The End of a Myth: Distributed Transactions Can Scale

NAM DB (Cont.)

VLDB 2017

Geoffrey Xue

The End of Slow Networks: It's Time for a Redesign

Carsten Binnig Andrew Crotty Alex Galakatos Tim Kraska Erfan Zamanian
Department of Computer Science, Brown University
{firstname_lastname}@brown.edu

1 year later...

The End of a Myth: Distributed Transactions Can Scale

Erfan Zamanian
Brown University
erfanz@cs.brown.edu

Carsten Binnig
Brown University
carsten_binnig@brown.edu

Tim Harris
Oracle Labs
timothy.l.harris@oracle.com

Tim Kraska
Brown University
tim_kraska@brown.edu

NAM DB - Introduction

Shared-nothing logically decoupled

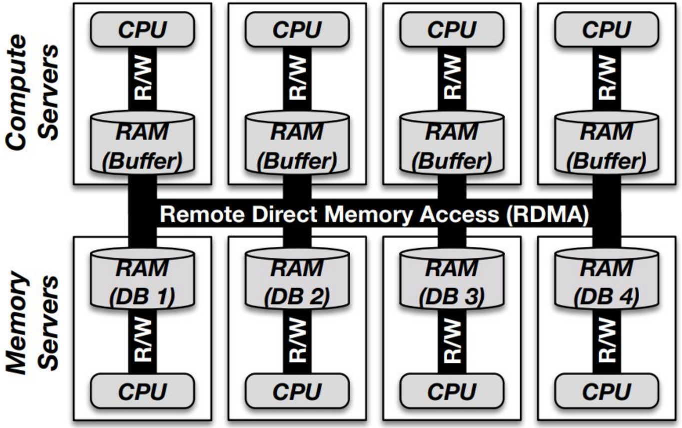
Compute Servers

Query Processor, Transaction Manager

- Each **thread** executes a transaction

Memory Servers

Stores Data, Metadata, Timestamps, etc.



The End of Slow Networks: It’s Time for a Redesign (2016)

Limitations

- Versioning and Snapshots don’t actually work
  - Aborts trxs requiring an older snapshot
- No fault tolerance

Transactions

NAM DB - Continued

Snapshot Isolation Baseline

Timestamp Oracle

Multi-versioning

Memory Management

Indexes, Catalog

Fault Tolerance

Snapshot Isolation Baseline

Assumptions

- Up-to-date Catalog Service (Data → Address lookup)
- Ignoring fault tolerance
- One version only

