Fa18] Facts, E.: Essential Facts About the Computer and Video Game Industry, http://www.theesa.com/wp-content/uploads/2018/05/EF2018_FINAL.pdf, Accessed: 2018-09-13, 2018.
- [Ke01] Kent, S. L.: The Ultimate History of Video Games: From Pong to Pokemon—the Story Behind the Craze That Touched Our Lives and Changed the World. Prima Communications, Inc., Rocklin, CA, USA, 2001, ISBN: 0761536434.
- [Wh07] White, W.; Demers, A.; Koch, C.; Gehrke, J.; Rajagopalan, R.: Scaling Games to Epic Proportions. In: Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data. SIGMOD '07, ACM, Beijing, China, pp. 31–42, 2007, ISBN: 978-1-59593-686-8, URL: <http://doi.acm.org/10.1145/1247480.1247486>.

⁵ In future work, we will seek to eliminate the need for native data structures altogether.

⁶ In addition to the raw generation process, the current implementation takes about 10 seconds to translate the generated data into a native OpenRA data structure. This additional times will disappear once we move more of the game engine logic to the database system.

⁷ Tested on a MacBookAir with a 2,2 GHz Intel Core i7, 8 GB 1600MHz DDR3 RAM and a SM0512G SSD.

Protobase: It's About Time for Backend/Database Co-Design

A Demo on Rapid Microservice Prototyping for Third-Party Dataset Analytics

Marcus Pinnecke,¹ Gabriel Campero,¹ Roman Zoun,¹ David Broneske,¹ Gunter Saake¹

Abstract: In this interactive demonstration, we show the current state of `PROTOBASE`, our main-memory analytic document store that is designed from scratch to enable rapid prototyping of efficient microservices that perform analytics and explorations on (third-party) JSON-like documents stored in a novel columnar binary-encoded format, called the `CABIN` file format. In contrast to other solutions, our database system exposes neither a particular query language, nor a fixed REST API to its clients. Instead, the entire user-defined backend logic, whose user code is written in Python, is placed inside a sandbox that runs in the systems process. `PROTOBASE` in turn exposes a *user-defined* REST API that the (frontend) application interacts with. Thus, our system acts as a backend server while at the same time avoids full exposure of its database to the clients. Consequently, a `PROTOBASE` instance (database + user code + REST API) serves as (the entire) microservice - potentially minimizing the number of systems running in a typical analytic software stack. In terms of execution performance, `PROTOBASE` therefore takes the inter-process communication overhead between backend and database system out of the picture *and* heavily utilizes columnar binary document storage to scale-up for analytic queries. Both features lead to a notable performance gain for non-trivial services, potentially minimizing the number of required nodes in a cloud setting, too. In our demo, we overview `PROTOBASE`'s internals, spot major design decisions, and show how to prototype a scholarly search engine managing the Microsoft Academic Graph, a real-world scientific paper graph of roughly 154 mio. documents.

Keywords: NoSQL; Document Stores; Analytics; Rapid Prototyping; Backend/Database Co-Design

1 Analytics on Schema-Free and Hierarchical Datasets

There is a common saying that the early bird catches the worm. This popular phrase serves as an advice on performing an action as soon as possible before competitors are able to do so to maximize the own fitness. Therefore, it is not surprising that service providers have a certain need to offer their solutions to the market quickly. Consequently, agile development methods² have established themselves as de-facto standard for many developer teams today. In such a context, tools that help the agile workflow are favored while cumbersome non-agile tools are typically avoided. Especially for agile microservice development, document stores rather than relational systems are often the first choice when it comes to large-scale data management. One reason for choosing document stores is the native support of the first-class citizen data exchange format for many (web) developers, the JavaScript Object Notation

¹ DBSE Working Group, University of Magdeburg, (pinnecke|campero|broneske|zoun|saake)@ovgu.de

² Manifesto for Agile Software Development: <https://agilemanifesto.org>

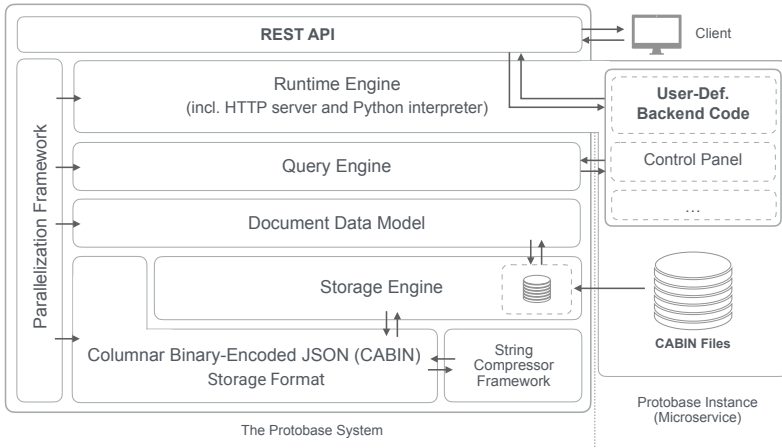


Fig. 1: PROTOBASE architecture including client request to and interaction with user code

(JSON) format³. Even if agile development is not an argument, it is likely that a team is confronted with JSON when working with a third-party data source, anyway. Since it is expressive, flexible, human readable, and easy to parse, JSON (or XML as an alternative) is a prominent serialization format for data exchange. However, *exploring* and *analyzing* large repositories of such JSON files can quickly become a time-consuming, non-trivial task [LKC⁺17, WZS⁺18].

A third-party provided JSON file (or *document*) is typically small: only a handful of properties of potentially incomplete records are contained on each tree level, and the typical maximum depth of a JSON tree is smallish, too. For example, a request to GitHub's^{4a} repository API may return only a single document of approx. 5 KB size. Such a document contains around 50 (mostly string-typed) properties, and one nested record with around 20 properties providing some information on the repository owner. Naturally, this is *not* a good starting point for any analytic system. Therefore, a typical document store is operational rather than analytical, and will store a single document as a single unit of information following a key-value pair serialization (maybe binary encoded, such as BSON^{4b} or UBJSON^{4c}). Since analytics favor columnar storage schemes instead of key-value storage schemes, it is common practice for analytics on a high number of documents to be based on several technologies in orchestration. Namely, a state-of-the-art software stack may consists of Kubernetes^{4d}, Docker^{4e}, MongoDB^{4f}, Elasticsearch^{4g}, Spark^{4h}, and others. Hence, many systems are needed to fulfill a particular task - which is a challenge by its own [LHB13] - even if the queries are quite simple.

³ RFC 7159, *The JSON Data Interchange Format (Marc 2014)*: <https://tools.ietf.org/html/rfc7159>

⁴ ^aGitHub: <https://github.com>, ^b Binary JSON: <http://www.bsonspec.org>, ^c Universal Binary JSON: <http://www.ubjson.org>, ^d Kubernetes: <https://www.kubernetes.io>, ^e Docker: <https://www.docker.com>, ^f MongoDB: <https://www.mongodb.com>, ^g Elasticsearch: <https://www.elastic.co/de>, ^h Spark: <https://spark.apache.org>

We overcome the limitation that naturally leads to key-value pair serialization: we convert a set of JSON files into a single file that is formatted columnwise and binary encoded, a CABIN file. Likewise, we focus on backend/database co-design for development teams that are issued with a particular task: quickly deploying fast-responding microservices to create novel value on third-party, schema-free, hierarchical, read-mostly datasets by providing analysis, aggregation and exploration features. As a case study, we decide for a scholar search engine motivated by our previous work [CJP⁺18]. Our experiences call for a PROTOBASE instance directly deployed as backend for a microservice backing a scholar search engine frontend (called PAN).

2 The PROTOBASE Document Store

PROTOBASE⁵ is our open-source document store that executes backend code for analytic applications in a sandbox inside the database system process (see Figure 1 for its architecture).

Overview Backend code is written in Python, exposes a REST-based user-defined service, and potentially accesses PROTOBASES query engine and storage engine directly. Backend code is executed by the runtime engine in PROTOBASE that handles HTTP requests, invokes user-defined code and manages delegation between components. However, backend code is primarily used for the user-defined microservice but may be intermixed with pre-defined components (such as PROTOBASES control panel) or other components. Underneath, the storage engine operates on a memory resistant database loaded once from a CABIN-formatted database image. To take advantage of modern processors asynchronous capabilities, the entire architecture stack is supported by our parallelization framework.

Instances The interaction between a backend and our database system does not require inter-process communication. This is a good fit for microservice prototypes due to (a) data marshalling can be kept to a minimum, and (b) less systems are required to get things done. In a sense, a service backed by PROTOBASE (an *instance*) acts as a specialized database system exposing a user-defined, domain-specific interface that frontends interact with.

Cabin Files Our data model is our novel *strict document* data model which is similar to the JSON data model (cf. [BRSV17] for a basic theoretical framework) but with two restrictions: (i) all property values of type array must contain elements of the same type, and (ii) arrays of arrays are invalid. Both (i) and (ii) allow us to efficiently recursively transform any strict document into a *columnar document* re-written to organize its data in a columnar fashion. In detail, a columnar document (a) stores keys and values in two separate columns grouped by value type, (b) aggressively performs *null* compression, and (c) converts properties mapping to arrays of objects to an equivalent (sparse) columnar table. Multiple documents are bundled, binary-encoded and lightweight compressed into a *Columnar Binary-Encoded JSON (CABIN)* file⁶.

⁵ PROTOBASE code repository: <https://github.com/proto4labs/protobufase>

⁶ Cabin file format specification: <http://www.cabinspec.org>

3 Demonstration Outline

In our demo, we show an example of (fullstack) prototyping of a (tiny) scientific research search engine (called PAN) using PROTOBASE and the Microsoft Academic Graph⁷ as dataset.

PAN is a web app rendering information fetched from a microservice. For ease of simplicity, PAN offers four features: (1) displaying publication information given a paper title, (2) listing publications within a certain time span given a specific author's name, (3) displaying quantitative information, and (4) lists publications via the *is-cited-by* relationship. Finally, we implement a microservice in PROTOBASE to provide a backend for PAN.

Outline The outline for our demonstration on working with PROTOBASE is as follows:

- (i) A brief introduction on fundamentals of PROTOBASE: the systems architecture and instance model (incl. structure, interaction user code and system, and database I/O).
- (ii) Showcasing the database meta information provided by PROTOBASE to get informed on the dataset at hand (e.g., property names, type conflicts, or schema information).
- (iii) Finally, we show several concepts and some code snippets to outline how to query, aggregate, and how to use the built-in cache and indexing mechanism in PROTOBASE.

The audience is invited to play around with PAN and PROTOBASE.

References

- [BRSV17] Pierre Bourhis, Juan L Reutter, Fernando Suárez, and Domagoj Vrgoč. JSON: Data Model, Query Languages and Schema Specification. In *Proceedings of the Symposium on Principles of Database Systems (ACM PODS)*, pages 123–135, 2017.
- [CJP⁺18] Gabriel Campero, Anusha Janardhana, Marcus Pinnecke, Yusra Shakeel, Jacob Krüger, Thomas Leich, and Gunter Saake. Exploring Large Scholarly Networks with Hermes. In *International Conference on Extending Database Technology (EDBT)*, pages 650–653, 2018.
- [LHB13] Harold Lim, Yuzhang Han, and Shivnath Babu. How to Fit when No One Size Fits. In *Conference on Innovative Data Systems Research (CIDR), Online Proceedings*, 2013.
- [LKC⁺17] Yinan Li, Nikos R Katsipoulakis, Badrish Chandramouli, Jonathan Goldstein, and Donald Kossmann. Mison: A Fast JSON Parser for Data Analytics. *Proceedings of the VLDB Endowment (VLDB)*, pages 1118–1129, 2017.
- [WZS⁺18] JunPing Wang, WenSheng Zhang, YouKang Shi, ShiHui Duan, and Jin Liu. Industrial Big Data Analytics: Challenges, Methodologies, and Applications. *arXiv preprint arXiv:1807.01016, Online Proceedings*, 2018.

⁷ Microsoft Academic Graph as part of the Open Academic Graph: <https://aminer.org/open-academic-graph>

Zuverlässige Verspätungsvorhersagen mithilfe von TAROT

Christoph Stach,¹ Corinna Giebler,¹ Simone Schmidt¹

Abstract: Bei der Einhaltung von Schadstoffwerten nehmen öffentliche Verkehrsmittel eine immer entscheidendere Rolle ein. Daher wird vermehrt darauf geachtet, deren Attraktivität zu erhöhen. Ein wichtiger Punkt hierbei ist die Vorhersagegenauigkeit von Verspätungen zu verbessern, damit Fahrgäste entsprechend planen können. Die aktuell angewandten Ansätze sind häufig ungenau, da sie die zur Verfügung stehenden Daten nicht ausreichend nutzen. In diesem Beitrag stellen wir daher mit TAROT ein System vor, das mittels prädiktiver Analysen die Vorhersagegenauigkeit von Verspätungen verbessert, indem es in den Modellen Verspätungsförpflanzungen berücksichtigt. Darüber hinaus ist es in der Lage, im Fall einer Störung augenblicklich auf ein besseres Vorhersagemodell umzusteigen und auf sowohl schleichende als auch abrupte Veränderungen automatisch zu reagieren. Die Vorteile dieser Eigenschaften lassen sich in unserem TAROT-Demonstrator anhand von vier repräsentativen Anwendungsszenarien zeigen. Auch wenn sich die gezeigten Szenarien alle auf die Verspätungsvorhersage von S-Bahnen beziehen, lassen sich die Konzepte von TAROT auch auf viele andere Anwendungsbereiche (z. B. zur Bestimmung von Produktionszeiten in der Industrie 4.0) anwenden.

Keywords: Verspätungsvorhersage; ÖPNV; deskriptive Analyse; prädiktive Analyse; Concept Drift.

1 Einleitung

Nicht erst seit den ausgesprochenen Dieselfahrverboten sind die Bürger für das Thema „Luftreinhaltung“ sensibilisiert. Eine Möglichkeit zur Reduktion der Luftbelastung stellen öffentliche Verkehrsmittel dar. Große Verspätungen und Ausfälle erschweren den Umstieg allerdings. Auch das Bundesministerium für Verkehr und digitale Infrastruktur hat diese Problematik erkannt und sucht nach Lösungen, mit deren Hilfe Fahrplanauskünfte genauer und Reiseplanungen erleichtert werden sollen [BMVI18].

Gegenwärtig werden zwar viele Fahrtdaten erfasst, deren Potential wird bei der Fahrplanung und der Verspätungsvorhersage jedoch nicht ausgenutzt. Abb. 1(a) stellt vereinfacht dar, wie die Fahrplanung erfolgt. Fahrtdaten werden in Echtzeit erfasst und historisiert. Wurden ausreichend Daten gesammelt, werden *deskriptive Data-Mining-Techniken* darauf angewandt, um ein Modell zu erstellen, aus dem hervorgeht, wie lange Fahrten abhängig von bestimmten Einflussfaktoren dauern. Mithilfe dieses Modells kann ein Fahrplan erstellt werden. Dieser Prozess muss regelmäßig wiederholt werden, um auf veränderte Bedingungen zu reagieren.

Für die Bestimmung einer Verspätung werden die so errechneten Fahrplandaten mit den aktuellen Fahrtdaten verglichen (siehe Abb. 1(b)). Die Vorhersagequalität ist für den nächsten

¹ Universität Stuttgart, Universitätsstraße 38, D-70569 Stuttgart, Vorname.Nachname@ipvs.uni-stuttgart.de

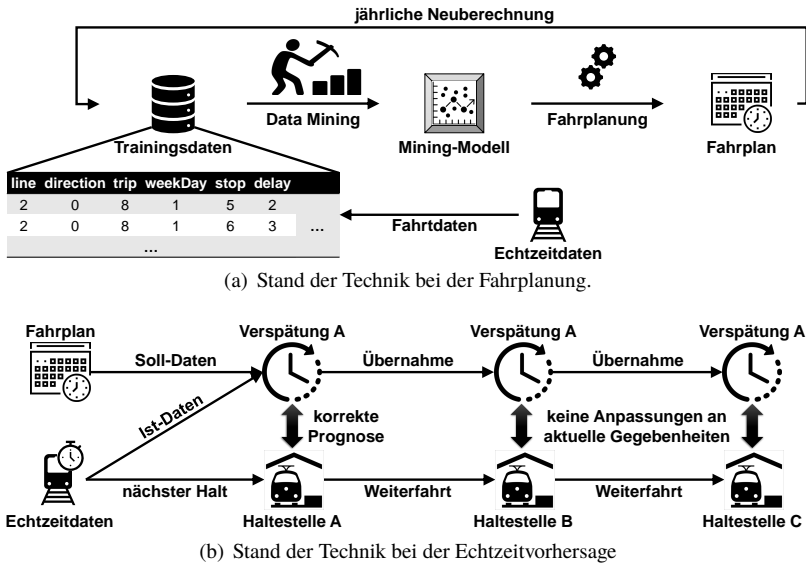


Abb. 1: Stand der Technik bei der Fahrplanung und Echtzeitvorhersage.

Halt sehr gut, verschlechtert sich aber mit jedem weiteren Halt, da die Verspätung nur übernommen wird und Verspätungsfortpflanzungen nicht berücksichtigt werden [WN08]. Während die Deutsche Bahn mittlerweile bessere Konzepte für die Verspätungsvorhersage nutzt, zeigten Gespräche mit lokalen Verkehrsbetrieben, dass beispielsweise für den S-Bahn-Betrieb weiterhin diese überholten Vorhersagetechniken zum Einsatz kommen.

2 Konzept von TAROT

Um eine bessere Vorhersage zu ermöglichen, führen wir *TAROT* ein. In *TAROT* wird unter Ausnutzung aller zur Verfügung stehenden Daten ein verbessertes Vorhersagemodell errechnet, angewandt und stetig verbessert. *TAROT* basiert auf der *BRAID-Architektur* [Gi18].

Abb. 2 zeigt die Architektur von *TAROT*. Eingehende Daten werden mittels *Kafka* in zwei *Topics* (Fahrt- und Störungsdaten) aufgeteilt. Die Fahrtdaten werden historisiert. Mit *Spark* werden daraus Vorhersagemodelle errechnet. Neben einem Modell für den Normalbetrieb wird für jede Störungsart (z. B., „Weichenstörung“ oder „Unfall“) ein eigenes Modell erstellt und in einem Modellspeicher abgelegt. Parallel zur Bestimmung der Soll-Daten werden die Echtzeitfahrtdaten als Ist-Daten an eine *Spark-Streaming-Komponente* weitergeleitet. Hierin ruft das *Spark-MLlib-Modul* das aktuell benötigte Modell ab und erstellt eine Verspätungsvorhersage. Sollte eine Störung eintreten, wird automatisch ein anderes Modell geladen. Die eingesetzten Vorhersagealgorithmen sind in der Lage, das Modell an *Concept Drifts* anzupassen. Neben der Vorhersage findet in *TAROT* auch eine Qualitätskontrolle

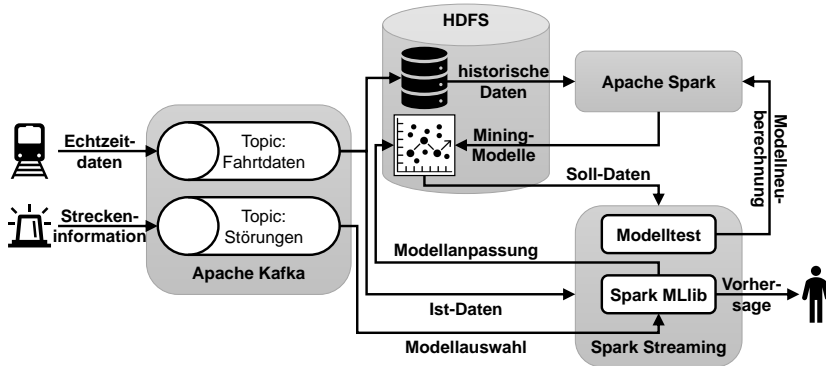


Abb. 2: Architektur von TAROT.

der Modelle statt. Sollte die Vorhersagequalität aufgrund einer abrupten Veränderung unter einen Grenzwert fallen, stößt TAROT automatisch eine vollständige Modellneuberechnung auf den historischen Daten, die seit der Veränderung gesammelt wurden, an.

3 Der TAROT-Demonstrator

Die Funktionsweise von TAROT kann anhand von vier Szenarien veranschaulicht werden. Die Szenarien verwenden jeweils synthetische Daten. Die zum Einsatz kommenden Datenmodelle orientieren sich an dem realen Datenmodell eines lokalen Verkehrsbetriebs. Es wurden allerdings irrelevante Attribute entfernt, um die Übersichtlichkeit zu erhöhen.

Demo-Szenario 1: Normalbetrieb. In dem ersten Szenario sollen die Grundfunktionalitäten von TAROT demonstriert werden. Hierfür wird auf historischen Daten ein Vorhersagemodell errechnet. Dieses Modell wird anschließend im Modellspeicher abgelegt und der Streaming-Komponente zur Verfügung gestellt. Diese ruft das Modell ab und wendet es auf eingehende Fahrdaten an. Dadurch lassen sich Verspätungsvorhersagen treffen.

In diesem Szenario wird davon ausgegangen, dass es zu keinen außergewöhnlichen Störungen kommt. Es wird zunächst lediglich der Normalbetrieb simuliert. Das heißt, es ergeben sich ausschließlich kleinere, betriebsbedingte Verzögerungen. Im Gegensatz zu der aktuell genutzten Technik zur Bestimmung der zu erwartenden Verspätung, wird in TAROT allerdings die Verspätungsförtpflanzung berücksichtigt. Hierdurch wird die Vorhersagegenauigkeit wesentlich verbessert, was sich insbesondere dann auswirkt, wenn zwischen der aktuellen Position und dem Ziel mehrere Haltestellen liegen.

Demo-Szenario 2: Reaktion auf Störungen. Wird über den Streckeninformationsdienst eine Störung gemeldet, ändert sich das Fahrverhalten aller Züge schlagartig. Dies wirkt sich augenblicklich auf den Betriebsablauf aus und verändert alle Fahrzeiten. TAROT adressiert dieses Problem dadurch, dass es für jede Störungsart ein eigenes Modell gibt.

In dem zweiten Szenario wird eine dieser Störungsarten gemeldet. Die Streaming-Komponente ruft daraufhin automatisch das passende Vorhersagemodell aus dem Modellspeicher ab und wendet das neue Modell an. Dadurch sind augenblicklich alle Vorhersagen an die veränderte Situation angepasst. In dem Demonstrator wird gezeigt, welche Verspätung mit dem alten Modell vorhergesagt worden wäre, und diese wird der des angepassten Modells gegenübergestellt. Dadurch wird der Vorteil des TAROT-Vorgehens verdeutlicht.

Demo-Szenario 3: Reaktion auf schleichende Veränderungen. Neben Störungen, die augenblicklich einen Effekt auf den Betriebsablauf haben, gibt es allerdings auch kleinere Veränderungen, die sich langsam, aber stetig auf die Fahrzeiten auswirken. So kann sich beispielsweise ein Defekt an der Steckle mit der Zeit verschlechtern, was dazu führt, dass Züge diesen Streckenabschnitt immer langsamer durchfahren müssen.

In dem dritten Szenario wird ein Datenstrom-basierter Vorhersagealgorithmus in der Streaming-Komponente eingesetzt, der Concept Drifts erkennen und das Vorhersagemodell daraufhin anpassen kann. Das angepasste Modell wird nicht nur in der Streaming-Komponente verwendet, sondern ebenfalls im Modellspeicher abgelegt. In dem Demonstrator wird gezeigt, wie diese kleineren Anpassungen die Vorhersagequalität langfristig verbessern.

