

Advanced Course

Distributed Systems

Reconfigurable RSMs



COURSE TOPICS



- ▶ Intro to Distributed Systems
- ▶ Basic Abstractions and Failure Detectors
- ▶ Reliable and Causal Order Broadcast
- ▶ Distributed Shared Memory
- ▶ Consensus, RSMs (Omni-Paxos, Raft, etc.)
- ▶ Dynamic Reconfiguration
- ▶ Time Abstractions and Interval Clocks (Spanner etc.)
- ▶ Consistent Snapshotting (Stream Data Management)
- ▶ Distributed ACID Transactions (Cloud DBs)

- Raft and ZooKeeper

Reconfiguration

MOTIVATION

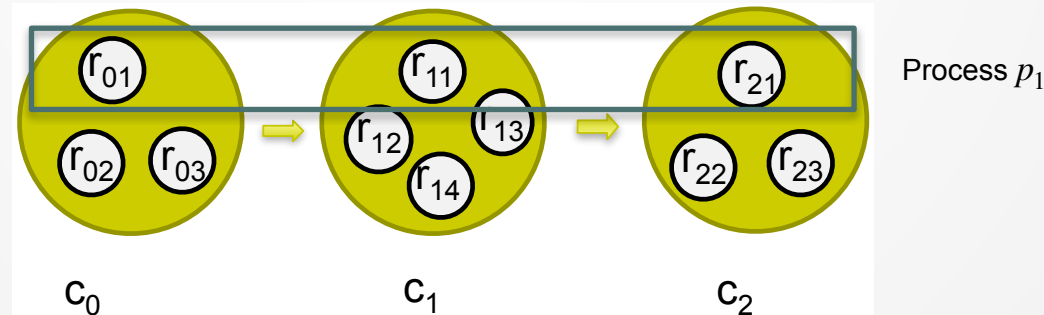
- A Replicated State Machine (RSM) is running on a set of N processes (typically 3 or 5)
 - Can tolerate up to $\lfloor N/2 \rfloor$ failures.
- Impossible to know if a process is faulty or just slow in Asynchronous model.
 - Need a way to replace any process.
- Scaling *up* (more powerful hardware) or *out* (more processes)

POLICY (WHEN) VS MECHANISM (HOW)

- External agent decides when to reconfigure (autonomous or human)
- The agent chooses the new configuration
 - E.g. $c_{old} = \{p_1, p_2, p_3\}$ and $c_{new} = \{p_1, p_2, p_4\}$
 - In general, c_{new} can be a completely new set of processes.
- Only concerned with the mechanism
 - Policy depends on application, deployment settings etc.

CONFIGURATIONS

- Each configuration c_i is conceptually an instance of Sequence Paxos, each with its own BLE instance.
 - Sequence Paxos and BLE instances of different configurations do not communicate!
- A process p that is part of c_i has a *replica instance* $r_{i,p}$
 - A process may have multiple replica instances in different configurations



STOP-SIGN

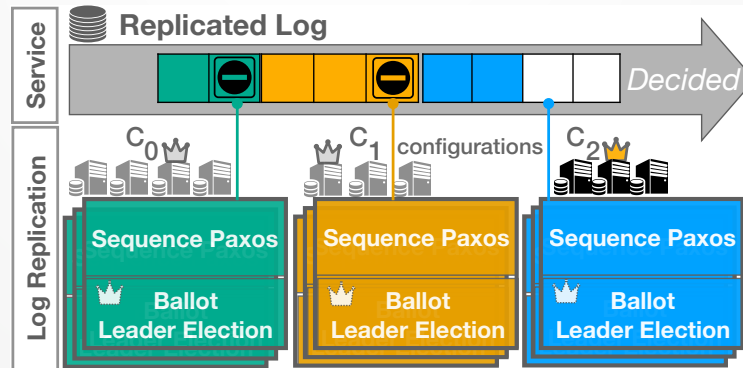
- Must safely stop the current configuration c_i before starting c_{i+1}
- A special **stop-sign (SS)** is proposed. Once it is chosen, the sequence in c_i cannot be extended and c_i is **stopped**. The sequence with SS as last command is the **final sequence** in c_0

