Zeus
Locality-aware distributed transactions

A. Katsarakis*, Y. Ma†, Z. Tan§, A. Bainbridge, M. Balkwill, A. Dragojevic, B. Grot*, B. Radunovic, Y. Zhang

*University of Edinburgh, †Fudan University, §UCLA, Microsoft Research

zeus-protocol.com

Thanks to: EPSRC Microsoft Research
Modern distributed datastores

Keep data
*in-memory, replicated, sharded*
across nodes of a datacenter
Modern distributed datastores

Keep data
in-memory, replicated, sharded
across nodes of a datacenter

Backbone of transactional cloud applications
Modern distributed datastores

Keep data
in-memory, replicated, sharded
across nodes of a datacenter

Backbone of transactional cloud applications

Demand
distributed reliable transactions (txs)
strongly-consistent and fault-tolerant
Keep data
in-memory, replicated, sharded
across nodes of a datacenter

Backbone of transactional cloud applications

Demand
distributed reliable transactions (txs)
strongly-consistent and fault-tolerant
high performance
Modern distributed datastores

Keep data
- in-memory, replicated, sharded
  across nodes of a datacenter

Backbone of transactional cloud applications

Demand
- distributed reliable transactions (txs)
  strongly-consistent and fault-tolerant
  high performance

Traditional distributed txs well-known as expensive
Observation

Many tx applications exhibit **dynamic locality**

network functions, peer-to-peer payments ...
Observation

Many tx applications exhibit **dynamic locality**

network functions, peer-to-peer payments ...

Example: cellular control plane

manages phone connectivity and

handovers among base stations
Many tx applications exhibit **dynamic locality**
network functions, peer-to-peer payments ...

**Example:** cellular control plane
manages phone connectivity and handovers among base stations

**Locality**
every phone user repeats txs:
same phone & nearest base-station
Observation

Many tx applications exhibit **dynamic locality**

network functions, peer-to-peer payments ...

Example: cellular control plane

manages phone connectivity and

handovers among base stations

**Locality**

every phone user repeats txs:

same phone & nearest base-station

**But locality is dynamic**

changes at run-time

e.g., user commutes → base-station changes
Observation

Many tx applications exhibit **dynamic locality** network functions, peer-to-peer payments ...

**Example:** cellular control plane manages phone connectivity and handovers among base stations

**Locality**

- every phone user repeats txs: same phone & nearest base-station

**But locality is dynamic**

- changes at run-time

*Can state-of-the-art datastores exploit dynamic locality?*
State-of-the-art reliable datastores

**Static sharding** (e.g., consistent hashing)

Objects placed randomly on fixed nodes

- Easy to locate and access objects
- Reliable txs regardless of access pattern
State-of-the-art reliable datastores

**Static sharding** (e.g., consistent hashing)

Objects placed randomly on fixed nodes
- easy to locate and access objects
- reliable txs regardless of access pattern

- expensive reliable txs
- mostly **remote accesses**
- some blocking (control flow, pointer chasing)

Adapted from *FaSST* [OSDI’16]
State-of-the-art reliable datastores

**Static sharding** (e.g., consistent hashing)

Objects placed randomly on fixed nodes

- easy to locate and access objects
- reliable txs regardless of access pattern

**Expensive reliable txs**

mostly **remote accesses**

- some blocking (control flow, pointer chasing)

- related objects on different shards

  **costly distributed commit**

Adapted from FaSST [OSDI’16]
State-of-the-art reliable datastores

**Static sharding** (e.g., consistent hashing)

Objects placed randomly on fixed nodes
- easy to locate and access objects
- reliable txs regardless of access pattern

**Expensive reliable txs**

- mostly remote accesses
- some blocking (control flow, pointer chasing)
- related objects on different shards
- costly distributed commit

Cannot exploit locality → expensive reliable txs
Enter Zeus

Distributed datastore: exploits locality for fast reliable txs
Inspired by multiprocessor’s hardware transactional memory
Enter Zeus

Distributed datastore: exploits locality for fast reliable txs
Inspired by multiprocessor’s hardware transactional memory

Basic idea💡

Each object has a single node **owner = data + exclusive write access**
the **owner changes dynamically** and is tracked by replicated directory
Enter Zeus

Distributed datastore: exploits locality for fast reliable txs
Inspired by multiprocessor’s hardware transactional memory

Basic idea

Each object has a single node owner = data + exclusive write access
the owner changes dynamically and is tracked by replicated directory

Coordinator executes a tx by acquiring ownership of all its objects
→ single-node commit
Enter Zeus

Distributed datastore: exploits locality for fast reliable txs
Inspired by multiprocessor’s hardware transactional memory

Basic idea

Each object has a single node owner = data + exclusive write access
the owner changes dynamically and is tracked by replicated directory

Coordinator executes a tx by acquiring ownership of all its objects
→ single-node commit

Ownership stays with coordinator
→ future txs on these objects enjoy local accesses
Enter Zeus

Distributed datastore: exploits locality for fast reliable txs
Inspired by multiprocessor’s hardware transactional memory

Basic idea

Each object has a single node $\text{owner} = \text{data} + \text{exclusive write access}$
the $\text{owner changes dynamically}$ and is tracked by replicated directory

Coordinator executes a tx by $\text{acquiring ownership}$ of all its objects
$\rightarrow \text{single-node commit}$

Ownership stays with coordinator
$\rightarrow \text{future txs on these objects enjoy local accesses}$

What are the exact steps?
Locality-aware txs in Zeus

1. Execute as the owner:
   - at object access: if (not owner) get ownership
   - local access

2. Local commit:
   commits tx: traditional single-node commit (updates not yet replicated)

3. Reliable commit:
   completes tx: updating replicas for availability
Locality-aware txs in Zeus

1. Execute as the owner
   a) at object access: if (not owner) get ownership

2.