Demo-Szenario 4: Reaktion auf schlechte Genauigkeit. Trotz dieser dauerhaften Anpassungen an den Vorhersagemodellen kann es dazu kommen, dass die Vorhersagequalität zu schlecht wird, z. B. wenn ein Concept Drift abrupt sehr große Veränderungen herbeiführt. Eine Anpassung, wie sie in Demo-Szenario 3 beschrieben ist, würde zu lange zu falschen Vorhersagen führen und wäre im Fall einer abrupten Veränderung daher zu langsam.

Die Akkuratess der Modelle kann von der Streaming-Komponente automatisch überprüft werden, sobald ein Zug am Zielbahnhof eintrifft. Ist diese zu gering, wird eine Neuberechnung des Modells angestoßen. Hierbei werden nur historische Daten berücksichtigt, die seit dem Auftreten des Concept Drifts angefallen sind. Dies wird im vierten Szenario demonstriert.

Danksagung. Die in diesem Beitrag vorgestellte Forschungsarbeit entstand aus dem PATRON-Forschungsauftrag, der von der Baden-Württemberg Stiftung finanziert wurde.

Literatur

- [BMVI18] Bundesministerium für Verkehr und digitale Infrastruktur: Digitalisierung kommunaler Verkehrssysteme, Förderrichtlinie, BMVI, 18. Jan. 2018.
- [Gi18] Giebler, C.; Stach, C.; Schwarz, H.; Mitschang, B.: BRAID — A Hybrid Processing Architecture for Big Data. In: Proceedings of the 7th International Conference on Data Science, Technology and Applications. S. 294–301, 2018.
- [WN08] Wendler, E.; Nießen, N.: Stochastische Modelle in der Eisenbahnbetriebswissenschaft. ZEVrail 132/1/2, S. 40–48, 2008.

Twoogle: Searching Twitter With MongoDB Queries

Wolfram Wingerath,¹ Felix Gessert,² Norbert Ritter³

Abstract: Modern real-time databases follow the same collection-based querying semantics as traditional database systems. Targeting interactive workloads, real-time databases do not only deliver a query's result upon request, but also produce a continuous stream of informational updates thereafter. In theory, building interactive, reactive, or collaborative applications should thus be simple with collection-based real-time queries as they bridge the gap between traditional database queries over static collections and continuous queries over dynamic data streams. In practice, though, building real-time applications is still considered challenging, since most real-time databases today provide only poor scalability, confusing interfaces for real-time data access, and reduced query expressiveness in comparison to their pull-based counterparts. In this demo, we illustrate that scalability, query expressiveness, and simplicity can go hand-in-hand for modern real-time databases. To this end, we present the social media search app Twoogle which is built on top of Baqend's real-time query API.

Keywords: Real-Time Databases, NoSQL, Scalability, Query Expressiveness, Real-Time Queries, Push-Based Data Access, Self-Maintaining Queries, Event Stream Queries, Search

1 Introduction

Traditional databases are optimized for pull-based queries that make information available only as a response to an explicit client request. While this access pattern is adequate for mostly static domains, it requires inefficient and slow workarounds (e.g. periodic polling) when data is in flow and clients need to stay up-to-date at all times. **Real-time databases** such as Firebase, Meteor, RethinkDB, and Parse address interactive and reactive applications by proactively pushing new information to their clients in realtime, i.e. as soon as new data has been committed to storage. But current implementations are either unscalable or avoid the complex queries that make them so appealing in theory: For example, the original Firebase only supports sorting by a single attribute, while Parse does not support sorted real-time queries at all and RethinkDB only supports sorted real-time queries with limit, but not with an offset clause [WGW⁺18]. In consequence, developers often find themselves evaluating complex performance trade-offs or compensating for missing query expressiveness when building reactive applications on top of a state-of-the-art real-time database.

¹ Baqend, Stresemannstraße 23, 22769 Hamburg, ww@baqend.com

² Baqend, Stresemannstraße 23, 22769 Hamburg, fg@baqend.com

³ Universität Hamburg, DBIS, Vogt-Kölln-Straße 30, 22527 Hamburg, ritter@informatik.uni-hamburg.de

In this demo, we illustrate how the real-time database in production at Baqend⁴ [Win18] combines the query expressiveness of a common NoSQL document store [GWFR16], a great ease-of-development, and push-based data delivery. To this end, we present **Twoogle**, an interactive real-time social media application.

2 Collection-Based Real-Time Queries

There are two distinct types of real-time database queries that differ in the way they expose data to the application: **Event stream queries** simply present the raw change events to the application, so that it can maintain an up-to-date copy of the result or apply custom business logic. **Self-maintaining queries** are more abstract as they do the result maintenance in a transparent manner and provide the client with the complete (updated) query result on every change. Using the latter, reactive behavior can be implemented without any notion of data streams or change events built into the business logic. Some static applications can be transformed into real-time applications simply by switching the underlying query mechanism – without even touching application code.

For query interfaces based on callback functions, the transition between static and real-time behavior is particularly smooth. When a query is executed asynchronously, the main thread of execution does not wait for the result to return. Instead, a callback function is provided at query time to specify how the result should be processed. When it is received, the callback function is invoked with the result as an argument.

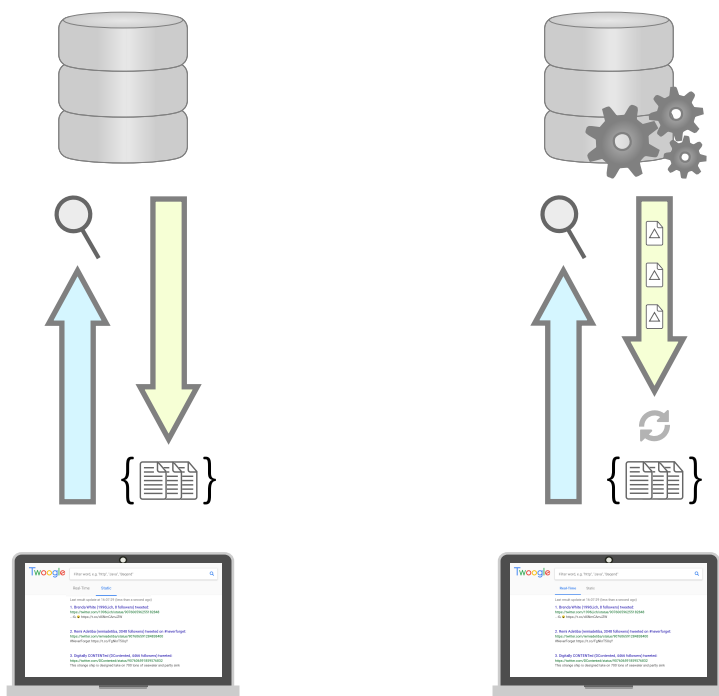
If the query is executed as a static ad hoc query (see Figure 1a), the callback function will only be called once. If the query is executed as a self-maintaining real-time query (see Figure 1b), on the other hand, the callback function will be invoked once when the initial result is returned and then again every time when the underlying data has changed; thus, the application effectively behaves as though a new static query was executed whenever it would return a different result than the last invocation. As the only application requirement, every callback function has to be implemented in idempotent fashion, so that an invocation overrides all effects of the previous invocation. For illustration, consider a search app or website where a callback function renders the result of a user-defined query. If the callback function is executed in the context of a self-maintaining query, it has to remove any result representations that were generated earlier or else the search result might grow indefinitely or might otherwise reflect the real-time updates incorrectly.

3 Demo Outline

As a showcase application, we implemented a social media search app named **Twoogle**⁵ [Win17] that provides an interface for searching a database of Twitter messages that is

⁴ Baqend: <https://www.baqend.com>.

⁵ To see a running version of Twoogle, visit <https://twoogle.info>.



(a) A static ad hoc query produces a single result that represents database state at query time. (b) A self-maintaining query yields a sequence of results, each reflecting the latest update.

Fig. 1: While a static ad hoc query is *pull-based*, a self-maintaining (real-time) query *pushes* updates to the client and presents a new result on every change.

permanently updated. A continuous process running on the Baqend application server receives live tweet messages as they are created by users anywhere on the world, each of which is inserted into the Tweet database collection. Twoogle users can then search all tweets in the continuously evolving database collection by typing a filter expression into the web interface. The currently active real-time search is canceled whenever the search query is updated.

The Twoogle user interface (UI) is depicted in Figure 2. Similar to other search engines, Twoogle displays the result of a user-defined filter query, newest matches first. The result is structured in pages, so that users can retrieve older messages by selecting a later page. A feature that is probably unique to Twoogle is the ability to toggle the **mode of query execution**: Users can choose between push-based real-time queries and pull-based static queries by clicking the corresponding button. When the query is executed in pull-based fashion, the search result is generated on page load and whenever the user modifies the query, for example by typing into the search field or navigating to another page. With a

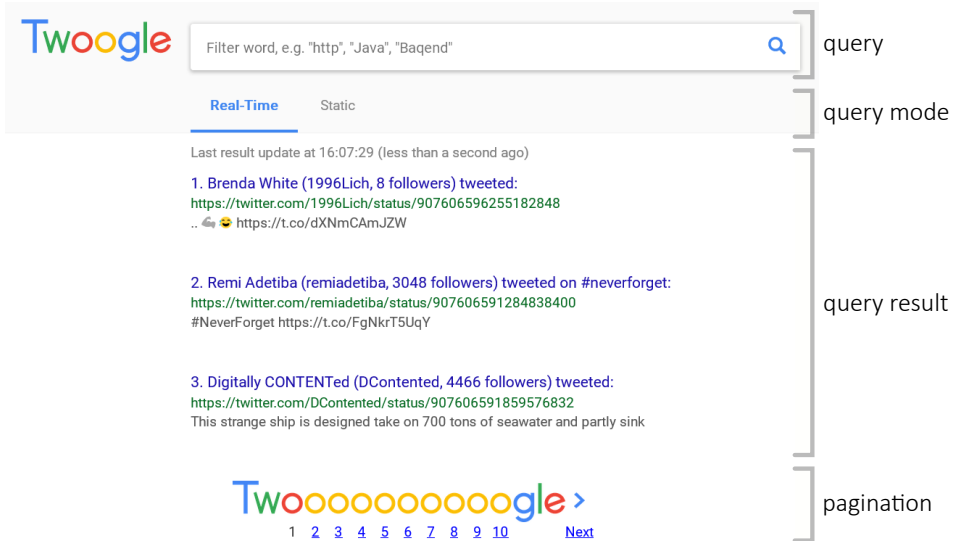


Fig. 2: Twoogle is a social media application that enables user-defined real-time queries over live Twitter messages.

self-maintaining query, in contrast, the user-defined result also refreshes itself whenever a new matching tweet is inserted into the Tweet collection or when a matching tweet is updated or leaves the result.

As part of the demonstration, we will show how self-maintaining queries turn a static version of Twoogle into a real-time version with only three simple lines of code. We will further demonstrate that event stream queries impose additional complexity as they require the developer to handle individual change events, but at the same time also enable more complex query patterns (e.g. real-time aggregations) and sophisticated notification mechanisms.

4 Future Directions

Baqend's real-time query engine currently supports MongoDB semantics for sorted filter queries with `limit` and `offset`, including query operators for content-based filtering through regular expressions (`$regex`), comparisons (e.g. `$eq`, `$ne`, `$gt`, `$gte`), logical combination of filter expressions (e.g. `$and`, `$or`, `$not`), evaluating matching conditions over array values (e.g. `$in`, `$elemMatch`, `$all`, `$size`), and various others (e.g. `$exists`, `$mod`). The current real-time query engine does not support aggregations or joins; prototypical implementations of full-text search and geo queries are being evaluated, but have not been rolled out into production at the time of writing. Extending the Baqend real-time query engine to these query types is work in progress and may therefore be addressed in future publications.

References

- [GWFR16] Felix Gessert, Wolfram Wingerath, Steffen Friedrich, and Norbert Ritter. NoSQL Database Systems: A Survey and Decision Guidance. *Computer Science - Research and Development*, 2016.
- [WGW⁺18] Wolfram Wingerath, Felix Gessert, Erik Witt, Steffen Friedrich, and Norbert Ritter. Real-Time Data Management for Big Data. In *Proceedings of the 21th International Conference on Extending Database Technology, EDBT 2018, Vienna, Austria, March 26-29, 2018*. OpenProceedings.org, 2018.
- [Win17] Wolfram Wingerath. Going Real-Time Has Just Become Easy: Baqend Real-Time Queries Hit Public Beta. *Baqend Tech Blog*, September 2017. Accessed: 2017-09-26.
- [Win18] Wolfram Wingerath. *Scalable Push-Based Real-Time Queries on Top of Pull-Based Databases*. PhD thesis, University of Hamburg, 2018.

Explore FREDDY: Fast Word Embeddings in Database Systems

Michael Günther¹, Zdravko Yanakiev¹, Maik Thiele¹, Wolfgang Lehner¹

Abstract: Word embeddings encode a lot of semantic as well as syntactic features and therefore are useful in many tasks especially in Natural Language Processing and Information Retrieval. FREDDY (Fast woRd EmbedDings Database sYstems), an extended PostgreSQL database system, allowing the user to analyze structured knowledge in the database relations together with unstructured text corpora encoded as word embedding by introducing novel operations for similarity calculation and analogy inference. Approximation techniques support these operations to perform fast similarity computations on high-dimensional vector spaces. This demo allows exploring the powerful query capabilities of FREDDY on different database schemes and a variety of word embeddings generated on different text corpora. From a systems perspective, the user is able to examine the impact of multiple approximation techniques and their parameters for similarity search on query execution time and precision.

Keywords: word embeddings, relational database, k nearest neighbor queries

1 Introduction

Word2vec [Mi13], GloVe [PSM14] and fastText [Bo16], all well-known models to produce word embeddings, are powerful techniques to study the syntactic and semantic relations between words by representing them in a low-dimensional vector. By applying algebraic operations on these vectors semantic relationships such as word analogies, gender-inflections, or geographical relationships can be easily recovered [LG14]. Word embeddings are useful in many tasks in Natural Language Processing and Information Retrieval, such as text mining and classification, sentiment analysis, sentence completion, or dictionary construction. Some recent papers also showed the potential of word embeddings to enable AI capabilities in relational databases [BBS17] and data curation tasks [TTO18].

The goal of the demo is to show how word embeddings could be utilized to augment and enrich the SQL query capabilities, e.g. to compare columns according to their semantic relation or to group rows according to the similarity. For this purpose, we implemented a web application for FREDDY [Gü18, GTL19]. We use pre-trained word embedding models of large text corpora such as Wikipedia but also domain-specific word embeddings that we trained ourselves. By exploiting this external knowledge during query processing we are able to apply inductive reasoning on text values stored in database relations. This is done by

¹ Technische Universität Dresden, Institute of Systems Architecture, Dresden Database Systems Group, Nöthnitzer Straße 46, 01187 Dresden, firstname.lastname@tu-dresden.de

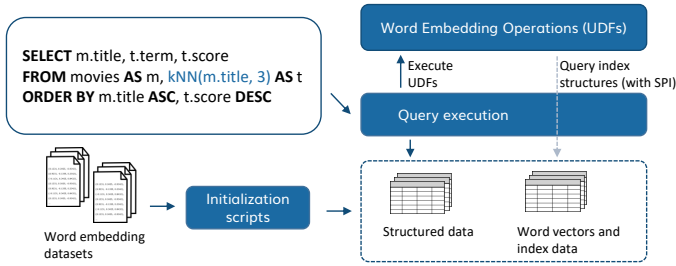


Fig. 1: System Overview

extending the standard SQL syntax of a PostgreSQL database system with novel functions that utilize the word embedding datasets to perform semantic similarity calculations. In this way, it is possible to search for the most similar terms, to group them into categories or to compare semantic relations between terms among each other, e.g. to answer so-called analogy queries. In the context of the IMDB³, FREDDY enables novel query types such as shown in Figure 1, a query that returns the top-3 nearest neighbors (3NN) of each movie in IMDB. Given the movie “Godfather” as input this results in “Scarface”, “Goodfellas” and “Untouchables”. FREDDY also supports semantical grouping, e.g. assigning movies to genres such as *film noir* or *road movie*.

Since most word embedding operations perform *k nearest neighbor search (kNN)* they easily can become a bottleneck for cases where the kNN search is done hundreds of thousands times within an SQL query. Therefore, FREDDY provides fast approximation techniques that exploit the trade-off between execution time and precision.

2 FREDDY: System Overview

FREDDY’s system architecture is sketched in Figure 1. For word embeddings datasets, which should be added, a script creates new relations for the different index structures. For exact distance computations, an additional relation is created storing terms and the respective normalized wordvec vector. In order to make use of these word embeddings within SQL we implemented the User Defined Functions (UDFs) in Figure 2 which operate on the index relations. Moreover, there are further UDFs which serve as helper functions, e.g. for the calculation of centroids for calculating exact cosine similarity values. Search functions like *kNN* provide the possibility of an exact computation as well, but can also be performed in an approximated manner to be applied on large input sets and tables. The UDFs for similarity calculations and search operations are implemented in C whereas interfaces are realized via the procedural script language PL/pgSQL. By using the PostgreSQL Server Programming Interface (SPI), the UDFs are able to run SQL commands inside the functions, e.g. to access the word vectors and index structures. All UDFs are bundled into a PostgreSQL extension.

³ <https://www.imdb.com>

cosine_similarity(token varchar, token varchar): Quantifies the similarity between two tokens

analogy(token_1 varchar, token_2 varchar, token_3 varchar): Solves analogy queries using the *PairDirection*, *3CosADD* or *3CosMul* method [LG14]

analogy_in(token_1 varchar, token_2 varchar, token_3 varchar, output_set varchar[]): Analogy queries with restricted result set

kNN(token varchar, k int): Searches for the k most similar tokens according to the input

kNN_in(token varchar, k int, output_set varchar[]): like kNN but restricting the result to a defined set of output tokens (e.g. to obtain results corresponding to a column in a database relation)

knn_batch(query_set varchar[], k integer): kNN function that runs multiple queries in batches (e.g. for JOIN operations based on similarity)

groups(tokens varchar[], groups varchar[]): Assigns input tokens to groups specified by other tokens according to their similarity

Fig. 2: Word Embedding Operations

3 Demonstration Outline

For this demonstration, we provide a web-client, different database schemas, various word embeddings and a selection of pre-defined queries⁴. Participants are also able to create their own queries. To gain detailed insights on how the different index structures and search functions perform and how their parameters affect result quality and query performance, we provide several widgets to select and adjust them.

Query Interface View The user can choose between different database schemes (1): IMDB, DBLP⁵ and Discogs⁶ music data. In addition, there is a drop-down menu to select different word embedding datasets from a selection including word2vec vectors trained on Google News articles and Wikipedia as well as vectors trained with GloVe[PSM14] on Common Crawl⁷ data. Executing the same query multiple times using different word embeddings leads to different result sets. In general, it is recommended to choose a word embedding dataset which is trained on a related topic according to the database schema. In the text field at (2), the query can be created manually or the users get inspired by one of the pre-defined example queries provided by the drop-down menu above. If a query is executed its result and the response time appears at (3). The demonstrator also keeps track of the previous query and its result. They can be retrieved and compared with the current ones using the tab menu above the result table. In a sidebar (see Figure 3b) the users can choose between the different index structures for similarity search and different analogy query types.

Performance View In a second view illustrated in Figure 3c, the demo user can perform time and precision measurements for kNN and analogy queries using different configurations and compare the results by employing different plots (1). Different visual features (e.g. color, size, ...) encode the index and search parameters. The notations are declared in the legend at (2). To obtain reliable measurements the queries are executed multiple times and the average values for the response time and the precision are obtained. At (3) the number of queries that should be executed and the neighborhood k are specified. The configuration of the search function is defined in the sidebar (Figure 3b) just as in the query view.

⁴ Screencast on our FREDDY website <https://www.db.inf.tu-dresden.de/research-projects/freddy/>

⁵ <https://dblp.uni-trier.de>

⁶ <https://www.discogs.com>

⁷ <http://commoncrawl.org/>

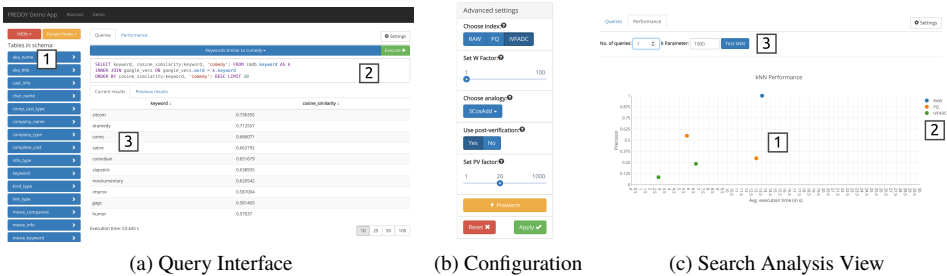


Fig. 3: The web-interface of FREDDY

Acknowledgments

This work is funded by the German Research Foundation (DFG) within the Research Training Group “Role-based Software Infrastructures for continuous-context-sensitive Systems” (GRK 1907) and by the Intel® AI Research.

References

- [BBS17] Bordawekar, Rajesh; Bandyopadhyay, Bortik; Shmueli, Oded: Cognitive Database: A Step towards Endowing Relational Databases with Artificial Intelligence Capabilities. CoRR, abs/1712.07199, 2017.
- [Bo16] Bojanowski, Piotr; Grave, Edouard; Joulin, Armand; Mikolov, Tomas: Enriching Word Vectors with Subword Information. CoRR, abs/1607.04606, 2016.
- [GTL19] Günther, Michael; Thiele, Maik; Lehner, Wolfgang: Fast Approximated Nearest Neighbor Joins For Relational Database Systems. Datenbanksysteme für Business, Technologie und Web (BTW 2019), 2019.
- [Gü18] Günther, Michael: FREDDY: Fast Word Embeddings in Database Systems. In: Proc. of the 2018 International Conference on Management of Data. ACM, pp. 1817–1819, 2018.
- [LG14] Levy, Omer; Goldberg, Yoav: Linguistic regularities in sparse and explicit word representations. In: Proc. of the 18th conference on computational natural language learning. pp. 171–180, 2014.
- [Mi13] Mikolov, Tomas; Sutskever, Ilya; Chen, Kai; Corrado, Greg S; Dean, Jeff: Distributed Representations of Words and Phrases and their Compositionality. In: Advances in Neural Information Processing Systems 26, pp. 3111–3119. Curran Associates, Inc., 2013.
- [PSM14] Pennington, Jeffrey; Socher, Richard; Manning, Christopher D.: GloVe: Global Vectors for Word Representation. In: Empirical Methods in Natural Language Processing (EMNLP). pp. 1532–1543, 2014.
- [TTO18] Thirumuruganathan, Saravanan; Tang, Nan; Ouzzani, Mourad: Data Curation with Deep Learning [Vision]: Towards Self Driving Data Curation. CoRR, abs/1803.01384, 2018.

