Replicated Database (Raft)

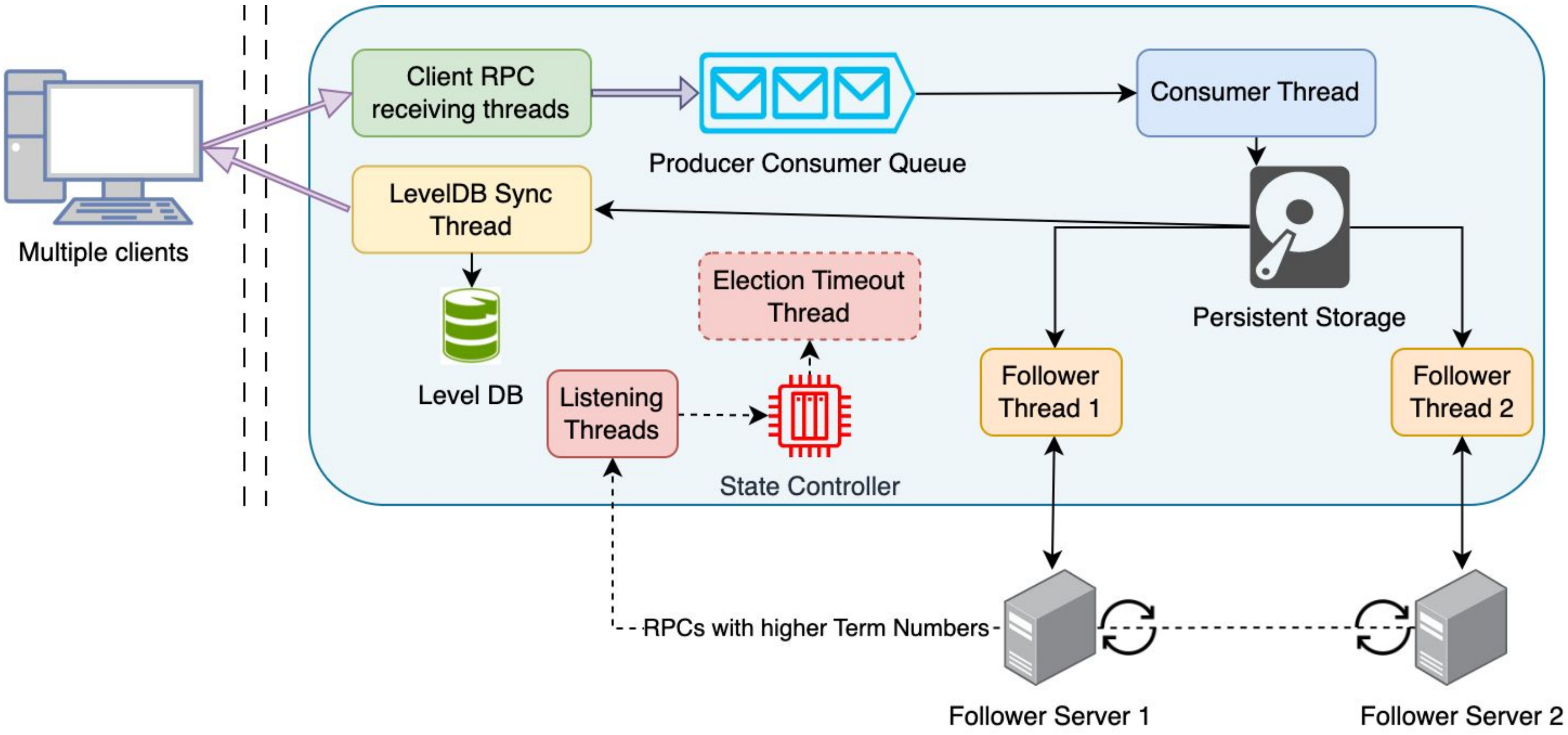
(Project Group 2/10)

Aadi Swadipto Mondal | Mohil Patel | Rahul Uday Chakwate

Key Design Decisions

- Replicated key value database is designed using Raft.
- Entire system is polling based (no signalling in-between threads). Threads poll and update shared (persistent) states and act based on their values. States (including persistent metadata) are updated atomically (using locks and atomic instructions).
- Log entries are lazily synced with followers (separate follower thread for each follower performs log sync).
- Client uses a token system to track “put” requests (always served by the leader).
- Election procedure is modified to ensure candidate with most updated log wins.