Round	Accepted by $r_{0,1}$	Accepted by $r_{0,2}$	Accepted by $r_{0,3}$
...	$\langle C_2, SS_0 \rangle$	$\langle C_2, SS_0 \rangle$	$\langle C_2, SS_0 \rangle$
$n=3$	$\langle C_2, SS_0 \rangle$	$\langle C_2, SS_0 \rangle$	$\langle C_2, SS_0 \rangle$
$n=2$		$\langle C_2 \rangle$	$\langle C_2 \rangle$
$n=1$	$\langle C_1 \rangle$		
$n=0$	$\langle \rangle$	$\langle \rangle$	$\langle \rangle$

- The final sequence in c_0 is $\sigma_0 = \langle C_2, SS_0 \rangle$. Any sequence in round $n > 3$ will be σ_0

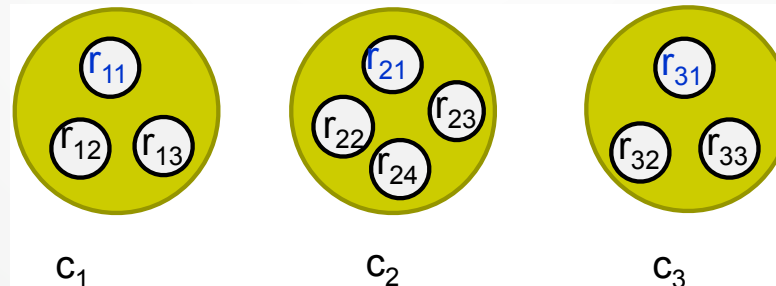
OMNI-PAXOS

- Omni-Paxos executes in one configuration until a reconfiguration occurs, then moves to new configuration.
- Processes transition to the new configuration asynchronously.
- A configuration is **active** once a majority of processes have started in the new configuration.
 - For safety, there can at most be one **running** configuration at all times.



CONFIGURATIONS

- Processes operate at different rates and the leader could fail before everybody have reached the stop-sign.
 - Thus, a process cannot just shut down its replica instance in c_i once it has seen the decided stop-sign.
- As a result, a process p can have multiple replica instances at the same time, each with different state.
 - e.g. p is **stopped** in c_1 , **running** in c_2 and **not-started** in c_3

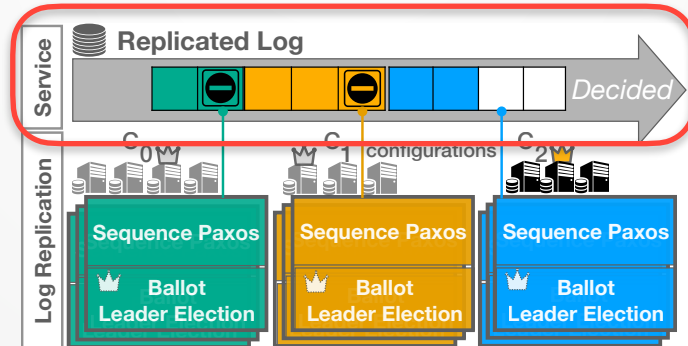


STARTING A NEW CONFIGURATION

- Once SS_i is decided, the new configuration c_{i+1} can start.
- SS_i contains complete information about c_{i+1} :
 - The set of processes in c_{i+1}
 - The new configuration number: cid
 - The identifier for each replica instance in c_{i+1}
- A process that is not part of c_i but added in c_{i+1} must get notified about the reconfiguration.
 - *Log migration*: to have the correct state, it **must** catch up the final sequence σ_i before starting its replica instance in c_{i+1}
- A process p that is part of both c_i and c_{i+1} will eventually see that SS_i is decided in c_i and start its replica instance in c_{i+1}

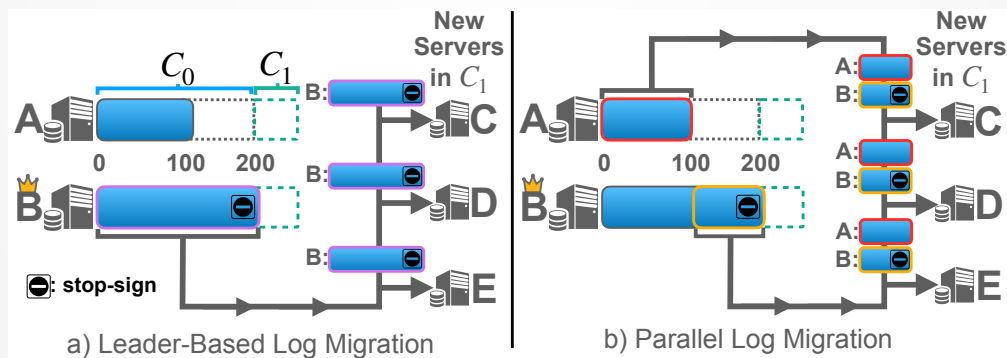
SERVICE LAYER

- The notification of reconfiguration and log migration to new processes are performed in the *service layer*.
 - On top of log replication.
- Advantages of having a separated service layer
 - Parallel log migration
 - Flexible transmission scheme
 - Can pull log entries from processes that have not even reached SS_i yet!