Information Retrieval for Precision Oncology

Jurica Ševa,¹ Julian Götze,² Mario Lamping,³ Damian Tobias Rieke^{3,4}, Reinhold Schäfer,⁵
Ulf Leser^{1,*}

Abstract: Diagnosis and treatment decisions in cancer increasingly depend on a detailed analysis of the mutational status of a patient's genome. This analysis relies on previously published information regarding the association of variations to disease progression and possible interventions. Clinicians to a large degree use biomedical search engines to obtain such information; however, the vast majority of search results in the common search engines focuses on basic science and is clinically irrelevant. We developed the *Variant-Information Search Tool*, a search engine designed for the targeted search of clinically relevant publications given a mutation profile. VIST indexes all PubMed abstracts, applies advanced text mining to identify mentions of genes and variants and uses machine-learning based scoring to judge the relevancy of documents. Its functionality is available through a fast and intuitive web interface. We also performed a comparative evaluation, showing that VIST's ranking is superior to that of PubMed or vector space models.

1 Introduction

Precision oncology denotes treatment schemes in cancer in which medical decisions depend on the individual molecular status of a patient [Ga13]. The most relevant molecular information is the set of variations (mutations) each individual carries. When faced with the variant profile of a patient, clinicians critically depend on accurate, up-to-date, and detailed information regarding the clinical relevance of the present variations. Finding such information is highly laborious and time-consuming, often taking hours or even days [Do17]. We demonstrate Variant-Information Search Tool (VIST), a search engine specifically developed to aid clinicians in precision oncology in their search for clinically relevant information for a (set of) variations or mutated genes. The core of VIST is its ranking function which, given a (set of) variation or a (set of) gene and a cancer entity, ranks those documents of its corpus highest which contain clinically relevant information. The main difficulty when developing a ranking function for such a novel and quickly emerging field are (a) the lack of gold standard data and (b) the complexity of the concept "clinical relevance", encompassing, among other, information about gene-mutation-drug associations, frequencies of variations within populations, mode of action of drugs and molecular functions. VIST copes with this

¹Department for Computer Science, Humboldt-Universität zu Berlin; ²University Hospital Tübingen;

³Charité – Universitätsmedizin Berlin; ⁴Berlin Institute of Health (BIH); ⁵Deutsches Krebsforschungszentrum; *Corresponding author: leser@informatik.hu-berlin.de.

complexity by using: (1) advanced information extraction to pre-filter documents based on the genes and variations they mention, and (2) machine learning (ML) document classifiers trained on a silver-standard corpus of clinically related documents. VIST furthermore offers several metadata filters (journal, year of publication), highlights key phrases (i.e., the clinically most important sentences) and mentions of query entities when displaying documents, links out to external databases, and allows mixing of entity and classical keyword search. VIST was developed in close interaction with medical experts and is freely available at <https://trriage.informatik.hu-berlin.de:8080/>. It is the first search engine directly targeting clinical relevance of documents which required the integration of ML methods into the ranking. This discerns it technically from other biomedical IR systems, such as GeneView [Th12] or DigSee [Ki13]. The algorithmic problem of finding clinically relevant documents for variation data was also studied in the recent TREC Precision Medicine evaluation [Ro17]. However, the precise task was different from what we target in VIST, as also general medical data and comorbidities of patients were included, which would be very sensitive to implement in a public search engine like VIST.

2 VIST

VIST is a document retrieval system which ranks PubMed abstracts according to their clinical relevance for (a set of) queried variation(s) and/or gene(s) and a cancer entity. When inserted into the index, documents undergo a comprehensive processing pipeline including textual preprocessing, metadata extraction, named entity recognition, classification regarding cancer-relatedness, cancer type, and clinical relevance, and keyphrase detection [Se18]. We detect gene mentions using GNormPlus, variations using tmVar, and drugs using tmChem. All documents and annotations are indexed using Solr. To rank documents against a query, we pre-rank all documents according to two scores: one for their relatedness to cancer in general, and one for clinical relevance. In both cases, we use supervised document classification trained on CIViC [Gr17] and OncoKB [Ch17] with tf-idf weighting. Results are computed by first retrieving all documents containing any of the given variants / genes and filtering for cancer type. Remaining documents are ranked according to a linear combination of "keyword score"(cosine similarity to query), "cancer score"(confidence of the cancer-relatedness classifier), and "clinic score"(confidence of the classifier for clinical relevance).

3 Evaluation

VIST was extensively evaluated to assess and optimize its performance. First, we used a prototype version of VIST to curate a new corpus of clinically (ir-)relevant documents for performing evaluation, resulting in 188 individual scores., of which 119 are used for evaluation; the others were removed due to inconsistent ratings. We assessed the accuracy of the clinical-relevance classifier on this data set using cross-validation. Finally, we used

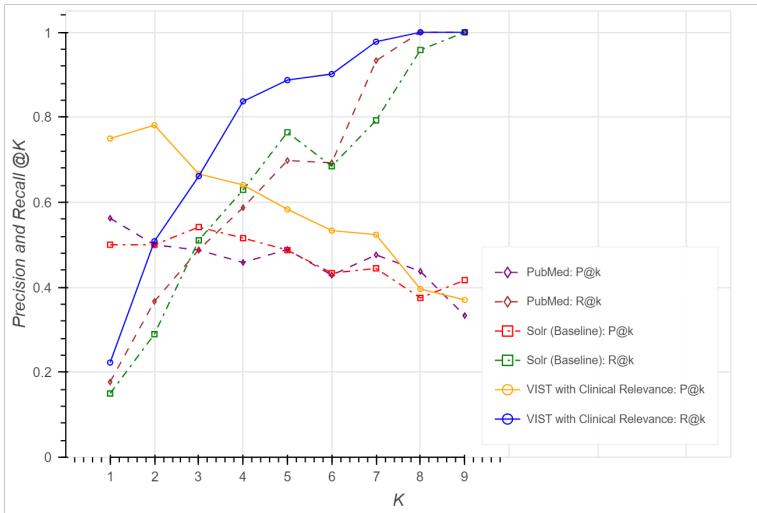


Fig. 1: Precision/Recall @ k averaged over all queries. k denotes the k 'th document in the result set that is also contained in the test set.

this set to compare the ranking performance of VIST with that of PubMed and a pure VSM. Due to lack of space, we here only report on the third evaluation. In this comparison we cannot compare absolute ranks of results as VIST performs result filtering, leading to different result set sizes. Instead, we use two metrics which are robust to different result set sizes. Regarding the ratio of the average rank of all relevant documents from our test data to the average rank of all irrelevant documents, VIST performs best in 11 out of the 16 queries and very close to the best in 5 out of 16 queries. Second, Figure 1 shows average precision@ k and recall@ k for all three systems; therein, k denotes the k 'th document in the ranked result that is also contained in the test set. Clearly, VIST outperforms VSM-ranking and PubMed in both regards.

4 Web Interface and Demonstration

The VIST web interface allows users to define queries and inspect matching documents. Additionally, it offers entity highlighting, various document filters, and a help page. The query shown below is taken from the evaluation queries, also available in the user interface as example query.

Starting a New Search. The initial query is of the format $Q: [keyword(s), gene(s), variant(s)]$. At least one item has to be specified. Matching abstracts are presented in a ranked order based on VIST relevance score. For each document, its title, PMID, publication year and ranking score are shown. The basic interface is shown in Figure 2.

The screenshot shows the VIST web interface. On the left, there is a search interface with filters for Genes, Journals, and Cancer Type. The main panel displays search results in a table with columns for Rank, Title, and Year. The right panel shows a detailed view of a search result, including the title 'Value of a molecular screening program to support clinical trial enrollment in Asian cancer patients: The Integrated Molecular Analysis of Cancer (IMAC) Study', an abstract, and a list of related genes and drugs with their MeSH keywords and relevance scores.

Fig. 2: VIST web interface: Left: Search interface and result overview. Right: Detailed search result with entity and keyphrase highlighting.

Filtering and highlighting of retrieved documents. Enabled as soon as a search yields a non-empty result. VIST allows narrowing returned results by (a) journals, (b) year of publication, and (c) cancer type.

Viewing Document Details. Key sentences and annotated entities are visually highlighted. Key sentences are represented with yellow background with varying transparency levels corresponding to confidence of the detection method. Found genes and drugs are linked to relevant databases (NCBI Genes and DrugBank, respectively). The interface also shows MeSH keywords and a link to the original publication.

References

- [Ch17] Chakravarty, Debyani et al.: OncoKB: A Precision Oncology Knowledge Base. *JCO Precision Oncology*, 1(1):1–16, 2017.
- [Do17] Doig, Kenneth D. et al.: PathOS: a decision support system for reporting high throughput sequencing of cancers in clinical diagnostic laboratories. *Genome Medicine*, 9(1):38, 2017.
- [Ga13] Garraway, Levi A et al.: Precision Oncology: An Overview. *Journal of Clinical Oncology*, 31(15):1803–1805, 2013.
- [Gr17] Griffith, Malachi et al.: CIViC is a community knowledgebase for expert crowdsourcing the clinical interpretation of variants in cancer. *Nature Genetics*, 49(2):170–174, 2017.
- [Ki13] Kim, Jeongkyun et al.: DigSee: disease gene search engine with evidence sentences (version cancer). *Nucleic Acids Research*, 41(W1):W510–W517, 2013.
- [Ro17] Roberts, Kirk et al.: Overview of the TREC 2017 Precision Medicine Track. *TREC*, pp. 1–12, 2017.
- [Še18] Ševa, Jurica et al.: Identifying Key Sentences for Precision Oncology Using Semi-Supervised Learning. In: *Proceedings of the BioNLP 2018 workshop*. pp. 35–46, 2018.
- [Th12] Thomas, Philippe et al.: GeneView: a comprehensive semantic search engine for PubMed. *Nucleic Acids Research*, 40(W1):W585–W591, 2012.

NeMeSys — Energy Adaptive Graph Pattern Matching on NUMA-based Multiprocessor Systems

Alexander Krause,¹ Annett Ungethuen,¹ Thomas Kissinger,¹ Dirk Habich,¹ Wolfgang Lehner¹

Abstract: NEMESYS is a NUMA-aware graph pattern processing engine, which leverages intelligent resource management for energy adaptive processing. With modern server systems incorporating an increasing amount of main memory, we can store graphs and compute analytical graph algorithms like graph pattern matching completely in-memory. Such server systems usually contain several powerful multiprocessors, which come with a high demand for energy. We demonstrate, that graph patterns can be processed in given performance constraints while saving energy, which would be wasted without proper controlling.

Keywords: Graph, Pattern Matching, NUMA, Adaptivity, Energy

1 Introduction

Graph pattern matching is an important, declarative, topology-based querying mechanism and a core primitive in graph analysis. Fundamentally, graph pattern matching is important to many applications such as analyzing hyper-links in the World Wide Web, fraud detection, biomolecular engineering, scientific computing, or social network analytics, only to name a few, as in Krause et al. [Kr17]. The query pattern is usually given as a graph-shaped pattern and the result is a set of matching subgraphs.

On the one hand, the calculation of graph patterns can get prohibitively expensive, because of a possibly high number of intermediate results. On the other hand, modern hardware systems feature main memory capacities of several terabytes, so that we are able to store and process graphs entirely in main memory. Based on that, we have built NEMESYS, a near memory graph pattern processing system being built upon the *Data-Oriented Architecture (DORA)* as used in Kissinger et al. [KHL18] to satisfy high performance demands in an efficient way. However, modern scale-up server systems usually contain several multiprocessors consuming high amounts of energy during data processing. Unfortunately, the energy efficiency of a graph pattern matching system is often not considered, because of the high performance demand for the pattern matching process.

¹ Technische Universität Dresden, Database Systems Group, Noethnitzer Straße 46, 01187 Dresden,
<firstname>.<lastname>@tu-dresden.de

To tackle that issue, we want to demonstrate at the BTW conference, that the energy control findings of Kissinger et al. [KHL18] for relational systems can be also transferred to the substantially more complex graph context. In our demo, we want to emphasize the importance of energy control and the negligible performance impact by using intelligent energy control loops.

2 System Description

From a hardware perspective, the scale-up approach is mainly characterized by the fact that separate memory domains per processor are implemented which are remotely accessible via an interconnect network resulting in a *non-uniform memory access (NUMA)* behavior. To tackle the limiting issues of increased latency and decreased bandwidth when accessing remote memory domains as shown by Kissinger et al. [KHL18], NEMESys leverages well-known DORA and *near-memory computing (NMC)* principles with its basic architecture as portrayed in Figure 1. NMC means, that we limit the scope of each worker to memory domains, which are directly connected to their socket (local instead of remote accesses). Moreover, the DORA approach defines, that only one worker at a time is allowed to touch a certain data partition. This architectural decision implicates data partitioning, which we apply following our guidelines from Krause et al. [Kr17]. During query processing, intermediate states are communicated between the workers with both local and inter-socket messages via the messaging interface, which fetches the correct receiver from the partition manager (cf. Figure 1). To tackle the energy adaptivity, NEMESys reuses the *Energy Control Loop* approach from Kissinger et al. [KHL18], which constantly monitors system utilization and applies appropriate core configurations.

In NEMESys, graph queries can be entered through a user interface, which are then parsed by a query compiler. Every query consists of a chained set of operators, which are derived from the given query pattern. When the query gets executed, every worker in the system

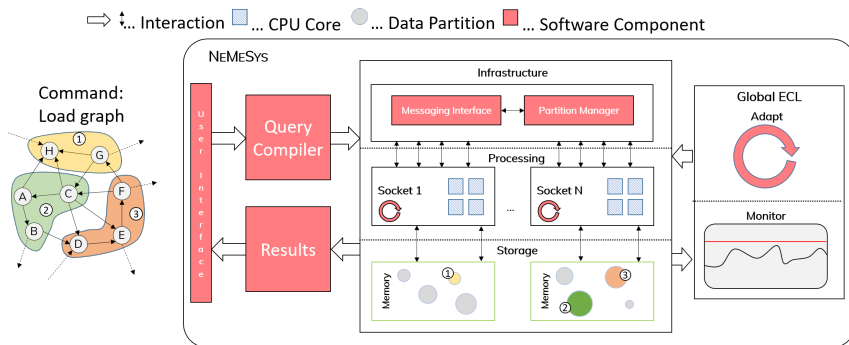


Fig. 1: NEMESys system design



Fig. 2: Operating system governed processing



Fig. 3: Full Demo UI with energy controlled processing

can fork the same operator code and apply it to any data partition on its socket to achieve maximal parallelism. Therefore high data locality is important for minimal inter-socket messages and thus less energy overhead for the network communication. An operator will forward its intermediate matching state to the next operator in the chain, until all messages in the system have been processed.

3 Demo Booth

We show the adaptivity of NeMeSys based on Wikidata², Wikipedias underlying knowledge graph. In contrast to Ungethüm et al. [Un17], our use case imposes increased complexity because of the NP hardness of graph pattern matching and thus demands for more sophisticated execution and adaption mechanisms. Our dataset is based on a filtered truthy statement dump³ with roughly 224M edges. We generated our query set based on the anonymized Wikidata SPARQL query logs⁴. From these logs, we created three different workload profiles for daily, hourly and minute wise query load fluctuation. The used query

² <https://www.wikidata.org/>

³ <https://dumps.wikimedia.org/wikidatawiki/entities/>

⁴ https://iccl.inf.tu-dresden.de/web/Wikidata_SPARQL_Logs/en

set contains approximately 1.2 M different graph pattern and arbitrarily length path queries with sizes ranging between 1 and 52 edges per query.

During the demo, the user can change the load profile and switch between using our energy control loop or letting the operating system take over with the controls at (1) in Figure 3. Our front end shows the arriving queries per second (2) and the graph at (3) displays the systems power draw over time. Switching between the operating system and our controlling mechanisms yields different energy consumption as shown in Figures 2 compared to Figure 3. One of the core components is the chart at (4). It displays the number of active cores per socket and color codes the currently configured frequency per core and socket, where grey means a sleeping CPU, green means a low frequency, yellow medium scaled frequency and red resides in the highest frequencies for that core or socket. The four charts at (5) the Work-Energy Profiles of Ungethüm et al. [Un17] are used to allow for a fine grained system configuration. The system chooses one configuration, such that the queued tasks do not exceed a performance threshold, which we defined as 1 s for this experiment, as shown at (6) with the task counter and speedometer.

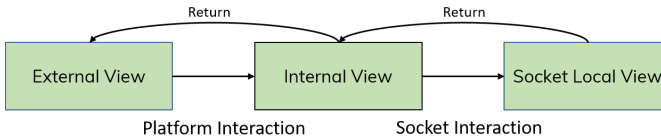


Fig. 4: Virtual Reality interaction guide

Our UI is able to display the most interesting information, however, it lacks the support of understanding the actual underlying hardware architecture. Thus, we visualize the system in an abstracted Virtual Reality (VR) environment, which the user can explore using VR glasses. Demo visitors can interact with the objects in the scene, e.g. dive into the server and select a specific component to display relevant related information. Figure 4 shows the possible user interactions to change scenes. The external view show meta information about the current experiment. Interacting with the shown platform leads the user to a new scene with visualized sockets and interconnects, where the user can choose to display statistics about the elements. The socket local can be reached via interacting with the sockets and yields per-core information.

References

- [KHL18] Kissinger, T.; Habich, D.; Lehner, W.: Adaptive Energy-Control for In-Memory Database Systems. In: SIGMOD. Pp. 351–364, 2018.
- [Kr17] Krause, A.; Kissinger, T.; Habich, D.; Voigt, H.; Lehner, W.: Partitioning Strategy Selection for In-Memory Graph Pattern Matching on Multiprocessor Systems. In: Euro-Par. Pp. 149–163, 2017.

- [Un17] Ungethüm, A.; Kissinger, T.; Mentzel, W.; Mier, E.; Habich, D.; Lehner, W.: Energy Elasticity on Heterogeneous Hardware using Adaptive Resource Reconfiguration. In: BTW. P. 615, 2017.

MAGPIE: A Scalable Data Storage System for Efficient High Volume Data Queries

Thomas Lindemann¹, Patrick Brinkmann², Fadi Dalbah², Christian Hakert², Philipp-Jan Honysz², Daniel Matuszczyk², Nikolas Müller², Alexander Schmulbach², Stefan Petyov Todorinski², Oliver Tüselmann², Shimon Wonsak², Jens Teubner¹

Abstract: Modern challenges in huge sized data storage and querying require new approaches in the field of data storage systems. With *MAGPIE*, we are introducing a hardware-software-co-design, which is efficient in querying data by distributed storage with storage-near pre-processing and designed to be scalable up to large dimensions.

Keywords: data analysis, big data, database, distributed computing, modern hardware

1 Introduction

Beyond the “Big Data” age, data is keeping on growing further year after year in all areas such as business, science and social media. Especially in the scientific field, the data sizes have exceeded all previous data sizes and are therefore often referred to as high-volume data.

The challenge is to develop new storage solutions which can store big amounts of data and make it accessible for fast analyses. With *MAGPIE*, we present our approach to aim the goals of scalable storing huge amounts of partitioned data and allowing efficient parallel distributed pre-processing. Figure 1 shows a comparison between classic database configuration and the approach that *MAGPIE* is based on. To achieve this, we have created a distributed hardware-software-co-design, which consists of SSD storage devices with storage-near embedded systems on the hardware side and a software framework, which allows to pass all operators that are able to process independently on the partitioned data in form of predicates to the storage nodes. Thus, we got a system that is scalable because of arbitrary partitioning on independent storage nodes and allows efficient querying on distributed storages through lightweight embedded systems in the storage layer.

¹ TU Dortmund University, Databases and Information Systems Group, {firstname.lastname}@cs.tu-dortmund.de

² TU Dortmund University, Student Project Group PG614, {firstname.lastname}@tu-dortmund.de

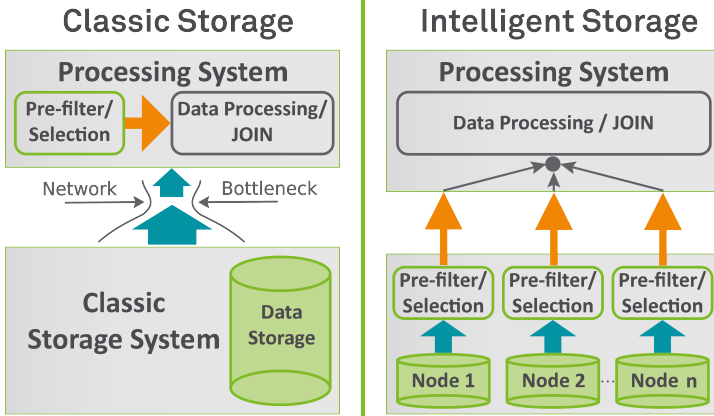


Fig. 1: Differences between the classical configuration and intelligent storage approach of *MAGPIE*

2 Hardware Construction Test Platform

To test our approach, we created a hardware platform in shape of a 2U 19"Blade with integrated embedded systems, SSD storage devices and power supplies with a custom designed integrated power control and energy consumption measurement. The system is by design scalable by just adding more similar blades through a fast network backend. We have used power efficient embedded systems for our evaluation system, because we expect that high performance processors have no advantage for the scans and filter operations. Figure 3 shows the test platform which has been created to make our experiments with the *MAGPIE* Framework on modern embedded hardware.

- AAEON UP2 Embedded Nodes:
 - 32 (8x 4) Cores N4200
 - 8x 2 MB L2 Cache
 - Base Clock: 1.1 GHz
 - Boost Clock: 2.4 GHz
 - 8x GPU Intel HD505
 - 64GB (8x 8GB) LPDDR4
 - Inter-Network: 8x 2 Gbit/s
 - Uplink to Client: 2x 10 GBit
 - Storage: 16x SATA
- SSD Storage:
 - 8 TB (16x 500 GB) SSD
 - Read speed: 16x 550 MB/s
 - Write speed: 16x 520 MB/s
 - Cache: 16x 512MB LPDDR4

Fig. 2: *MAGPIE* System Specifications

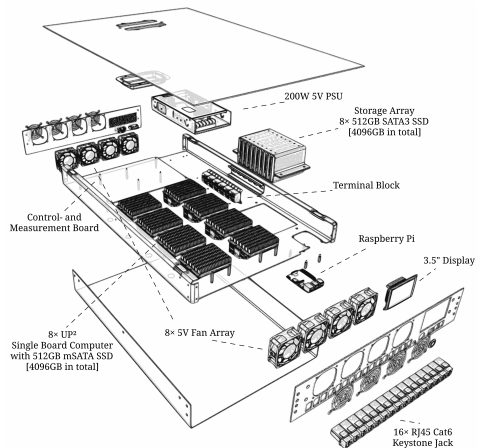


Fig. 3: *MAGPIE* Hardware Construction

3 Storage-sided Distributed Data Pre-processing with *MAGPIE*

There's already a lot of research on distributed storage near data processing done by commercial companies, one of the most popular examples is IBM's PureData System for Analytics, which is based on the rebranded Netezza Architecture. It all comes down to the AMPP (Asymmetrical Massive Parallel Processing) engine. [IB14] A similar approach is shown by Oracle in their Oracle Exadata Database Machines, which has been engineered to be the highest performing and most available platform for running Oracle Database. [Or16] The disadvantage is that these systems are of a commercial nature and not open for research on different processing strategies. An energy-aware approach is shown in [LWA13], the authors introduce IBEX, an FPGA accelerator that allows a number of more complex operators to be pushed down into the storage engine of a database. The result of this is increased performance for certain queries and reduced power consumption. Compared to the FPGA approach of IBEX, *MAGPIE* is much more flexible but still energy-aware.

