

THE RAFT CONSENSUS ALGORITHM

DIEGO ONGARO AND JOHN OUSTERHOUT

STANFORD UNIVERSITY

MAY 2015



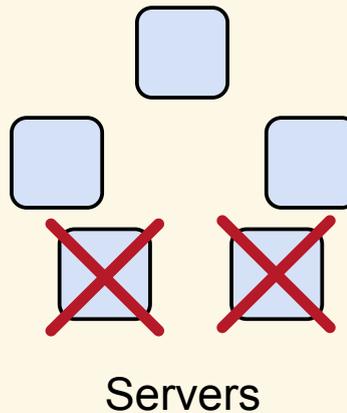
raftconsensus.github.io

MOTIVATION

- Goal: shared key-value store (state machine)
- Host it on a single machine attached to network
 - Pros: easy, consistent
 - Cons: prone to failure
- With Raft, keep consistency yet deal with failures

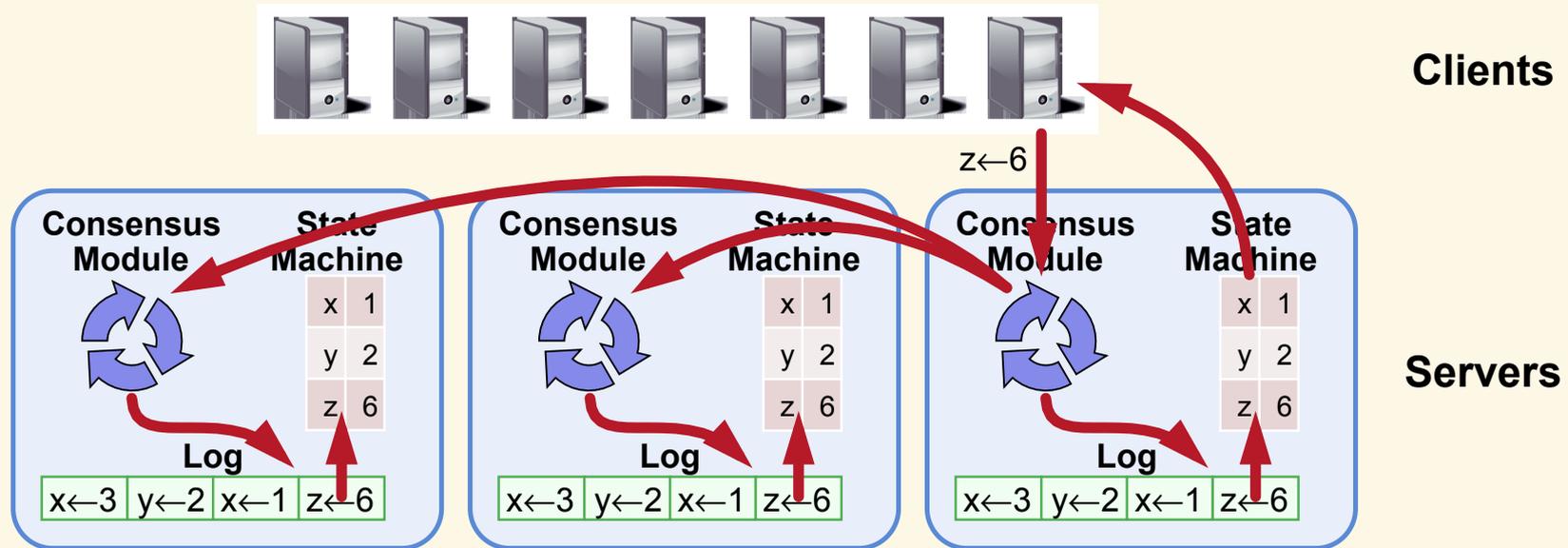
WHAT IS CONSENSUS

- Agreement on shared state (single system image)
- Recovers from server failures autonomously
 - Minority of servers fail: no problem
 - Majority fail: lose availability, retain consistency



- Key to building consistent storage systems

REPLICATED STATE MACHINES



- **Replicated log** \Rightarrow replicated state machine
 - All servers execute same commands in same order
- Consensus module ensures proper log replication
- System makes progress as long as any majority of servers up
- Failure model: fail-stop (not Byzantine), delayed/lost msgs

PAXOS PROTOCOL

- Leslie Lamport, 1989
- Nearly synonymous with consensus

“The dirty little secret of the NSDI community is that at most five people really, truly understand every part of Paxos ;-).”

–NSDI reviewer

“There are significant gaps between the description of the Paxos algorithm and the needs of a real-world system...the final system will be based on an unproven protocol.”

–Chubby authors

RAFT'S DESIGN FOR UNDERSTANDABILITY

We wanted an algorithm optimized for building real systems

- Must be correct, complete, and perform well
- Must also be **understandable**

“What would be easier to understand or explain?”

- Fundamentally different decomposition than Paxos
- Less complexity in state space
- Less mechanism

RAFT OVERVIEW

1. Leader election

- Select one of the servers to act as cluster leader
- Detect crashes, choose new leader

2. Log replication (normal operation)

- Leader takes commands from clients, appends to its log
- Leader replicates its log to other servers (overwriting inconsistencies)

3. Safety

- Only a server with an up-to-date log can become leader

RAFTSCOPE VISUALIZATION

CORE RAFT REVIEW

1. Leader election

- Heartbeats and timeouts to detect crashes
- Randomized timeouts to avoid split votes
- Majority voting to guarantee at most one leader per term

2. Log replication (normal operation)

- Leader takes commands from clients, appends to its log
- Leader replicates its log to other servers (overwriting inconsistencies)
- Built-in consistency check simplifies how logs may differ

3. Safety

- Only elect leaders with all committed entries in their logs
- New leader defers committing entries from prior terms

CONCLUSION

- Consensus widely regarded as difficult
- Raft designed for understandability
 - Easier to teach in classrooms
 - Better foundation for building practical systems
- Pieces needed for a practical system:
 - Cluster membership changes (simpler in dissertation)
 - Log compaction (expanded tech report/dissertation)
 - Client interaction (expanded tech report/dissertation)
 - Evaluation
(dissertation: understandability, correctness, leader election & replication performance)

QUESTIONS

raftconsensus.github.io

[raft-dev mailing list](#)