HyPer on Cloud 9

Thomas Neumann

Technische Universität München

February 10, 2016

HyPer

Technische Universität München

HyPer is the main-memory database system developed in our group

- a very fast database system with ACID transactions
- quite comprehensive SQL support (SQL92 plus many SQL99 features)
- queries are compiled into machine code using LLVM
- very little overhead, excellent performance

You can try it out (or download a demo) online:
`http://www.hyper-db.com`

We use it for teaching, too. (Easy to use web interface for students).

Hyper in the Cloud

Currently HyPer is primarily run on a single dedicated node

- but of course we looked at running HyPer in the cloud
- Wolf Rödiger will talk about distributed processing next
- I will concentrate on issues we found during experiments
- some of them quite surprising; could be relevant for your project, too
- not all of them have perfect solutions yet

Costs of Virtualization

Cloud applications usually run on virtualized machined

- shields the application from the actual hardware
- simplified management, allows for easy migration, etc.
- several different techniques to do that (containers, hypervisors, etc.)
- what does that actually cost?
- in-memory processing is CPU bound
- in an ideal world virtualization would be (nearly) free

Evaluation Systems

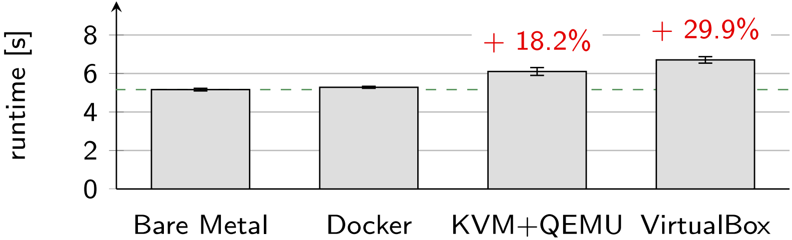
Local evaluation system

- Intel Core i7-3770 (Ivy Bridge), 3.40 GHz (3.90 GHz maximum turbo)
- 4 cores/8 threads
- 32 GB DDR3 1600 MHz memory
- Ubuntu 14.10 host and Ubuntu 14.10 guests
- Virtualization environments: Docker, KVM+QEMU, VirtualBox

Google Compute Engine (GCE)

- n1-standard-8 Compute Engine Instance
- Sandy Bridge CPU, 2.60 GHz
- 8 virtual cores
- 30 GB memory
- Ubuntu 14.10 guest

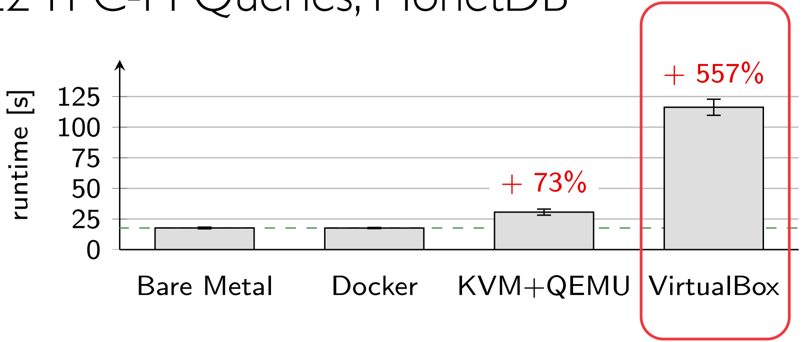
22 TPC-H Queries, HyPer



Workload

- TPC-H scale factor 10
- run the 22 TPC-H queries, 10 repetitions, mean execution time
- HyPer with 4 worker threads, intra-query parallel execution
- GCE n1-standard-8 (4 threads): 8.06s

22 TPC-H Queries, MonetDB



Workload

- TPC-H scale factor 10
- run the 22 TPC-H queries, 10 repetitions, mean execution time
- MonetDB with 4 worker threads, intra-query parallel execution
- GCE n1-standard-8 (4 threads): 37.79s

Some Microbenchmarks

TLB miss latency/page fault latency

- 10% overhead in KVM+QEMU and VirtualBox

System calls

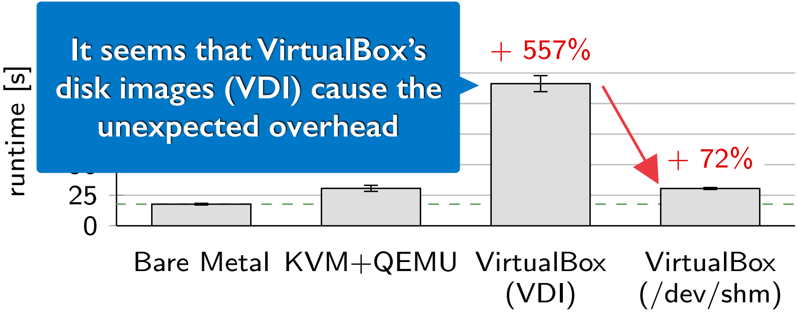
- 60% overhead in KVM+QEMU and VirtualBox

Overheads are too similar in KVM+QEMU and VirtualBox to explain the performance difference between the virtualization environments for MonetDB!

