BYSHARD: Sharding in a Byzantine Environment

Jelle Hellings[1,2] Mohammad Sadoghi[1]

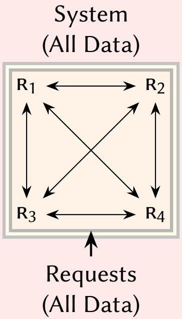
[1]Exploratory Systems Lab
Department of Computer Science
University of California, Davis

ExpoLab Creativity Unfolded
UC DAVIS UNIVERSITY OF CALIFORNIA

[2]Department of Computing and Software
McMaster University

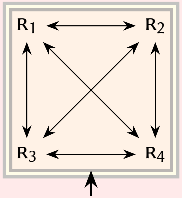
McMaster University

Motivation: High-performance resilient system



Motivation: High-performance resilient system

System
(All Data)

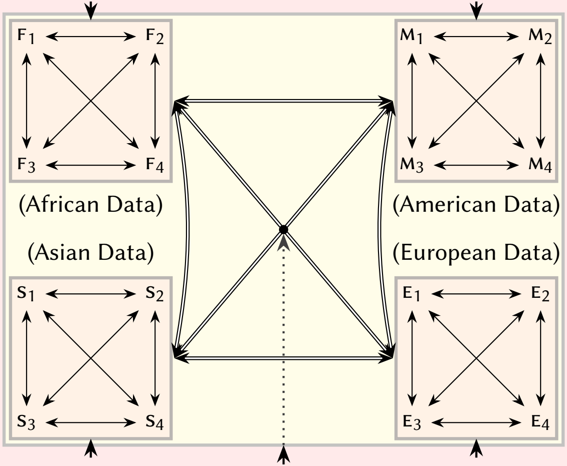


Requests
(All Data)

⟹

Requests
(African Data)

Requests
(American Data)



Requests
(Asian Data)

Requests
(Mixed Data)

Requests
(European Data)

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 multishard 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

τ = "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

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

$$\sigma_1 = \text{“}\textsc{Lock}(\textit{Ana})\text{; if } \textit{Ana} \text{ has \$500, then forward } \sigma_2 \text{ to } \mathcal{S}_b \text{ (commit vote) else } \textsc{Release}(\textit{Ana}) \text{ (abort vote).”}$$

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)."

vote-step σ_1 at $\mathcal{S}_a$ —*vote commit*→ **vote-step** σ_2 at $\mathcal{S}_b$

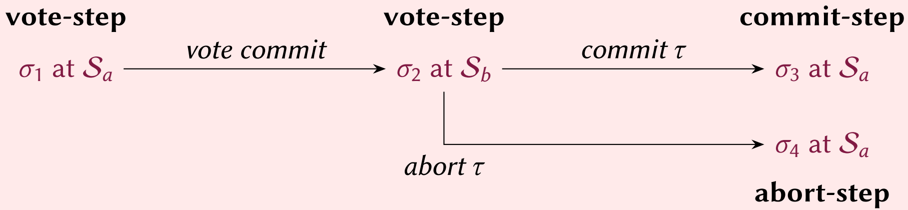
Example of the orchestrate-execute model

Shard accounts by first letter of name

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

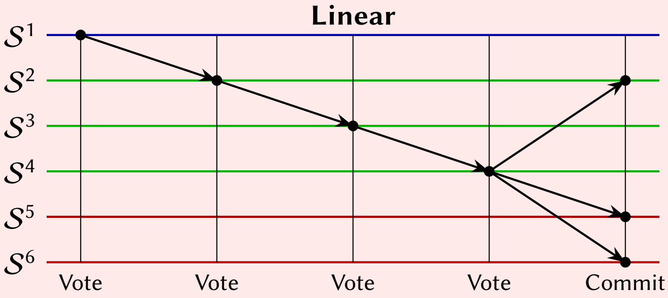
σ_3 = "remove \$400 from *Ana* and RELEASE(*Ana*)."

σ_4 = "RELEASE(*Ana*)."



The orchestration methods of ByShard

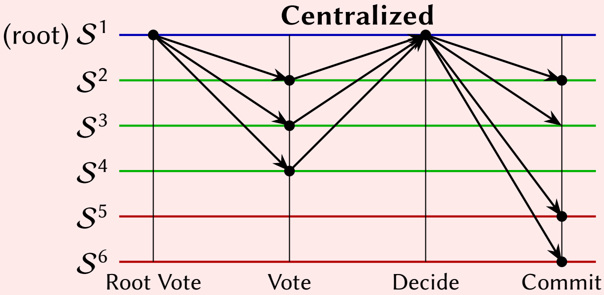
Orchestration ≈ two-phase commit, except that *shards never fail*.



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*.

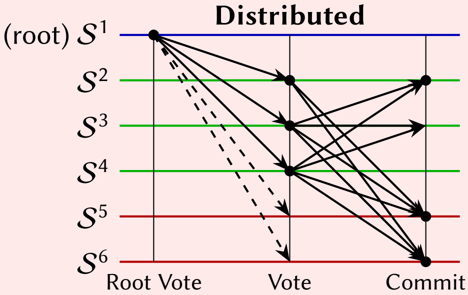


Vote-steps in *parallel*, decide *centralized*, commit or abort in *parallel*.

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

The orchestration methods of ByShard

Orchestration ≈ 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*.

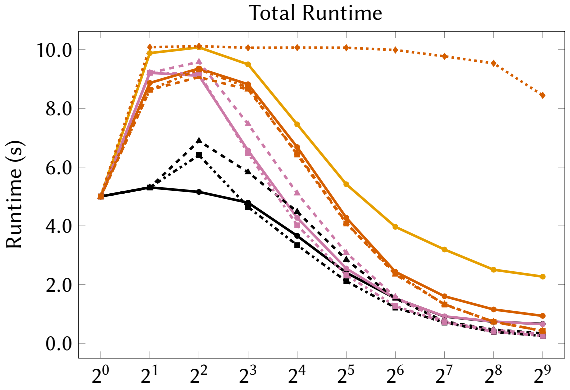
- Higher isolation levels via *two-phase locking*:
  - read uncommitted execution (degree 1 isolation): only *write locks*;
  - read committed execution (degree 2 isolation): *read locks* during steps;
  - serializable execution (degree 3 isolation): *read and write locks*.

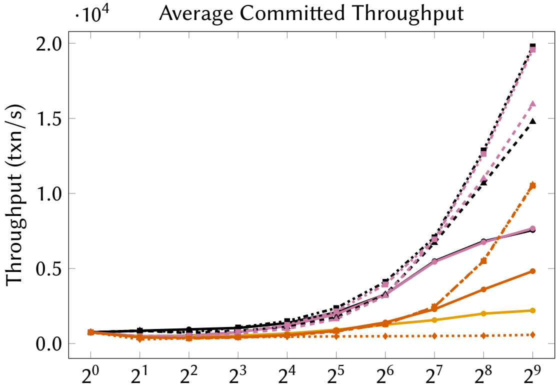
- Blocking locks (with linear orchestration) versus non-blocking locks.

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		
Centralized	CIFu	CIFs		CSnb	AHL	(reference committee)
Distributed	DIFu	DIFs		DSnb		





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.