Snapshot Isolation Baseline*

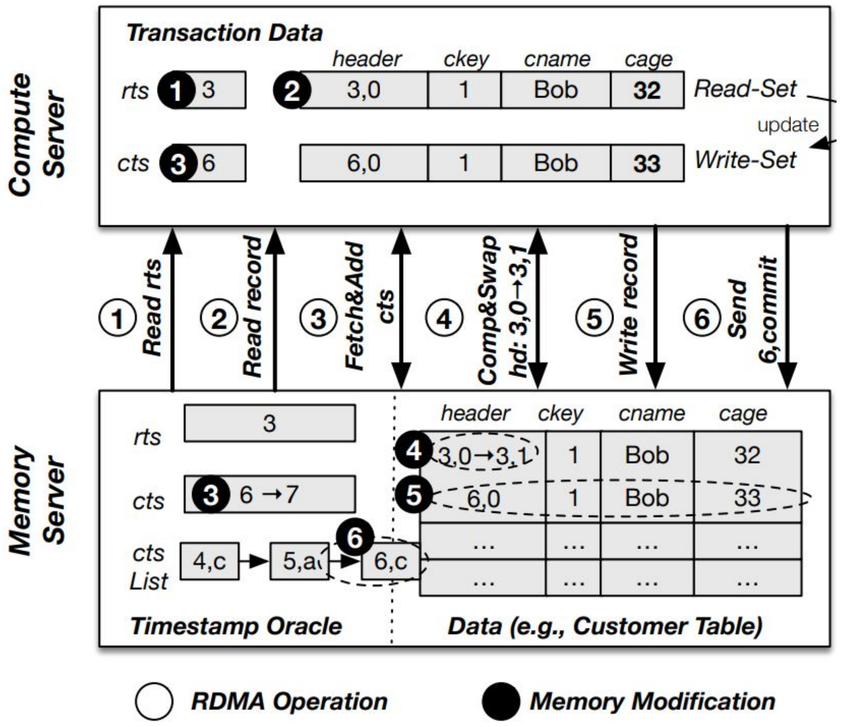
Read*

1. Get **read timestamp** from global
2. Execute transaction, build read/write sets

Commit

3. Get unique **commit timestamp***
4. Locks **record blocks** in write set
5. Writes records
6. Send commit status

*Different from 2016 paper



Timestamp Oracle

Scan tail of completed to update read

Snapshot Isolation Baseline

Challenges

- Global fetch-and-add timestamps do not scale well
- Stragglers/Long transactions → High aborts from read update slowdown
- No Fault-tolerance

Timestamp Oracle

Timestamp Vector

- For **n** threads
- Stored in single memory server

$$T_R = \langle t_1, t_2, t_3, ..., t_n \rangle$$

Record's Latest Update

- Thread identifier
- Commit Timestamp

$$T_C = \langle i, t_i \rangle$$

Timestamp Oracle - Snapshot Isolation

Read

$$T_R = \langle t_1, t_2, t_3, ..., t_n \rangle$$

1. Get **read timestamp** ~~from global~~ by getting copy of **timestamp vector**
2. Execute transaction, build read/write sets. For each record:
    a. Get newest record $T_C = \langle i, t_i \rangle$
    b. Ensure that record is **visible** to read snapshot $t \leq t_i$
    c. Otherwise, grab older version

