Rethinking Distributed Databases for Modern Networks

Carsten Binnig
In the past ...

Network Communication was evil: Must be avoided at all cost

Distributed DBMS Mantra: Data-Locality first!

- Complex partitioning schemes to leverage data-locality
- Complex communication avoiding schemes (e.g. semi-join reducers, relaxed consistency protocols)

<table>
<thead>
<tr>
<th></th>
<th>DDR3 -1600</th>
<th>1Gb Eth.</th>
<th>Net/RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latency (μs)</td>
<td>0.1</td>
<td>100</td>
<td>1000</td>
</tr>
<tr>
<td>Throughput (GB/s)</td>
<td>51.2 (4 channels)</td>
<td>0.125</td>
<td>~400</td>
</tr>
</tbody>
</table>
BUT modern networks ... make it possible to achieve network bandwidth similar to the main memory bandwidth and it does no longer ruin your budget.
Distributed Systems are getting more balanced!
Distributed DBMSs: Just Upgrade Network?
Workload: standard TPC-C, with 50 warehouses per server.
27 machines of type: Two Xeon E7-4820 processors (each with 8 cores), 128 GB RAM
28 machines of type: Two Xeon E5-2660 processors (each with 8 cores), 256 GB RAM
How do we redesign DBMSs?
Problems of shared nothing

- Message Passing between nodes using IP stack (IPoIB)
- Bottlenecks due to load imbalance / skew
The Network-Attached-Memory Database Architecture (NAM-DB)

- Use RDMA for ALL communication
- Separate state and compute -> scalability & load balancing
NAM-DB: Naïve OLTP Protocol  
(based on Generalized SI)

<table>
<thead>
<tr>
<th>Client (Compute Server)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) <strong>RDMA-READ</strong> “Read TS”</td>
</tr>
<tr>
<td>2) <strong>RDMA-READ</strong> n (version/pointers)-pairs</td>
</tr>
<tr>
<td>3) <strong>RDMA-READ</strong> payload according to “Read TS” (abort if version is no longer available)</td>
</tr>
<tr>
<td>**4) <strong>RDMA-Atomic-Increment</strong> of “Commit TS”</td>
</tr>
<tr>
<td>5) For every record in write-set</td>
</tr>
<tr>
<td>5a) <strong>RDMA-Compare-And-Swap</strong> (64bit) the read record version to “Commit TS”. If fails, roll-back all changed records.</td>
</tr>
<tr>
<td>5b) <strong>RDMA-Write</strong> payload</td>
</tr>
<tr>
<td>5c) <strong>RDMA-Write</strong> to install new version by simply replacing n (record/version) pairs</td>
</tr>
<tr>
<td>5) <strong>RDMA-Send</strong> “Commit TS” to append “Commit TS” to “Committed TSs”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Memory Servers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read TS (64 bit): t₆</td>
</tr>
<tr>
<td>Commit TS (64 bit): t₉</td>
</tr>
<tr>
<td>Committed TSs: t₈</td>
</tr>
</tbody>
</table>

**TSs are bottleneck**
Alternative: Timestamp Vectors

Commit Timestamps

<table>
<thead>
<tr>
<th>Client-ID (8-bits)</th>
<th>TS (56 bits)</th>
</tr>
</thead>
</table>

Read Timestamps (vector)

- Highest committed TS by client 0 (56 bits)
- Highest committed TS by client 1 (56 bits)
- Highest committed TS by client 2 (56 bits)
- Highest committed TS by client 3 (56 bits)
- Highest committed TS by client k (56 bits)

Committed TSs can be distributed!
Example: Record and TS Vector

**Read-TS**

- \( t_{10} \)  
  - *Client_0*
- \( t_8 \)  
  - *Client_1*
- \( t_9 \)  
  - *Client_2*

**Record** *(Multiple versions)*

- \( c_0:t_{11} \)  
  - \( p_5 \)
- \( c_1:t_6 \)  
  - \( p_6 \)
- \( c_1:t_4 \)  
  - \( p_3 \)
- \( c_2:t_t \)  
  - \( p_2 \)
Example: Record and TS Vector

<table>
<thead>
<tr>
<th>Read-TS</th>
<th>Record (Multiple versions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_{10}</td>
<td>c_0 : t_{11}</td>
</tr>
<tr>
<td>t_8</td>
<td>p_5</td>
</tr>
<tr>
<td>t_9</td>
<td>c_1 : t_6</td>
</tr>
<tr>
<td></td>
<td>p_6</td>
</tr>
<tr>
<td></td>
<td>c_1 : t_4</td>
</tr>
<tr>
<td></td>
<td>p_3</td>
</tr>
<tr>
<td></td>
<td>c_2 : t_t</td>
</tr>
<tr>
<td></td>
<td>p_2</td>
</tr>
</tbody>
</table>

- **Similar to vector-clocks** but not really the same (Read-TS is a vector, a version consist of a single TS)
- **Can still guarantee SI** not only generalized SI
- **Avoids problems with long-running transactions and stale-reads**
OLTP: Scale-out Experiment on TPC-C

Binnig et al.: The End of A Myth: Distributed Transactions Can Scale. VLDB 2017

Shared-nothing (No RDMA)

Workload: standard TPC-C, with 50 warehouses per server.
27 machines of type: Two Xeon E7-4820 processors (each with 8 cores), 128 GB RAM
28 machines of type: Two Xeon E5-2660 processors (each with 8 cores), 256 GB RAM
OLTP: Scale-out Experiment on TPC-C

All Distributed transactions

- Shared-nothing (No RDMA)
- NAM-DB (wo locality)