Figure 4 shows a schematic of the *MAGPIE* Framework. The software concept is designed of three main components, the user node (client), the master worker node (master) and the worker nodes (slaves). The client is supposed to run on the workstation, it is the interface between the user/application querying the database and the *MAGPIE* cluster and receives the query result from the user. Selected operations which are possible to run distributed on all worker nodes are forwarded to the master node. The master is also a worker node itself but keeps track of the tables in a catalog, after doing the lookup which workers have partitions of the tables involved in the sub-queries, it forwards the queries to the respective worker node. It is also possible to maintain the catalog redundantly to avoid having a single point of failure. In theory, every worker node can be a master node in case of a malfunction of the current active master. To avoid the master worker to become a bottleneck, the results of the queries of all worker slaves are sent directly to the user client.

After the worker have completed the scans and filters on all their partitions for the current query id, they send an acknowledge to the master, the master worker collects all the acknowledge messages and forwards one commit to the user client when all slaves completed their tasks for this id. We implemented the scans algorithms, the predicate filter, the buffer manager, the catalogs and the TCP communication from scratch, due to performance optimization.

4 Comparison to state-of-the-art Hardware and Software

To evaluate the performance of the Hardware-Software-Co-Design of *MAGPIE*, we have run similar test workloads on state-of-the-art hardware and common used software. Thus, we executed different test scenarios with a generated 100GB TPC-H dataset on a dual-socket AMD EPYC 7501 32-Core Server with two CPUs, 32 cores each, one thread per core and stored the input data on a ramdisk to avoid a bottleneck with the hard drives.

The comparison software of choice was PostgreSQL 9.6.10 on x86_64-pc-linux-gnu, compiled by gcc (Debian 6.3.0-18+deb9u1) 6.3.0 20170516, 64-bit. We tested different filters and aggregations, as well as a modified TPC-H Q6 scan query. The plots in Figure 5 show the results on *MAGPIE* and the comparison system.

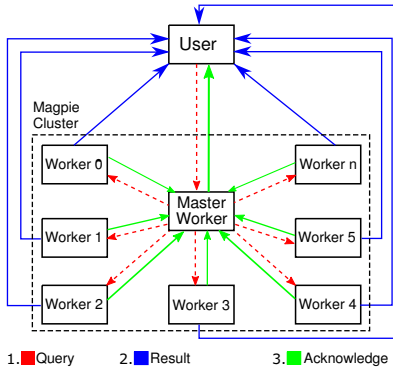


Fig. 4: *MAGPIE* Software Architecture

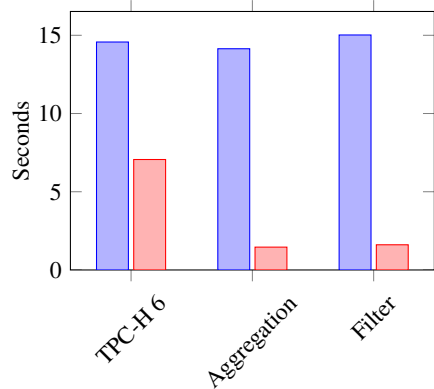


Fig. 5: Execution Time of Queries Comparison on Magpie vs. Reference Server w/ PostgreSQL

5 Demo Outline

For our demo, we are planning to bring along our *MAGPIE* test system as an exhibition piece. In our demo we want to show how *MAGPIE* processes different queries. We can also query a schema listing of the distributed TPC-H data which is stored on the system nodes. Moreover, we created a graphical interface for our demo that allows to execute queries on the *MAGPIE* system interactively and makes an output of the results. For a running query, we will log the system's execution time and power consumption. Furthermore, we want to observe the system load during execution.

References

- [IB14] IBM PureData System for Analytics Architecture, <https://www.redbooks.ibm.com/redpapers/pdfs/redp4725.pdf>.
- [LWA13] Louis Woods, Jens Teubner; Alonso, Gustavo: Less Watts, More Performance: An Intelligent Storage Engine for Data Appliances. Proceedings of the 2013 ACM SIGMOD Conference on Management of Data, April 2013.
- [Or16] Oracle Exadata Database Machine, <https://www.oracle.com/technetwork/database/exadata/exadatatechnicaldeepdive-3518309.pdf>.

DICE: Density-based Interactive Clustering and Exploration

Daniyal Kazempour,¹ Maksim Kazakoy Peer Kröger,¹ Thomas Seidl¹

Abstract: Clustering algorithms are mostly following the pipeline to provide input data, and hyperparameter values. Then the algorithms are executed and the output files are generated or visualized. We provide in our work an early prototype of an interactive density-based clustering tool named DICE in which the users can change the hyperparameter settings and immediately observe the resulting clusters. Further the users can browse through each of the single detected clusters and get statistics regarding as well as a convex hull profile for each cluster. Further DICE keeps track of the chosen settings, enabling the user to review which hyperparameter values have been previously chosen. DICE can not only be used in scientific context of analyzing data, but also in didactic settings in which students can learn in an exploratory fashion how a density-based clustering algorithm like e.g. DBSCAN behaves.

Keywords: Density-based clustering. Interactive. Exploration. Hyperparameters.

1 Introduction

The density based clustering algorithm DBSCAN [Es96] enables the detection of arbitrarily shaped clusters which are density connected. The density is regulated through two hyperparameters, one which represents the neighborhood range of data points, denoted with ε -range and another one which states how many points have to be located within this ε -range, namely *minPts*. With both of the hyperparameters the users can control how dense the clusters shall be which are detected. However, finding a good hyperparameter setting is a tedious task. With a classic DBSCAN implementation one would re-run the algorithm with different parameters, visualize the output and repeat the whole procedure. It may be facilitated by utilizing a method like OPTICS [An99] which generates a plot making it easier to identify suitable hyperparameter settings. A data mining framework like e.g. ELKI [Sc15] provides besides a rich selection of data mining algorithms and index structures, and a high scale of configurability, visualizations and additional information to the discovered clusters. With DICE we aim to deliver a system which gives *immediate* feedback on which the users can intervene, based on provided information, inspecting like e.g. single clusters, or keeping track of changes made. Such interactive tools have been developed like e.g. VISA [As07] in context of subspace clustering. In one of our previous works [Ka18] we presented an interactive tool which makes it possible not only to set one hyperparameter setting for

¹ Ludwig-Maximilians-Universität München, Lehrstuhl für Datenbanksystem und Data Mining, Oettingenstraße 67, 80538, München, {kazempour,kroeger,seidl}@dbs.ifi.lmu.de

the clustering algorithm in the beginning but also set different parameter values at different iteration steps, going back and forth, exploring different outcomes. We aim to target in this work with DICE the interactive clustering in context of density based methods. Our major contributions are: (1) An interactive version of a density-based clustering method, (2) Inspecting single clusters getting statistics and additional information such as convex hulls of a clusters and (3) Keeping track of hyperparameter changes made.

2 DICE - Interaction scheme

DICE abides to the following interaction scheme as seen in Figure 1. First data and initial ε and $minPts$ parameters are provided and the algorithm is executed. To provide a glimpse of what happens behind the interactive user interface, in a initial step all pairwise distances among all points in the data set are computed. This follows the YOCCDO-principle: You Only Compute Distances Once. Based on this distance matrix, filters are applied returning a new matrix where all pairwise entries which have a distance above ε threshold are set to zero, while those being below are set to one, yielding an adjacency matrix. On this matrix the connected components are computed. The users see on the main screen the clusters and outliers with the initially provided parameters. As a reaction the users can now change either one or both of the parameters and are provided with an immediate visualization. This visualization-(re)action cycle can be pursued indefinitely.

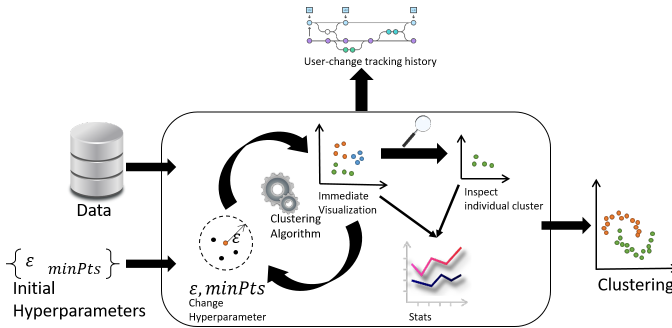


Fig. 1: Interaction scheme of DICE.

On this occasion it may happen that the users loose track of which hyperparameter settings have been explored so far. For this purpose track-records are provided enabling the users to see which e.g. ε and which $minPts$ parameters have been chosen in the previous steps, further it is shown how many clusters emerged at which parameter settings. If the users wish, they can slide through each of the detected clusters being displayed in the main window separately. Here histograms are provided, one which represents on the vertical axis the number of data points and on the horizontal axis, specific distances to other data points. The intention is to show users the distribution of pairwise distances within a cluster. In a much simpler fashion (compared to e.g. OPTICS) it enables to detect further clusters within a cluster, by looking for hills and valleys within the histogram. Besides zooming into

single areas within a cluster, which is provided natively by matplotlib, we thought of another concept to focus on a cluster. In the single-cluster view, users can apply “convex hull” and are provided first with a plot showing all layers of convex hulls of the inspected cluster. Then, by each time clicking on that button, one convex hull and the points belonging to it is removed from the view of that cluster, enabling users to inspect a cluster layer-by-layer. The entire described process scheme can be repeated until a desired clustering of the data is found.

3 Demonstration outline

In this section we elaborate on the “screenplay” of the demo. We first start DICE with a given data set and initial hyperparameter settings. This will open the main window first as seen in Figure 2 (center). Since DICE is an interactive tool, we will make the demo session interactive and strongly encourage the attendants to interact with the tool or to suggest e.g. which hyperparameters to choose. Since we may try out many different hyperparameter settings we will highlight the issue of losing-track which hyperparameters have been chosen. In a second step we reveal the hyperparameter tracking capabilities of DICE as seen in Figure 2 (left and right).

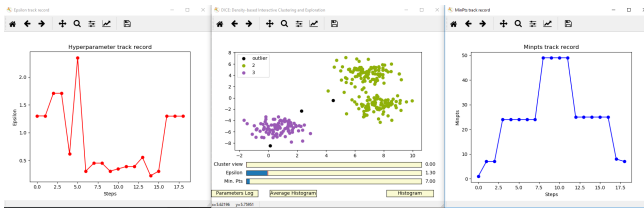


Fig. 2: Main window of DICE (center) with the track record for different ε hyperparameter settings (left) and different $minPts$ hyperparameter settings (right).

In a third step we go deeper into the analysis process by inspecting single clusters more in detail as it can be seen in Figure 3 (left). Choosing a cluster, we inspect it by first looking at the clusters histogram highlighting which information can be obtained (Figure 3 (right)). Further we show that even on single cluster level, we can change the hyperparameters to observe in how far they affect the currently observed cluster. In the fourth and last step we demonstrate how DICE determines the convex hull layers being represented in a separate plot (Figure 3 (center)). By clicking repeatedly on the “Convex Hull” button the participants can observe how like the layers of an onion the convex hulls are removed.

4 Concluding remarks and Outlook

Being aware that DICE is in an early prototype stage, it provides various points of interaction and visualization. DICE enables to inspect single clusters being supported with histograms

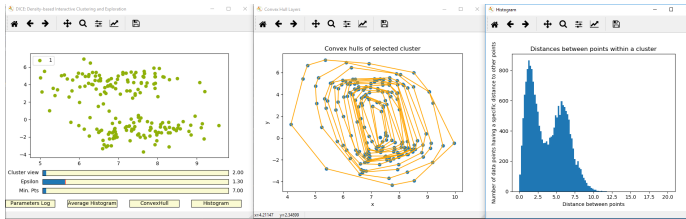


Fig. 3: Visualization of a single cluster (left) with its convex hull layer plot (center) and with a distance plot (right).

on frequencies of pairwise distances. In its current stage it is not suitable for high-volume or high-dimensional data sets. For such further research on the computation schemes are necessary regarding the high-volume aspect. For the high-dimensionality aspect a careful choice of visualization techniques and histograms is required. Beyond the two mentioned aspects different approaches such as minimum spanning trees (MST) for each cluster may be added, since they are (compared to convex hulls) applicable in high-dimensional data. In context of didactic means, DICE is a tool which enables students to analyze data as well as to learn and understand how density based methods work.

Acknowledgement

This work has been funded by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A. The authors of this work take full responsibilities for its content.

References

- [An99] Ankerst, M.; Breunig, M. M.; Kriegel, H.-P.; Sander, J.: OPTICS: Ordering Points to Identify the Clustering Structure. *SIGMOD Rec.* 28/2, pp. 49–60, June 1999, ISSN: 0163-5808.
- [As07] Assent, I.; Krieger, R.; Müller, E.; Seidl, T.: VISA: Visual Subspace Clustering Analysis. *SIGKDD Explor. Newsl.* 9/2, pp. 5–12, Dec. 2007, ISSN: 1931-0145.
- [Es96] Ester, M.; Kriegel, H.-P.; Sander, J.; Xu, X.: A Density-based Algorithm for Discovering Clusters a Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In: *KDD'96*, Portland, Oregon, pp. 226–231, 1996.
- [Ka18] Kazempour, D.; Beer, A.; Lohrer, J.; Kaltenthaler, D.; Seidl, T.: PARADISO: an interactive approach of parameter selection for the mean shift algorithm. In: *SSDBM 2018*, Bozen-Bolzano, Italy, July 09-11, 2018. 26:1–26:4, 2018.
- [Sc15] Schubert, E.; Koos, A.; Emrich, T.; Züfle, A.; Schmid, K. A.; Zimek, A.: A Framework for Clustering Uncertain Data. *PVLDB* 8/12, pp. 1976–1979, 2015.

Processing Large Raster and Vector Data in Apache Spark

Stefan Hagedorn,¹ Timo Räth,¹ Oliver Birli,² Kai-Uwe Sattler¹

Abstract: Spatial data processing frameworks in many cases are limited to vector data only. However, an important type of spatial data is raster data which is produced by sensors on satellites but also by high resolution cameras taking pictures of nano structures, such as chips on wafers. Often the raster data sets become large and need to be processed in parallel on a cluster environment. In this paper we demonstrate our STARK framework with its support for raster data and functionality to combine raster and vector data in filter and join operations. To save engineers from the burden of learning a programming language, queries can be formulated in SQL in a web interface. In the demonstration, users can use this web interface to inspect examples of raster data using our extended SQL queries on a Apache Spark cluster.

1 Introduction

An enormous number of spatial and spatio-temporal data objects is produced every second by thousands or even millions of sources. This data is represented as vector objects (i. e. a single point, a sequence of points that form a line string or polygon) or raster data. Raster data sets are created, e. g. by sensors and cameras where each pixel of the camera sensor is stored as a value in the data set.

Large raster data sets are produced by satellites observing the Earth and by high resolution cameras taking pictures of nano-sized structures – as performed in the Nano Positioning and Measurement Machines (NPMM)³[Ba14; Ha02]. Depending on camera resolution and magnification, the NPMM200 at TU Ilmenau produces data sets up to 17 TB per object, which cannot be handled by single node DBMSs anymore.

For managing and processing raster data special purpose DBMSs have been proposed in the past, e. g. SciDB [Br10] or RasDaMan [Ba98]. Popular platforms for very large data sets are Hadoop MapReduce and Apache Spark. However, since their generic data model is not able to utilize characteristics of spatial objects (neighborhood, distances, etc.), extensions that add spatial *vector*-only data support have been proposed [EM15; YWS16]. Rasterframes⁴ is the only system that supports raster data. The combination of raster and vector data, however, is limited to range queries only.

¹ Technische Universität Ilmenau, Databases & Information Systems, Ilmenau, first.last@tu-ilmenau.de

² Technische Universität Ilmenau, PMS, Ilmenau, first.last@tu-ilmenau.de

³ <https://www.tu-ilmenau.de/cc-npmm/>

⁴ <http://rasterframes.io/>

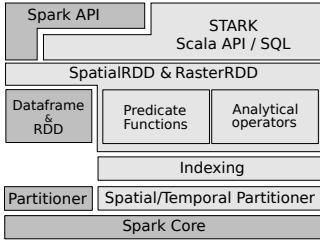


Fig. 1: STARK’s architecture. All components use standard Spark APIs to integrate into the platform.

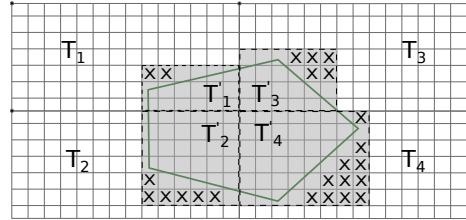


Fig. 2: Result tiles of a combination with a vector object – each with different dimensions. Pixels marked with X mean NULL or NODATA.

Typically, engineers need to transform, filter, and join their raster data sets with additional data. In the scenario of precision measurement at nano scale, a use case is to inspect the values (e. g. depth images of chips on a wafer, micromechanical structures, optical assemblies) to find defects in the components. This requires to apply transformations to the original raster tiles to correct skews. Furthermore, engineers need to manually navigate through the images by selecting areas (rectangles and polygons) of special interest to overlap them with other (reference) images and use it for further processing. Analysis on the images include measurements of geometries and inspection of material properties (white light interferometry or spectroscopic measurements). For this, appropriate functions need to be implemented.

In an example Earth observation use case, a raster data set with temperature measurements has to be joined with the vector data set of country borders, to calculate the average temperature value per country.

In this paper we demonstrate our STARK⁵ framework [HGS17] along with a web interface to interactively explore raster and vector data using SQL and to visualize query results.

2 The STARK Framework

STARK is tightly integrated with Apache Spark by leveraging Scala language features so that users who are familiar with the RDD API can intuitively use STARK’s functions. Besides the Scala API based on the core RDDs, STARK is integrated into SparkSQL and implements SQL functions to filter, join, and aggregate vector and raster data. The overall architecture is shown in Fig. 1. In its core STARK implements two specialized classes that extend Spark’s RDD: SpatialRDD and RasterRDD. A SpatialRDD is used to store spatio-temporal vector data sets, while a RasterRDD represents a raster data set consisting of a set of tiles. A Tile type is a rectangle that stores the actual payload data, the pixels of the raster, in a generic array. In addition to the payload data, a Tile stores some meta data: the coordinates of the upper left corner point as well as the width and height (number of pixels)

⁵ <https://github.com/dbis-ilm/stark/>

of the represented rectangle and the resolution. Vector data objects are represented using a `STObject` class that stores the geometric object (point, polygon, etc.) as well as a temporal component.

A filter operator on a raster data set expects a `STObject` (or a plain rectangle) as query range. As a result, the operator finds all tiles that match the query range according to the provided predicate (intersects, contains, etc.) and returns them as a new `RasterRDD`. The explicit storage of the meta information (position, width, height) in each tile allows for an independent data parallel processing of all tiles. As shown in Fig. 2, a tile may only partially intersect with a polygonal query range and hence, the resulting tiles may be of different extents. This new information is saved into the resulting tiles T'_1, T'_2 etc. The rectangular result tiles of the filter operation represent the minimum bounding rectangle of the intersection with the query range and the input tile. Pixel values in the result tile that do not match the predicate, are set to `NULL` or a `NODATA` constant.

Besides the filter, `RasterRDDs` and `SpatialRDDs` can be joined with another `SpatialRDD`. Similar to the filter, the join operation finds all tiles (or `STObjects`) from the left input that have a join partner in the right input. In `STARK`, tiles are vectorized into rectangles using their stored meta information when their relation with instances of `STObject` has to be computed in filters and joins.

Operators are supported by spatial and temporal partitioners to assign data objects to worker nodes respecting their spatial and temporal neighborhood. Operators additionally benefit from a partition-local spatial or temporal indexing.

The web frontend is specifically designed to let users explore the raster and vector data sets with SQL. Users can define their relations by providing the path in the file system and the schema. The formulation of the spatial filter conditions is aided by a graphical selection tool that displays the spatial component (raster or vector) and lets users draw their region of interest. The SQL query is executed on a Spark cluster using our `STARK` framework. Results of the query can either be displayed in tabular form or as pictures (from raster tiles), on maps (vector objects), or charts. Users can select what and how attributes from the result should be visualized. Image rendering is performed by `STARK` (rather than by the browser) to account for large data sets.

3 Demonstration

We present our prototype designed to support scientists and engineers working with large spatial raster and vector data sets. A screenshot of the web frontend is shown in Fig. 3. During the demonstration users can interact with the web application: We will prepare real high resolution raster data sets captured by the NPMM as well as satellite images. Users can run queries (our predefined or their own ones) that show the various features of the `STARK` framework itself as well as the capabilities of the web interface. The queries

The screenshot displays the STARK web interface. At the top, 'Datasets' are listed: 'hdfs://data/nano/messung1/' pointing to 'wafers' and 'hdfs://data/regions/' pointing to 'regions'. Below is the 'Query' section with a text area containing an SQL query: `SELECT w.tile, r_maxvalue(w.tile) FROM wafers w, regions r WHERE r_intersects(w.tile, r.geo) AND st_contains($refgeo, r.geo)`. A red circle highlights the 'Geo Select' button, with a red arrow pointing to the `$refgeo` variable in the query. To the right is the 'Schema' section showing the structure of 'wafers' (tile: Tile) and 'regions' (id: Int, name: String, geo: STObject). Below the query is the 'Results' section, which includes a 'Table' view showing a thumbnail of a wafer and a 'Visualization' view showing a bar chart of 'r_maxvalue(w.tile)' for tiles 19 to 29. The 'Query Statistics' section on the right indicates 'Progress' at 100%, 'Elapsed Time' of 52 seconds, and 'No. Rows' of 1583252.

Fig. 3: The user interface for querying raster data with SQL. With Geo Select users can use a graphical tool to select a region of interest, referred to in the `$refgeo` variable.

demonstrate STARK's unique feature of combined processing of raster and vector data on the Spark platform whereas the web interface lowers the entrance barrier of creating and submitting jobs to a cluster. Furthermore, users can choose from a list of available charts how results should be visualized and dynamically add them to the web page.

References

- [Ba14] Balzer, G. F.: Entwicklung und Untersuchungen zur 3-D-Nanopositioniertechnik in großen Bewegungsbereichen, PhD thesis, TU Ilmenau, 2014.
- [Ba98] Baumann, P.; Dehmel, A.; Furtado, P.; Ritsch, R.; Widmann, N.: The multidimensional database system RasDaMan. In: SIGMOD Rec. 1998.
- [Br10] Brown, P. G.: Overview of sciDB: Large Scale Array Storage, Processing and Analysis. In: SIGMOD. Pp. 963–968, 2010.
- [EM15] Eldawy, A.; Mokbel, M. F.: SpatialHadoop: A MapReduce Framework for Spatial Data. In: ICDE. Seoul, 2015.
- [Ha02] Hausotte, T.: Nanopositionier- und Nanomessmaschine, PhD thesis, TU Ilmenau, 2002.
- [HGS17] Hagedorn, S.; Goetze, P.; Sattler, K.-U.: The STARK Framework for Spatio-Temporal Data Analytics on Spark. In: BTW. Pp. 123–142, 2017.
- [YWS16] Yu, J.; Wu, J.; Sarwat, M.: A demonstration of GeoSpark: A cluster computing framework for processing big spatial data. In: ICDE. Pp. 1410–1413, 2016.

jHound: Large-Scale Profiling of Open JSON Data

Mark Lukas Möller,¹ Nicolas Berton,² Meike Klettke,¹ Stefanie Scherzinger,³ Uta Störl⁴

Abstract: We present jHound, a tool for profiling large collections of JSON data, and apply it to thousands of data sets holding open government data. jHound reports key characteristics of JSON documents, such as their nesting depth. As we show, jHound can help detect structural outliers, and most importantly, badly encoded documents: jHound can pinpoint certain cases of documents that use string-typed values where other native JSON datatypes would have been a better match. Moreover, we can detect certain cases of maladaptively structured JSON documents, which obviously do not comply with good data modeling practices. By interactively exploring particular example documents, we hope to inspire discussions in the community about what makes a good JSON encoding.