Database files in MonetDB

- Database stored on disk and mapped to memory on access
- When data is hot (accessed multiple times), it is cached in main memory and performance should be the same as a main-memory access

Storage Location of MonetDB Files



Workload

- TPC-H scale factor 10
- run the 22 TPC-H queries, 10 repetitions, mean execution time
- MonetDB with 4 worker threads, intra-query parallel execution
- GCE n1-standard-8 (4 threads): 37.79s

Cooperative Behavior

Cloud infrastructure relies upon cooperative behavior

- virtualized applications utilize ideal resources
- combined system looks more powerful than it really is
- hogging resources is bad for cloud processing

Main resource for HyPer (and probably most other DBMSs):

- main memory
- not only for the "real" data, but also for intermediate results
- some queries have large intermediate results
- allocate a lot of memory

Allocating and Releasing Memory

Memory allocation is a multi-step process:

1. pages are created in the page table, but (usually) not physically assigned
2. upon access, pages are assigned and zeroed out
3. upon release, page table is modified again

Problem: this does not scale

- a lot of locking and overhead within the kernel
- costs more than a factor 2 performance with large queries on large hosts
- even more pronounced on Windows, there "freeing" pages is surprisingly expensive

Solutions?

- never release memory? (i.e., do your own memory management)
- best performance, but hogs resources
- non-portable techniques? (madvise etc.) Unsatisfying.

Durability

For ACID, we must 1) not report commit until the WAL record hits durable media, and 2) avoid data corruption during writes.

- difficult to guarantee even on bare metal
- often requires battery-backed RAID controllers to avoid all issues (including tearing)

How do we guarantee that on virtualized hardware?

- nearly impossible
- the (virtualized) hardware lies to us
- an inconveniently timed power failure can lead to a disaster
- Microsoft installed special hooks for their cloud DBMS, but that is not publicly usable
- other vendors probably, too, but not generic solutions exists

Conclusion

Databases in the cloud are an interesting topic

- many effects that we see are implementation artifacts
- choice of virtualization product has a large impact
- many open issues
- concerning both performance and correctness
- but apparently many people are happy with living dangerously

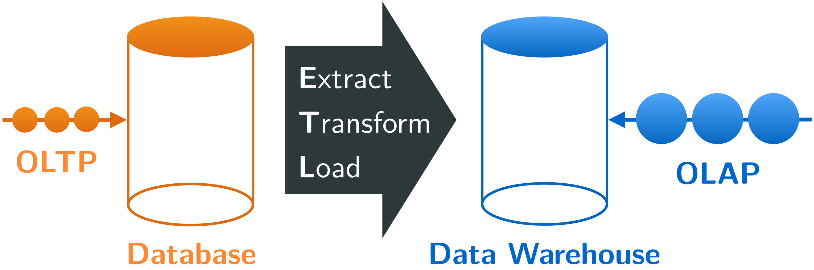
Still many open research questions.

HyPer on Cloud 9

Wolf Rödiger

Technische Universität München

Traditional Data Warehouse



HyPer: Hybrid OLTP & OLAP

HyPer

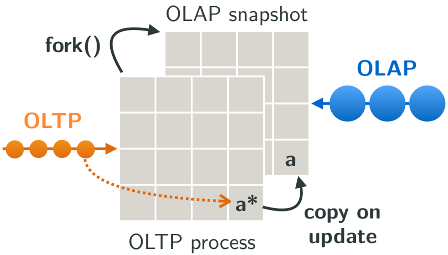
- **OLTP & OLAP** on the same data at the same time
- Efficient **snapshotting**
- Data-centric **code generation**