EFFICIENT HAND-OVER

- Since we stop and start configurations, there could be periods of down-time e.g. when new servers are still catching up the log and a majority in the new configuration cannot start yet.
- Important with an efficient hand-over procedure.
 - Flexible and parallel log migration
 - Snapshots



CORRECTNESS

- Must maintain Sequence Consensus invariant across different configurations: *If a proposal with sequence v is chosen, then every higher-numbered proposal that is chosen has v as a prefix.*
- What we have done:
 - Safely stop current configuration c_i before starting c_{i+1}
 - Decide stop-sign as any command using Sequence Paxos. Once chosen, c_i cannot be extended.
 - Require all processes to have the final sequence σ_i before starting in c_{i+1} (log migration)
- Conceptually, we have just **extended the round number** from n to (cid, n) where cid is the configuration number. We made the **round number totally-ordered** across configurations.

ORDERING ROUNDS TOTALLY

Round	Accepted by $r_{c1,1}$	Accepted by $r_{c1,2}$	Accepted by $r_{c1,4}$
...			
$n=(c_1, 3)$	$\langle C_2, SS_0, C_3, C_5 \rangle$	$\langle C_2, SS_0, C_3, C_5 \rangle$	
$n=(c_1, 2)$			$\langle C_2, SS_0, C_3, C_4 \rangle$
$n=(c_1, 1)$	$\langle C_2, SS_0, C_3 \rangle$	$\langle C_2, SS_0, C_3 \rangle$	$\langle C_2, SS_0, C_3 \rangle$
$n=(c_1, 0)$	$\langle C_2, SS_0 \rangle$	$\langle C_2, SS_0 \rangle$	$\langle C_2, SS_0 \rangle$
Round	Accepted by $r_{c0,1}$	Accepted by $r_{c0,2}$	Accepted by $r_{c0,3}$
...	$\langle C_2, SS_0 \rangle$	$\langle C_2, SS_0 \rangle$	$\langle C_2, SS_0 \rangle$
$n=(c_0, 3)$	$\langle C_2, SS_0 \rangle$	$\langle C_2, SS_0 \rangle$	$\langle C_2, SS_0 \rangle$
$n=(c_0, 2)$		$\langle C_2 \rangle$	$\langle C_2 \rangle$
$n=(c_0, 1)$	$\langle C_1 \rangle$		
$n=(c_0, 0)$	$\langle \rangle$	$\langle \rangle$	$\langle \rangle$

SUMMARY

- Reconfiguring an RSM is relatively straight forward.
 - Must avoid “split-brain” problem by first safely stopping the current configuration.
 - Round numbers are totally-ordered across configurations.
- Service layer allows for efficient hand-over with flexible and parallel log migration
- The Omni-Paxos stack is now completed:
 - Service layer for efficient reconfiguration.
 - Sequence Paxos for safely replicating a log.
 - Ballot Leader Election for liveness even in partial connectivity.

Raft

In Search of an Understandable Consensus Algorithm
Ongaro et al.

TERMINOLOGY

- **Sequence Paxos**

- v_a The accepted sequence
- The Decided sequence
- Round/ballot number
- Process
- n_{prom} , n_L
- Element in a sequence