Timestamp Oracle - Optimizations

Detached Fetch Thread → Fetch/cache T_R

Timestamp Vector Compression → Compute Server granularity

- Threads share slot. Atomically update between each other, like original

Partitioning → Loses strict monotonicity

Multi-versioning

Current, Old-Version, Overflow

Current → Old-Version

- Handled on commit

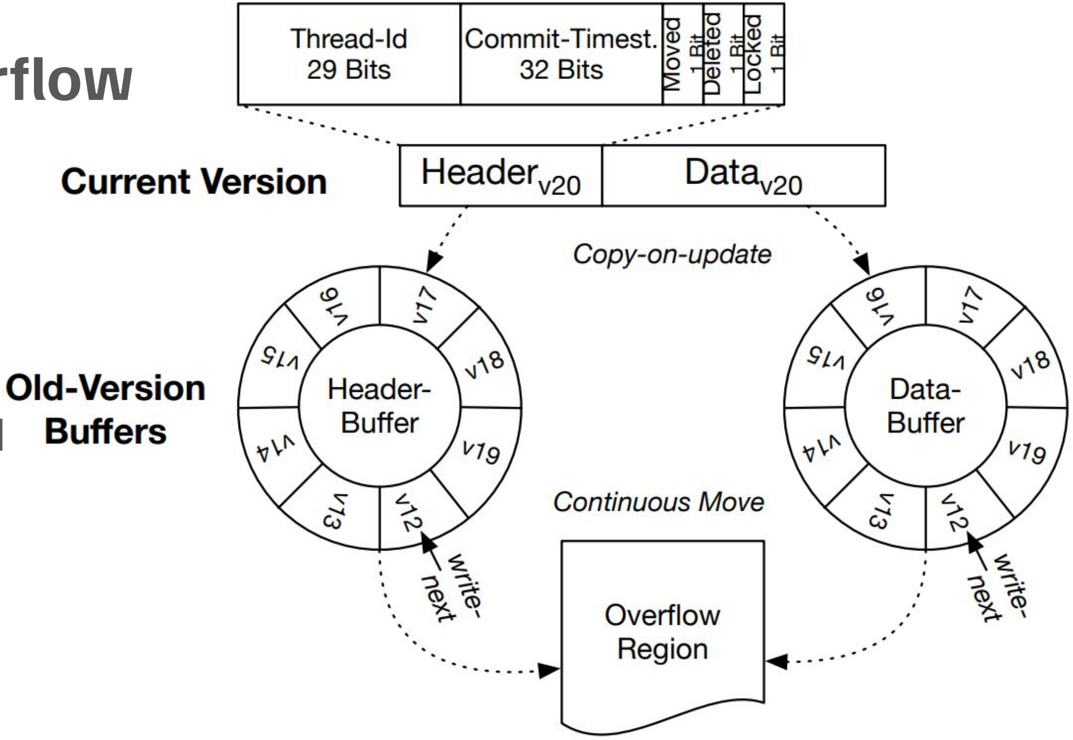
Old-Version → Overflow

- Detached version-mover thread
- Sets “moved” bit to 1

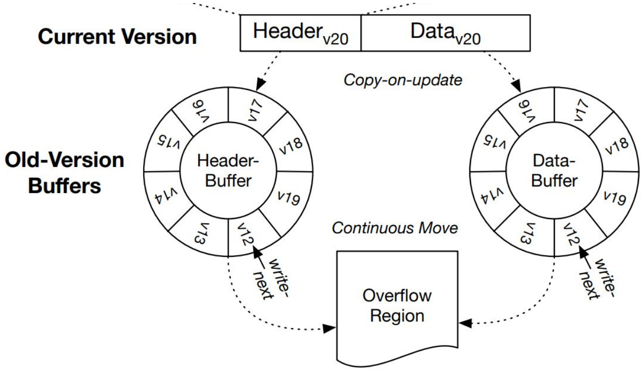
Overflow → Deletion: Timer-based

Table: Key → 4 pointers

$$T_C = \langle i, t_i \rangle$$



Multi-versioning - Current → Old-Version



Commit

1. *Get unique commit timestamp*
2. *Locks record blocks in write set*
3. Writes records. **For each record**
    a. Look at pointer for record to evict
    b. If record has "moved" bit 1, replace. Otherwise, advance
    c. Copy existing "current version" to old
4. *Send commit status*

Memory Management

Pinning/Registering is expensive

Allocates large region first

When compute adds row, allow extend as needed

Garbage Collection (See multi-versioning)

Indexes, Catalog

Indexes

Hash Index → Linked List, Partitioned, One-sided RDMA

Put in single memory region to avoid pointer chasing

B+ Tree Index → Partitioned, Two-sided RDMA

Catalog

Lookup for tables and indexes, partitioned, Two-sided RDMA

Cache, update if **catalog version** metadata is stale on fetch

Fault Tolerance - Memory Servers

Log

Threads RDMA write log to multiple memory servers

In the form $< T, S >$

- Timestamp vector
- Statement executed

Detached Checkpoint Thread → Writes checkpoints to disk to truncate log

Fault Tolerance - Memory Servers

Commit

1. *Get unique commit timestamp*
2. *Locks record blocks in write set*
3. **Ensure that log is persisted**
4. *Writes records. For each record*
    a. *Look at pointer for record to evict*
    b. *If record has "moved" bit 1, replace. Otherwise, advance*
    c. *Copy existing "current version" to old*
5. *Send commit status*

Fault Tolerance - Memory Servers

Stateful. Forces complete halt of system, recovery using single compute server

Recovery

1. Start up single compute server recovery
2. Read logs from memory from last checkpoint
3. Partially order by logged read timestamp snapshot
4. Replay merged log back to memory servers