OLTP

> **100,000** TPC-C TX/s

OLAP

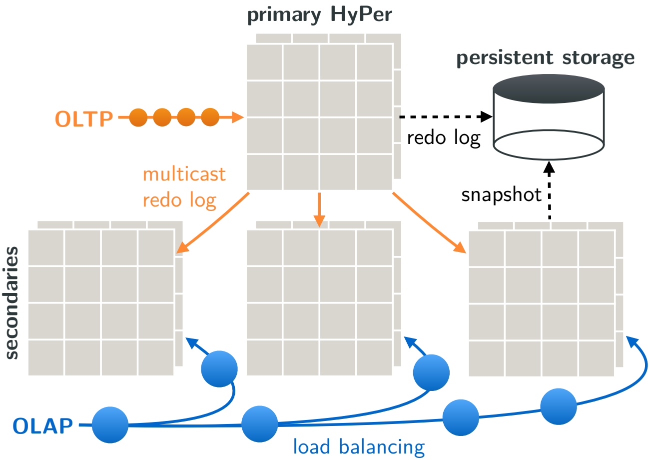
- **Best-of-breed** response times
- **Real-time** analytics



Wishlist for HyPer in the Cloud

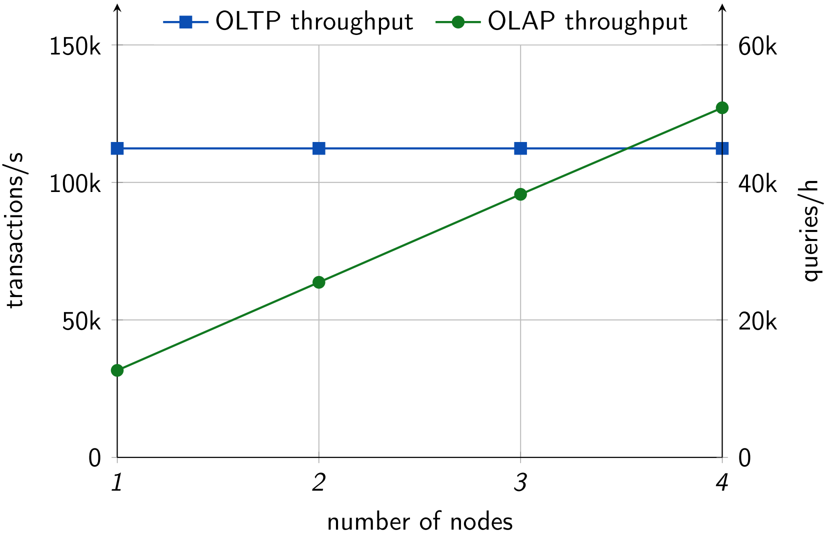
- Scale **query performance** with the cluster size
- Sustain **transaction performance**
- Scale **capacity** to process larger workloads
- **Elastically** add servers to the cluster
- Provide **high availability**

1st Design: Full Replication via Redo Log Multicasting

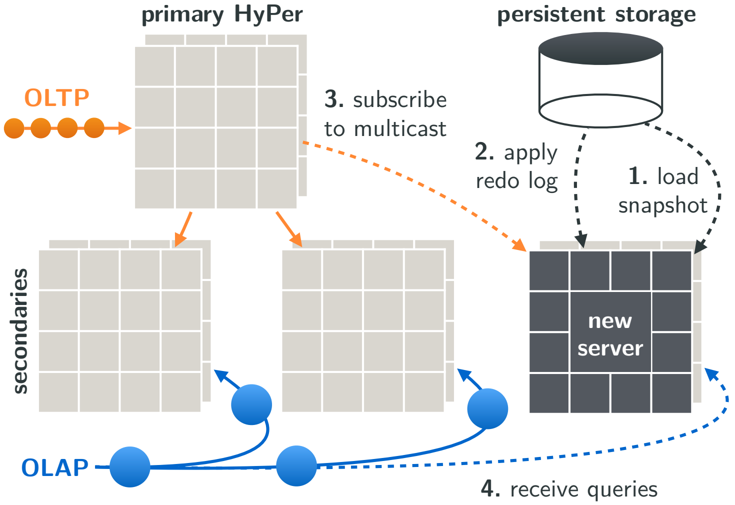


Tobias Mühlbauer, Wolf Rödiger, Angelika Reiser, Alfons Kemper, Thomas Neumann, *ScyPer: Elastic OLAP Throughput on Transactional Data*, DanaC 2012.