3.
Locality-aware txs in Zeus

1. Execute as the owner
   a) at object access: if (not owner) get ownership

How to get ownership reliably?

2.

3.
Ownership protocol

1) Coordinator gets ownership from current owner
Ownership protocol

1) Coordinator gets ownership from current owner
2) Keeps consistent directory replicas

Conflicts: logical timestamps, fault tolerance: idempotent replays as in Hermes [ASPLOS’20]
Ownership protocol

1) Coordinator gets ownership from current owner
2) Keeps consistent directory replicas

1. Coordinator sends object ownership invalidations (through a directory replica) to all arbiters
Ownership protocol

1) Coordinator gets ownership from current owner
2) Keeps consistent directory replicas

1. Coordinator sends object ownership **invalidations** (through a directory replica) to all arbiters
2. Arbiters **acknowledge** the coordinator directly
Ownership protocol

1) Coordinator gets ownership from current owner
2) Keeps consistent directory replicas

1. Coordinator sends object ownership invalidations (through a directory replica) to all arbiters
2. Arbiters acknowledge the coordinator directly

Ownership is acquired & coordinator proceeds with tx
Ownership protocol

1) Coordinator gets ownership from current owner
2) Keeps consistent directory replicas

1. Coordinator sends object ownership invalidations (through a directory replica) to all arbiters
2. Arbiters acknowledge the coordinator directly
3. Coordinator sends validations informing arbiters for acquisition
Ownership protocol

1) Coordinator gets ownership from current owner
2) Keeps consistent directory replicas

1. Coordinator sends object ownership invalidations (through a directory replica) to all arbiters
2. Arbiters acknowledge the coordinator directly
3. Coordinator sends validations informing arbiters for acquisition

Conflicts: logical timestamps, fault tolerance: idempotent replays as in Hermes [ASPLOS’20]
Ownership protocol

1) Coordinator gets ownership from current owner
2) Keeps consistent directory replicas among arbiters

1. Coordinator sends object ownership invalidations (through a directory replica) to all arbiters
2. Arbiters acknowledge the coordinator directly
3. Coordinator sends validations informing arbiters for acquisition

Conflicts: logical timestamps, fault tolerance: idempotent replays

Correctness verified under conflicts and faults
Locality-aware txs in Zeus

1. Execute as the owner
   a) at object access: if (not owner) get ownership

Locality + ownership stays with coordinator
Locality-aware txs in Zeus

1. Execute as the owner
   a) at object access: if (not owner) get ownership
   b) local access

Locality + ownership stays with coordinator
Localization-aware txs in Zeus

1. Execute as the owner
   a) at object access: if (not owner) get ownership
   b) local access
Locality-aware txs in Zeus

1. Execute as the owner
   a) at object access: \textbf{if (not owner) get ownership}
   b) local access

2. Local commit
   \textbf{commits tx}: traditional single-node commit
Locality-aware txs in Zeus

1. Execute as the owner
   a) at object access: if (not owner) get ownership
   b) local access

2. Local commit
   commits tx: traditional single-node commit

3. Reliable commit
   completes tx: updating replicas
Locality-aware txs in Zeus

1. Execute as the owner
   a) at object access: if (not owner) get ownership
   b) local access

2. Local commit
   commits tx: traditional single-node commit

3. Reliable commit
   completes tx: updating replicas

Great! But how efficient is reliable commit?
1. Committed tx → no conflicts → fast tx completion
   - coordinator sends updates to replicas and waits for ACKs
   - read-only txs: no updates → no reliable commit
1. Committed tx $\rightarrow$ no conflicts $\rightarrow$ fast tx completion
   - coordinator sends updates to replicas and waits for ACKs
   - read-only txs: no updates $\rightarrow$ no reliable commit

2. No conflicts $\rightarrow$ no aborts $\rightarrow$ pipelined txs (no waiting for replication)
   - subsequent txs use local state with certainty & issue updates
   - coordinator sequences updates, which replicas apply in order
1. Committed tx → no conflicts → fast tx completion
   - coordinator sends updates to replicas and waits for ACKs
   - read-only txs: no updates → no reliable commit

2. No conflicts → no aborts → pipelined txs (no waiting for replication)
   - subsequent txs use local state with certainty & issue updates
   - coordinator sequences updates, which replicas apply in order