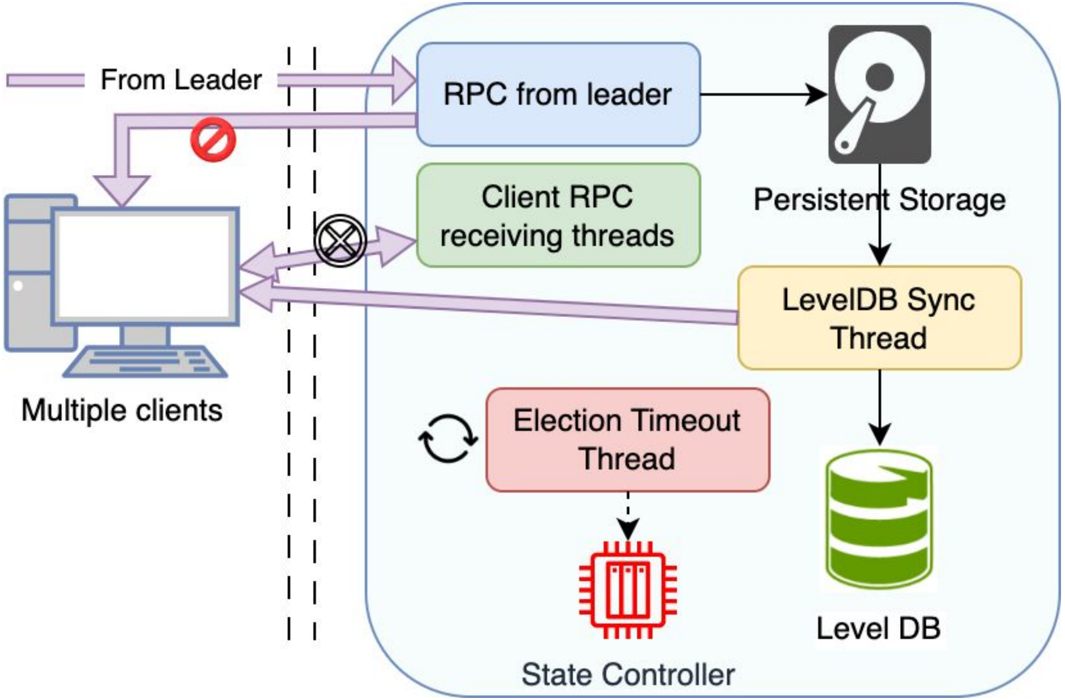
Leader Design



Leader Design (continued)

- Client requests are received by the GRPC server. “get” requests are served directly from that thread, and the “put” requests are queued in a producer-consumer queue.
- Separate consumer thread picks entries from the queue and write it to the leader’s log
- Each follower machine has a separate syncing thread which polls on log and pick entries as they arrive. Additionally they also update the majority bits.
- Whichever follower syncing thread received majority for that entry updates the last commit index.
- State sync thread polls on last commit index and apply committed entries to LevelDB, & also send +ve acks to the client.
- Lastly, leader’s GRPC server listens for higher term RPCs from other machines and steps down if received.

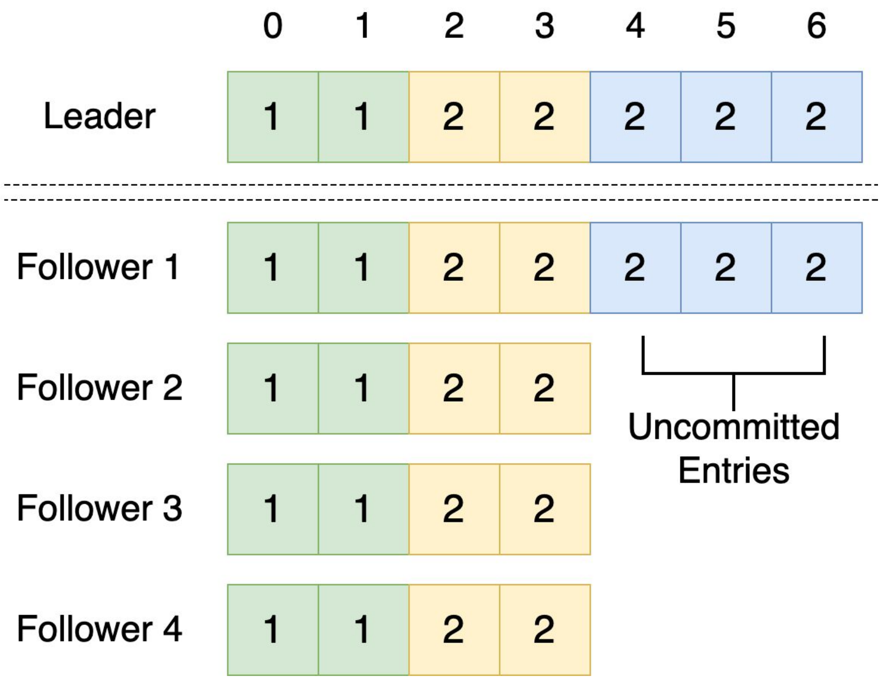
Follower Design



- GRPC server thread listens for RPC
- If no RPC received for a while, election timeout thread triggers election
- AppendEntries RPC used by leader to new log entries
- Last commit index updated in AppendEntries RPC based on leader's last commit index & log index being written
- In case any old entries are overwritten in AppendEntries RPC, we send a -ve ack to the client