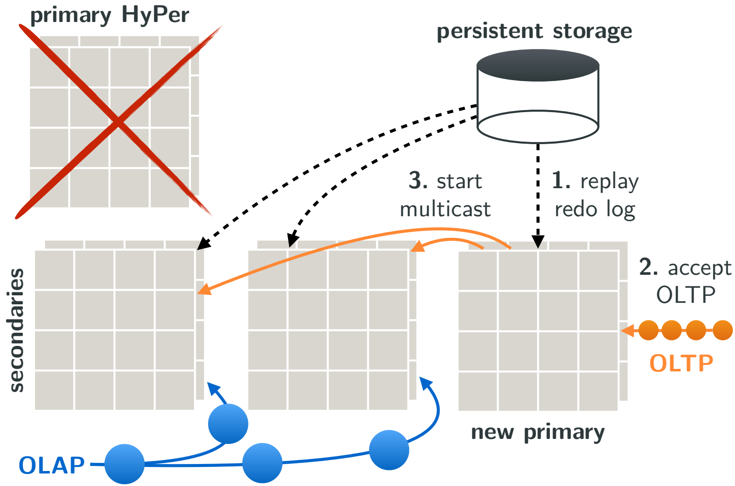
Scale query throughput, sustain transaction performance



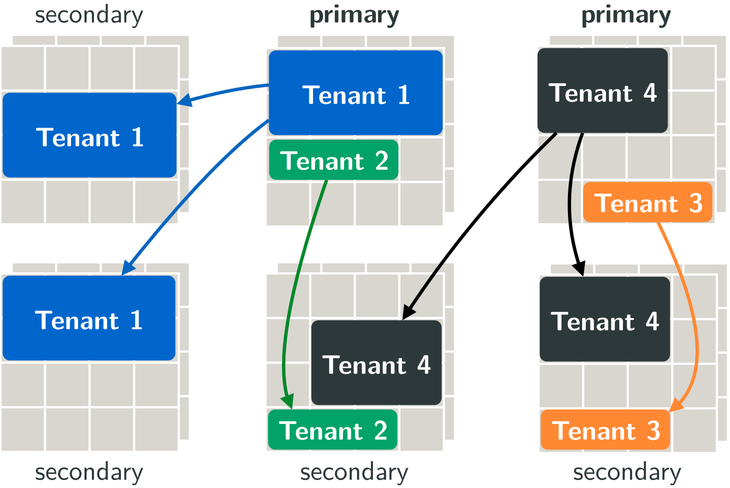
Elasticity: Materialized snapshots and multicasting



High Availability: Secondaries can replace the primary



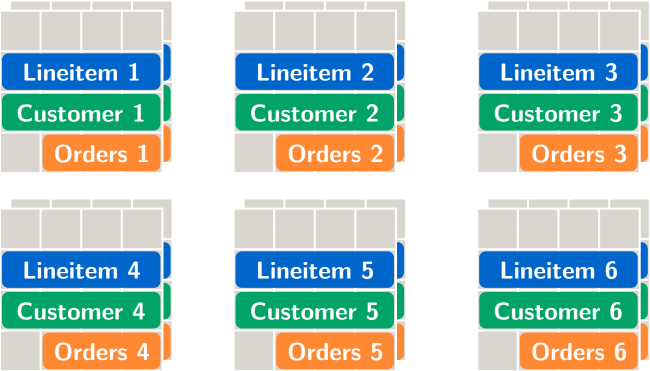
Multi-Tenancy: Flexible database-as-a-service deployment



1st Design: Full Replication via Redo Log Multicasting

- ✓ Scales **query throughput** with the cluster size
- ✓ Sustains **transaction performance**
- ✓ Allows to **elastically** add servers to the cluster
- ✓ Provides **high availability**
- ✗ Limited to workloads that fit into a **single server**
- ✗ Same **query response times** as a single server

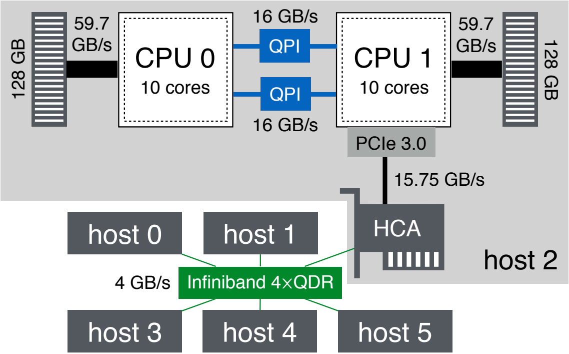
2nd Design: Query Processing on Fragmented Relations



- **Fragment relations** across servers to use the **combined capacity** of the cluster, making room for larger workloads

Wolf Rödiger, Tobias Mühlbauer, Alfons Kemper, Thomas Neumann, *High-Speed Query Processing over High-Speed Networks*, VLDB 2016.