Fault tolerance: idempotent replays
Reliable commit

1. Committed tx → no conflicts → fast tx completion
   - coordinator sends updates to replicas and waits for ACKs
   - read-only txs: no updates → no reliable commit

2. No conflicts → no aborts → pipelined txs (no waiting for replication)
   - subsequent txs use local state with certainty & issue updates
   - coordinator sequences updates, which replicas apply in order

Very efficient! Correctness verified under faults
Recap: txs in Zeus

Locality-aware, distributed and reliable
Recap: txs in Zeus

Locality-aware, distributed and reliable

1. Execute as the owner

   if (not the owner)
   get ownership

   occasionally

Local access

common case
Recap: txs in Zeus

Locality-aware, distributed and reliable

1. Execute as the owner
   if (not the owner)
   get ownership
   occasionally

   Local access
   common case

2. Local commit
   read-only tx completion

Coordinator
Directory
Owner
Recap: txs in Zeus

Locality-aware, distributed and reliable

1. Execute as the owner
   - if (not the owner)
     - get ownership
   - occasionally

2. Local access
   - common case

3. Local commit
   - read-only tx completion

3. Reliable commit
   - write tx completion
   - Non-abortable, fast and pipelined
Recap: txs in Zeus

Locality-aware, distributed and reliable

1. Execute as the owner
   - if (not the owner)
     - get ownership
   - occasionally

Local access
   - common case

2. Local commit
   - read-only tx completion

3. Reliable commit
   - write tx completion
   - Non-abortable, fast and pipelined

Directory

Cooperator

Owner

Replicas

Local read-only txs at all replicas
Recap: txs in Zeus

Locality-aware, distributed and reliable

1. Execute as the owner
   - if (not the owner) get ownership
   - occasionally
   - Local access (common case)

2. Local commit
   - read-only tx completion
   - Local read-only txs at all replicas

3. Reliable commit
   - write tx completion
   - Non-abortable, fast and pipelined

Awesome! Does it translate into performance?
Performance

6 nodes, 3-way replication, Zeus 40Gb (no RDMA)
Performance

6 nodes, 3-way replication, Zeus 40Gb (no RDMA)
Performance

6 nodes, 3-way replication, Zeus 40Gb (no RDMA)

Zeus: within 9% of ideal
Performance

6 nodes, 3-way replication, Zeus 40Gb (no RDMA)

Zeus: within 9% of ideal
Performance

6 nodes, 3-way replication,
Zeus 40Gb (no RDMA)

Zeus: within 9% of ideal

Up to 40M.tx/s and 2x state-of-the-art
FaSST [OSDI’16], FaRM [SOSP’15]
which use 56Gb RDMA
Performance

6 nodes, 3-way replication, Zeus 40Gb (no RDMA)

Handovers

<table>
<thead>
<tr>
<th>Million txs/sec</th>
<th>Ideal (all local)</th>
<th>Zeus (real-world locality)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>9%</td>
</tr>
</tbody>
</table>

TATP

<table>
<thead>
<tr>
<th>% write txs needing ownership</th>
<th>Zeus [40Gb (no RDMA)]</th>
<th>FaSST [56Gb RDMA]</th>
<th>FaRM [56Gb RDMA]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>40</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Zeus: within 9% of ideal

Up to 40M.tx/s and 2x state-of-the-art

Paper: more benchmarks, ownership, latency ...
Conclusion

**State-of-the-art** reliable txs over **static sharding**: cannot exploit dynamic locality
- remote accesses
- costly **distributed commit**
State-of-the-art reliable txs over static sharding:
cannot exploit dynamic locality
remote accesses
costly distributed commit

Zeus’ reliable txs exploit locality via dynamic ownership:
local accesses in the common case
single-node commit
  - local for read-only txs
  - fast and pipelined for write txs
Conclusion

**State-of-the-art** reliable txs over **static sharding**:
- cannot exploit dynamic locality
- **remote accesses**
- costly **distributed commit**

**Zeus’** reliable txs exploit locality via **dynamic ownership**:
- **local accesses** in the common case
- **single-node commit**
  - **local** for read-only txs
  - **fast and pipelined** for write txs

**Performance** 10s millions txs/second
- up to **2x state-of-the-art**

[zeus-protocol.com](http://zeus-protocol.com)
TLA+ specification, Q&A ...
State-of-the-art reliable txs over static sharding:
cannot exploit dynamic locality
remote accesses
costly distributed commit

Zeus’ reliable txs exploit locality via dynamic ownership:
local accesses in the common case
single-node commit
- local for read-only txs
- fast and pipelined for write txs

Performance 10s millions txs/second
up to 2x state-of-the-art

Bonus: programmability!

zeus-protocol.com
TLA+ specification, Q&A ...
Conclusion

State-of-the-art reliable txs over static sharding:
cannot exploit dynamic locality
remote accesses
costly distributed commit

Zeus’ reliable txs exploit locality via dynamic ownership:
local accesses in the common case
single-node commit
- local for read-only txs
- fast and pipelined for write txs

Performance 10s millions txs/second
up to 2x state-of-the-art

Bonus: programmability!

Reliable txs with locality? Use Zeus!

zeus-protocol.com
TLA+ specification, Q&A …