

BYSHARD: Sharding in a Byzantine Environment

Jelle Hellings^{1,2} Mohammad Sadoghi¹

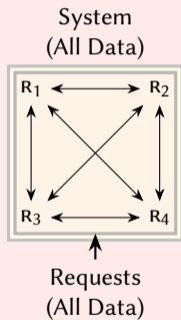
¹Exploratory Systems Lab
Department of Computer Science
University of California, Davis



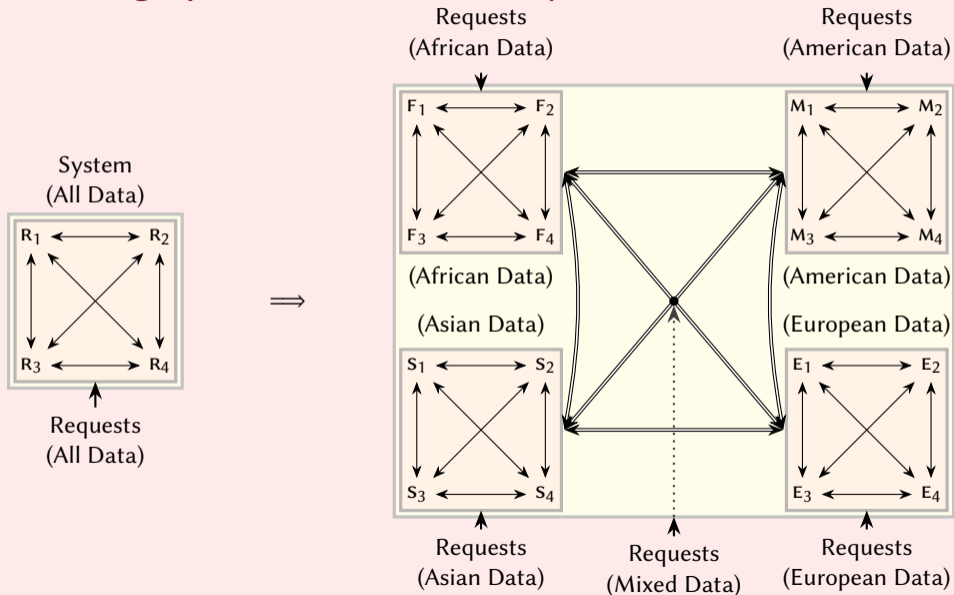
²Department of Computing and Software
McMaster University



Motivation: High-performance resilient system



Motivation: High-performance resilient system



Ingredients of sharding and fault-tolerance

Multi-shard transaction execution of τ

Replication of τ among shards: two-phase commit.

Concurrency control to guarantee consistent execution of τ : two-phase locking.

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

Ingredients of sharding and fault-tolerance

Multi-shard transaction execution of τ

Replication of τ among shards: two-phase commit.

Concurrency control to guarantee consistent execution of τ : 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 multi-shard transaction τ 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* τ .

Example of the orchestrate-execute model

Shard accounts by first letter of name

$\tau =$ “if *Ana* has \$500 and *Bo* has \$200, then
move \$400 from *Ana* to *Bo*.”

Example of the orchestrate-execute model

Shard accounts by first letter of name

τ = “if *Ana* has \$500 and *Bo* has \$200, then
move \$400 from *Ana* to *Bo*.”

σ_1 = “LOCK(*Ana*); if *Ana* has \$500, then forward σ_2 to \mathcal{S}_b (commit vote)
else RELEASE(*Ana*) (abort vote).”

vote-step

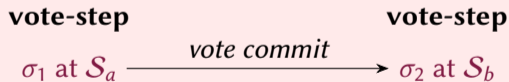
σ_1 at \mathcal{S}_a

Example of the orchestrate-execute model

Shard accounts by first letter of name

τ = “if *Ana* has \$500 and *Bo* has \$200, then
move \$400 from *Ana* to *Bo*.”

σ_2 = “LOCK(*Bo*); if *Bo* has \$200, then add \$400 to *Bo*; RELEASE(*Bo*); and
forward σ_3 to \mathcal{S}_a (commit)
else RELEASE(*Bo*) and forward σ_4 to \mathcal{S}_a (abort).”