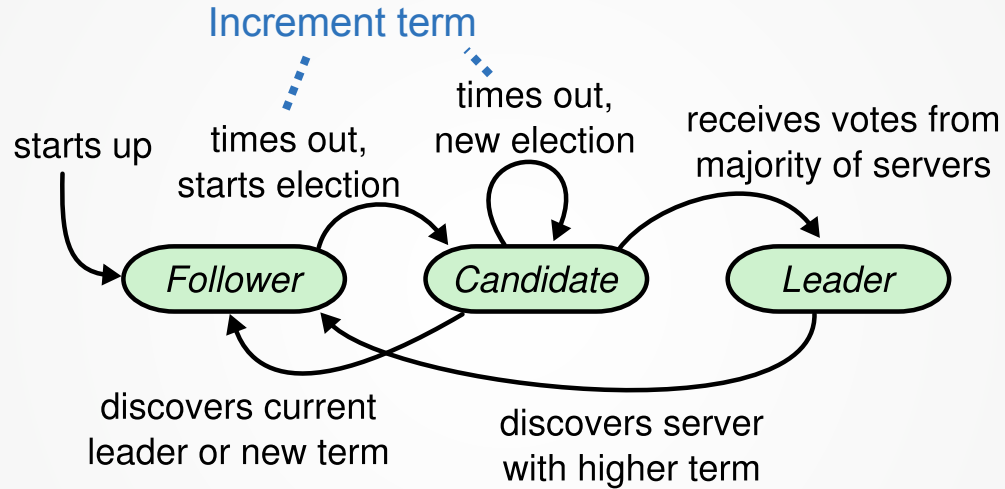
- **Raft**

- The Log
- The committed prefix of Log
- Term
- Server
- Highest Term
- Entry

RAFT DECOMPOSITION

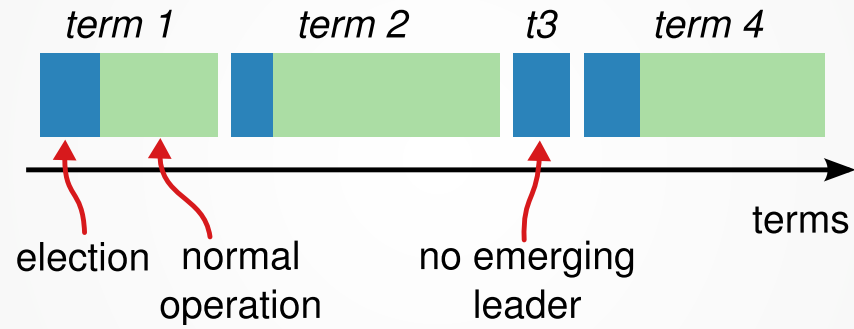
- Leader Election
 - Elect one server as the leader. Detect crashes and choose new leader
 - **Only servers with up-to-date logs can become the leader**
 - The leader election and sequence consensus are fused in one protocol.
 - Incorporates the prepare phase in the leader election algorithm.
- Log replication
 - Leader replicates its log to other servers, overwrites inconsistencies to keep logs consistent
 - Consistent replication is done differently from Sequence Paxos using a *log reconciliation* mechanism.

SERVER STATES



- The servers use remote procedure call (RPC) for communication.
 - RequestVoteRPC
- Each server gives only one vote per term (round)
 - Server p votes for server q if the latest log entry of q has higher term or same term but higher index. In this case, the log of q is more ***up-to-date*** than p .
- Majority of votes required to win.
- Terms are not unique => could be **split votes** with no winner
 - Retry RequestVoteRPC with higher term after some random time.

