Fast Networks
and the Next Generation of
Transactional Database Systems

Tim Kraska <kraska@mit.edu>
Erfan Zamanian
<erfan_zamanian_dolati@brown.edu>

Is graduating very soon and is looking industry jobs
Imagine a new world
A better world
and with **Scalable Distributed Transactions**
For centuries app & system developers try to avoid distributed transactions with huge pain
- Static- and dynamic data partitioning
- Data-aware scheduling
- Replication schemes
- ....
But if distributed transaction would scale, all this pain would go away
Yet, the relieve does not come from....
But changes in the network
RAM

Memory Bus (4 channels)
51.2 GB/s Half-duplex (total)

Memory Controller

CPU 1

Core 1  Core 2  Core 3  Core 4
Core 5  Core 6  Core 7  Core 8

PCle Gen3 (40 lanes), 39.4 GB/s full-duplex
78.8 GB/s total

Dualport RNIC (16 PCI lanes)

FDR 4x (2ports)
13.6 GB/s full-duplex
27.3 GB/s total

Assuming equal read/write workload
CPU 1

- Core 1
- Core 2
- Core 3
- Core 4
- Core 5
- Core 6
- Core 7
- Core 8

Memory Bus (4 channels)
51.2 GB/s Half-duplex (total)

Memory Controller

PCle Gen3 (40 lanes), 39.4 GB/s full-duplex
78.8 GB/s total

RAM

Dualport RNIC (16 PCI lanes)
Two FDR 4x (2ports)
27.3 GB/s full-duplex
54.5 GB/s total

Dualport RNIC (16 PCI lanes)

Assuming equal read/write workload
QPI (2 links)
32 GB/s
Full-duplex
64 GB/s total

CPU 1
Core 1
Core 2
Core 3
Core 4
Core 5
Core 6
Core 7
Core 8

CPU 2
Core 1
Core 2
Core 3
Core 4
Core 5
Core 6
Core 7
Core 8

Memory Bus (4 channels)
51.2 GB/s Half-duplex (total)

PCIe Gen3 (40 lanes), 39.4 GB/s full-duplex
78.8 GB/s total

Dualport RNIC
(16 PCI lanes)

Dualport RNIC
(16 PCI lanes)

Two FDR 4x (2ports)
27.3 GB/s full-duplex
54.5 GB/s total

Dualport RNIC
(16 PCI lanes)

Dualport RNIC
(16 PCI lanes)

Two FDR 4x (2ports)
27.3 GB/s full-duplex
54.5 GB/s total
Total Bandwidth over 2 Sockets with **Four** Dual-port FDR 4x NICs

2 servers, each with: Xeon E5-2660 v2 CPUs, 256GB of DDR3-1600 RAM, **FOUR** 2-port InfiniBand FDR 4x NIC per machine
Intel is working on 400Gbs NIC

“There is no shared nothing if you have 400Gbs NIC”

--- David Cohen, Intel
Why are distributed transactions considered not scalable

- Message Overhead
- Using up network BW
- Higher latency of distributed trxs
But with the next generation of networks (e.g., Infiniband HDR)

- Low/zero CPU overhead (2-sided RDMA: low overhead, 1-sided RDMA: zero overhead)
- High bandwidth (is getting on par with memory BW)
- Low latency (5x-30x better than 10Gb Ethernet)
- Higher latency of distributed trxs

Message Overhead

Using up network BW

Higher latency of distributed trxs
But with the next generation of networks (e.g., InfiniBand HDR)

Is Higher Latency a problem?
1. No contention point → No
2. Contention point → Kinda, but in this case the workload is per definition not scalable (more later)

Higher latency of distributed trxs

Low latency

(5x-30x better than 10Gb Ethernet)
Can we simply upgrade the network with IPoIB and we are done???
Quick Experiment

**Shared-nothing In-Memory Distributed DB**
- Data partitioned horizontally on servers
- One server acts as the trx’s coordinator
- Generalized Snapshot Isolation guarantees
- 100% distributed transactions

1 NIC (single-port) Ethernet
**1.25 GB/s**

IP over IB

1x NIC (dual-port) IB FDR 4x
**13.64 GB/s**

---

S. Elnikety et al. Database replication using generalized snapshot isolation. In SRDS, 2005
Y. Lin et al. Middleware based data replication providing snapshot isolation. In SIGMOD, 2005
Ethernet vs IPoIB
Transaction Throughput (in 1000 trx/s)

8 servers, each with: Xeon E5-2660 v2 CPUs, 256GB of DDR3-1600 RAM, one 2-port InfiniBand FDR 4x NIC
Workload: 100% distributed transactions, every transaction spans 3 servers
Ethernet vs IPoIB Transaction Throughput (in 1000 trx/s)

8 servers, each with: Xeon E5-2660 v2 CPUs, 256GB of DDR3-1600 RAM, one 2-port InfiniBand FDR 4x NIC
Workload: 100% distributed transactions, every transaction spans 3 servers
What are the key problems

