The End of a Myth: Distributed Transactions Can Scale

Erfan Zamanian, Carsten Binnig, Tim Harris, and Tim Kraska

Rafael Ketsetsides
Why distributed transactions?
Cheaper for the same processing power
The problem with distributed transactions
Adding more machines doesn’t increase performance

![Graph showing scalability of NAM-DB](image_url)
Adding more machines doesn’t increase performance

Figure 4: Scalability of NAM-DB

Throughput (M trxs/sec) vs. Cluster Size

- Clustered with TCP/IP

Accurate representation of a DB administrator
Why can’t they scale?
Background: TCP/IP
TCP/IP is computationally expensive

Complex design

---

TCP/IP

- SYN
- SYN/ACK
- ESTABLISHED
- FIN/ACK
- FIN+ACK/ACK
- ACK/

TCP/IP states:
- CLOSED
- LISTEN
- SYN
- LISTEN/-
- SYN/SYN+ACK
- SYN/SENT
- CLOSE/
- FIN/ACK
- DATA EXCHANGE OCCURS
- ESTABLISHED
- FIN/ACK
- TIME WAIT
- CONNECT/SYN
- SYN/SYN+ACK
- LISTEN
- SYN+ACK
- SYN/SYN+ACK
- FIN WAI 1
- FIN WAI 2
- LAST ACK
- Passive open
- Active open
- CLOSE WAIT
- FIN+ACK/ACK
- FIN/ACK
- FIN WAI 1
- FIN WAI 2
- FIN/ACK
- TIME WAIT
- CLOSING
- CLOSE/FIN
- LAST ACK
- PASSIVE OPEN
- SYN/SYN+ACK
- LISTEN
- FIN+ACK/ACK
- FIN/ACK
- FIN WAI 1
- FIN WAI 2
- FIN/ACK
- TIME WAIT
- CLOSING
- CLOSE/FIN
- LAST ACK
TCP/IP is computationally expensive

Fixed window size causes linear overhead
How do we replace TCP/IP?
Background: RDMA
Remote Direct Memory Access (RDMA)
RDMA is great!

- Low latency
- High throughput
- Supported by InfiniBand
RDMA is hard to use

**One-sided communication:**
Receiver is not notified of connection
RDMA is hard to use

One-sided communication:
Receiver is not notified of connection

So far, most solutions that use RDMA, only do so for part of the design
We need a system redesign
Main design

*Actually, the hash table might point to a different memory server*
set when entry is moved to overflow region

set when entry may be deleted

set when entry is being committed

Data entries

<table>
<thead>
<tr>
<th>Thread-Id</th>
<th>Commit-Timest.</th>
<th>Moved 1 Bit</th>
<th>Deleted 1 Bit</th>
<th>Locked 1 Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>29 Bits</td>
<td>32 Bits</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Old-version Buffers

In the following, we explain further optimizations to make NAM-DB for certain broadcast operations. This direction leverages the message ordering guarantees provided by In-tems. We are currently investigating if we can solve this by partitioning the network bandwidth of the server so that every machine now only stores a fraction of the vector. Unfortunately, partitioning will improve the bandwidth per server as every machine now needs to update only a single slot in its local memory. While this reduces the network load, the runtime of each transaction is only reduced by the reduced network load, the runtime of each transaction is not significantly reduced. Since the number of transaction execution threads per compute server is bounded (if we use one dedicated thread per execution thread on one machine share one timestamp slot), the timestamp oracle remains scalable.

Further Optimizations

In our evaluation, we found that with RDMA, the latency for messages of up to 8-byte value that can be atomically updated by a remote compare-and-swap operation is negligible. Moreover, we also store other data in the header section that makes sense to store the data in a fixed-length field that has the maximal required length inside a record. The data section is a full copy of the version information of a record and is set during the commit phase. Moreover, we also store other data in the header section that can be atomically updated by a remote compare-and-swap operation.

5.1 Scalable Memory Management

In the section of a record that refers to the variable-length part, which could be supported by storing an additional pointer in the data section that pointer can then be continuously copied to an old-version buffer. That way, slots in the old-version buffer are shared between transaction execution threads. As a result, the old-version buffer can be shared between transaction execution threads. As a result, the old-version buffer can be shared between transaction execution threads.

5.2 Scalable Record Layout

In Figure 3: Version Management and Record Layout, we show the layout of a record.

Old-version Buffers

- **Data-Buffer**
- **Old-Version Buffers**
- **Head**
- **Tail**
- **Moved**

For each record, we store a 32-bit value: a deleted-bit indicates if a version was already moved from the record. The next version of the transaction execution thread that installed the version in the record is marked for deletion and can be safely garbage collected.