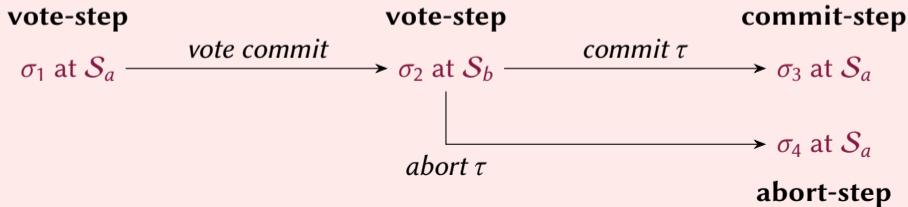
Example of the orchestrate-execute model

Shard accounts by first letter of name

τ = “if *Ana* has \$500 and *Bo* has \$200, then
move \$400 from *Ana* to *Bo*.”

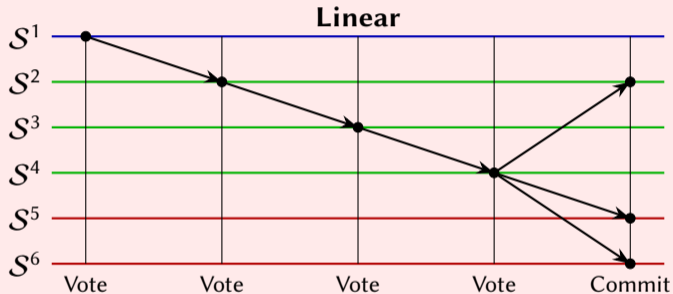
σ_3 = “remove \$400 from *Ana* and `RELEASE(Ana)`.”

σ_4 = “`RELEASE(Ana)`.”



The orchestration methods of BYSHARD

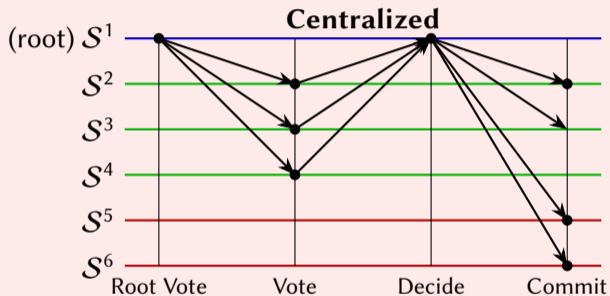
Orchestration \approx two-phase commit, except that *shards never fail*.



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

The orchestration methods of BYSHARD

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

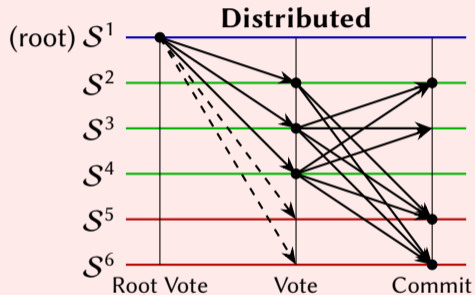


Vote-steps in *parallel*, decide *centralized*, commit or abort in *parallel*.

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

The orchestration methods of BYSHARD

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



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

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

The execution methods of BYSHARD

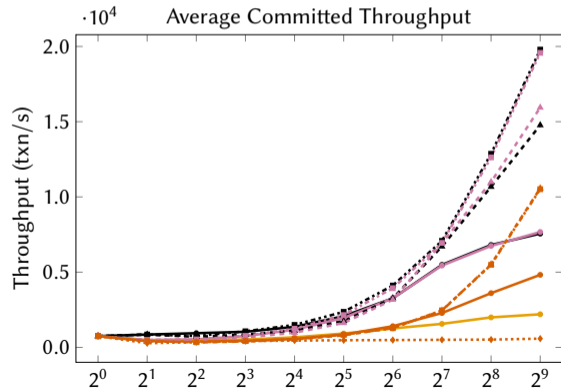
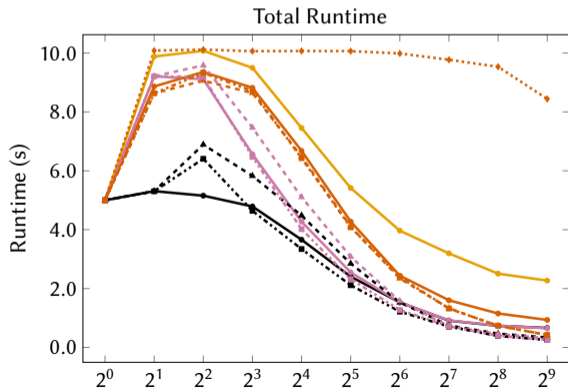
Execution updates state and performs *concurrency control*:

- ▶ 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

	Isolation-Free execution (write uncommitted)				Lock-based execution Serializable		
Linear	● LIFu	● LIFs	● LSB	● LSNB	◆ AHL (reference committee)		
Centralized	▲ CIFu	▲ CIFs	▲ CSNB	▲ DSNB			
Distributed	■ DIFu	■ DIFs					



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.