• Remote-Direct-Memory-Access (RDMA) → Requires a new memory layout

• **New bottlenecks**
  (e.g., creating global timestamps for snapshot isolation)
New-Generation Network: RDMA

RDMA: **Remote Direct Memory Access**

Three modes: IPoIB, one-sided, two-sided
New-Generation Network: RDMA

RDMA: **Remote Direct Memory Access**
Three modes: IPoIB, **one-sided**, two-sided

![Diagram showing conventional network and 1-sided RDMA](image-url)
4 Key Results from 4 Years

NAM-Architecture
Carsten Binnig, Andrew Crotty, Alex Galakatos, Tim Kraska, Erfan Zamanian: The End of Slow Networks: It’s Time for a Redesign. PVLDB 9(7): 528-539 (2016)

Scalable SI Protocol

Active Replication

Contention-centric Clustering
4 Key Results from 4 Years

NAM-Architecture
Carsten Binnig, Andrew Crotty, Alex Galakatos, Tim Kraska, Erfan Zamanian: The End of Slow Networks: It's Time for a Redesign. PVLDB 9(7): 528-539 (2016)

Scalable SI Protocol

Active Replication

Contention-centric Clustering
The Network-Attached-Memory (NAM) Architecture

Carsten Binnig, Andrew Crotty, Alex Galakatos, Tim Kraska, Erfan Zamanian: The End of Slow Networks: It's Time for a Redesign. PVLDB 9(7): 528-539 (2016)
NAM Architecture

• **Memory Servers**
  • Expose their memory to compute servers
  • Store multiple versions of each record

• **All trxns are distributed by default**

• **A logical separation, not a physical**
  • Physical co-location to leverage locality as an optimization

• **Optimized for Remote-Direct-Memory-Access (RDMA)**
  (no messages) → avoids CPU overhead

• **Benefits:**
  • Compute and storage can independently scale
  • Can efficiently handle data imbalance
  • Helps us to understand how far RDMA and fast networks can go and where messages are really needed.
4 Key Results from 4 Years

NAM-Architecture

Scalable SI Protocol

Active Replication

Contention-centric Clustering
Goal

Explore if we can design an entire SI protocol using one-sided RDMA read/write

(RPC/2-sided is an optimization)
An RDMA Friendly Memory Layout

1) Fast scan and retrieval of single key for the most recent version
   - Record offsets as identifiers (memory location is known from the record-id)
   - Requires to re-organize data from time to time

2) Circular buffer containing version number and payload pointers for older versions

See paper for more details and other optimizations
Timestamp Oracle

Needed to determine the currently valid version-id

Global counter updated w/ atomic RDMA

- Doesn’t scale with number of servers
- As slow as the slowest transaction
- Requires either
  - Several roundtrip of atomic RDMA
  - 2-sided send/receive
Solution: Timestamp Vector

• Similar to vector-clocks but with two important differences:
  1. **Read-TS is a vector, a version consist of a <single thread-id, commit-counter> pair**
     • Version numbers are small (2 integers)
     • Faster comparisons
  2. **Monotonically increasing**: no branching, no need for epochs

• Still SI-guarantees

• Avoids problems with long-running transactions and stale-reads

* For monotonicity NIC has to ensure left-to-right write order and no DMA re-ordering
The Basic RSI Protocol

**Client (Compute Server)**

1) **RDMA-READ** Timestamp
2) for \( r \in \text{read-set} \)
   1) **RDMA-READ** current version
   2) Check Version, if needed **RDMA-READ** older version from buffer
3) Increment local thread TS
4) for \( w \in \text{write-set} \):
   1) **RDMA-Compare-And-Swap** record header (set lock-bit).
   2) If compare-and-swap fails or latest version is new, abort and revert changes
   3) If not move, **RDMA-WRITE** record to circular buffer
   4) **RDMA-WRITE** version in place
5) **RDMA-WRITE** to update Thread-Counter

**Distr. Shared Storage**

**Timestamp**

<table>
<thead>
<tr>
<th>Thread ID</th>
<th>Last commit TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
</tr>
</tbody>
</table>

**Data**

(versions’ headers & payloads)

________________
________________
________________

38
Scale-Out Experiment on TPC-C

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
Scale-Out Experiment on TPC-C

All Distributed transactions

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
Scale-Out Experiment on TPC-C

90% local transactions, 10% distributed

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
Scale-Out Experiment on TPC-C

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”
4 Key Results from 4 Years

NAM-Architecture
Carsten Binnig, Andrew Crotty, Alex Galakatos, Tim Kraska, Erfan Zamanian: The End of Slow Networks: It's Time for a Redesign. PVLDB 9(7): 528-539 (2016)

Scalable SI Protocol

Active Replication

Contention-centric Clustering
Context of this Work

- Cluster over Local area network (LAN)
- Striped master model

Node 1

Node 2

Node 3
All Existing Replication Protocols
Trade CPU for Network Bandwidth
Active-Passive Replication (i.e. log shipping)