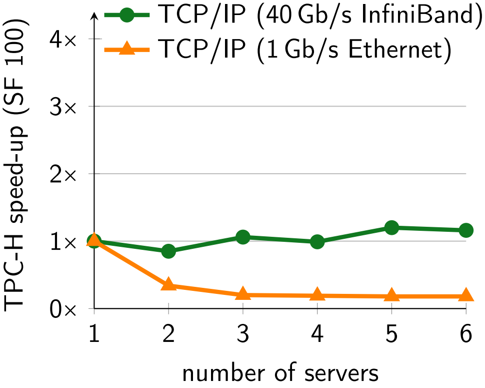
Main Challenges



- **Challenge 1:** Network is a **bottleneck** to query processing
- **Challenge 2:** Utilize all **cores** and all **servers**

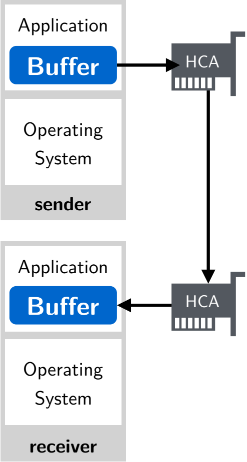
Challenge 1: Network is a Bottleneck

- **Network** is bottleneck for distributed processing
- Manual schema-tuning is **time-consuming** and **workload-dependent**
- Faster network hardware is not enough, **software has to change**



Remote Direct Memory Access

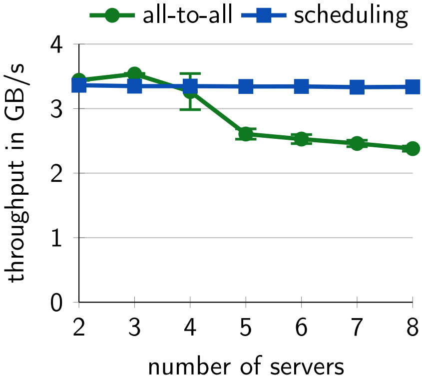
- **Zero-copy** network communication:
  - High network throughput
  - Almost no CPU cost
  - Less memory bus traffic
- Less CPU cost, higher throughput than **TCP**
- RDMA achieves **full speed** at only 4 % CPU load
- Recently used to implement a **distributed radix join***



*Barthels et al., *Rack-Scale In-Memory Join Processing using RDMA*, SIGMOD 2015.

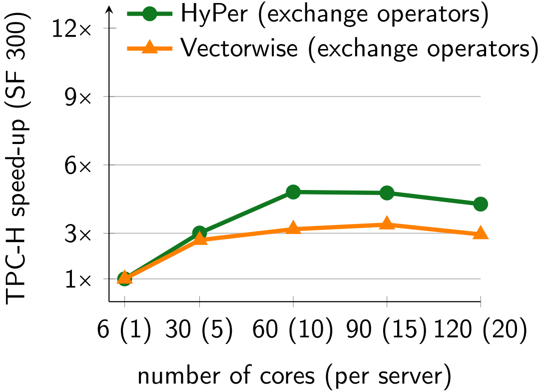
Network Scheduling

- Uncoordinated network communication causes **switch contention**
- Communicate in a strict **round-robin** fashion
- Synchronize via **low-latency** RDMA operations
- Network scheduling improves throughput by **40 %** for an 8-server InfiniBand cluster



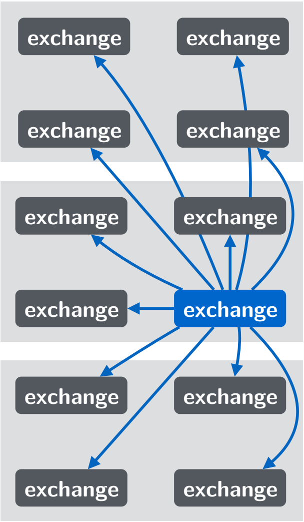
Challenge 2: Utilize all cores and all servers

- **Exchange** partitions tuples for joins and aggregations
- Traditionally used to parallelize for multiple **cores** and **servers**
- **But:** Exchange does **not scale** well due to inflexible design



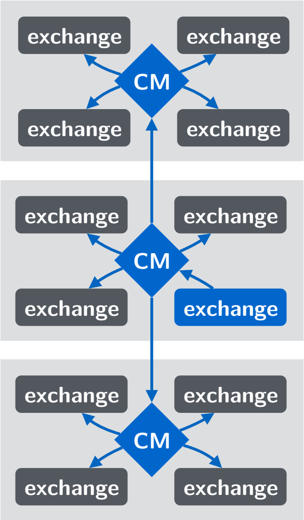
Classic Exchange

- **Fixed** degree of parallelism
- An exchange needs a buffer for **every other** exchange: **#buffers per server = servers × cores²**
- **2,400 buffers ≈ 1 GB** of main memory per server
- **Also:** Each join key value is processed by a **specific** exchange operator
- **Heavy hitters** are assigned to a **single exchange**