1 Introduction

In an effort towards facilitating government transparency, the German government passed the e-government law in 2013. In particular, this law regulates the provision of *machine readable data* by government institutions. However, a W3C working draft on publishing open government data⁵ recommends to use *human-readable* data formats instead. In this demo, we focus on the data exchange format JSON, since it is both machine and human-readable, and becoming increasingly popular: At the time of writing, the site www.data.gov alone is hosting over 14,000 JSON documents⁶, capturing just about anything from census data to consumer complaints. However, the documents come without a proper schema declaration (such as a JSON Schema description). Many documents even come without *any* meta-data regarding the utmost basics, such as the semantics of property names. For data consumers who intend to programmatically process this data, this poses a serious challenge.

To understand the document structure and to detect bad encoding (more on this later), we introduce jHound, a profiler for large collections of JSON data. While our main motivation is to analyze open government data, our tool is not restricted to this use case.

¹ University of Rostock, mark.moeller2@uni-rostock.de

² ENSEIRB-MATMECA, Bordeaux, France

³ OTH Regensburg, stefanie.scherzinger@oth-regensburg.de

⁴ University of Applied Sciences Darmstadt, uta.stoerl@h-da.de

⁵ Publishing Open Data, <https://www.w3.org/TR/gov-data/>, W3C Working Draft September 2009.

⁶ According to our experience, open government data changes frequently on the hosting sites, due to additions and removals of files. We have noticed that statements regarding the total number of files hosted come with a short expiration date. The number reported here dates back to December 6th, 2018.

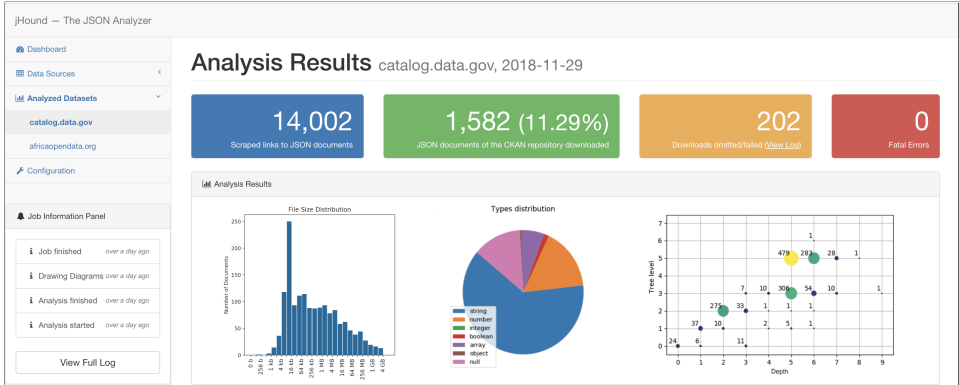


Fig. 1: The web frontend of jHound, showing the analysis results page.

Related Work. Profiling relational data is an established research area (cf. [Ab18] for an overview), and there is a range of established commercial tools. However, tooling for JSON data has not yet matured to the same extent. One active direction of research is schema extraction for JSON document collections (cf. [FSS18]). While this is certainly related, a profiler additionally extracts descriptive statistics such as distributions and outliers.

Contributions and Structure. We next describe the jHound workflow. We highlight some key findings of jHound in real-world data, and conclude with an outline of our demo.

2 Analyzing Open Government Data

To analyze JSON data with jHound, we point the tool to a CKAN repository, `catalog.data.gov` for instance. As seen in the top left of the screenshot in Figure 1, jHound identifies over 14 thousand documents. We used jHound to download a sample with over 1.5 thousand JSON documents. An additional 202 documents could not be downloaded, causing warnings due to timeouts or broken links. We then exclude any documents that have been miscategorized, i.e., they have been labeled as JSON, but do not comply to the format (49 in this scenario).

The results of the analysis are shown in the lower half of Figure 1: (1) jHound reports the distribution of file sizes in a histogram. Exploring the extremes reveals peculiar cases. For instance, among the very small files are documents that merely consist of an empty JSON array. Also, there are files in the gigabyte range. This is a vital information for anybody planning to programmatically process this data, since they need to allocate enough main memory, or choose streaming algorithms. In fact, in our explorations of open government data, we have encountered JSON documents with 60GB in size. (2) A pie chart visualizes the overall distribution of data types among the property values. These types are derived from the encoding syntax, so the value "3.14" is categorized as type `string`, even though it encodes a numerical value. jHound does not yet derive types based on semantic analysis.

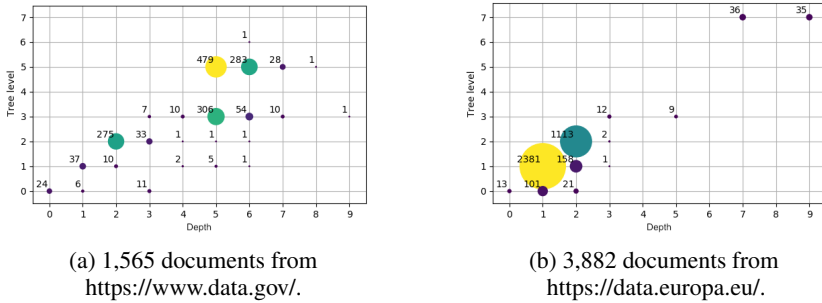


Fig. 2: The maximum document depth versus the level where most of the nodes resides.

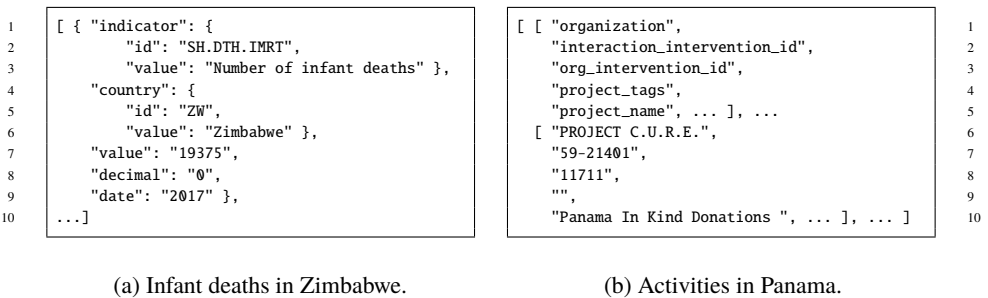


Fig. 3: Excerpts of open data documents provided at <https://data.humdata.org/>.

(3) A bubble chart relates the maximal nesting depth of documents and the tree level at which most of the nodes resides. Figure 2(a) shows the chart in isolation. For instance, there are 24 documents that are not nested, among them structural anomalies, like the documents that merely contain an empty JSON array, and therefore no payload data.

In Figure 2, we compare data from different hosting sites. In Fig. 2(b), the majority of documents has a nesting depth of one. These documents are not nested, but encode flat tuples via key-value pairs. Thus, these documents might have just as well been stored as relational tuples. It is a mere conjecture at this point, but we suspect that these JSON documents were generated from relational data.

Figure 3 shows excerpts of JSON documents with maladaptive structure that we have discovered with jHound: In the left document, all property values are encoded as strings, even though the values in lines 7 through 9 would be more naturally encoded using JSON type number. Similar with the document to the right. Structurally, both documents are unusual. For the left document, we could not find any documentation on the nesting strategy, or on the

meaning of the decimal-property in line 8. This problem of piecing together information is even more striking in the right document, which seems to be a blunt line-by-line translation of CSV data into JSON format. By matching positions in the nested arrays (lines 3 and 8), we painstakingly extract that the intervention id is 11711. It is possible to detect malformed documents by inspecting their structure and property values.

3 Demo Outline

Our demo shows the interactive workflow of analyzing open government data with jHound:

1. We start with documents downloaded from CKAN repositories (e.g. from <https://data.humdata.org/>, <https://www.data.gov/>, and <https://www.govdata.de/>).
2. In dialogue with our audience, we drill down into the analysis results, starting with the analysis results page, as shown in Figure 1:
 - We configure filters for the pie chart showing the type distribution, e.g. considering only JSON documents with over 90% string-typed values.
 - Having identified interesting bubbles in the bubble chart, we can further identify and inspect the raw JSON datasets. In doing so, we come across maladaptive encodings, like those shown in Figure 3.
 - For individual datasets, we can extract a schema description, using an implementation from our earlier work [K117].

4 Conclusion

jHound is a tool for scraping, downloading, and analyzing JSON documents. jHound compiles descriptive statistics, allowing us to systematically explore large collections of JSON documents. We plan to open-source our *python*-implementation of jHound.

Acknowledgements: The article is published in the scope of the project "*NoSQL Schema Evolution und Big Data Migration at Scale*" which is funded by the *Deutsche Forschungsgemeinschaft (DFG)* under the number 385808805.

References

- [Ab18] Abedjan, Ziawasch; Golab, Lukasz; Naumann, Felix; Papenbrock, Thorsten: Data Profiling. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2018.
- [FSS18] Frozza, Angelo Augusto; dos Santos Mello, Ronaldo; de Souza da Costa, Felipe: An Approach for Schema Extraction of JSON and Extended JSON Document Collections. In: Proc. IEEE IRI'18. 2018.
- [K117] Klettke, Meike; Awolin, Hannes; Störl, Uta; Müller, Daniel; Scherzinger, Stefanie: Uncovering the evolution history of data lakes. In: Proc. SCDM'07. 2017.

Big graph analysis by visually created workflows

M. Ali Rostami¹, Eric Peukert¹, Moritz Wilke¹, Erhard Rahm¹

Abstract: The analysis of large graphs has received considerable attention recently but current solutions are typically hard to use. In this demonstration paper, we report on an effort to improve the usability of the open-source system Gradoop for processing and analyzing large graphs. This is achieved by integrating Gradoop into the popular open-source software KNIME to visually create graph analysis workflows, without the need for coding. We outline the integration approach and discuss what will be demonstrated.

Keywords: Graph analysis; Workflow; Gradoop; KNIME; Visualization

1 Introduction

The analysis of big (graph) data becomes increasingly relevant for a variety of fields from bioinformatics to business intelligence. Big data processing frameworks make the computational part of this task possible but mostly require advanced software development and data engineering skills, which many data analysts and scientists lack. Workflow-oriented tools with graphical interfaces promise an easier usage for this group of users but often lack the ability to tackle large amounts of data. Therefore it is important to combine both approaches: user-friendly creation of data analysis workflows with the scalability of a big data processing framework in the background.

Gradoop [Ju15; Ju18] is an open-source framework for the analysis of large graphs that is developed by the Database Group of the University Leipzig and the Competence Center for Scalable Data Services and Solutions (ScaDS) Leipzig. It uses an extension of the flexible property graph model and provides a variety of graph operators and graph mining algorithms. It is build on top of Apache Flink for parallel processing and scalability to large data volumes. KNIME [Ba07], on the other hand, is a well-known open source software for data analysis for the creation and deployment of analytical workflows and applications. KNIME allows users with minimal experience in writing code to select a large set of different operators as nodes from a graphical interface and to combine them on a virtual canvas to define executable workflows.

To achieve both scalability and usability, we developed a KNIME integration for Gradoop operators within the BIGGR project [Ro19]. The aim of this demonstration paper is to show

¹ University of Leipzig, ScaDS Dresden Leipzig, Augustusplatz 10, 04109 Leipzig, Germany, {rostami, peukert, wilke, rahm}@informatik.uni-leipzig.de

how the tools KNIME and Gradoop can be used in combination to load, transform and analyze large graph data and to visualize the results.

2 Gradoop Integration in KNIME

The integration approach makes the GRADOOP operators available as nodes within KNIME and allows either the local or the remote execution of the Gradoop operators and workflows [Ro19]. The majority of real-world data is still stored in column based or unstructured formats, not in a property graph format which is directly readable with GRADOOP. Hence the first task in a graph-based analysis is to transfer given data in a graph format, e.g. to specify which rows in a table are regarded as *vertices* and which relations qualify as *edges*. KNIME supports various ways of reading and processing data and is build around a tabular data representation, so the data preparation step can be tackled with it. However, it requires knowledge about the internal data format of GRADOOP which is a hurdle.

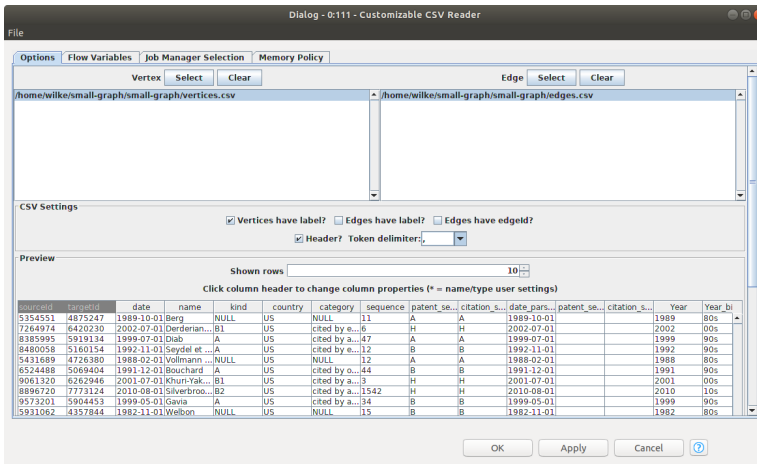


Fig. 1: CSV import: a graph of patent citations is read from a file containing the vertices and a file which contains the source and target id of a citation.

To overcome this, data transformation nodes between GRADOOP and KNIME are necessary. As a first working solution, we provide a Knime node for importing CSV data and transforming into a property graph to start an analysis workflow. Fig. 1 shows the CSV import functionality: multiple files can be chosen and configured to be transformed into sets of vertices or edges. Further efforts will deal with supporting recently proposed operators for graph transformation [KPR19] and to exchange data between the local machine running KNIME and a cluster which stores graph data in a distributed manner.

3 Demonstration

In our demonstration, we show workflows of large graph analytics using KNIME and Gradoop. We consider two datasets: the medium-sized citation networks of research papers and authors as well as a large patent citation graph consisting of 6.6 million US patents and 94.7 million citations between them.²

We build several workflows for the analysis of these large graphs. First, we see how a summary of graphs can be generated as a starting point for further operators. The patent graph needs several steps of preprocessing to obtain a handy set of data. For example, we can first determine connected components and apply further operators on selected components. The resulting graphs from different processes are visualized either completely if possible or in a clustered way. We see how the visualizations give us a first insight into data and help to decide on further analysis steps.

3.1 Research Papers

In this workflow, we first read a publication graph from a JSON data source. After a reduction, the connected components are computed and one of them is visualized. There are two visualization nodes, called **Big Graph View**. For an interactive visualization, large graphs are written out to some format (like CSV) and the path will be given to the view nodes.

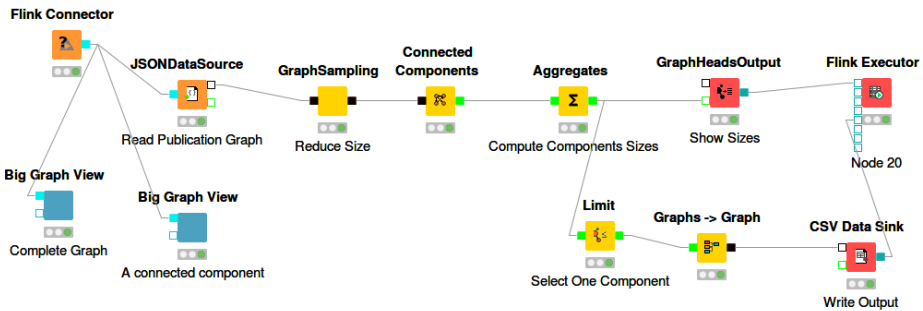


Fig. 2: A Gradoop workflow in KNIME.

3.2 US Patent - Citation Network

Goal of the analysis is to find and collapse so-called patent families. This structure reflects the patent holders not to claim a single patent for a new invention, but to submit several

² <http://www.patentsview.org>

slightly modified patents in order to prevent rivals from adopting the invention easily. These patent families clutter the space of patents, hence clustering them makes further analysis easier because it reduces the size and complexity of the graph.

The workflow uses a GRADOOP matching operator which uses the Cypher[Ju17] query (listing 1) language to detect pairs of patents from the same assignee that have been claimed in the same year and are connected with a citation.

```
MATCH (p1:PATENT)<-[e1:citation]-(p2:PATENT)
          (p1)-[:assignedBy]->(a1:ASSIGNEE)<-[:assignedBy]-(p2)
WHERE e1.years_difference = 0
```

List. 1: Cypher-query to identify citations between patents from the same assignee in the same year.

Afterwards the connected components algorithm is used to combine the pairs into stars that are assumed to be a patent family. Finally the families can be fused in the original graph resulting in only a single vertex representing what was a star before. both graphs (patent families and reduced graph) are stored for further analysis and processing afterwards.

4 Acknowledgements

This work was funded by the German Federal Ministry of Education and Research within the projects BIGGR (BMBF 01IS16030B) and ScaDS Dresden/Leipzig (BMBF 01IS14014B).

References

- [Ba07] Berthold, M. R.; et al.: KNIME: The Konstanz Information Miner. In: Proc. 31st Annual Conference of the Gesellschaft für Klassifikation e.V. Pp. 319–326, 2007.
- [Ju15] Junghanns, M.; Petermann, A.; Gómez, K.; Rahm, E.: Gradoop: scalable graph data management and analytics with Hadoop, 2015, arXiv: 1506.00548.
- [Ju17] Junghanns, M.; Kießling, M.; Averbuch, A.; Petermann, A.; Rahm, E.: Cypher-based graph pattern matching in Gradoop. In: Proc. Fifth International Workshop on Graph Data-management Experiences & Systems. ACM, p. 3, 2017.
- [Ju18] Junghanns, M.; Kiessling, M.; Teichmann, N.; Gómez, K.; Petermann, A.; Rahm, E.: Declarative and distributed graph analytics with GRADOOP. Proceedings of the VLDB Endowment 11/12, pp. 2006–2009, 2018.
- [KPR19] Kricke, M.; Peukert, E.; Rahm, E.: Graph data transformations in GRADOOP. In: Proc. BTW Conference. 2019.

- [Ro19] Rostami, M. A.; Kricke, M.; Peukert, E.; Kuehne, S.; Wilke, M.; Dienst, S.; Rahm, E.: BIGGR: Bringing Gradoop to Applications. *Datenbank-Spektrum* 19/1, to appear, 2019.

When is Harry Potters Birthday? — Question Answering on Linked Data

Nadine Steinmetz¹, Ann-Katrin Arning¹, Kai-Uwe Sattler¹

Abstract: Question Answering (QA) systems provide a user-friendly way to access any type of knowledge base and especially for Linked Data sources to get insight into semantic information. But understanding natural language, transferring it to a formal query and finding the correct answer is a complex task. The challenge is even harder when the QA system aims to be easily adaptable regarding the underlying information. This goal can be achieved by an approach that is independent from the knowledge base. Thereby, the respective data can be replaced or updated without changes on the system itself. With this paper we present our QA approach and the demonstrator which is able to consume natural language questions of general knowledge (not specific to a topic or domain).

1 Motivation

Question answering (QA) is a research discipline at the intersection of natural language processing (NLP), information retrieval, and database processing aiming at answering questions formulated in natural languages. Though, early QA approaches dating back to the sixties, this field has gotten great attention and research made a significant progress over the last few years. Most well-known examples are personal assistants with voice recognition such as Apple Siri, Amazon Alexa or Microsoft Cortana. These systems are able not only to execute spoken commands but also to answer (simple) questions in natural language. However, answering complex questions beyond asking for facts as well as dealing with ambiguities are still challenging problems.

QA systems work usually in a sequence of the following steps: (1) question parsing and focus detection, (2) question classification, (3) query generation, (4) answer candidate generation (query execution), and (5) result ranking. Furthermore, QA systems make use of structured databases / knowledge bases as sources for answers. Here, particularly RDF databases are useful by allowing a schema-agnostic approach where the schema has not to be known for query generation because all facts are represented by triples. In addition, RDF allows to exploit more advanced semantic concepts such as semantic equivalence, similarity or inference mechanisms. Finally, large collections of Linked Data based on a core set such as DBpedia, Wikidata or YAGO represent a great source for answering a wide range of (not only) factoid questions.

¹ TU Ilmenau, Databases & Information Systems Group, Ilmenau, Germany, *first.last@tu-ilmenau.de*

In our work, we focus on the steps of query generation and answer generation by using a *generic* approach. Compared to existing works our query generation approach is independent from the underlying knowledge base. This is done by representing queries through basic graph patterns whose mapping to knowledge base-specific properties or labels are determined at runtime.

In this paper we demonstrate a prototype implementation of our QA system which is able to answer common (but also complex) natural language questions (in English language) on DBpedia as an example knowledge base.

2 System Architecture

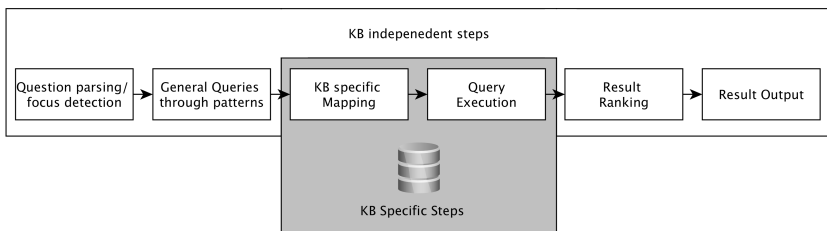


Fig. 1: Overall process of Question Answering

Our approach for question answering consists of seven consecutive steps. Within our algorithm two tasks are dependent on the underlying knowledge base as the concrete properties, entities and ontology classes or categories are requested – as shown in Figure 1. All other steps are independent from the knowledge base.

Question parsing and focus detection. The natural language phrase is analyzed for part-of-speech (POS) and subsequently the question type and the resulting focus are identified.

Generation of general queries with the phrases of the natural language question according to pre-defined patterns. General queries are generated using the extracted phrases from the question and adding the required variables as detected by the previous step.

Mapping subject/predicate/object of the general question. The natural language phrases used for the queries are mapped to the underlying knowledge base to retrieve the actual entities, properties and classes/categories required for the SPARQL query. The general queries are then replaced by executable SPARQL queries using the retrieved resources.

Query execution All generated queries are executed on the underlying knowledge bases and results are retrieved.

Result ranking Each result is scored according to plausibility compared to the identified question type and the expected result type.

Output of the highest ranked SPARQL query and the corresponding result The result of the highest ranked query is provided as assumed correct answer for the question. Our demonstrator also provides all executed queries and the retrieved results together with the computed scores.

Please see [SAS19] for further details on our approach and system architecture.

3 Demonstration

How many children does Jane Fonda have?

ANSWER (1.0)

Focus of the Question: Jane Fonda How Many Have Children

SPARQL: (9)

```
SELECT (COUNT(DISTINCT ?y) as ?x) FROM <http://dbpedia.org> WHERE { <http://dbpedia.org/resource/Jane_Fonda> <http://dbpedia.org/ontology/child> ?y . }
```

The Answer

to your question 'How many children does Jane Fonda have?'

