BYSHARD: Sharding in a Byzantine Environment

Jelle Hellings$^{1,2}$ Mohammad Sadoghi$^1$

$^1$Exploratory Systems Lab
Department of Computer Science
University of California, Davis

$^2$Department of Computing and Software
McMaster University
Motivation: High-performance resilient system
Motivation: High-performance resilient system

System (All Data)

Requests (All Data)

(African Data)

Requests (African Data)

(American Data)

Requests (American Data)

(Asian Data)

Requests (Asian Data)

(Mixed Data)

Requests (Mixed Data)

(European Data)

Requests (European Data)
Ingredients of sharding and fault-tolerance

Multi-shard transaction execution of $\tau$

Replication of $\tau$ among shards: two-phase commit.

Concurrency control to guarantee consistent execution of $\tau$: two-phase locking.

One needs *computations* within a shard and *communication* between shards.
Ingredients of sharding and fault-tolerance

Multi-shard transaction execution of $\tau$

- Replication of $\tau$ among shards: two-phase commit.
- Concurrency control to guarantee consistent execution of $\tau$: two-phase locking.

One needs *computations* within a shard and *communication* between shards.

Fault-tolerant shards

Each shard is a cluster of replicas that can be faulty.

- Consensus for each *computation* within shards.
- Cluster-sending for any *communication* between shards.

Consensus is costly: Minimize its use.
BySHARD: A resilient sharding framework

Processing multishard transaction $\tau$ via the orchestrate-execute model:

- Processing is broken down into three types of shard-steps: vote, commit, and abort.
- Each shard-step is performed via one consensus step.
- Transfer control between steps using cluster-sending.

Execution method determines the local operations of a shard-step:

locks, checking conditions, updating state, ….

Orchestration method determines how control is transferred between shard-steps:

perform votes, collect votes, decide commit or abort $\tau$. 
Example of the orchestrate-execute model

Shard accounts by first letter of name

$$\tau = \text{“if } Ana \text{ has $500 and } Bo \text{ has $200, then move $400 from } Ana \text{ to } Bo\text{.”}$$
Example of the orchestrate-execute model

Shard accounts by first letter of name

\[ \tau = \text{"if } Ana \text{ has } $500 \text{ and } Bo \text{ has } $200, \text{ then move } $400 \text{ from } Ana \text{ to } Bo." } \]

\[ \sigma_1 = \text{"Lock}(Ana); \text{ if } Ana \text{ has } $500, \text{ then forward } \sigma_2 \text{ to } S_b \text{ (commit vote)} \]  
\[ \text{else } \text{RELEASE}(Ana) \text{ (abort vote)."} \]

**vote-step**

\( \sigma_1 \text{ at } S_a \)
Example of the orchestrate-execute model

Shard accounts by first letter of name

\[ \tau = \text{“if Ana has $500 and Bo has$200, then move $400 from Ana to Bo.”} \]

\[ \sigma_2 = \text{“Lock}(Bo)\text{; if Bo has$200, then add $400 to Bo; Release}(Bo)\text{; and forward } \sigma_3 \text{ to } S_a \text{ (commit) } \]

\[ \text{else Release}(Bo) \text{ and forward } \sigma_4 \text{ to } S_a \text{ (abort).} \]

\[ \begin{align*}
\text{vote-step} \quad \sigma_1 \text{ at } S_a & \quad \xrightarrow{\text{vote commit}} \quad \sigma_2 \text{ at } S_b \\
\end{align*} \]
Example of the orchestrate-execute model

Shard accounts by first letter of name

\( \tau = \text{“if Ana has } 500 \text{ and Bo has } 200, \text{ then move } 400 \text{ from Ana to Bo.”} \)

\( \sigma_3 = \text{“remove } 400 \text{ from Ana and RELEASE(Ana).”} \)

\( \sigma_4 = \text{“RELEASE(Ana).”} \)
The orchestration methods of BySHARD

Orchestration $\approx$ two-phase commit, except that shards never fail.

Vote-steps in sequence, decide centralized, commit or abort in parallel.
The orchestration methods of BySHARD

Orchestration ≈ two-phase commit, except that *shards never fail.*

Vote-steps in *parallel*, decide *centralized*, commit or abort in *parallel*.

Lemma 4.2. Decide with a *single* consensus step, independent of the number of votes.
The orchestration methods of BySHARD

Orchestration \approx \text{two-phase commit, except that \textit{shards never fail}.}

Vote-steps in \textit{parallel}, decide \textit{decentralized}, commit or abort in \textit{parallel}.

Lemma 4.2. Decide with a \textit{single} consensus step, independent of the number of votes.
The execution methods of BYSHARD

Execution updates state and performs *concurrency control*:

- Write uncommitted execution (degree 0 isolation) for *free*.

- Higher isolation levels via *two-phase locking*:
  - read uncommitted execution (degree 1 isolation): only *write locks*;
  - read committed execution (degree 2 isolation): *read locks* during steps;
  - serializable execution (degree 3 isolation): *read and write locks*.

- Blocking locks (with linear orchestration) versus non-blocking locks.

Theorem 5.3. Obtaining and releasing locks does *not cost additional* consensus steps.
Performance evaluation

<table>
<thead>
<tr>
<th>Isolation-Free execution (write uncommitted)</th>
<th>Lock-based execution</th>
<th>Serializable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>LSb</td>
<td>AHl (reference committee)</td>
</tr>
<tr>
<td>Centralized</td>
<td>LSB</td>
<td></td>
</tr>
<tr>
<td>Distributed</td>
<td>DSb</td>
<td></td>
</tr>
</tbody>
</table>

**Total Runtime**

- **Runtime (s)**
  - 10.0
  - 8.0
  - 6.0
  - 4.0
  - 2.0
  - 0.0

- **Throughput (txn/s)**
  - 1.0
  - 0.5
  - 0.0

**Average Committed Throughput**

- 2.0
- 1.5
- 1.0
- 0.5
- 0.0
Conclusion

BYSHARD: a *general-purpose* framework for sharded resilient systems.

Eighteen *high-performance* multi-shard transaction processing protocols.

Fine-grained control over isolation level and performance *per* transaction.