1. Perform txn logic locally

Primary

Backup A

Backup B
Active-Passive Replication (i.e. log shipping)

1. Perform txn logic locally

2. Send “redo log” to backups
Active-Passive Replication (i.e. log shipping)

1. Perform txn logic locally

2. Send “redo log” to backups

3. Apply “redo log” to replica copies
Active-Passive Replication (i.e. log shipping)

1. Perform txn logic locally

2. Send “redo log” to backups

3. Apply “redo log” to replica copies

4. Receive acks and apply txn updates to primary copy
Active-Active Replication

- Sequencer sends **inputs of transactions** to replicas
- Each replica runs transactions **deterministically**
CPU vs. Network Tradeoff

<table>
<thead>
<tr>
<th></th>
<th>Active-Passive</th>
<th>Active-Active</th>
</tr>
</thead>
</table>

- **CPU demand**
- **Network demand**
CPU vs. Network Tradeoff

<table>
<thead>
<tr>
<th></th>
<th>Active-Passive</th>
<th>Active-Active</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Demand</td>
<td>Replay logs</td>
<td>&lt; Duplicate execution</td>
</tr>
</tbody>
</table>
## CPU vs. Network Tradeoff

<table>
<thead>
<tr>
<th>CPU Demand</th>
<th>Active-Passive</th>
<th>Active-Active</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Demand</td>
<td>Replay logs</td>
<td>Duplicate execution</td>
</tr>
<tr>
<td>Network Demand</td>
<td>Send logs</td>
<td>Send input</td>
</tr>
</tbody>
</table>

**Active-Passive**
- CPU Demand: Replay logs
- Network Demand: Send logs

**Active-Active**
- CPU Demand: Duplicate execution
- Network Demand: Send input

**CPU demand** vs. **Network demand**
## CPU vs. Network Tradeoff

<table>
<thead>
<tr>
<th></th>
<th>Active-Passive</th>
<th>Active-Active</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU Demand</strong></td>
<td>Replay logs</td>
<td>Duplicate execution</td>
</tr>
<tr>
<td><strong>Network Demand</strong></td>
<td>Send logs</td>
<td>Send input</td>
</tr>
</tbody>
</table>

### Diagram Description

- **CPU Demand**: Active vs. Passive
- **Network Demand**: Send logs vs. Send input

### Active/Active (Log Shipping)

- CPU Demand: Active
- Network Demand: Active

### Active/Passive

- CPU Demand: Passive
- Network Demand: Active

The diagram illustrates the tradeoff between CPU and network demand in different configurations.
New Idea: **Active Memory** Replication

Key idea: Coordinator **directly updates** memory states of backup nodes using **one-sided RDMA**

![Diagram showing CPU demand and network demand with Active/Active and Active/Passive (Log Shipping) options]
The Network-Attached-Memory (NAM) Architecture
Challenge: Fault Tolerance

All or nothing replication
Challenge: Fault Tolerance

All or nothing replication

Solution: **UNDO logging**

Write UNDO log to a backup *before* writing the actual data
**Evaluation: Network Bandwidth**

2-phase locking not SI

Workload: YCSB, 1KB records w/ 10 columns, 5M records / server. Each txn has 10 RMW ops
8 nodes: 2 Xeon E7-4820 processors (each with 2x10 cores), 256 GB RAM, 3 way-replication
Just for Ippokratis:
Comparison Against Query Fresh

<table>
<thead>
<tr>
<th>Throughput (k txns/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>QF (sync)</td>
</tr>
<tr>
<td>QF (pipelined)</td>
</tr>
<tr>
<td>QF (multi-primary)</td>
</tr>
<tr>
<td>Log Shipping</td>
</tr>
<tr>
<td>Active-Memory</td>
</tr>
</tbody>
</table>

4 Key Results from 4 Years

NAM-Architecture

Scalable SI Protocol

Active Replication

Contenion-centric Clustering
Traditional Partitioning

```
update("phil", bal-10k)
update("rose", bal+8k)
update("henry", bal+2k)
```

```
bal1 = select("dave")
bal2 = select("jack")
bal3 = select("henry")
return bal1 + bal2 + bal3
```

```
bal1 = select("rose")
bal2 = select("bob")
return bal1 + bal2
```

```
bal1 = select("rose")
bal2 = select("bob")
return bal1 + bal2
```

```
bal1 = select("rose")
bal2 = select("bob")
return bal1 + bal2
```

Chiller Partitioning

Conclusions

“There is no shared nothing, if you have 400Gbs NIC”

NAM-Architecture
Carsten Binnig, Andrew Crotty, Alex Galakatos, Tim Kraska, Erfan Zamanian: The End of Slow Networks: It’s Time for a Redesign. PVLDB 9(7): 528-539 (2016)

Active Replication

Scalable SI Protocol

Contention-centric Clustering
Conclusions

“There is no shared nothing, if you have 400Gbs NIC”


Erfan Zamanian erfan_zamanian_dolati@brown.edu