2 (1.0)

Fig. 2: The system frontend showing the question, the extracted key words and the respective answer.

SPARQL: (9)

```
SELECT (COUNT(DISTINCT ?y) as ?x) FROM <http://dbpedia.org> WHERE { <http://dbpedia.org/resource/Jane_Fonda> <http://dbpedia.org/ontology/child> ?y . }
```

```
SELECT (COUNT(DISTINCT ?y) as ?x) FROM <http://dbpedia.org> WHERE { <http://dbpedia.org/resource/Jane_Fonda> <http://dbpedia.org/ontology/child> ?y . }
```

```
SELECT (COUNT(DISTINCT ?z) as ?x) FROM <http://dbpedia.org> WHERE { <http://dbpedia.org/resource/Jane_Fonda> <http://dbpedia.org/ontology/parent> ?z . }
```

```
SELECT (COUNT(DISTINCT ?y) as ?x) FROM <http://dbpedia.org> WHERE { <http://dbpedia.org/resource/Jane_Fonda> <http://dbpedia.org/ontology/parent> ?y . }
```

```
SELECT (COUNT(DISTINCT ?z) as ?x) FROM <http://dbpedia.org> WHERE { <http://dbpedia.org/resource/Jane_Fonda> <http://dbpedia.org/ontology/spouse> ?z . }
```

```
SELECT (COUNT(DISTINCT ?y) as ?x) FROM <http://dbpedia.org> WHERE { <http://dbpedia.org/resource/Jane_Fonda> <http://dbpedia.org/ontology/spouse> ?y . }
```

```
SELECT DISTINCT ?x FROM <http://dbpedia.org> WHERE { <http://dbpedia.org/resource/Jane_Fonda> <http://dbpedia.org/ontology/child> ?x . }
```

```
SELECT DISTINCT ?x FROM <http://dbpedia.org> WHERE { <http://dbpedia.org/resource/Jane_Fonda> <http://dbpedia.org/ontology/parent> ?x . }
```

```
SELECT DISTINCT ?x FROM <http://dbpedia.org> WHERE { <http://dbpedia.org/resource/Jane_Fonda> <http://dbpedia.org/ontology/spouse> ?x . }
```

Fig. 3: The system frontend showing all generated SPARQL queries.

For the demonstration of our system the frontend allows a natural language input and provides the result of the most relevant query including more details about the generated queries and the respective answers. Our demonstrator is able to handle different types of questions, such as “How many”, “Who”, “When”, “Where”, “List all”, “Show me”, etc.

The underlying knowledge base is DBpedia. This allows questions on many different topics without restriction on a specific domain.

Figure 2 shows the frontend when asked for “How many children does Jane Fonda have?” and the respective correct answer. The demonstrator also presents the extracted key words resulting from the first step of analyzing the question.

Next, all generated queries for the input question are shown to the user. In this way the user can comprehend the different mapping results for the extracted key words as well as different operators (e.g. COUNT for the given example of questions asking for “How many”). For the given example, each query returns a result from the knowledge base. Figure 4 shows the frontend presenting all retrieved results and the respective score for each result. For the given example the demonstrator shows that the results with the correct answer type (a number for the question “How many”) received the highest overall score.

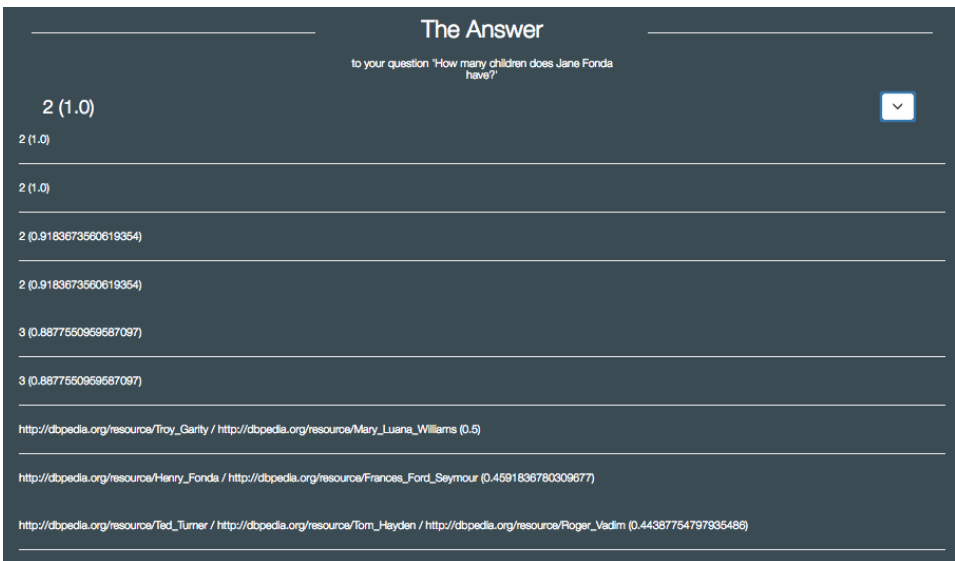


Fig. 4: The system frontend showing all retrieved results and their scores.

References

- [SAS19] Steinmetz, Nadine; Arning, Ann-Katrin; Sattler, Kai-Uwe: From Natural Language Questions to SPARQL Queries: A Pattern-based Approach. In: Datenbanksysteme für Business, Technologie und Web (BTW), 18. Fachtagung des GI-Fachbereichs Datenbanken und Informationssysteme (DBIS), 4.–8.3.2019 in Rostock, Germany. Proceedings. March 2019.

Autorenverzeichnis

A

Allen, David, 377
Arning, Ann-Katrin, 289, 565

B

Bartnik, Adrian, 127
Behrens, Diogo, 361
Beier, Felix, 453
Beltcheva, Olga, 337
Benndorf, Dirk, 507
Berton, Nicolas, 555
Birli, Oliver, 551
Boehm, Matthias, 267
Brehmer, Sven, 507
Bremer, Lars, 419
Brinkmann, Patrick, 543
Broneske, David, 215, 507, 515

C

Campero, Gabriel Durand, 515
Chen, Xiao, 215
Chkalova, Mariya, 419
Crenze, Uwe, 337

D

Dalbah, Fadi, 543
Durand, Gabriel Campero, 215

E

Eggert, Lars, 97
Endres, Markus, 499
Evfimievski, Alexandre, 267

F

Fenske, Wolfram, 507

G

Gessert, Felix, 523
Giebler, Corinna, 519
Glombiewski, Nikolaus, 77
Goetze, Julian, 533
Graefe, Goetz, 77
Gröger, Christoph, 435
Günemann, Stephan, 247
Günther, Michael, 225, 529

H

Habich, Dirk, 35, 537
Hagedorn, Stefan, 551
Hakert, Christian, 543
Hegenbarth, Yvonne, 399
Heyenbrock, Thomas, 247
Heyer, Robert, 507
Hodler, Amy, 377
Honysz, Philipp-Jan, 543
Hoos, Eva, 435
Hunger, Michael, 377

I

Ilyas, Ihab, 27

J

Jahns, Volker, 337
Jugel, Uwe, 361

K

Kastner, Johannes, 499
Kazakov, Maksim, 547
Kazempour, Daniyal, 547
Keil, Jan Martin, 205
Kemper, Alfons, 107, 247, 313
Kessler, Johannes, 503

Kiefer, Cornelia, 149
Kischkel, André, 337
Kissinger, Thomas, 537
Klauck, Stefan, 97
Klettke, Meike, 555
Knebel, Sven, 97
Knobloch, Martin, 377
Krause, Alexander, 537
Kricke, Matthias, 193
Kröger, Peer, 547

L

Lamping, Mario, 533
Lehner, Wolfgang, 35, 225, 529, 537
Leis, Viktor, 313
Leser, Ulf, 533
Li, Yang, 215
Lindemann, Thomas, 543
Lofi, Christoph, 169
Lumpp, Thomas, 359
Lyon, William, 377

M

Markl, Volker, 127
Matuszczyk, Daniel, 543
Mitschang, Bernhard, 149
Möller, Mark Lukas, 555
Monte, Bonaventura Del, 127
Muehlbauer, Tobias, 313
Müller, Jens, 453
Müller, Nikolas, 543

N

Needham, Mark, 377
Neumann, Thomas, 57, 107, 247, 313

O

Oberhofer, Martin, 419
O'Grady, Daniel, 511

Oukid, Ismail, 477

P

Papenbrock, Thorsten, 467
Passing, Linnea, 107
Peukert, Eric, 193, 559
Pietrzyk, Johannes, 35
Pinnecke, Marcus, 507, 515
Plauth, Max, 97

R

Rabl, Tilmann, 127
Radke, Bernhard, 57
Rahm, Erhard, 193, 559
Ranitovic, Nemanja, 499
Reimann, Peter, 149
Reinke, Mark, 337
Reinwald, Berthold, 267
Renkes, Frank, 29
Rieke, Damian Tobias, 533
Rinderle-Ma, Stefanie, 25
Ristow, Gerald, 399
Ritter, Norbert, 523
Rostami, M. Ali, 559

S

Saake, Gunter, 215, 507, 515
Santos, Juan De Dios, 361
Santry, Douglas, 97
Sattler, Kai-Uwe, 289, 551, 565
Sauer, Caetano, 487
Schaefer, Reinhold, 533
Schallert, Kay, 507
Scherzinger, Stefanie, 555
Schmidt, Simone, 519
Schmulbach, Alexander, 543
Schüle, Maximilian, 107, 247
Seeger, Bernhard, 77
Seidl, Thomas, 547

Seva, Jurica, 533
Simonis, Frédéric, 247
Sommer, Christian, 29
Specht, Günther, 503
Stach, Christoph, 519
Steinmetz, Nadine, 289, 565
Stolze, Knut, 453
Störl, Uta, 555
Strobl, Marius, 97

T

Teubner, Jens, 543
Thiele, Maik, 225, 529
Todorinski, Stefan Petyov, 543
Torre, Manuel Valle, 169
Trautmann, Evelyn, 361
Tschuggnall, Michael, 503
Tüselmann, Oliver, 543

U

Ungethüm, Annett, 35, 537

V

Vogelsgesang, Adrian, 313
Voigt, Hannes, 377

W

Waizenegger, Tim, 359
Wilke, Moritz, 559
Wingerath, Wolfram, 523
Wonsak, Shimon, 543

Y

Ye, Mengmeng, 169

Z

Zoun, Roman, 215, 507, 515

GI-Edition Lecture Notes in Informatics

- P-1 Gregor Engels, Andreas Oberweis, Albert Zündorf (Hrsg.): Modellierung 2001.
- P-2 Mikhail Godlevsky, Heinrich C. Mayr (Hrsg.): Information Systems Technology and its Applications, ISTA'2001.
- P-3 Ana M. Moreno, Reind P. van de Riet (Hrsg.): Applications of Natural Language to Information Systems, NLDB'2001.
- P-4 H. Wörn, J. Mühlhng, C. Vahl, H.-P. Meinzer (Hrsg.): Rechner- und sensor-gestützte Chirurgie; Workshop des SFB 414.
- P-5 Andy Schürr (Hg.): OMER – Object-Oriented Modeling of Embedded Real-Time Systems.
- P-6 Hans-Jürgen Appelpath, Rolf Beyer, Uwe Marquardt, Heinrich C. Mayr, Claudia Steinberger (Hrsg.): Unternehmen Hochschule, UH'2001.
- P-7 Andy Evans, Robert France, Ana Moreira, Bernhard Rumpe (Hrsg.): Practical UML-Based Rigorous Development Methods – Countering or Integrating the extremists, pUML'2001.
- P-8 Reinhard Keil-Slawik, Johannes Magenheim (Hrsg.): Informatikunterricht und Medienbildung, INFOS'2001.
- P-9 Jan von Knop, Wilhelm Haverkamp (Hrsg.): Innovative Anwendungen in Kommunikationsnetzen, 15. DFN Arbeitstagung.
- P-10 Mirjam Minor, Steffen Staab (Hrsg.): 1st German Workshop on Experience Management: Sharing Experiences about the Sharing Experience.
- P-11 Michael Weber, Frank Kargl (Hrsg.): Mobile Ad-Hoc Netzwerke, WMAN 2002.
- P-12 Martin Glinz, Günther Müller-Luschnat (Hrsg.): Modellierung 2002.
- P-13 Jan von Knop, Peter Schirmbacher and Viljan Mahni_ (Hrsg.): The Changing Universities – The Role of Technology.
- P-14 Robert Tolksdorf, Rainer Eckstein (Hrsg.): XML-Technologien für das Semantic Web – XSW 2002.
- P-15 Hans-Bernd Bludau, Andreas Koop (Hrsg.): Mobile Computing in Medicine.
- P-16 J. Felix Hampe, Gerhard Schwabe (Hrsg.): Mobile and Collaborative Business 2002.
- P-17 Jan von Knop, Wilhelm Haverkamp (Hrsg.): Zukunft der Netze –Die Verletzbarkeit meistern, 16. DFN Arbeitstagung.
- P-18 Elmar J. Sinz, Markus Plaha (Hrsg.): Modellierung betrieblicher Informationssysteme – MobIS 2002.
- P-19 Sigrid Schubert, Bernd Reusch, Norbert Jesse (Hrsg.): Informatik bewegt – Informatik 2002 – 32. Jahrestagung der Gesellschaft für Informatik e.V. (GI) 30.Sept.-3. Okt. 2002 in Dortmund.
- P-20 Sigrid Schubert, Bernd Reusch, Norbert Jesse (Hrsg.): Informatik bewegt – Informatik 2002 – 32. Jahrestagung der Gesellschaft für Informatik e.V. (GI) 30.Sept.-3. Okt. 2002 in Dortmund (Ergänzungsband).
- P-21 Jörg Desel, Mathias Weske (Hrsg.): Promise 2002: Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen.
- P-22 Sigrid Schubert, Johannes Magenheim, Peter Hubwieser, Torsten Brinda (Hrsg.): Forschungsbeiträge zur "Didaktik der Informatik" – Theorie, Praxis, Evaluation.
- P-23 Thorsten Spitta, Jens Borchers, Harry M. Sneed (Hrsg.): Software Management 2002 – Fortschritt durch Beständigkeit
- P-24 Rainer Eckstein, Robert Tolksdorf (Hrsg.): XMIDX 2003 – XML-Technologien für Middleware – Middleware für XML-Anwendungen
- P-25 Key Pousttchi, Klaus Turowski (Hrsg.): Mobile Commerce – Anwendungen und Perspektiven – 3. Workshop Mobile Commerce, Universität Augsburg, 04.02.2003
- P-26 Gerhard Weikum, Harald Schöning, Erhard Rahm (Hrsg.): BTW 2003: Datenbanksysteme für Business, Technologie und Web
- P-27 Michael Kroll, Hans-Gerd Lipinski, Kay Melzer (Hrsg.): Mobiles Computing in der Medizin
- P-28 Ulrich Reimer, Andreas Abecker, Steffen Staab, Gerd Stumme (Hrsg.): WM 2003: Professionelles Wissensmanagement – Erfahrungen und Visionen
- P-29 Antje Düsterhöft, Bernhard Thalheim (Eds.): NLDB'2003: Natural Language Processing and Information Systems
- P-30 Mikhail Godlevsky, Stephen Liddle, Heinrich C. Mayr (Eds.): Information Systems Technology and its Applications
- P-31 Arslan Brömme, Christoph Busch (Eds.): BIOSIG 2003: Biometrics and Electronic Signatures

- P-32 Peter Hubwieser (Hrsg.): Informatische Fachkonzepte im Unterricht – INFOS 2003
- P-33 Andreas Geyer-Schulz, Alfred Taudes (Hrsg.): Informationswirtschaft: Ein Sektor mit Zukunft
- P-34 Klaus Dittrich, Wolfgang König, Andreas Oberweis, Kai Rannenber, Wolfgang Wahlster (Hrsg.): Informatik 2003 – Innovative Informatikanwendungen (Band 1)
- P-35 Klaus Dittrich, Wolfgang König, Andreas Oberweis, Kai Rannenber, Wolfgang Wahlster (Hrsg.): Informatik 2003 – Innovative Informatikanwendungen (Band 2)
- P-36 Rüdiger Grimm, Hubert B. Keller, Kai Rannenber (Hrsg.): Informatik 2003 – Mit Sicherheit Informatik
- P-37 Arndt Bode, Jörg Desel, Sabine Rathmayer, Martin Wessner (Hrsg.): DeLFI 2003: e-Learning Fachtagung Informatik
- P-38 E.J. Sinz, M. Plaha, P. Neckel (Hrsg.): Modellierung betrieblicher Informationssysteme – MobIS 2003
- P-39 Jens Nedon, Sandra Frings, Oliver Göbel (Hrsg.): IT-Incident Management & IT-Forensics – IMF 2003
- P-40 Michael Rebstock (Hrsg.): Modellierung betrieblicher Informationssysteme – MobIS 2004
- P-41 Uwe Brinkschulte, Jürgen Becker, Dietmar Fey, Karl-Erwin Großpietsch, Christian Hochberger, Erik Maehle, Thomas Runkler (Edts.): ARCS 2004 – Organic and Pervasive Computing
- P-42 Key Pousttchi, Klaus Turowski (Hrsg.): Mobile Economy – Transaktionen und Prozesse, Anwendungen und Dienste
- P-43 Birgitta König-Ries, Michael Klein, Philipp Obreiter (Hrsg.): Persistence, Scalability, Transactions – Database Mechanisms for Mobile Applications
- P-44 Jan von Knop, Wilhelm Haverkamp, Eike Jessen (Hrsg.): Security, E-Learning, E-Services
- P-45 Bernhard Rumpe, Wolfgang Hesse (Hrsg.): Modellierung 2004
- P-46 Ulrich Flegel, Michael Meier (Hrsg.): Detection of Intrusions of Malware & Vulnerability Assessment
- P-47 Alexander Prosser, Robert Krimmer (Hrsg.): Electronic Voting in Europe – Technology, Law, Politics and Society
- P-48 Anatoly Doroshenko, Terry Halpin, Stephen W. Liddle, Heinrich C. Mayr (Hrsg.): Information Systems Technology and its Applications
- P-49 G. Schiefer, P. Wagner, M. Morgenstern, U. Rickert (Hrsg.): Integration und Datensicherheit – Anforderungen, Konflikte und Perspektiven
- P-50 Peter Dadam, Manfred Reichert (Hrsg.): INFORMATIK 2004 – Informatik verbindet (Band 1) Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 20.-24. September 2004 in Ulm
- P-51 Peter Dadam, Manfred Reichert (Hrsg.): INFORMATIK 2004 – Informatik verbindet (Band 2) Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 20.-24. September 2004 in Ulm
- P-52 Gregor Engels, Silke Seehusen (Hrsg.): DELFI 2004 – Tagungsband der 2. e-Learning Fachtagung Informatik
- P-53 Robert Giegerich, Jens Stoye (Hrsg.): German Conference on Bioinformatics – GCB 2004
- P-54 Jens Borchers, Ralf Kneuper (Hrsg.): Softwaremanagement 2004 – Outsourcing und Integration
- P-55 Jan von Knop, Wilhelm Haverkamp, Eike Jessen (Hrsg.): E-Science und Grid Ad-hoc-Netze Medienintegration
- P-56 Fernand Feltz, Andreas Oberweis, Benoit Otjacques (Hrsg.): EMISA 2004 – Informationssysteme im E-Business und E-Government
- P-57 Klaus Turowski (Hrsg.): Architekturen, Komponenten, Anwendungen
- P-58 Sami Beydeda, Volker Gruhn, Johannes Mayer, Ralf Reussner, Franz Schweiggert (Hrsg.): Testing of Component-Based Systems and Software Quality
- P-59 J. Felix Hampe, Franz Lehner, Key Pousttchi, Kai Rannenber, Klaus Turowski (Hrsg.): Mobile Business – Processes, Platforms, Payments
- P-60 Steffen Friedrich (Hrsg.): Unterrichtskonzepte für informatische Bildung
- P-61 Paul Müller, Reinhard Gotzhein, Jens B. Schmitt (Hrsg.): Kommunikation in verteilten Systemen
- P-62 Federrath, Hannes (Hrsg.): „Sicherheit 2005“ – Sicherheit – Schutz und Zuverlässigkeit
- P-63 Roland Kaschek, Heinrich C. Mayr, Stephen Liddle (Hrsg.): Information Systems – Technology and its Applications

- P-64 Peter Liggesmeyer, Klaus Pohl, Michael Goedicke (Hrsg.): Software Engineering 2005
- P-65 Gottfried Vossen, Frank Leymann, Peter Lockemann, Wolfrid Stucky (Hrsg.): Datenbanksysteme in Business, Technologie und Web
- P-66 Jörg M. Haake, Ulrike Lucke, Djamshid Tavangarian (Hrsg.): DeLFI 2005: 3. deutsche e-Learning Fachtagung Informatik
- P-67 Armin B. Cremers, Rainer Manthey, Peter Martini, Volker Steinhage (Hrsg.): INFORMATIK 2005 – Informatik LIVE (Band 1)
- P-68 Armin B. Cremers, Rainer Manthey, Peter Martini, Volker Steinhage (Hrsg.): INFORMATIK 2005 – Informatik LIVE (Band 2)
- P-69 Robert Hirschfeld, Ryszard Kowalczyk, Andreas Polze, Matthias Weske (Hrsg.): NODe 2005, GSEM 2005
- P-70 Klaus Turowski, Johannes-Maria Zaha (Hrsg.): Component-oriented Enterprise Application (COAE 2005)
- P-71 Andrew Torda, Stefan Kurz, Matthias Rarey (Hrsg.): German Conference on Bioinformatics 2005
- P-72 Klaus P. Jantke, Klaus-Peter Fähnrich, Wolfgang S. Wittig (Hrsg.): Marktplatz Internet: Von e-Learning bis e-Payment
- P-73 Jan von Knop, Wilhelm Haverkamp, Eike Jessen (Hrsg.): "Heute schon das Morgen sehen"
- P-74 Christopher Wolf, Stefan Lucks, Po-Wah Yau (Hrsg.): WEWoRC 2005 – Western European Workshop on Research in Cryptology
- P-75 Jörg Desel, Ulrich Frank (Hrsg.): Enterprise Modelling and Information Systems Architecture
- P-76 Thomas Kirste, Birgitta König-Riess, Key Pousttchi, Klaus Turowski (Hrsg.): Mobile Informationssysteme – Potentiale, Hindernisse, Einsatz
- P-77 Jana Dittmann (Hrsg.): SICHERHEIT 2006
- P-78 K.-O. Wenkel, P. Wagner, M. Morgens-tern, K. Luzi, P. Eisermann (Hrsg.): Land- und Ernährungswirtschaft im Wandel
- P-79 Bettina Biel, Matthias Book, Volker Gruhn (Hrsg.): Softwareengineering 2006
- P-80 Mareike Schoop, Christian Huemer, Michael Rebstock, Martin Bichler (Hrsg.): Service-Oriented Electronic Commerce
- P-81 Wolfgang Karl, Jürgen Becker, Karl-Erwin Großpietsch, Christian Hochberger, Erik Maehle (Hrsg.): ARCS'06
- P-82 Heinrich C. Mayr, Ruth Breu (Hrsg.): Modellierung 2006
- P-83 Daniel Huson, Oliver Kohlbacher, Andrei Lupas, Kay Nieselt and Andreas Zell (eds.): German Conference on Bioinformatics
- P-84 Dimitris Karagiannis, Heinrich C. Mayr, (Hrsg.): Information Systems Technology and its Applications
- P-85 Witold Abramowicz, Heinrich C. Mayr, (Hrsg.): Business Information Systems
- P-86 Robert Krimmer (Ed.): Electronic Voting 2006
- P-87 Max Mühlhäuser, Guido Rößling, Ralf Steinmetz (Hrsg.): DELFI 2006: 4. e-Learning Fachtagung Informatik
- P-88 Robert Hirschfeld, Andreas Polze, Ryszard Kowalczyk (Hrsg.): NODe 2006, GSEM 2006
- P-90 Joachim Schelp, Robert Winter, Ulrich Frank, Bodo Rieger, Klaus Turowski (Hrsg.): Integration, Informationslogistik und Architektur
- P-91 Henrik Stormer, Andreas Meier, Michael Schumacher (Eds.): European Conference on eHealth 2006
- P-92 Fernand Feltz, Benoît Otjacques, Andreas Oberweis, Nicolas Poussing (Eds.): AIM 2006
- P-93 Christian Hochberger, Rüdiger Liskowsky (Eds.): INFORMATIK 2006 – Informatik für Menschen, Band 1
- P-94 Christian Hochberger, Rüdiger Liskowsky (Eds.): INFORMATIK 2006 – Informatik für Menschen, Band 2
- P-95 Matthias Weske, Markus Nüttgens (Eds.): EMISA 2005: Methoden, Konzepte und Technologien für die Entwicklung von dienstbasierten Informationssystemen
- P-96 Saartje Brockmans, Jürgen Jung, York Sure (Eds.): Meta-Modelling and Ontologies
- P-97 Oliver Göbel, Dirk Schadt, Sandra Frings, Hardo Hase, Detlef Günther, Jens Nedon (Eds.): IT-Incident Mangament & IT-Forensics – IMF 2006