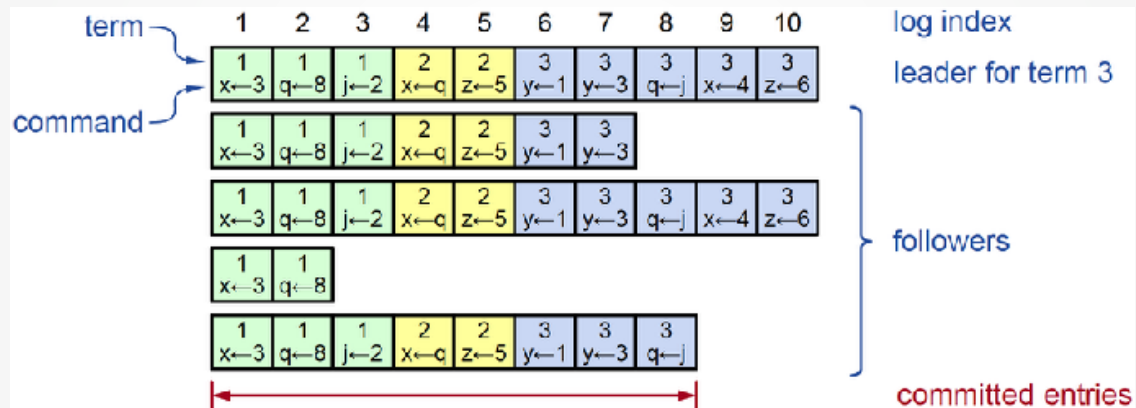
EXECUTION



LOG REPLICATION

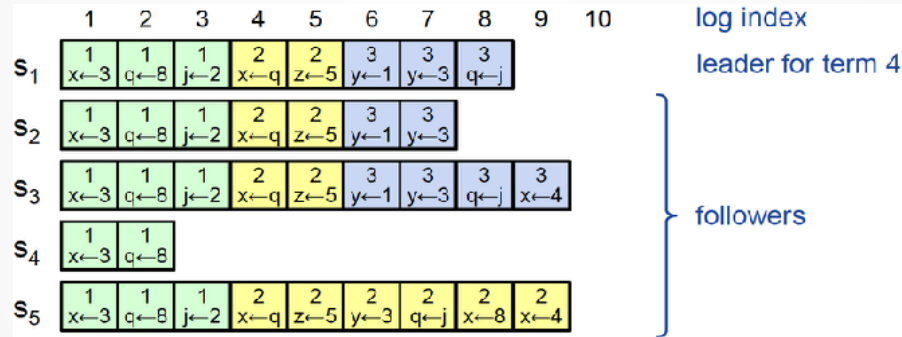
- Client sends commands to leader who appends them to its log.
- Leader sends **AppendEntriesRPC** to all followers (similar to $\langle \text{Accept} \rangle$ in Sequence Paxos)
- Entry is **committed** if AppendEntriesRPC successfully returns from a majority.
- Notify followers of committed index in the next AppendEntriesRPC (similar to $\langle \text{Decide} \rangle$)

LOG STRUCTURE



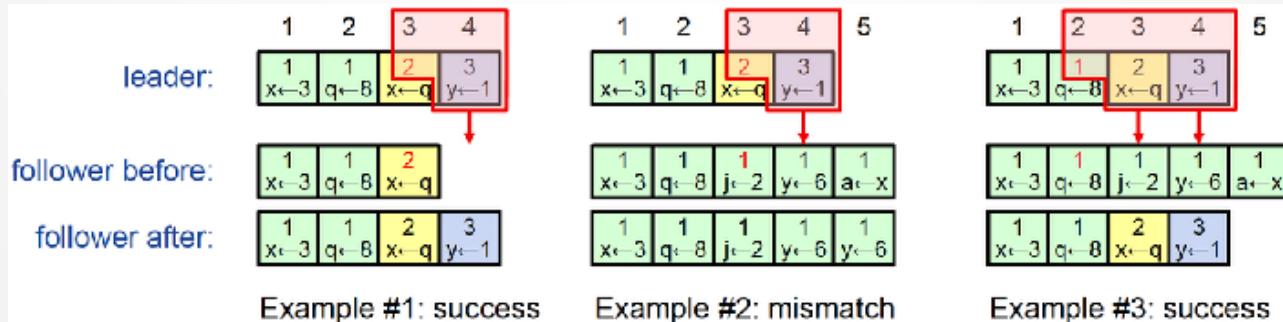
INCONSISTENCIES

Crashes and network partitions may result in inconsistent logs.



LOG RECONCILIATION

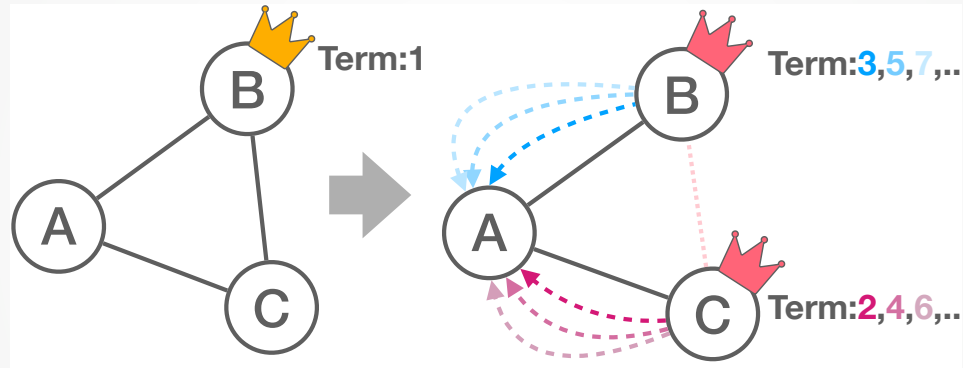
- *Correctness invariant*: Log entries on different servers with same index and term must store the same command, and the logs are identical in all preceding entries.
 - If a given entry is committed, all preceding entries are also committed.
- AppendEntriesRPC include $\langle index, term \rangle$ of entry directly preceding new one(s).
 - Follower must have matching preceding entry; otherwise reject the AppendEntriesRPC and leader retries with lower index.