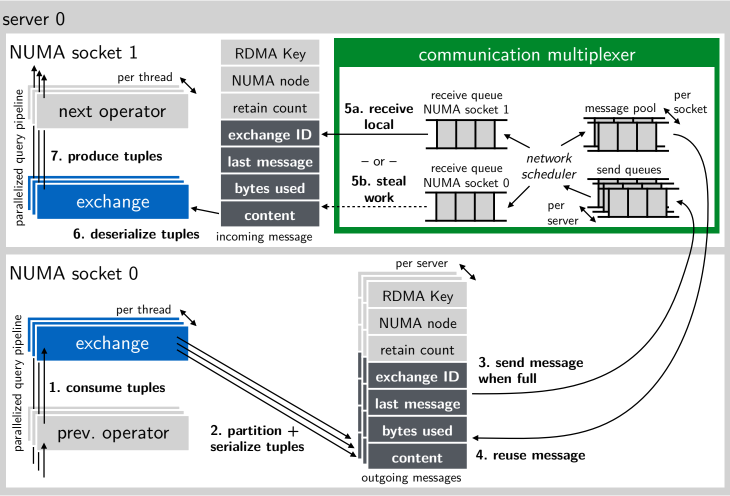
Decoupled Exchange

- Communicate **indirectly** via communication multiplexers
- Address servers not threads:
  - Decreases **memory consumption**
  - Reduces negative impact of **heavy hitters**
  - Improves applicability of **broadcast** optimization
- Local **load balancing** via **work stealing*** important for good scalability

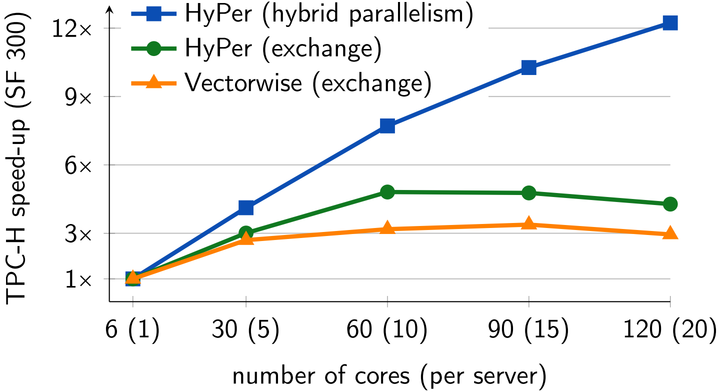


*Leis et al., *Morsel-driven parallelism: a NUMA-aware query evaluation framework for the many-core age*, SIGMOD 2014.

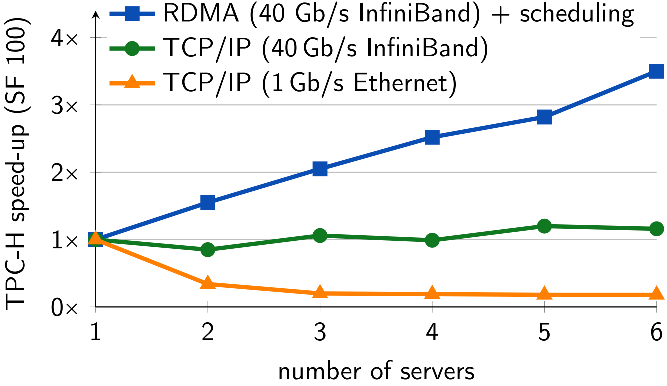
Hybrid Parallelism = RDMA-based Communication + Decoupled Exchange Operators



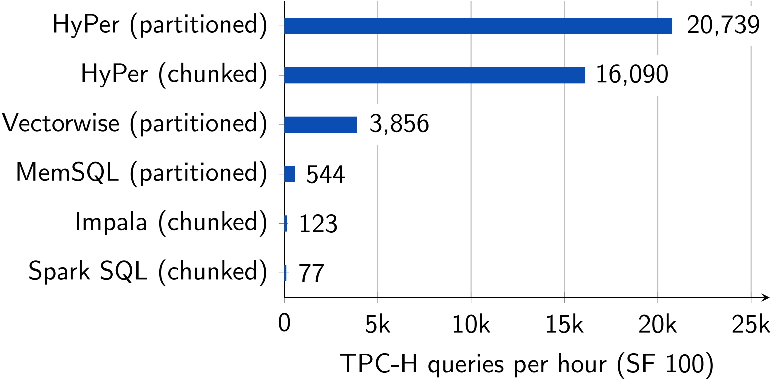
Hybrid parallelism scales in the number of cores ...