- P-98 Hans Brandt-Pook, Werner Simonsmeier und Thorsten Spitta (Hrsg.): Beratung in der Softwareentwicklung – Modelle, Methoden, Best Practices
- P-99 Andreas Schwill, Carsten Schulte, Marco Thomas (Hrsg.): Didaktik der Informatik
- P-100 Peter Forbrig, Günter Siegel, Markus Schneider (Hrsg.): HDI 2006: Hochschuldidaktik der Informatik
- P-101 Stefan Böttinger, Ludwig Theuvsen, Susanne Rank, Marlies Morgenstern (Hrsg.): Agrarinformatik im Spannungsfeld zwischen Regionalisierung und globalen Wertschöpfungsketten
- P-102 Otto Spaniol (Eds.): Mobile Services and Personalized Environments
- P-103 Alfons Kemper, Harald Schöning, Thomas Rose, Matthias Jarke, Thomas Seidl, Christoph Quix, Christoph Brochhaus (Hrsg.): Datenbanksysteme in Business, Technologie und Web (BTW 2007)
- P-104 Birgitta König-Ries, Franz Lehner, Rainer Malaka, Can Türker (Hrsg.) MMS 2007: Mobilität und mobile Informationssysteme
- P-105 Wolf-Gideon Bleek, Jörg Raasch, Heinz Züllighoven (Hrsg.) Software Engineering 2007
- P-106 Wolf-Gideon Bleek, Henning Schwentner, Heinz Züllighoven (Hrsg.) Software Engineering 2007 – Beiträge zu den Workshops
- P-107 Heinrich C. Mayr, Dimitris Karagiannis (eds.) Information Systems Technology and its Applications
- P-108 Arslan Brömme, Christoph Busch, Detlef Hühnlein (eds.) BIOSIG 2007: Biometrics and Electronic Signatures
- P-109 Rainer Koschke, Otthein Herzog, Karl-Heinz Rödiger, Marc Ronthaler (Hrsg.) INFORMATIK 2007 Informatik trifft Logistik Band 1
- P-110 Rainer Koschke, Otthein Herzog, Karl-Heinz Rödiger, Marc Ronthaler (Hrsg.) INFORMATIK 2007 Informatik trifft Logistik Band 2
- P-111 Christian Eibl, Johannes Magenheimer, Sigrid Schubert, Martin Wessner (Hrsg.) DeLFI 2007: 5. e-Learning Fachtagung Informatik
- P-112 Sigrid Schubert (Hrsg.) Didaktik der Informatik in Theorie und Praxis
- P-113 Sören Auer, Christian Bizer, Claudia Müller, Anna V. Zhdanova (Eds.) The Social Semantic Web 2007 Proceedings of the 1st Conference on Social Semantic Web (CSSW)
- P-114 Sandra Frings, Oliver Göbel, Detlef Günther, Hardo G. Hase, Jens Nedon, Dirk Schadt, Arslan Brömme (Eds.) IMF2007 IT-incident management & IT-forensics Proceedings of the 3rd International Conference on IT-Incident Management & IT-Forensics
- P-115 Claudia Falter, Alexander Schliep, Joachim Selbig, Martin Vingron and Dirk Walthert (Eds.) German conference on bioinformatics GCB 2007
- P-116 Witold Abramowicz, Leszek Maciszek (Eds.) Business Process and Services Computing 1st International Working Conference on Business Process and Services Computing BPSC 2007
- P-117 Ryszard Kowalczyk (Ed.) Grid service engineering and management The 4th International Conference on Grid Service Engineering and Management GSEM 2007
- P-118 Andreas Hein, Wilfried Thoben, Hans-Jürgen Appelrath, Peter Jensch (Eds.) European Conference on ehealth 2007
- P-119 Manfred Reichert, Stefan Strecker, Klaus Turowski (Eds.) Enterprise Modelling and Information Systems Architectures Concepts and Applications
- P-120 Adam Pawlak, Kurt Sandkuhl, Wojciech Cholewa, Leandro Soares Indrusiak (Eds.) Coordination of Collaborative Engineering - State of the Art and Future Challenges
- P-121 Korbinian Herrmann, Bernd Bruegge (Hrsg.) Software Engineering 2008 Fachtagung des GI-Fachbereichs Softwaretechnik
- P-122 Walid Maalej, Bernd Bruegge (Hrsg.) Software Engineering 2008 - Workshopband Fachtagung des GI-Fachbereichs Softwaretechnik

- P-123 Michael H. Breitner, Martin Breunig, Elgar Fleisch, Ley Pousttchi, Klaus Turowski (Hrsg.)
Mobile und Ubiquitäre Informationssysteme – Technologien, Prozesse, Marktfähigkeit
Proceedings zur 3. Konferenz Mobile und Ubiquitäre Informationssysteme (MMS 2008)
- P-124 Wolfgang E. Nagel, Rolf Hoffmann, Andreas Koch (Eds.)
9th Workshop on Parallel Systems and Algorithms (PASA)
Workshop of the GI/ITG Special Interest Groups PARS and PARVA
- P-125 Rolf A.E. Müller, Hans-H. Sundermeier, Ludwig Theuvsen, Stephanie Schütze, Marlies Morgenstern (Hrsg.)
Unternehmens-IT: Führungsinstrument oder Verwaltungsbürde
Referate der 28. GIL Jahrestagung
- P-126 Rainer Gimnich, Uwe Kaiser, Jochen Quante, Andreas Winter (Hrsg.)
10th Workshop Software Reengineering (WSR 2008)
- P-127 Thomas Kühne, Wolfgang Reisig, Friedrich Steimann (Hrsg.)
Modellierung 2008
- P-128 Ammar Alkassar, Jörg Siekmann (Hrsg.)
Sicherheit 2008
Sicherheit, Schutz und Zuverlässigkeit
Beiträge der 4. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik e.V. (GI)
2.-4. April 2008
Saarbrücken, Germany
- P-129 Wolfgang Hesse, Andreas Oberweis (Eds.)
Sigsand-Europe 2008
Proceedings of the Third AIS SIGSAND European Symposium on Analysis, Design, Use and Societal Impact of Information Systems
- P-130 Paul Müller, Bernhard Neumair, Gabi Dreo Rodosek (Hrsg.)
1. DFN-Forum Kommunikationstechnologien Beiträge der Fachtagung
- P-131 Robert Krimmer, Rüdiger Grimm (Eds.)
3rd International Conference on Electronic Voting 2008
Co-organized by Council of Europe, Gesellschaft für Informatik und E-Voting, CC
- P-132 Silke Seehusen, Ulrike Lucke, Stefan Fischer (Hrsg.)
DeLFI 2008:
Die 6. e-Learning Fachtagung Informatik
- P-133 Heinz-Gerd Hegering, Axel Lehmann, Hans Jürgen Ohlbach, Christian Scheideler (Hrsg.)
INFORMATIK 2008
Beherrschbare Systeme – dank Informatik Band 1
- P-134 Heinz-Gerd Hegering, Axel Lehmann, Hans Jürgen Ohlbach, Christian Scheideler (Hrsg.)
INFORMATIK 2008
Beherrschbare Systeme – dank Informatik Band 2
- P-135 Torsten Brinda, Michael Fothe, Peter Hubwieser, Kirsten Schlüter (Hrsg.)
Didaktik der Informatik – Aktuelle Forschungsergebnisse
- P-136 Andreas Beyer, Michael Schroeder (Eds.)
German Conference on Bioinformatics GCB 2008
- P-137 Arslan Brömme, Christoph Busch, Detlef Hühlein (Eds.)
BIOSIG 2008: Biometrics and Electronic Signatures
- P-138 Barbara Dinter, Robert Winter, Peter Chamoni, Norbert Gronau, Klaus Turowski (Hrsg.)
Synergien durch Integration und Informationslogistik
Proceedings zur DW2008
- P-139 Georg Herzwurm, Martin Mikusz (Hrsg.)
Industrialisierung des Software-Managements
Fachtagung des GI-Fachausschusses Management der Anwendungsentwicklung und -wartung im Fachbereich Wirtschaftsinformatik
- P-140 Oliver Göbel, Sandra Frings, Detlef Günther, Jens Nedon, Dirk Schadt (Eds.)
IMF 2008 - IT Incident Management & IT Forensics
- P-141 Peter Loos, Markus Nüttgens, Klaus Turowski, Dirk Werth (Hrsg.)
Modellierung betrieblicher Informationssysteme (MobIS 2008)
Modellierung zwischen SOA und Compliance Management
- P-142 R. Bill, P. Korduan, L. Theuvsen, M. Morgenstern (Hrsg.)
Anforderungen an die Agrarinformatik durch Globalisierung und Klimaveränderung
- P-143 Peter Liggesmeyer, Gregor Engels, Jürgen Münch, Jörg Dörr, Norman Riegel (Hrsg.)
Software Engineering 2009
Fachtagung des GI-Fachbereichs Softwaretechnik

- P-144 Johann-Christoph Freytag, Thomas Ruf, Wolfgang Lehner, Gottfried Vossen (Hrsg.)
Datenbanksysteme in Business, Technologie und Web (BTW)
- P-145 Knut Hinkelmann, Holger Wache (Eds.)
WM2009: 5th Conference on Professional Knowledge Management
- P-146 Markus Bick, Martin Breunig, Hagen Höpfner (Hrsg.)
Mobile und Ubiquitäre Informationssysteme – Entwicklung, Implementierung und Anwendung
4. Konferenz Mobile und Ubiquitäre Informationssysteme (MMS 2009)
- P-147 Witold Abramowicz, Leszek Maciaszek, Ryszard Kowalczyk, Andreas Speck (Eds.)
Business Process, Services Computing and Intelligent Service Management
BPSC 2009 · ISM 2009 · YRW-MBP 2009
- P-148 Christian Erfurth, Gerald Eichler, Volkmar Schau (Eds.)
9th International Conference on Innovative Internet Community Systems
I²CS 2009
- P-149 Paul Müller, Bernhard Neumair, Gabi Dreo Rodosek (Hrsg.)
2. DFN-Forum
Kommunikationstechnologien
Beiträge der Fachtagung
- P-150 Jürgen Münch, Peter Liggesmeyer (Hrsg.)
Software Engineering
2009 - Workshopband
- P-151 Armin Heinzl, Peter Dadam, Stefan Kirm, Peter Lockemann (Eds.)
PRIMIUM
Process Innovation for Enterprise Software
- P-152 Jan Mendling, Stefanie Rinderle-Ma, Werner Esswein (Eds.)
Enterprise Modelling and Information Systems Architectures
Proceedings of the 3rd Int'l Workshop EMISA 2009
- P-153 Andreas Schwill, Nicolas Apostolopoulos (Hrsg.)
Lernen im Digitalen Zeitalter
DeLFI 2009 – Die 7. E-Learning Fachtagung Informatik
- P-154 Stefan Fischer, Erik Maehle, Rüdiger Reischuk (Hrsg.)
INFORMATIK 2009
Im Focus das Leben
- P-155 Arslan Brömme, Christoph Busch, Detlef Hühnlein (Eds.)
BIOSIG 2009:
Biometrics and Electronic Signatures
Proceedings of the Special Interest Group on Biometrics and Electronic Signatures
- P-156 Bernhard Koerber (Hrsg.)
Zukunft braucht Herkunft
25 Jahre »INFOS – Informatik und Schule«
- P-157 Ivo Grosse, Steffen Neumann, Stefan Posch, Falk Schreiber, Peter Stadler (Eds.)
German Conference on Bioinformatics 2009
- P-158 W. Claudepein, L. Theuvsen, A. Kämpf, M. Morgenstern (Hrsg.)
Precision Agriculture
Reloaded – Informationsgestützte Landwirtschaft
- P-159 Gregor Engels, Markus Luckey, Wilhelm Schäfer (Hrsg.)
Software Engineering 2010
- P-160 Gregor Engels, Markus Luckey, Alexander Pretschner, Ralf Reussner (Hrsg.)
Software Engineering 2010 –
Workshopband
(inkl. Doktorandensymposium)
- P-161 Gregor Engels, Dimitris Karagiannis, Heinrich C. Mayr (Hrsg.)
Modellierung 2010
- P-162 Maria A. Wimmer, Uwe Brinkhoff, Siegfried Kaiser, Dagmar Lück-Schneider, Erich Schweighofer, Andreas Wiebe (Hrsg.)
Vernetzte IT für einen effektiven Staat
Gemeinsame Fachtagung
Verwaltungsinformatik (FTVI) und
Fachtagung Rechtsinformatik (FTRI) 2010
- P-163 Markus Bick, Stefan Eulgem, Elgar Fleisch, J. Felix Hampe, Birgitta König-Ries, Franz Lehner, Key Pousttchi, Kai Rannenberg (Hrsg.)
Mobile und Ubiquitäre Informationssysteme
Technologien, Anwendungen und Dienste zur Unterstützung von mobiler
Kollaboration
- P-164 Arslan Brömme, Christoph Busch (Eds.)
BIOSIG 2010: Biometrics and Electronic Signatures
Proceedings of the Special Interest Group on Biometrics and Electronic Signatures

- P-165 Gerald Eichler, Peter Kropf, Ulrike Lechner, Phayung Meesad, Herwig Unger (Eds.)
10th International Conference on Innovative Internet Community Systems (I²CS) – Jubilee Edition 2010 –
- P-166 Paul Müller, Bernhard Neumair, Gabi Dreo Rodosek (Hrsg.)
3. DFN-Forum Kommunikationstechnologien Beiträge der Fachtagung
- P-167 Robert Krimmer, Rüdiger Grimm (Eds.)
4th International Conference on Electronic Voting 2010
co-organized by the Council of Europe, Gesellschaft für Informatik and E-Voting.CC
- P-168 Ira Diethelm, Christina Dörge, Claudia Hildebrandt, Carsten Schulte (Hrsg.)
Didaktik der Informatik
Möglichkeiten empirischer Forschungsmethoden und Perspektiven der Fachdidaktik
- P-169 Michael Kerres, Nadine Ojstersek Ulrik Schroeder, Ulrich Hoppe (Hrsg.)
DeLFI 2010 - 8. Tagung der Fachgruppe E-Learning der Gesellschaft für Informatik e.V.
- P-170 Felix C. Freiling (Hrsg.)
Sicherheit 2010
Sicherheit, Schutz und Zuverlässigkeit
- P-171 Werner Esswein, Klaus Turowski, Martin Juhrisch (Hrsg.)
Modellierung betrieblicher Informationssysteme (MobIS 2010)
Modellgestütztes Management
- P-172 Stefan Klink, Agnes Koschmider Marco Mevius, Andreas Oberweis (Hrsg.)
EMISA 2010
Einflussfaktoren auf die Entwicklung flexibler, integrierter Informationssysteme
Beiträge des Workshops der GI-Fachgruppe EMISA (Entwicklungsmethoden für Informationssysteme und deren Anwendung)
- P-173 Dietmar Schomburg, Andreas Grote (Eds.)
German Conference on Bioinformatics 2010
- P-174 Arslan Brömme, Torsten Eymann, Detlef Hühnlein, Heiko Roßnagel, Paul Schmücker (Hrsg.)
perspeGktive 2010
Workshop „Innovative und sichere Informationstechnologie für das Gesundheitswesen von morgen“
- P-175 Klaus-Peter Fähnrich, Bogdan Franczyk (Hrsg.)
INFORMATIK 2010
Service Science – Neue Perspektiven für die Informatik
Band 1
- P-176 Klaus-Peter Fähnrich, Bogdan Franczyk (Hrsg.)
INFORMATIK 2010
Service Science – Neue Perspektiven für die Informatik
Band 2
- P-177 Witold Abramowicz, Rainer Alt, Klaus-Peter Fähnrich, Bogdan Franczyk, Leszek A. Maciaszek (Eds.)
INFORMATIK 2010
Business Process and Service Science – Proceedings of ISSS and BPSC
- P-178 Wolfram Pietsch, Benedikt Krams (Hrsg.)
Vom Projekt zum Produkt
Fachtagung des GI-Fachausschusses Management der Anwendungsentwicklung und -wartung im Fachbereich Wirtschafts-informatik (WI-MAW), Aachen, 2010
- P-179 Stefan Gruner, Bernhard Rumpe (Eds.)
FM+AM'2010
Second International Workshop on Formal Methods and Agile Methods
- P-180 Theo Härder, Wolfgang Lehner, Bernhard Mitschang, Harald Schöning, Holger Schwarz (Hrsg.)
Datenbanksysteme für Business, Technologie und Web (BTW) 14. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme“ (DBIS)
- P-181 Michael Clasen, Otto Schätzel, Brigitte Theuvsen (Hrsg.)
Qualität und Effizienz durch informationsgestützte Landwirtschaft, Fokus: Moderne Weinwirtschaft
- P-182 Ronald Maier (Hrsg.)
6th Conference on Professional Knowledge Management
From Knowledge to Action
- P-183 Ralf Reussner, Matthias Grund, Andreas Oberweis, Walter Tichy (Hrsg.)
Software Engineering 2011
Fachtagung des GI-Fachbereichs Softwaretechnik
- P-184 Ralf Reussner, Alexander Pretschner, Stefan Jähnichen (Hrsg.)
Software Engineering 2011
Workshopband
(inkl. Doktorandensymposium)

- P-185 Hagen Höpfner, Günther Specht, Thomas Ritz, Christian Bunse (Hrsg.)
MMS 2011: Mobile und ubiquitäre Informationssysteme Proceedings zur 6. Konferenz Mobile und Ubiquitäre Informationssysteme (MMS 2011)
- P-186 Gerald Eichler, Axel Küpper, Volkmar Schau, Hacène Fouchal, Herwig Unger (Eds.)
11th International Conference on Innovative Internet Community Systems (I²CS)
- P-187 Paul Müller, Bernhard Neumair, Gabi Dreo Rodosek (Hrsg.)
4. DFN-Forum Kommunikationstechnologien, Beiträge der Fachtagung 20. Juni bis 21. Juni 2011 Bonn
- P-188 Holger Rohland, Andrea Kienle, Steffen Friedrich (Hrsg.)
DeLFI 2011 – Die 9. e-Learning Fachtagung Informatik der Gesellschaft für Informatik e.V. 5.–8. September 2011, Dresden
- P-189 Thomas, Marco (Hrsg.)
Informatik in Bildung und Beruf INFOS 2011
14. GI-Fachtagung Informatik und Schule
- P-190 Markus Nüttgens, Oliver Thomas, Barbara Weber (Eds.)
Enterprise Modelling and Information Systems Architectures (EMISA 2011)
- P-191 Arslan Brömme, Christoph Busch (Eds.)
BIOSIG 2011
International Conference of the Biometrics Special Interest Group
- P-192 Hans-Ulrich Heiß, Peter Pepper, Holger Schlingloff, Jörg Schneider (Hrsg.)
INFORMATIK 2011
Informatik schafft Communities
- P-193 Wolfgang Lehner, Gunther Piller (Hrsg.)
IMDM 2011
- P-194 M. Clasen, G. Fröhlich, H. Bernhardt, K. Hildebrand, B. Theuvsen (Hrsg.)
Informationstechnologie für eine nachhaltige Landwirtschaft Fokus Forstwirtschaft
- P-195 Neeraj Suri, Michael Waidner (Hrsg.)
Sicherheit 2012
Sicherheit, Schutz und Zuverlässigkeit Beiträge der 6. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik e.V. (GI)
- P-196 Arslan Brömme, Christoph Busch (Eds.)
BIOSIG 2012
Proceedings of the 11th International Conference of the Biometrics Special Interest Group
- P-197 Jörn von Lucke, Christian P. Geiger, Siegfried Kaiser, Erich Schweighofer, Maria A. Wimmer (Hrsg.)
Auf dem Weg zu einer offenen, smarten und vernetzten Verwaltungskultur Gemeinsame Fachtagung Verwaltungsinformatik (FTVI) und Fachtagung Rechtsinformatik (FTRI) 2012
- P-198 Stefan Jähnichen, Axel Küpper, Sahin Albayrak (Hrsg.)
Software Engineering 2012
Fachtagung des GI-Fachbereichs Softwaretechnik
- P-199 Stefan Jähnichen, Bernhard Rumpe, Holger Schlingloff (Hrsg.)
Software Engineering 2012
Workshopband
- P-200 Gero Mühl, Jan Richling, Andreas Herkersdorf (Hrsg.)
ARCS 2012 Workshops
- P-201 Elmar J. Sinz Andy Schürr (Hrsg.)
Modellierung 2012
- P-202 Andrea Back, Markus Bick, Martin Breunig, Key Pousttchi, Frédéric Thiesse (Hrsg.)
MMS 2012: Mobile und Ubiquitäre Informationssysteme
- P-203 Paul Müller, Bernhard Neumair, Helmut Reiser, Gabi Dreo Rodosek (Hrsg.)
5. DFN-Forum Kommunikationstechnologien
Beiträge der Fachtagung
- P-204 Gerald Eichler, Leendert W. M. Wienhofen, Anders Kofod-Petersen, Herwig Unger (Eds.)
12th International Conference on Innovative Internet Community Systems (I²CS 2012)
- P-205 Manuel J. Kripp, Melanie Volkamer, Rüdiger Grimm (Eds.)
5th International Conference on Electronic Voting 2012 (EVOTE2012)
Co-organized by the Council of Europe, Gesellschaft für Informatik and E-Voting.CC
- P-206 Stefanie Rinderle-Ma, Mathias Weske (Hrsg.)
EMISA 2012
Der Mensch im Zentrum der Modellierung
- P-207 Jörg Desel, Jörg M. Haake, Christian Spannagel (Hrsg.)
DeLFI 2012: Die 10. e-Learning Fachtagung Informatik der Gesellschaft für Informatik e.V.
24.–26. September 2012