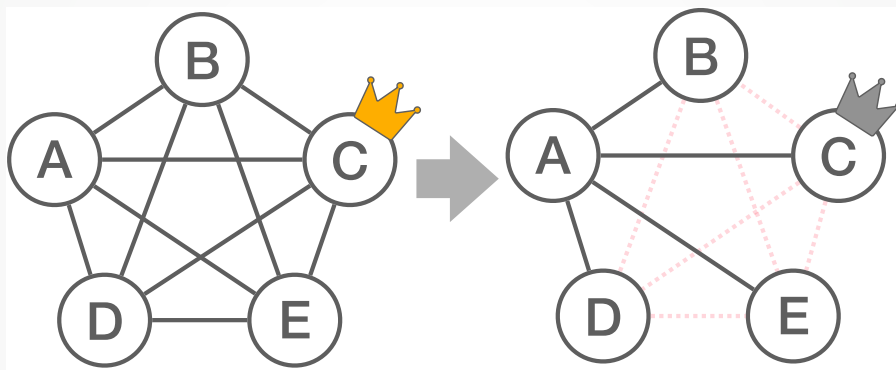
RAFT AND SEQUENCE PAXOS

- Raft and Sequence Paxos are both sequence consensus algorithms.
 - Replicate a growing log.
 - Leader must have highest round or term number.
- Raft differs from Sequence Paxos on:
 - Leader Election: unique ballot numbers in BLE vs. Split votes and randomised retries in Raft.
 - Raft incorporates the prepare phase as part of electing a leader
 - A server must have the most up-to-date log to win election.
 - In Sequence Paxos, any server can become the leader. Will get synchronized in the Prepare phase.
 - Log Reconciliation