Fault Tolerance - Compute Servers

Compute Servers are stateless, but failure can result in abandoned locks

Monitoring Compute Server → Compute Server monitoring Peer

1. Detect that peer Compute Server fails
2. Read logs made by execution threads, find locks
3. Release abandoned locks

Evaluation

TPC-C Benchmark, or the Order Entry Benchmark

Cluster A → 57 total, 28 type 1 (compute), 29 type 2 (memory + timestamp oracle)

Cluster B → 8 total for other testing

Single InfiniBand FDR 4X, Mellanox Connect IB (2011, 54.54 Gb/s)

Hash and B+ tree indexes included

Evaluation - System Scalability

Cluster A

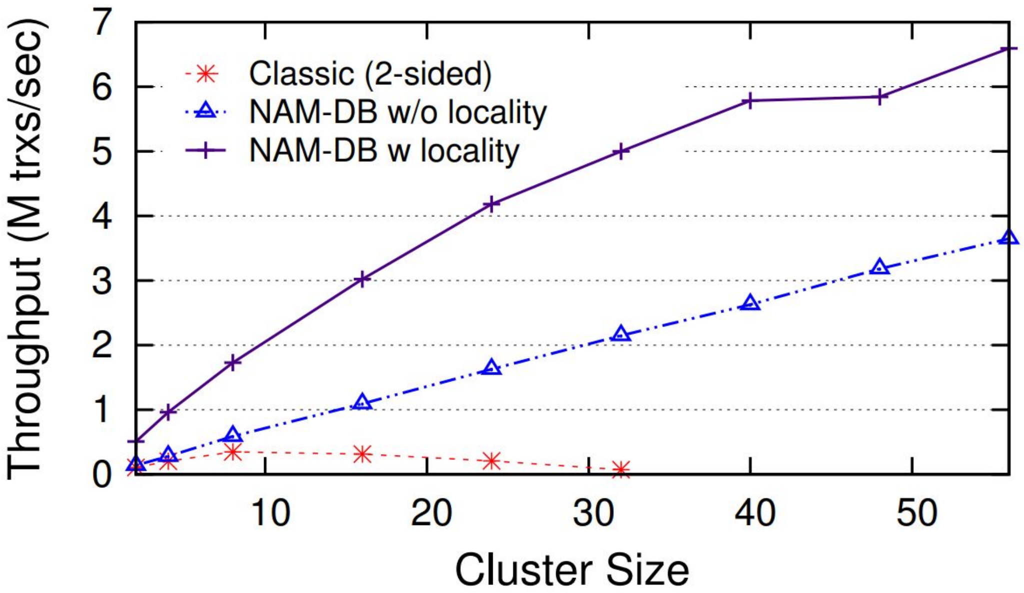
Compute, **T**hreads/**C**, **M**emory

Without locality: 28 C, 60 T/C, 28 M

With locality: 56 C, 30 T/C, 56 M

- Paired per physical machine

2-sided: Two-sided RDMA 2PC



Evaluation - Timestamp Scalability

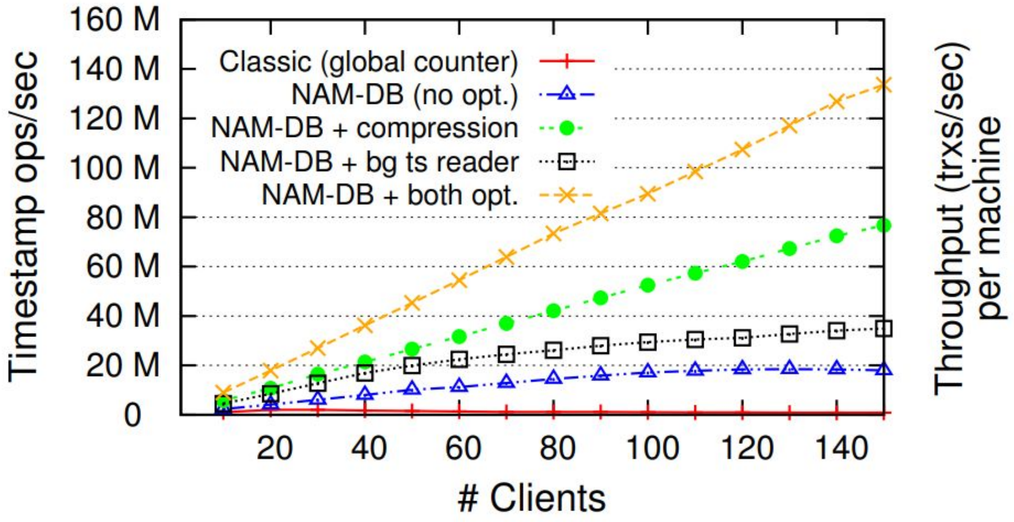