- P-208 Ursula Goltz, Marcus Magnor, Hans-Jürgen Appelrath, Herbert Matthies, Wolf-Tilo Balke, Lars Wolf (Hrsg.)
INFORMATIK 2012
- P-209 Hans Brandt-Pook, André Fleer, Thorsten Spitta, Malte Wattenberg (Hrsg.)
Nachhaltiges Software Management
- P-210 Erhard Plödereder, Peter Dencker, Herbert Klenk, Hubert B. Keller, Silke Spitzer (Hrsg.)
Automotive – Safety & Security 2012
Sicherheit und Zuverlässigkeit für automobile Informationstechnik
- P-211 M. Clasen, K. C. Kersebaum, A. Meyer-Aurich, B. Theuvsen (Hrsg.)
Massendatenmanagement in der Agrar- und Ernährungswirtschaft
Erhebung - Verarbeitung - Nutzung
Referate der 33. GIL-Jahrestagung 20. – 21. Februar 2013, Potsdam
- P-212 Arslan Brömme, Christoph Busch (Eds.)
BIOSIG 2013
Proceedings of the 12th International Conference of the Biometrics Special Interest Group
04.–06. September 2013
Darmstadt, Germany
- P-213 Stefan Kowalewski, Bernhard Rumpe (Hrsg.)
Software Engineering 2013
Fachtagung des GI-Fachbereichs Softwaretechnik
- P-214 Volker Markl, Gunter Saake, Kai-Uwe Sattler, Gregor Hackenbroich, Bernhard Mitschang, Theo Härder, Veit Köppen (Hrsg.)
Datenbanksysteme für Business, Technologie und Web (BTW) 2013
13. – 15. März 2013, Magdeburg
- P-215 Stefan Wagner, Horst Lichter (Hrsg.)
Software Engineering 2013
Workshopband
(inkl. Doktorandensymposium)
26. Februar – 1. März 2013, Aachen
- P-216 Gunter Saake, Andreas Henrich, Wolfgang Lehner, Thomas Neumann, Veit Köppen (Hrsg.)
Datenbanksysteme für Business, Technologie und Web (BTW) 2013 – Workshopband
11. – 12. März 2013, Magdeburg
- P-217 Paul Müller, Bernhard Neumair, Helmut Reiser, Gabi Dreo Rodosek (Hrsg.)
6. DFN-Forum Kommunikationstechnologien
Beiträge der Fachtagung
03.–04. Juni 2013, Erlangen
- P-218 Andreas Breiter, Christoph Rensing (Hrsg.)
DeLFI 2013: Die 11 e-Learning Fachtagung Informatik der Gesellschaft für Informatik e.V. (GI)
8. – 11. September 2013, Bremen
- P-219 Norbert Breier, Peer Stechert, Thomas Wilke (Hrsg.)
Informatik erweitert Horizonte
INFOS 2013
15. GI-Fachtagung Informatik und Schule
26. – 28. September 2013
- P-220 Matthias Horbach (Hrsg.)
INFORMATIK 2013
Informatik angepasst an Mensch, Organisation und Umwelt
16. – 20. September 2013, Koblenz
- P-221 Maria A. Wimmer, Marijn Janssen, Ann Macintosh, Hans Jochen Scholl, Efthimos Tambouris (Eds.)
Electronic Government and Electronic Participation
Joint Proceedings of Ongoing Research of IFIP EGOV and IFIP ePart 2013
16. – 19. September 2013, Koblenz
- P-222 Reinhard Jung, Manfred Reichert (Eds.)
Enterprise Modelling and Information Systems Architectures (EMISA 2013)
St. Gallen, Switzerland
September 5. – 6. 2013
- P-223 Detlef Hühnlein, Heiko Roßnagel (Hrsg.)
Open Identity Summit 2013
10. – 11. September 2013
Kloster Banz, Germany
- P-224 Eckhart Hanser, Martin Mikusz, Masud Fazal-Baqaie (Hrsg.)
Vorgehensmodelle 2013
Vorgehensmodelle – Anspruch und Wirklichkeit
20. Tagung der Fachgruppe Vorgehensmodelle im Fachgebiet Wirtschaftsinformatik (WI-VM) der Gesellschaft für Informatik e.V.
Lörrach, 2013
- P-225 Hans-Georg Fill, Dimitris Karagiannis, Ulrich Reimer (Hrsg.)
Modellierung 2014
19. – 21. März 2014, Wien
- P-226 M. Clasen, M. Hamer, S. Lehnert, B. Petersen, B. Theuvsen (Hrsg.)
IT-Standards in der Agrar- und Ernährungswirtschaft Fokus: Risiko- und Krisenmanagement
Referate der 34. GIL-Jahrestagung
24. – 25. Februar 2014, Bonn

- P-227 Wilhelm Hasselbring,
Nils Christian Ehmke (Hrsg.)
Software Engineering 2014
Fachtagung des GI-Fachbereichs
Softwaretechnik
25. – 28. Februar 2014
Kiel, Deutschland
- P-228 Stefan Katzenbeisser, Volkmar Lotz,
Edgar Weippl (Hrsg.)
Sicherheit 2014
Sicherheit, Schutz und Zuverlässigkeit
Beiträge der 7. Jahrestagung des
Fachbereichs Sicherheit der
Gesellschaft für Informatik e.V. (GI)
19. – 21. März 2014, Wien
- P-229 Dagmar Lück-Schneider, Thomas
Gordon, Siegfried Kaiser, Jörn von
Lucke, Erich Schweighofer, Maria
A. Wimmer, Martin G. Löhe (Hrsg.)
Gemeinsam Electronic Government
ziel(gruppen)gerecht gestalten und
organisieren
Gemeinsame Fachtagung
Verwaltungsinformatik (FTVI) und
Fachtagung Rechtsinformatik (FTRI)
2014, 20.-21. März 2014 in Berlin
- P-230 Arslan Brömme, Christoph Busch (Eds.)
BIOSIG 2014
Proceedings of the 13th International
Conference of the Biometrics Special
Interest Group
10. – 12. September 2014 in
Darmstadt, Germany
- P-231 Paul Müller, Bernhard Neumair,
Helmut Reiser, Gabi Dreo Rodosek
(Hrsg.)
7. DFN-Forum
Kommunikationstechnologien
16. – 17. Juni 2014
Fulda
- P-232 E. Plödereder, L. Grunske, E. Schneider,
D. Ull (Hrsg.)
INFORMATIK 2014
Big Data – Komplexität meistern
22. – 26. September 2014
Stuttgart
- P-233 Stephan Trahasch, Rolf Plötzner, Gerhard
Schneider, Claudia Gayer, Daniel Sassiati,
Nicole Wöhrle (Hrsg.)
DeLFI 2014 – Die 12. e-Learning
Fachtagung Informatik
der Gesellschaft für Informatik e.V.
15. – 17. September 2014
Freiburg
- P-234 Fernand Feltz, Bela Mutschler, Benoît
Ottjacques (Eds.)
Enterprise Modelling and Information
Systems Architectures
(EMISA 2014)
Luxembourg, September 25-26, 2014
- P-235 Robert Giegerich,
Ralf Hofestädt,
Tim W. Nattkemper (Eds.)
German Conference on
Bioinformatics 2014
September 28 – October 1
Bielefeld, Germany
- P-236 Martin Engstler, Eckhart Hanser,
Martin Mikusz, Georg Herzwurm (Hrsg.)
Projektmanagement und
Vorgehensmodelle 2014
Soziale Aspekte und Standardisierung
Gemeinsame Tagung der Fachgruppen
Projektmanagement (WI-PM) und
Vorgehensmodelle (WI-VM) im
Fachgebiet Wirtschaftsinformatik der
Gesellschaft für Informatik e.V., Stuttgart
2014
- P-237 Detlef Hühnlein, Heiko Roßnagel (Hrsg.)
Open Identity Summit 2014
4.–6. November 2014
Stuttgart, Germany
- P-238 Arno Ruckelshausen, Hans-Peter
Schwarz, Brigitte Theuvsen (Hrsg.)
Informatik in der Land-, Forst- und
Ernährungswirtschaft
Referate der 35. GIL-Jahrestagung
23. – 24. Februar 2015, Geisenheim
- P-239 Uwe Aßmann, Birgit Demuth, Thorsten
Spitta, Georg Püschel, Ronny Kaiser
(Hrsg.)
Software Engineering & Management
2015
17.-20. März 2015, Dresden
- P-240 Herbert Klenk, Hubert B. Keller, Erhard
Plödereder, Peter Dencker (Hrsg.)
Automotive – Safety & Security 2015
Sicherheit und Zuverlässigkeit für
automobile Informationstechnik
21.–22. April 2015, Stuttgart
- P-241 Thomas Seidl, Norbert Ritter,
Harald Schöning, Kai-Uwe Sattler,
Theo Härder, Steffen Friedrich,
Wolfram Wingerath (Hrsg.)
Datenbanksysteme für Business,
Technologie und Web (BTW 2015)
04. – 06. März 2015, Hamburg

- P-242 Norbert Ritter, Andreas Henrich, Wolfgang Lehner, Andreas Thor, Steffen Friedrich, Wolfram Wingerath (Hrsg.)
Datenbanksysteme für Business, Technologie und Web (BTW 2015) – Workshopband
02. – 03. März 2015, Hamburg
- P-243 Paul Müller, Bernhard Neumair, Helmut Reiser, Gabi Dreo Rodosek (Hrsg.)
8. DFN-Forum
Kommunikationstechnologien
06.–09. Juni 2015, Lübeck
- P-244 Alfred Zimmermann, Alexander Rossmann (Eds.)
Digital Enterprise Computing (DEC 2015)
Böblingen, Germany June 25-26, 2015
- P-245 Arslan Brömme, Christoph Busch, Christian Rathgeb, Andreas Uhl (Eds.)
BIOSIG 2015
Proceedings of the 14th International Conference of the Biometrics Special Interest Group
09.–11. September 2015
Darmstadt, Germany
- P-246 Douglas W. Cunningham, Petra Hofstedt, Klaus Meer, Ingo Schmitt (Hrsg.)
INFORMATIK 2015
28.9.-2.10. 2015, Cottbus
- P-247 Hans Pongratz, Reinhard Keil (Hrsg.)
DeLFI 2015 – Die 13. E-Learning Fachtagung Informatik der Gesellschaft für Informatik e.V. (GI)
1.–4. September 2015
München
- P-248 Jens Kolb, Henrik Leopold, Jan Mendling (Eds.)
Enterprise Modelling and Information Systems Architectures
Proceedings of the 6th Int. Workshop on Enterprise Modelling and Information Systems Architectures, Innsbruck, Austria
September 3-4, 2015
- P-249 Jens Gallenbacher (Hrsg.)
Informatik
allgemeinbildend begreifen
INFOS 2015 16. GI-Fachtagung
Informatik und Schule
20.–23. September 2015
- P-250 Martin Engstler, Masud Fazal-Baqaie, Eckhart Hanser, Martin Mikusz, Alexander Volland (Hrsg.)
Projektmanagement und Vorgehensmodelle 2015
Hybride Projektstrukturen erfolgreich umsetzen
Gemeinsame Tagung der Fachgruppen Projektmanagement (WI-PM) und Vorgehensmodelle (WI-VM) im Fachgebiet Wirtschaftsinformatik der Gesellschaft für Informatik e.V., Elmshorn 2015
- P-251 Detlef Hühnlein, Heiko Roßnagel, Raik Kuhlisch, Jan Ziesing (Eds.)
Open Identity Summit 2015
10.–11. November 2015
Berlin, Germany
- P-252 Jens Knoop, Uwe Zdun (Hrsg.)
Software Engineering 2016
Fachtagung des GI-Fachbereichs Softwaretechnik
23.–26. Februar 2016, Wien
- P-253 A. Ruckelshausen, A. Meyer-Aurich, T. Rath, G. Recke, B. Theuvsen (Hrsg.)
Informatik in der Land-, Forst- und Ernährungswirtschaft
Fokus: Intelligente Systeme – Stand der Technik und neue Möglichkeiten
Referate der 36. GIL-Jahrestagung
22.-23. Februar 2016, Osnabrück
- P-254 Andreas Oberweis, Ralf Reussner (Hrsg.)
Modellierung 2016
2.–4. März 2016, Karlsruhe
- P-255 Stefanie Betz, Ulrich Reimer (Hrsg.)
Modellierung 2016 Workshopband
2.–4. März 2016, Karlsruhe
- P-256 Michael Meier, Delphine Reinhardt, Steffen Wendzel (Hrsg.)
Sicherheit 2016
Sicherheit, Schutz und Zuverlässigkeit
Beiträge der 8. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik e.V. (GI)
5.–7. April 2016, Bonn
- P-257 Paul Müller, Bernhard Neumair, Helmut Reiser, Gabi Dreo Rodosek (Hrsg.)
9. DFN-Forum
Kommunikationstechnologien
31. Mai – 01. Juni 2016, Rostock

- P-258 Dieter Hertweck, Christian Decker (Eds.)
Digital Enterprise Computing (DEC 2016)
14.–15. Juni 2016, Böblingen
- P-259 Heinrich C. Mayr, Martin Pinzger (Hrsg.)
INFORMATIK 2016
26.–30. September 2016, Klagenfurt
- P-260 Arslan Brömme, Christoph Busch,
Christian Rathgeb, Andreas Uhl (Eds.)
BIOSIG 2016
Proceedings of the 15th International
Conference of the Biometrics Special
Interest Group
21.–23. September 2016, Darmstadt
- P-261 Detlef Rätz, Michael Breidung, Dagmar
Lück-Schneider, Siegfried Kaiser, Erich
Schweighofer (Hrsg.)
Digitale Transformation: Methoden,
Kompetenzen und Technologien für die
Verwaltung
Gemeinsame Fachtagung
Verwaltungsinformatik (FTVI) und
Fachtagung Rechtsinformatik (FTRI) 2016
22.–23. September 2016, Dresden
- P-262 Ulrike Lucke, Andreas Schwill,
Raphael Zender (Hrsg.)
DeLFI 2016 – Die 14. E-Learning
Fachtagung Informatik
der Gesellschaft für Informatik e.V. (GI)
11.–14. September 2016, Potsdam
- P-263 Martin Engstler, Masud Fazal-Baqaie,
Eckhart Hanser, Oliver Linssen, Martin
Mikusz, Alexander Volland (Hrsg.)
Projektmanagement und
Vorgehensmodelle 2016
Arbeiten in hybriden Projekten: Das
Sowohl-als-auch von Stabilität und
Dynamik
Gemeinsame Tagung der Fachgruppen
Projektmanagement (WI-PM) und
Vorgehensmodelle (WI-VM) im
Fachgebiet Wirtschaftsinformatik
der Gesellschaft für Informatik e.V.,
Paderborn 2016
- P-264 Detlef Hühnlein, Heiko Roßnagel,
Christian H. Schunck, Maurizio Talamo
(Eds.)
Open Identity Summit 2016
der Gesellschaft für Informatik e.V. (GI)
13.–14. October 2016, Rome, Italy
- P-265 Bernhard Mitschang, Daniela
Nicklas, Frank Leymann, Harald
Schöning, Melanie Herschel, Jens
Teubner, Theo Härder, Oliver Kopp,
Matthias Wieland (Hrsg.)
Datenbanksysteme für Business,
Technologie und Web (BTW 2017)
6.–10. März 2017, Stuttgart
- P-266 Bernhard Mitschang, Norbert Ritter,
Holger Schwarz, Meike Klettke, Andreas
Thor, Oliver Kopp, Matthias Wieland
(Hrsg.)
Datenbanksysteme für Business,
Technologie und Web (BTW 2017)
Workshopband
6.–7. März 2017, Stuttgart
- P-267 Jan Jürjens, Kurt Schneider (Hrsg.)
Software Engineering 2017
21.–24. Februar 2017, Hannover
- P-268 A. Ruckelshausen, A. Meyer-Aurich,
W. Lentz, B. Theuvsen (Hrsg.)
Informatik in der Land-, Forst- und
Ernährungswirtschaft
Fokus: Digitale Transformation –
Wege in eine zukunftsfähige
Landwirtschaft
Referate der 37. GIL-Jahrestagung
06.–07. März 2017, Dresden
- P-269 Peter Dencker, Herbert Klenk, Hubert
Keller, Erhard Plödereder (Hrsg.)
Automotive – Safety & Security 2017
30.–31. Mai 2017, Stuttgart
- P-270 Arslan Brömme, Christoph Busch,
Antitza Dantcheva, Christian Rathgeb,
Andreas Uhl (Eds.)
BIOSIG 2017
20.–22. September 2017, Darmstadt
- P-271 Paul Müller, Bernhard Neumair, Helmut
Reiser, Gabi Dreo Rodosek (Hrsg.)
10. DFN-Forum Kommunikationstechnologien
30. – 31. Mai 2017, Berlin
- P-272 Alexander Rossmann, Alfred
Zimmermann (eds.)
Digital Enterprise Computing
(DEC 2017)
11.–12. Juli 2017, Böblingen

- P-273 Christoph Igel, Carsten Ullrich, Martin Wessner (Hrsg.)
BILDUNGSRÄUME
DeLFI 2017
Die 15. e-Learning Fachtagung Informatik der Gesellschaft für Informatik e.V. (GI)
5. bis 8. September 2017, Chemnitz
- P-274 Ira Diethelm (Hrsg.)
Informatische Bildung zum Verstehen und Gestalten der digitalen Welt
13.–15. September 2017, Oldenburg
- P-275 Maximilian Eibl, Martin Gaedke (Hrsg.)
INFORMATIK 2017
25.–29. September 2017, Chemnitz
- P276 Alexander Volland, Martin Engstler, Masud Fazal-Baqaie, Eckhart Hanser, Oliver Linssen, Martin Mikusz (Hrsg.)
Projektmanagement und Vorgehensmodelle 2017
Die Spannung zwischen dem Prozess und den Menschen im Projekt
Gemeinsame Tagung der Fachgruppen Projektmanagement und Vorgehensmodelle im Fachgebiet Wirtschaftsinformatik der Gesellschaft für Informatik e.V. in Kooperation mit der Fachgruppe IT-Projektmanagement der GPM e.V., Darmstadt 2017
- P-277 Lothar Fritsch, Heiko Roßnagel, Detlef Hühnlein (Hrsg.)
Open Identity Summit 2017
5.–6. October 2017, Karlstad, Sweden
- P-278 Arno Ruckelshausen, Andreas Meyer-Aurich, Karsten Borchard, Constanze Hofacker, Jens-Peter Loy, Rolf Schwerdtfeger, Hans-Hennig Sundermeier, Helga Floto, Brigitte Theuvsen (Hrsg.)
Informatik in der Land-, Forst- und Ernährungswirtschaft
Referate der 38. GIL-Jahrestagung
26.–27. Februar 2018, Kiel
- P-279 Matthias Tichy, Eric Bodden, Marco Kuhrmann, Stefan Wagner, Jan-Philipp Steghöfer (Hrsg.)
Software Engineering und Software Management 2018
5.–9. März 2018, Ulm
- P-280 Ina Schaefer, Dimitris Karagiannis, Andreas Vogelsang, Daniel Méndez, Christoph Seidl (Hrsg.)
Modellierung 2018
21.–23. Februar 2018, Braunschweig
- P-281 Hanno Langweg, Michael Meier, Bernhard C. Witt, Delphine Reinhardt (Hrsg.)
Sicherheit 2018
Sicherheit, Schutz und Zuverlässigkeit
25.–27. April 2018, Konstanz
- P-282 Arslan Brömme, Christoph Busch, Antitza Dantcheva, Christian Rathgeb, Andreas Uhl (Eds.)
BIOSIG 2018
Proceedings of the 17th International Conference of the Biometrics Special Interest Group
26.–28. September 2018
Darmstadt, Germany
- P-283 Paul Müller, Bernhard Neumair, Helmut Reiser, Gabi Dreo Rodosek (Hrsg.)
11. DFN-Forum Kommunikationstechnologien
27.–28. Juni 2018, Günzburg
- P-284 Detlef Krömker, Ulrik Schroeder (Hrsg.)
DeLFI 2018 – Die 16. E-Learning Fachtagung Informatik
10.–12. September 2018, Frankfurt a. M.
- P-285 Christian Czarniecki, Carsten Brockmann, Eldar Sultanow, Agnes Koschmider, Annika Selzer (Hrsg.)
Workshops der INFORMATIK 2018 - Architekturen, Prozesse, Sicherheit und Nachhaltigkeit
26.–27. September 2018, Berlin
- P-286 Martin Mikusz, Alexander Volland, Martin Engstler, Masud Fazal-Baqaie, Eckhart Hanser, Oliver Linssen (Hrsg.)
Projektmanagement und Vorgehensmodelle 2018
Der Einfluss der Digitalisierung auf Projektmanagementmethoden und Entwicklungsprozesse
Düsseldorf 2018

- P-287 A. Meyer-Aurich, M. Gandorfer, N. Barta,
A. Gronauer, J. Kantelhardt, H. Floto (Hrsg.)
Informatik in der Land-, Forst- und
Ernährungswirtschaft
Fokus: Digitalisierung für
landwirtschaftliche Betriebe in
kleinstrukturierten Regionen – ein
Widerspruch in sich?
Referate der 39. GIL-Jahrestagung
18.–19. Februar 2019, Wien
- P-289 Torsten Grust, Felix Naumann, Alexander
Böhm, Wolfgang Lehner, Jens Teubner,
Meike Klettke, Theo Härder, Erhard
Rahm, Andreas Heuer, Holger Meyer
(Hrsg.)
Datenbanksysteme für Business,
Technologie und Web (BTW 2019)
4.–8. März 2019 in Rostock
- P-291 Michael Räckers, Sebastian Halsbenning,
Detlef Rätz, David Richter,
Erich Schweighofer (Hrsg.)
Digitalisierung von Staat und Verwaltung
Gemeinsame Fachtagung
Verwaltungsinformatik (FTVI) und
Fachtagung Rechtsinformatik (FTRI) 2019
6.–7. März 2019 in Münster
- P-292 Steffen Becker, Ivan Bogicevic, Georg
Herzwurm, Stefan Wagner (Hrsg.)
Software Engineering and Software
Management 2019
18.–22. Februar 2019 in Stuttgart

The titles can be purchased at:

Köllen Druck + Verlag GmbH

Ernst-Robert-Curtius-Str. 14 · D-53117 Bonn

Fax: +49 (0)228/9898222

E-Mail: druckverlag@koellen.de

Gesellschaft für Informatik e.V. (GI)

publishes this series in order to make available to a broad public recent findings in informatics (i.e. computer science and information systems), to document conferences that are organized in cooperation with GI and to publish the annual GI Award dissertation.

Broken down into

- seminars
- proceedings
- dissertations
- thematics

current topics are dealt with from the vantage point of research and development, teaching and further training in theory and practice. The Editorial Committee uses an intensive review process in order to ensure high quality contributions.

The volumes are published in German or English.

Information: <http://www.gi.de/service/publikationen/lni/>

ISSN 1617-5468

ISBN 978-3-88579-683-1

“BTW 2019” is the 18th event in a conference series focusing on a broad range of database topics from a variety of perspectives. With its emphasis on lively discussions and cross-fertilization of academia and industry, it provides a valuable platform to further the state of the art in database foundations and techniques for high-performance query processing and optimization, cloud data management, text and graph processing, and big data analytics, among others. This volume contains contributions from the keynotes, the refereed scientific and industrial program, as well as the refereed demonstration program and from the winners of the GI-FB-DBIS PhD thesis awards.