Workload: standard TPC-C, with 50 warehouses per server.
27 machines of type: Two Xeon E7-4820 processors (each with 8 cores), 128 GB RAM
28 machines of type: Two Xeon E5-2660 processors (each with 8 cores), 256 GB RAM
OLTP: Scale-out Experiment on TPC-C

90% local transactions, 10% distributed

- Shared-nothing (No RDMA)
- NAM-DB (wo locality)
- NAM-DB (w locality)

Workload: standard TPC-C, with 50 warehouses per server.
27 machines of type: Two Xeon E7-4820 processors (each with 8 cores), 128 GB RAM
28 machines of type: Two Xeon E5-2660 processors (each with 8 cores), 256 GB RAM
OLTP: Scale-out Experiment on TPC-C

- Shared-nothing (No RDMA)
- NAM-DB (wo locality)
- NAM-DB (w locality)

Workload: standard TPC-C, with 50 warehouses per server.
27 machines of type: Two Xeon E7-4820 processors (each with 8 cores), 128 GB RAM
28 machines of type: Two Xeon E5-2660 processors (each with 8 cores), 256 GB RAM

FaRM: From the paper “No compromises: distributed transactions with consistency, availability, and performance”
Many (Important) Details Left Out

How to find records? *(see next slides)*

Fault-Tolerance, availability, and durability
*(NVM, replication and additional checks to undo-transactions of failed clients)*

Many, many possible optimizations *(caches in compute server, extend RDMA verbs by programmable NICs)*
NAM-DB: Remote Table Access

How to enable efficient access of remote tables (key and range lookups) on memory servers?

Key Question: How to design of tree-based indexes (i.e., B-tree like indexes) for RDMA?
NAM-DB: Remote Indexes
Ziegler et al.: Designing Distributed Tree-based Indexes for RDMA. SIGMOD’19

Index Distribution: How to distribute remote indexes across memory servers?

- **Coarse-grained Distribution**

- **Fine-grained Distribution**

Index Access: How to implement index accesses from compute servers?

- **One-Sided RDMA**: Memory-based (READ / WRITE)
- **Two-Sided RDMA**: RPC-based (SEND / RCV)
NAM-DB: Index Design Space

The “Design Matrix” for RDMA-based Indexes:

Index Distribution

<table>
<thead>
<tr>
<th>Index Access</th>
<th>Two-Sided</th>
<th>One-Sided</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><img src="image1" alt="Coarse-grained Distribution" /></td>
<td><img src="image2" alt="Fine-grained Distribution" /></td>
</tr>
<tr>
<td></td>
<td><img src="image3" alt="Remote Pointers" /></td>
<td><img src="image4" alt="Remote Pointers" /></td>
</tr>
</tbody>
</table>

*Assuming that each RDMA access needs to visit a different server*
Design 1: Coarse-Grained / 2-sided

1. Request key / range (2-sided)
2. Traverse index (on server)
3. Send result (2-sided)

Only one roundtrip BUT sensitive to skewness
Design 2: Fine-grained / 1-sided

1. Read Node (one-sided)
2. Read Node (one-sided)
3. Read Leave(s) (one-sided)

Multiple roundtrips BUT better load balancing
Design 3: Hybrid (Fine/Coarse)

1. Request key (2-sided)
2. Traverse tree (Server thread)
3. Send pointer (2-sided)
4. Read Leave(s) (one-sided)

One roundtrip for index traversal + Multiple reads of data for better load balancing
NAM-DB: Evaluation (Indexes)

Index Workloads:

<table>
<thead>
<tr>
<th>Workload</th>
<th>Point Queries</th>
<th>Range Queries (sel=s)</th>
<th>Inserts</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>100%</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>95%</td>
<td>5%</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>50%</td>
<td>50%</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>50%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Throughput (Workload A+B, Skewed):

Setup:
- 4 Memory Servers
- 6 Compute Servers
- No co-location
- Data 100M unique keys
Networks are becoming smart

Smart NICs & Switches

Software-defined-Networking

Use network to offload computation from a distribute DBMS → In-Network-Processing (INP) of SQL operators?
A motivating example

- Data warehousing scenario: star schema
- Fact table **not co-partitioned** with dimensions B and C

### Star Schema:

```
SELECT * FROM A JOIN B JOIN C
```

![Diagram of a star schema with a fact table A and dimensions B and C.](Image)
Traditional Distributed Execution

Steps:
1. Shuffle table B & build HT
2. Shuffle table A & probe HT of B
3. Shuffle table C & build HT
4. Shuffle intermediate A $\bowtie$ B & probe HT of C

Observation:
Re-shuffling of large fact table A is expensive
Moreover, skew is a problem

Non-uniform foreign-key distribution → network skew

Network link to one node is getting congested
- Increased shuffling time
- Increased processing time on straggler
Case for In-network Processing

Steps:
1. Send table B and C to switch and build HT in switch
2. Stream fact table A through switch & probe HTs

Take away:
• Avoids re-shuffling of large fact table A
• Not sensitive to skew
Current P4 switches (e.g., Barefoot Tofino) have many limitations

Our own prototype switch:

- FPGA chosen as processing unit
- Based on network focused FPGA dev board (NetFPGA SUME)
- 2 x 4GB DDR3 memory @ 800MHz
INP-Join: Experimental Evaluation

**Query:** A ⋈ B ⋈ C ⋈ D

**Data:** A: $5 \times 10^6$ to $5 \times 10^9$ tuples - B, C & D: $50 \times 10^6$ tuples

---

**Without Skew**

**With Skew**
Conclusions and Future Work

The next generation of high-speed networks requires us to rethink distributed database systems.

Network-Attached Memory (NAM) as a general distributed architecture to take advantage of fast networks.

Other workloads: Streaming, ML, Graphs, ...

Networks are not only getting “faster” but also “smarter”