High-Speed Query Processing over High-Speed Networks

Wolf Rödiger, Tobias Mühlbauer, Alfons Kemper, Thomas Neumann
Traditional Data Warehouse

Transactions → Database → Extract Transform Load → Data Warehouse → Analytics
Traditional Data Warehouse

• Isolate business-critical **transactions** from analytical **queries**

• **ETL process** to update the data warehouse

• Periodic refresh leads to **data staleness**
HyPer

- **Analytics**
  Excellent response times

- **Transactions**
  100k TPC-C transaction/s

- **Both workloads** on the same state in one system

- Code generation, MVCC
Scale the **HyPer** main-memory database system to a **cluster** of machines
Full Replication

- **Replicate** the data from a primary server
- **Improved** query throughput
- **Same** main-memory, **same** response times as a single server
Horizontal Partitioning

- **Partition** the data across servers
- **Increases** main-memory capacity
- **But can we also** speed up query processing?
Speed-Up? Easy!

... as long as the system is slow on one server.
Unfortunately, HyPer is Fast
Negative Speed-Up

- Queries **shuffle data** for joins and aggregations
- **Low bandwidth** is main bottleneck
- More servers = **less performance**

Graph:
- TPC-H speed-up
- Gigabit Ethernet
- Number of servers: 1, 2, 3, 4, 5, 6
- Speed-up: 0x, 1x, 2x, 3x, 4x
Scale HyPer to a cluster and it should be fast
Can’t we just avoid communication?
Can’t we just avoid communication?

- **Partition** the data (H-Store/VoltDB do this for transactions)
- **Partition-crossing** queries problematic
- Partitioning depends on the **workload**
Can’t we just use faster network hardware?
Can’t we just use faster network hardware?

- **Low bandwidth** is main bottleneck
- InfiniBand offers up to $100 \times$ the bandwidth
- Existing software can use IPoIB unchanged
Can’t we just use faster network hardware?

- **New bottlenecks:**
  - TCP/IP stack processing
  - Interrupts
  - Context switches
  - Multiple memory transfers

![TPC-H speed-up graph](image)
Software has to Change

- For **slow** networks, more servers = less performance
- New bottlenecks emerge for **faster** networks
- **Software** has to change as well
Two Types of Networks
Two Types of Networks

QPI
• Connects **NUMA** sockets in a server
• **32 GB/s** bandwidth
• **0.2 µs** latency
• **Cache-coherent**

InfiniBand QDR
• Connects **servers** in a cluster
• **4 GB/s** bandwidth
• **1.3 µs** latency
• **Not** cache-coherent
Hybrid Parallelism

On each server:

- Use flexible worker threads instead of exchange operators
- **Work stealing** per CPU
- Work stealing across **NUMA** sockets

Between servers:

- Use Remote Direct Memory Access (RDMA) instead of TCP
- **Decoupled** exchange operators
- Network **scheduling**

* Leis et al., Morsel-driven parallelism, SIGMOD 2014
TCP over InfiniBand

- TCP is **compute-bound** at the receiver
- Even with **large MTUs** and **TCP offloading**
- Using a **separate core** for interrupts improves throughput by **53%**
- Still **compute-bound**

**Chart:**
- Sender: 59% busy, 41% idle
- Receiver: 98% busy, 2% idle
Remote Direct Memory Access

- **Bypasses** operating system and application
- **Zero-copy** network communication:
  - Achieves full network throughput
  - Almost no **CPU** cost
  - Less **data copying**
Remote Direct Memory Access

TCP via IPoIB

- Sender: 59% busy, 41% idle
- Receiver: 98% busy

RDMA

- Sender: 3% busy, 97% idle
- Receiver: 3% busy, 97% idle

+35% throughput

-98% CPU time
Classic Exchange

- A buffer **per exchange**: 
  - # buffers per server = \text{servers} \times \text{cores}^2
  - **1 GB**/server for 6 hosts and 20 cores
- **Skew** is huge problem:
  - Join key assigned to **fixed** exchange
  - **No work stealing**
Decoupled Exchange

- Use **communication multiplexers** (CM)
- **Address servers** not individual cores:
  - Decreases **memory** consumption (2.5 MB instead of 1 GB)
  - Reduces negative impact of **skew**
  - Makes **broadcast** more applicable
Decoupled Exchange

TPC-H speed-up

- Vectorwise Vortex (exchange)
- HyPer (exchange)
- HyPer (decoupled exchange)

number of servers

6 (1)  30 (5)  60 (10)  90 (15)  120 (20)

TPC-H speed-up

0x  5x  10x  15x

3x

3x
Network Scheduling

- Uncoordinated all-to-all transfers cause switch contention
- Make sure a server sends to at most one server
- Low-latency inline RDMA messages for network scheduling
Network Scheduling

![Graph showing throughput in GB/s vs number of servers. The graph compares two scheduling methods: all-to-all and scheduling. The y-axis represents throughput in GB/s, ranging from 0 to 4, and the x-axis represents the number of servers, ranging from 2 to 8. The graph indicates that scheduling consistently maintains a higher throughput compared to all-to-all as the number of servers increases.]
Summary

- For **slow** networks, more servers = less performance.
- New bottlenecks emerge for **faster** networks.
- **Hybrid parallelism** optimizes for both types of networks.

![Graph showing TPC-H speed-up with number of servers for Gigabit Ethernet, InfiniBand, and Hybrid Parallelism.]
How do we compare?

<table>
<thead>
<tr>
<th>Database</th>
<th>TPC-H queries/h</th>
</tr>
</thead>
<tbody>
<tr>
<td>HyPer Scale-Out</td>
<td>20,739</td>
</tr>
<tr>
<td>Vectorwise Vortex</td>
<td>3,856</td>
</tr>
<tr>
<td>MemSQL</td>
<td>544</td>
</tr>
<tr>
<td>Impala</td>
<td>123</td>
</tr>
<tr>
<td>Spark SQL</td>
<td>77</td>
</tr>
</tbody>
</table>
Future Work

What about low-latency distributed transactions?
Backup
Elasticity
High Availability
Hot/cold approach

- Multicast redo log
- Persistent storage
- Distributed queries on global TX-consistent snapshots
Parallelism

theory

practice