… and in the number of servers in the cluster



How do we compare?

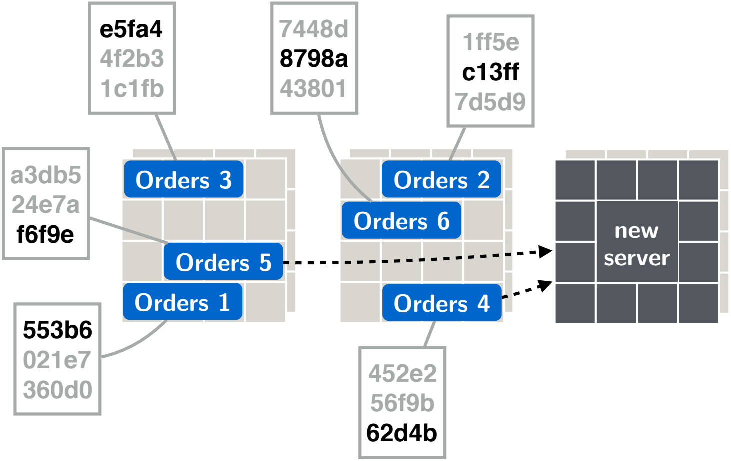


Note: MemSQL takes several seconds for query compilation, but these compilation times are not included in this experiment.

2nd Design: Query Processing on Fragmented Relations

- ✓ Utilizes the combined **capacity** of the cluster
- ✓ Reduces query **response times**
- ✓ Scales **query throughput** with the cluster size
- ? How can we **add servers** with minimal disruption?
- ? How can we survive **server failures**?
- ? How can we sustain the **transaction performance**?

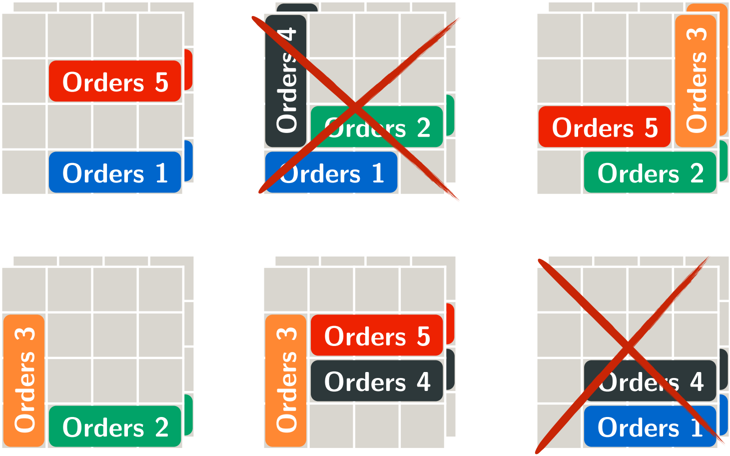
Outlook: Elasticity via Highest Random Weight Hashing



- Redistribute data when servers are **added/removed**
- Avoids reshuffling the **whole database**, balances load evenly

Mukherjee et al., *Distributed Architecture of Oracle Database In-memory*, VLDB 2015.

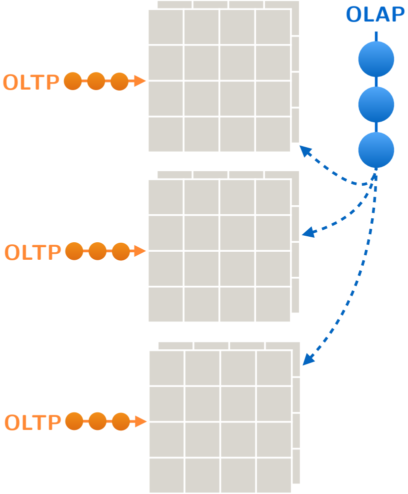
Outlook: High availability via HDFS-style replication



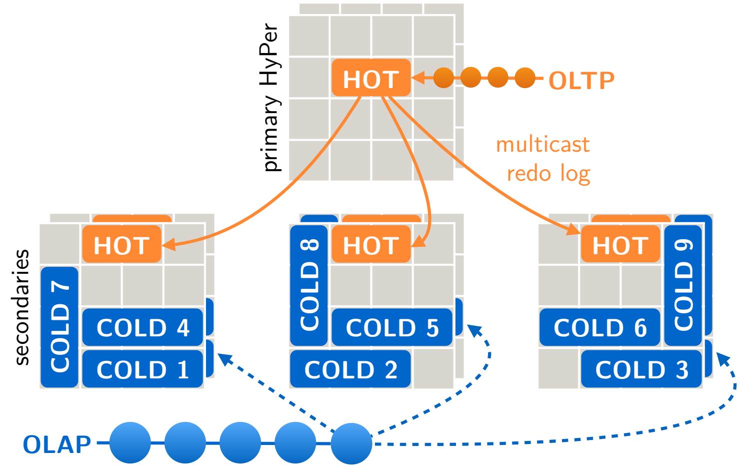
- Replication factor of x allows for $x - 1$ server failures

