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

¹ Hasso Plattner Institute, University of Potsdam, Germany
² NetApp

March, 2019
Motivation

In scale-out database systems, queries must be routed to individual servers.
Motivation

In scale-out database systems, queries must be routed to individual servers.

**Direct Communication**

+ Latency
- Requires smart clients or static backends

**Central Dispatcher**

+ Simple clients / dynamic backends
- Central dispatcher is potential bottleneck
Motivation - Use Cases for Central Dispatching

- Horizontal Partitioning / Sharded Database

- Partially Replicated Database System
  - Maximize throughput by balancing the load evenly while minimizing memory footprint

---
Motivation – Central Dispatching from a Network Perspective

```python
>>> import psycopg2
>>> conn = psycopg2.connect("dbname='tpch' host='dispatcher'")
```

- **Logical view**

![Logical view diagram](image)
# Motivation - Central Dispatching from a Network Perspective

```python
>>> import psycopg2

>>> conn = psycopg2.connect("dbname='tpch' host='dispatcher'")
```

**Logical view**

![Logical View Diagram]

**Physical view**

![Physical View Diagram]
Whether the dispatcher becomes a bottleneck depends on the workload

- Number and size of queries/messages
- Ratio of processed tuples and result set size
Motivation

- Whether the dispatcher becomes a bottleneck depends on the workload
  - Number and size of queries/messages
  - Ratio of processed tuples and result set size

- “Transferring a large amount of data out of a database system to a client program is a common task.”
  

- Needed for statistical analyses or machine learning in clients
- Main bottleneck is *network bandwidth*
Research Goals

- Integration of a TCP connection hand-over by means of a reprogrammable network switch into a database

- Comparison of query-based dispatching approaches in terms of
  - Throughput scaling
  - Processing flexibility
Dispatcher Implementations

- Traditional architecture with two separate TCP connections:
  client $\leftrightarrow$ dispatcher $\leftrightarrow$ database
  1. **HAProxy** - free and open source TCP/HTTP load balancer
  2. **Hyrise dispatcher**

- Using a reprogrammable switch to perform TCP connection hand-over
  3. **Prism**: exchange most packets directly between client and backend

Client query is initially sent/routed to Prism Controller

- Prism Controller **forwards connection** to an appropriate backend and **reprograms the switch**
- Backend processes query and **sends result directly to the client** (bypassing the Prism Controller)
- Backend hands back connection to Prism Controller
Experimental Evaluation

- 10Gb and 40Gb Ethernet experiments
  - Hyrise with a stored procedure
  - wrk - HTTP benchmarking tool
  - mSwitch - software switch

- Honda et al. mSwitch: A Highly-Scalable, Modular Software Switch. SOSR 2015.
Experimental Evaluation with two Clients and Backends

- **10 GbE results**

  - Using TCP hand-over outperforms traditional approaches for large payloads

  - Scales up to bandwidth: \( \min(\Sigma \text{clients}, \Sigma \text{backends}) \)

  - Limited by bandwidth of central dispatcher

- Using TCP hand-over outperforms traditional approaches for large payloads
Experimental Evaluation with two Clients and Backends

- 10 GbE results

- Using TCP hand-over outperforms traditional approaches for large payloads

- Hyrise dispatcher performs best for small payload sizes up to 4kB

**Graph**

- Scales up to bandwidth: min(\(\Sigma\) clients, \(\Sigma\) backends)

- Limited by bandwidth of central dispatcher

**Payload Throughput [Gb/s]**

- Prism
- Dispatcher
- HAProxy

**Throughput for 512 B payload**

- Prism: 50 Mb/s
- Dispatcher: 63 MB/s
- HAProxy: 42 MB/s
Eris: Coordination-Free Consistent Transactions Using In-Network Concurrency Control

Jialin Li
University of Washington
jlili@cs.washington.edu

Ellis Michael
University of Washington
cmichael@cs.washington.edu

Dan R. K. Ports
University of Washington
drk@cs.washington.edu

ABSTRACT
Distributed storage systems aim to provide strong consistency and isolation guarantees on an architecture that is partitioned across multiple shards for scalability and replicability for fault tolerance. Traditionally, achieving all of these goals has required an expensive combination of atomic commitment and replication protocols—introducing extensive coordination overhead. Our system, Eris, takes a different approach. It moves a zero area of concurrency control functionality, which we term multi-sequence execution, onto the datacenter network itself. This network primitive takes on the responsibility for consistently ordering transactions, and a new lightweight transaction protocol ensures atomicity.

The result is that Eris avoids both replication and transaction coordination overhead: we show that it can process a large class of distributed transactions in a single round-trip from the client to the storage system without any explicit coordination between shards or replicas in the normal case. It provides atomicity, consistency, and fault tolerance with less than 10% overhead—achieving throughput 3.6-39x higher and latency 72-89% lower than a conventional design on standard benchmarks.

CCS CONCEPTS
• Information systems → Database transaction processing

Offload full SQL query segments onto a programmable dataplane (published @ CIDR 2019)

The Case for Network-Accelerated Query Processing

Alberto Lerner  Rana Hussein  Philippe Cudre-Mauroux
eXascalb Infolab, U. of Fribourg—Switzerland

ABSTRACT
The fastest plans in MPP databases are usually those with the least amount of data movement across nodes, as data is not processed while in transit. The network switches that connect MPP nodes are hard-wired to perform packet-forwarding logic only. However, as a recent paradigm shift, network devices are becoming “programmable.” The quotes here are necessary. Switches are not becoming general-purpose computers (just yet). But note the set of tasks they can perform can be encoded in software.

In this paper we explore this programmability to accelerate OLAP queries. We determined that we can offload onto the switch some very common and expensive query patterns. Thus, for the first time, moving data through the network device can pay off. We report our preliminary results here that we can improve response times on even the best agreed upon plan by more than 20x using 10Gbps networks. We also see the promise of linear performance improvement with faster switches. The use of programmable switches can open new possibilities of architecture and datacenter-wide database systems, with implications across the stack.

1. INTRODUCTION

Networking is an area in constant evolution. New protocols keep arising from emerging fields such as virtualization [22], and new emerging fields such as Barefoot Tofino [1], Cavium Xpliant [2], and Cisco Quantas [19].

Figure 1: (a) A multi-action table programmed to forward or to drop a packet according to its destination MAC address. (b) Architecture of a programmable switched dataplane with buffering.

with a cop in the table wing, for instance, exact matching. Other types of matches are also possible. The switch engine executes simple instructions over a packet or table data. Examples of such instructions are simple arithmetic or moving data within a packet. The MAU is programmable in the sense that one can specify its lookup tables, the type of lookups to perform, and the processing done at a match event, as we illustrate in Figure 1(a). We say that a MAC...
Scale-out database systems use central query dispatchers to hide backend complexity, but may be a bandwidth bottleneck.

We compared dispatching architectures for database systems:
- Traditional dispatcher performs best for small payload sizes
- Prism’s connection overhead pays off for larger payloads

-> Hybrid approach with on-demand connection hand-over for large results
Thanks

Stefan Klauck
stefan.klauck@hpi.de
http://epic.hpi.de