In the record layout, we distinguish between the old-version buffers and the current version.

- **Old-Version Buffers**
- **Current Version**

The header section describes the metadata of a record. In our design decisions are made to make distributed transactions scalable rather than optimize for locality. Therefore, we use continuous memory regions for the most recent versions, transferring the most recent version from compute servers. The header encodes different variables: The first variable is the metadata of a record. In the header, we store the payload. The data section is a full copy of the record's payload. The data section is a full copy of the record's payload.

In this section, we first discuss the details of the multi-phase garbage collection algorithm. Subsequently, we explain the memory layout in more detail. In the following, we first explain the memory layout in more detail.
Data entries

set when entry is moved to overflow region

set when entry may be deleted

set when entry is being committed

The scheme to store multiple versions of a database record

For each record, we store a 29-bit value: a timestamp, a 32-bit, and a 5-bit. The timestamp is locked, the 32-bit is to prevent overflow, and the 5-bit is used to store the version.

The first version, v1, is stored in a dedicated memory region. Whenever a record is updated by a transaction (i.e., a new version needs to be inserted), the current version is moved to an overflow region.

In this section, we first discuss the details of the multi-versioning scheme that contains additional metadata and a version information of a record and are set during the commit phase. Moreover, we also store other data in the header section that is used for version management, each represented by a record which represents a particular version. Currently, we continuously copied to an memory server is shown in Figure 3. The main idea is that the reduced network load, the runtime of each transaction is heavily reduced, leading instead to a lower abort rate.

In the following, we explain further optimizations to make those two optimizations, a single timestamp server is already able to sustain over 140 million trxs/sec using a single dual-port FDR 4x NIC.

In our evaluation, we found that with pre-fetching increases the staleness of the beginning of each transaction can cause a high network load for large transaction execution thread pools on large clusters. However, due to pre-fetching increases the staleness of the beginning of each transaction can cause a high network load for large transaction execution thread pools on large clusters.

In the following, we explain further optimizations to make our current implementation, we use an atomic RDMA fetch-and-add operation to increase the counter value. Since the number of transaction execution threads per compute server is bounded (if we use one dedicated thread per transaction execution threads on one machine share one timestamp slot having only one slot)

Old-Version Buffers

Current Version

Header, Data

Copy-on-update

Continuous Move

Overfow Region

unwanted entries are lazily GCed in continuous chunks

In this section, we first discuss the details of the multi-versioning scheme that contains additional metadata and a version information of a record and are set during the commit phase. Moreover, we also store other data in the header section that is used for version management, each represented by a record which represents a particular version. Currently, we continuously copied to an memory server is shown in Figure 3. The main idea is that the reduced network load, the runtime of each transaction is heavily reduced, leading instead to a lower abort rate.

In the following, we explain further optimizations to make those two optimizations, a single timestamp server is already able to sustain over 140 million trxs/sec using a single dual-port FDR 4x NIC.

In our evaluation, we found that with pre-fetching increases the staleness of the beginning of each transaction can cause a high network load for large transaction execution thread pools on large clusters. However, due to pre-fetching increases the staleness of the beginning of each transaction can cause a high network load for large transaction execution thread pools on large clusters.
Main design

- Read before fetching data
- Each cell is the commit timestamp of a thread
- Stored in a single Memory server

Optimizations

- Fetched by a dedicated thread in each Compute server (big ts reader)
- Threads in a Compute server might share commit timestamp (compression)
Further notes

- Compute and Memory servers might coexist, taking advantage of locality
- Timestamp vectors may be partitioned
- Secondary indexes (B+-trees, hash tables)
Is it good enough?
Linear scalability

![Graph: Linear scalability](image)

- **Classic (2-sided)**
- **NAM-DB w/o locality**
- **NAM-DB w locality**

Cluster Size vs. Throughput (M trxs/sec)
Linear scalability

![Graph showing Linear scalability](image)

- **Throughput (M trxs/sec)** vs **Cluster Size**
  - Classic (2-sided)
  - NAM-DB w/o locality
  - NAM-DB w locality

![Diagram](image)
Experiments
Setup

- TPC-C benchmark
- 2011-released InfiniBand (FDR)
- Two clusters, with 8 and 57 machines
Experiment 1: System scalability

![Graph showing throughput and latency vs. cluster size]
Experiment 2: Scalability of the Oracle

Figure 9 shows the total number of performed operations (a) Throughput and (b) Abort Rate for different configurations. The graph illustrates the performance of Classic (global counter), NAM-DB (no opt.), NAM-DB + compression, NAM-DB + bg ts reader, and NAM-DB + both opt. as the number of clients increases. The x-axis represents the number of clients, while the y-axis shows the throughput in terms of timestamp ops/sec. The performance is measured across the different configurations, demonstrating the impact of optimizing RDMA queue pairs on scalability.
Experiment 3: Effect of Locality

![Graph showing throughput and latency vs probability of distribution for NAM-DB with and without locality.](image)

The graphs illustrate the effect of locality on throughput and latency. The x-axis represents the probability of distribution, while the y-axis shows the throughput (in million transactions per second) and latency (in microseconds) for transactions under different skew conditions.