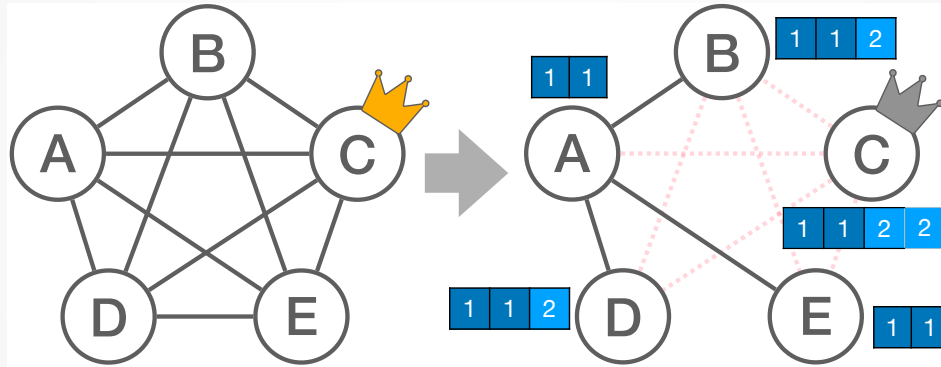
CHAINED SCENARIO



QUORUM-LOSS SCENARIO

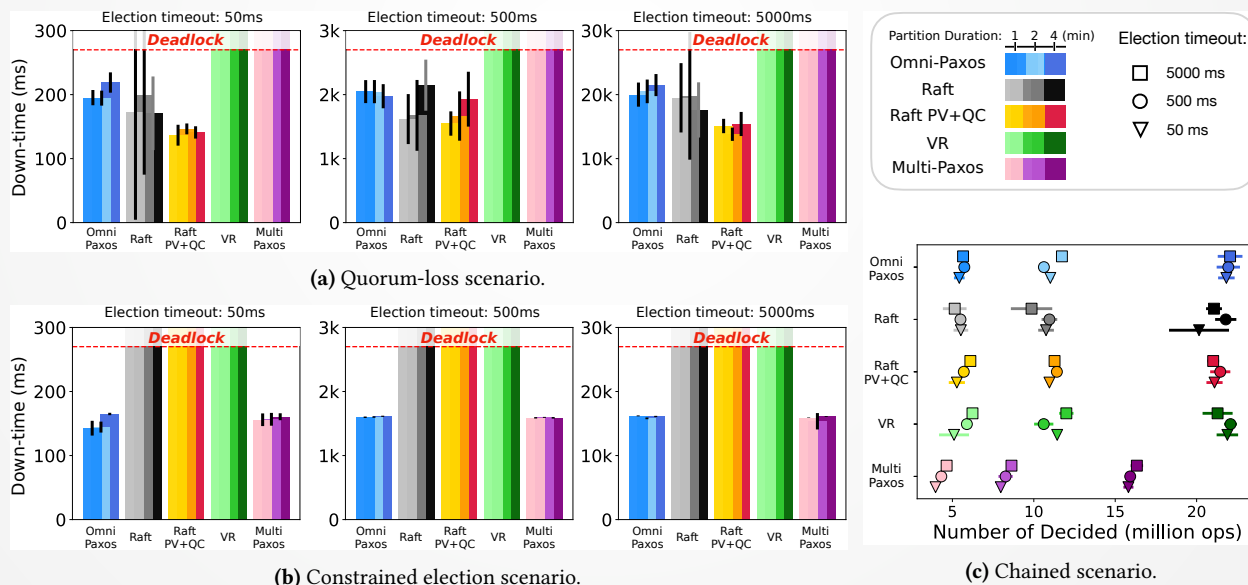


CONSTRAINED ELECTION SCENARIO



LEADER ELECTION EXPERIMENTS

- No existing protocol can tolerate all partial connectivity scenarios except for Omni-Paxos.
- Omni-Paxos recovers in constant number of election timeouts

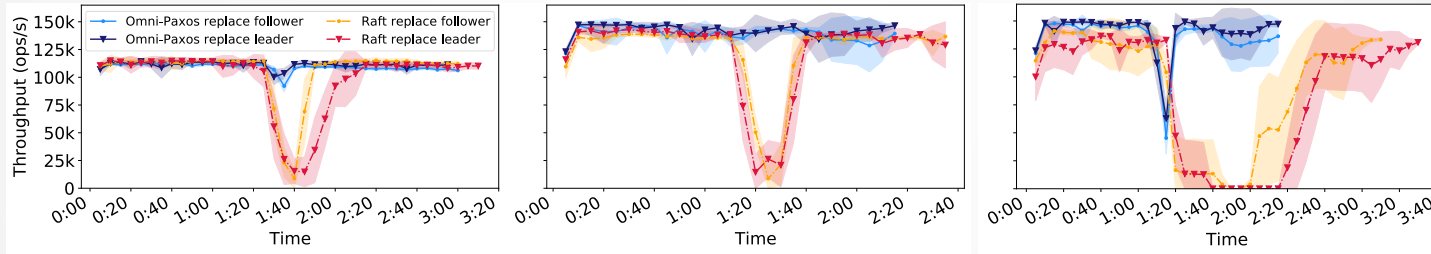


RAFT RECONFIGURATION

- Omni-Paxos: stop current configuration, then start new one.
 - Log migration to new servers in service layer.
- Raft uses a “joint-consensus” approach.
 - Intermediate configuration with both old and new configuration: $C_{old} \rightarrow C_{old,new} \rightarrow C_{new}$
 - In $C_{old,new}$ commands can continued to be decided, but must get majority from both C_{old} and C_{new}
 - Leader can be any server in C_{old} or C_{new}
 - New servers catch up the log following the normal log replication protocol. When majority in both C_{old} and C_{new} has caught up, only use C_{new}

RECONFIGURATION EXPERIMENTS

- Raft leader gets overloaded: must migrate log to all new servers.
 - Down-time if leader is replaced.
- Omni-Paxos: parallel log migration in service layer reduces down-time.



(a) Replace single server with $CP = 500$.

(b) Replace single server with $CP = 50k$.

(c) Replace majority with $CP = 50k$.

Figure 11: Reconfiguration experiments. The shaded areas show the 95% CI using the t -distribution.

SUMMARY

- Raft is designed to be understandable.
- Incorporates leader election, log replication and reconfiguration all into a single protocol.
- Log requirement in leader election causes problems with partial connectivity.
- Performing log migration in log replication results in leader-bottleneck.