Cluster B

7 Compute, 1 Memory

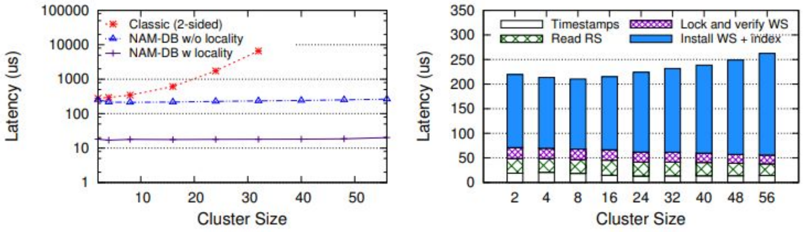
Detached Fetch Thread

Timestamp Vector Compression

Theoretical limit: ~~60~~ 500 clients

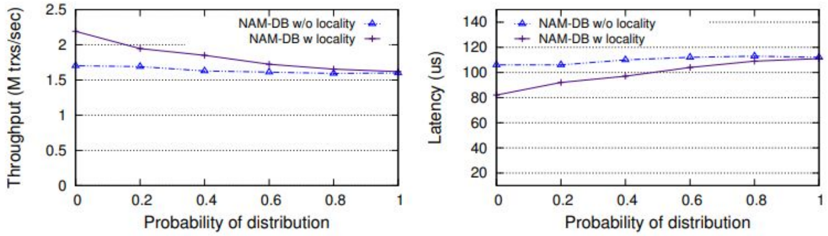


Evaluation - Locality, Contention, RDMA Queue Pairs



(a) Latency (b) Breakdown for NAM-DB

Figure 5: Latency and Breakdown



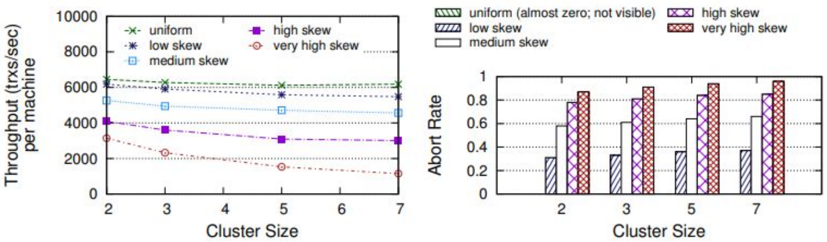
(a) Throughput (b) Latency

Figure 6: Effect of Locality

Probability of Distribution → Chance of remote access

Locality adds ~30%, no longer order-of-magnitude difference

Evaluation - Locality, Contention, RDMA Queue Pairs



(a) Throughput (b) Abort Rate

Figure 8: Effect of Contention

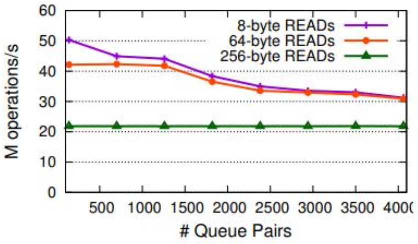


Figure 9: Scalability of QPs

Extremely high skew workloads are inherently unscalable

High Queue Pairs may overflow NIC cache, but is not limit for most workloads

Thank you!

Questions?