Outlook: Sustain transaction performance

- Distributed transactions are overly **expensive**:
  - Two-phase commit
  - Global locking
  - Deadlock detection
- H-Store model: **partitioned execution** of transactions
- Works well for **TPC-C**
- Drawbacks:
  - Requires **schema tuning**
  - Not for all workloads



Outlook: Hot/cold approach for distributed transactions



distributed queries on global TX-consistent snapshots

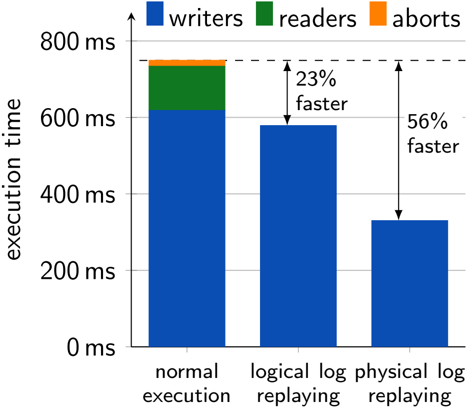
HyPer on Cloud 9

- ✓ Scale **capacity** to process larger workloads
- ✓ Reduce query **response times**
- ✓ Scale **query throughput** with the cluster size
- ✓ **Elastically** add servers to the cluster
- ✓ Provide **high availability**
- ✓ Sustain **transaction performance**

Backup Slides

Keeping secondaries up-to-date

- **Logical log:** transaction identifier and parameters
- **Physical log:** insert, update and delete operations



Replay log on secondaries

- Logical log excludes:
  - Readers
  - Aborts
  - Redo logging
  - Undo logging
- Physical log also excludes:
  - Read operations
  - Costly transaction logic

Redo Log Multicasting

- Independent of cluster size
- UDP unreliable, PGM reliable

TPC-C transactions

- ~60,000 log entries/s
- ~1,500 B physical log entry/TX
- ~250 B logical log entry/TX

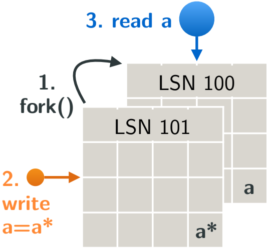
1 GbE vs. InfiniBand

- 1 GbE needs:
  - Group commits
  - LZ4 compression (~50 %)
- InfiniBand (IPoIB) is sufficient; PGM is CPU-bound

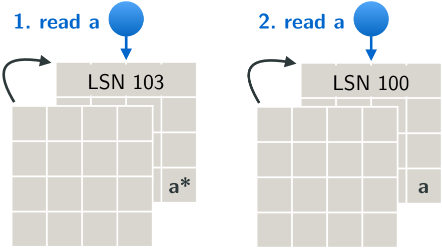
1 GbE	PGM
Bandwidth [Mbit/s]	675
Packets [1,000/s]	43
Latency [μs]	100.4

InfiniBand 4×QDR	PGM
Bandwidth [Mbit/s]	1,832
Packets [1,000/s]	112
Latency [μs]	13.5

Guaranteeing correct results



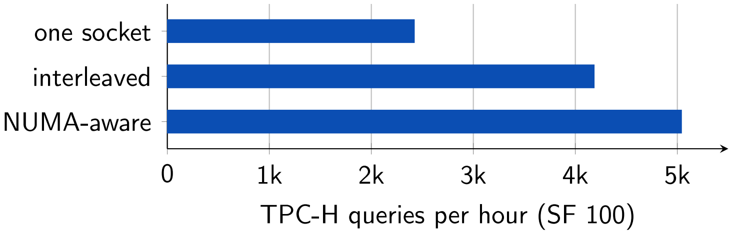
- Order-preserving serializability violation



- Diverging distributed reads

- **Logical time** defined by log sequence number (LSN)
- **Global TX-consistent snapshots** avoid consistency problems

NUMA



- **NUMA-aware** allocation of message buffers improves TPC-H performance by a **factor of 2** for a 4-socket server
- Our communication multiplexer provides **NUMA-local message buffers** to the decoupled exchange operators