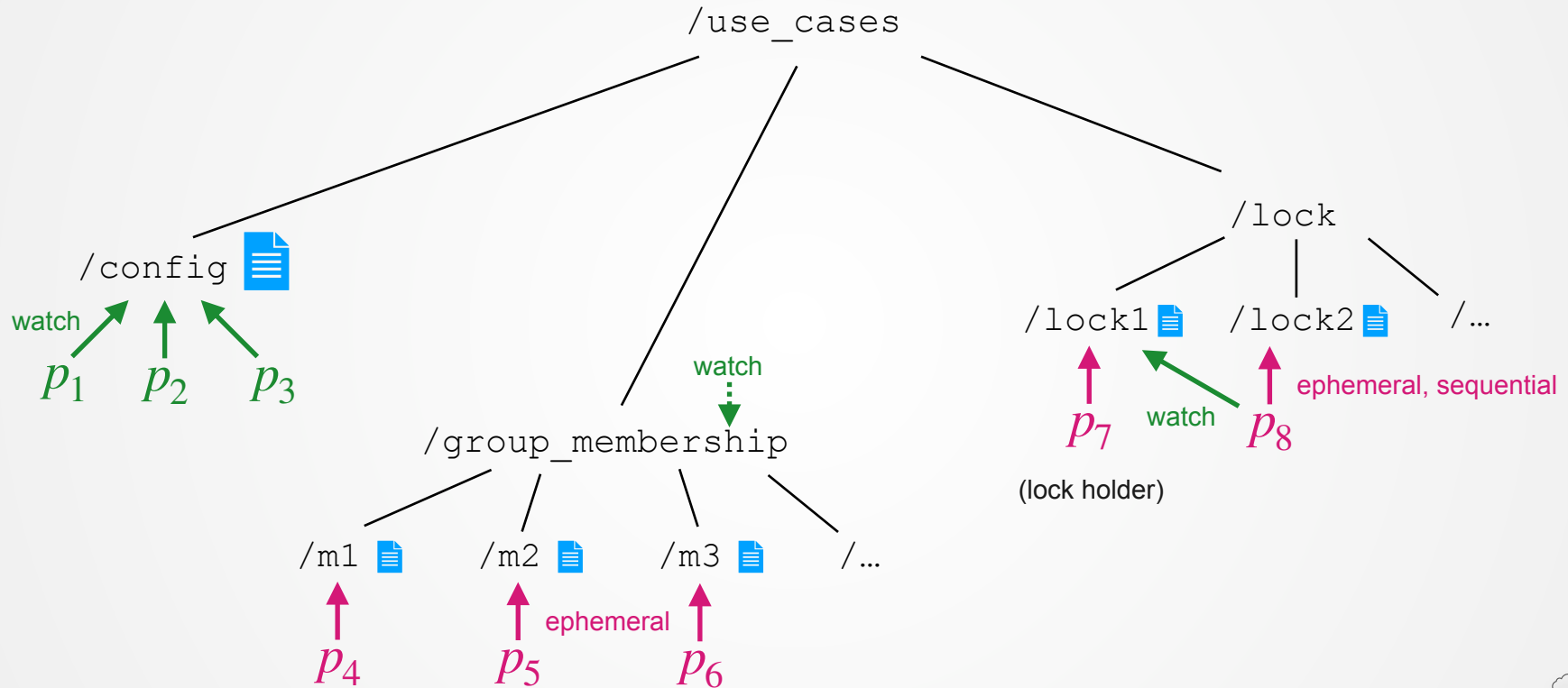
ZooKeeper

- A distributed coordination service.
 - A complete and general-purpose system.
 - File system API: hierarchical structure of nodes
 - Lock service, group membership, leader election, etc.
- Widely used: Apache Hadoop, Kafka, Flink, Spark etc.
- Based on ZooKeeper Atomic Broadcast (Zab)
 - Original was similar to Sequence Paxos but later became closer to Raft

CONSISTENCY

- Totally-ordered writes.
 - Do not support linearizable reads due to performance.
 - This would require reading via the leader or a majority.
 - Instead, we allow any replica to serve read from its local state.
- FIFO client order:
 - “read-your-writes”: read might stall until preceding write is complete.
 - Read after read: must guarantee that the second read is at least as updated as the first. But different replicas could serve these requests and thus might also stall.
- Can use *sync* operation to perform a linearizable read that is decided in the log.

COMMON USE CASES AND PATTERNS



SUMMARY

- Omni-Paxos first stops the current configuration by deciding stop-sign, before starting the new configuration.
 - Parallel log migration in the service layer, decoupled from log replication.
- Raft: designed for understandability
 - Monolithic: log replication, leader election and reconfiguration all in a single protocol
 - Cannot handle partial connectivity and leader-bottleneck during reconfiguration.
- ZooKeeper: a general-purpose distributed coordination service
 - File system API: group membership, lock service, etc.
 - Weaker consistencies for performance.