- **Throughput (M trxs/sec):**
  - NAM-DB w/o locality
  - NAM-DB w locality

- **Latency (us):**

As the probability of distribution increases, the throughput decreases and latency increases. The locality optimization shows a significant improvement in both throughput and latency compared to the non-locality case.
Experiment 4: Effect of Contention

![Graph](image)

- **Cluster Size** vs. **Abort Rate**
  - **Uniform**
  - **Low Skew**
  - **Medium Skew**
  - **High Skew**
  - **Very High Skew**

- **Cluster Size** values: 2, 3, 5, 7

**Legend**:
- Uniform (almost zero; not visible)
- High skew
- Low skew
- Medium skew
- Very high skew
Related Work

FaRMs design is centered around locality. While this paper was under review, like NAM-DB, FaSST also focuses on scalability but the authors took a different approach by building an efficient RPC abstraction on top of 1-to-1 communications.

The case that distributed transactions can now scale, whereas existing DBMSs do not play an important role. RAC does not directly take advantage of the redesign of the full database stack. SpinningJoins sug-
gests a new architecture for RDMA. Different from our work, it does not be that influential on performance. Due to their decision not to establish queue pairs (QPs) per NIC limit the scalability. The reason is that queue pairs are not needed to be established for every client-thread (as currently done in NAM-DB) but rather a key/value stores. We leverage some of these re-
results to build our distributed indexes in NAM-DB, but transac-
tions and query processing are not discussed in these papers.

FaRM [14, 13] is another related project which was published recently and varied the number of queue pairs over flow the cache, potentially causing the performance to de-
crease. Moreover, at every round, a client thread chooses one of its queue pairs randomly, and issues a fixed-
size READ to the server.

Different from our work, FaSST implements serializability, whereas we show how to scale snapshot isolation, a property which provides better performance for read-heavy workloads and the design is likely more sensitive to data locality. Fi-

Finally, SQLServer [28] uses RDMA to extend the impact of treating RDMA as a first-class citizen, they treat RDMA as an afterthought. Moreover, they use a centralized commit manager to coordinate distributed transactions, which is likely not even support it).

Industrial-strength products have also adopted RDMA in their systems, their system is not able to take full advantage of verbs. This design minimizes the size of queue pair state stored leveraging the NIC as co-processors to access remote mem-
ory, and the design is likely more sensitive to data locality. Fi-

Another recent work [29] is similar to our design since it

8-byte (64-byte) READs. However, with 256-byte READs,

The number of queue pairs has almost no impact. In this case,

Overall performance, but mainly for small messages. For ex-

The number of queue pairs has almost no impact. In this case,

Related Projects

Figure 7: Scalability of Oracle TPC-C

Figure 9 shows the total number of performed operations

Exp.5: Scalability of RDMA Queue Pairs

(b) Abort Rate

(b) Abort Rate

(a) Throughput

(b) Abort Rate

(a) Throughput

(a) Throughput

20

2500

3000

3500

4000

500

1000

1500

2000

2500

3000

3500

4000

0

50

100

150

200

250

300

350

400

M operations/s

# Queue Pairs

(b) Abort Rate

(a) Throughput

20

2500

3000

3500

4000

500

1000

1500

2000

2500

3000

3500

4000

0

50

100

150

200

250

300

350

400

M operations/s

# Queue Pairs

Figure 5: Scalability of Oracle TPC-C

Figure 9: Scalability of QPs

8-byte READs

64-byte READs

256-byte READs

Figure 5: Scalability of Oracle TPC-C

Experiment 5: Scalability of RDMA

Queue Pairs

8-byte READs

64-byte READs

256-byte READs
Gaps in the logic
Future work
Future work

• Optimize for OLAP
• Reliably emulate large clusters and perform experiments
• Analyze performance, and optimize constants
• Explore collocation methods
• Explore secondary indexes
Thank you!
Media sources

E. Zamanian et al. The end of a myth: Distributed transactions can scale

C. Binnig et al. The end of slow networks: It’s time for a redesign