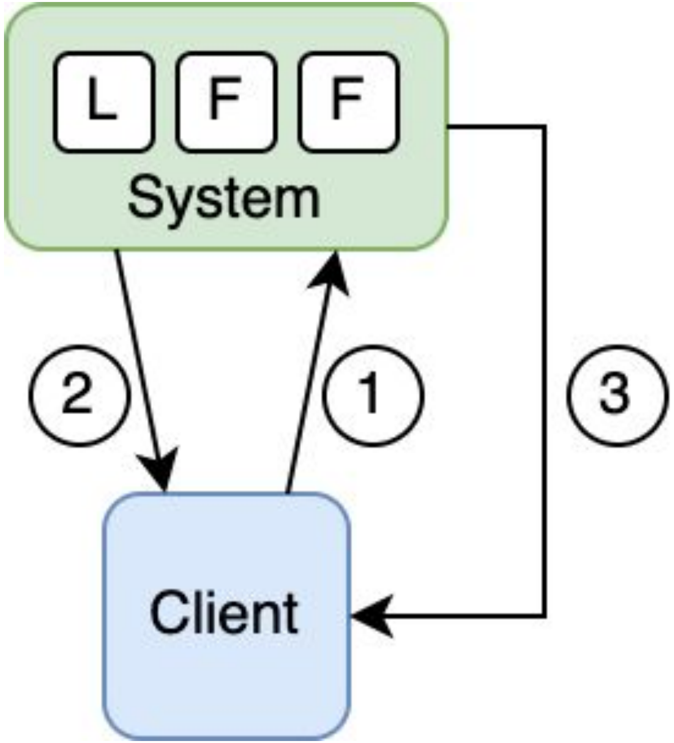
Election Procedure

- System follows the standard raft election procedure, where each machine can start a new election after election timeout and request votes
- In request vote RPC, we check last log term and log length as expected in standard Raft
- Additionally in our system, the candidate steps down if any vote is denied (even if majority is received)
- This design is aimed to minimize the overwrites and thus reduce -ve acks send to the client

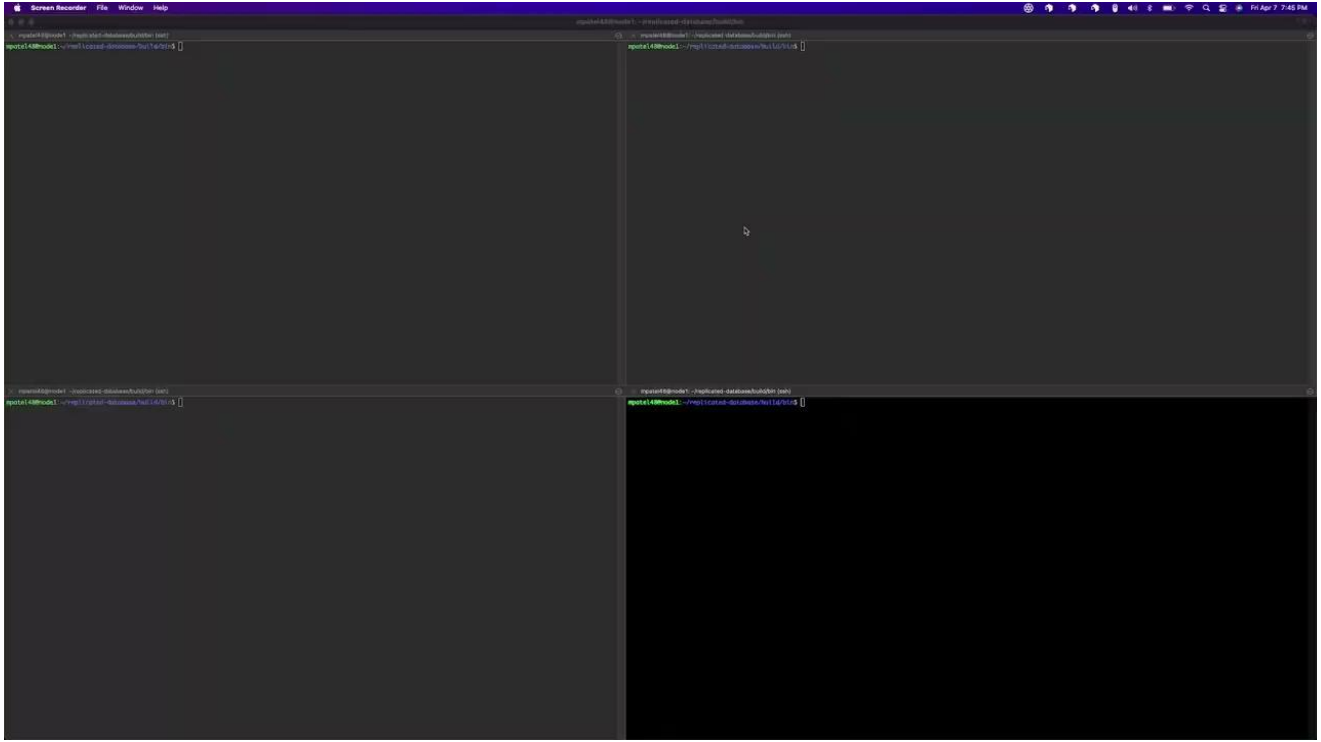


Client Token System

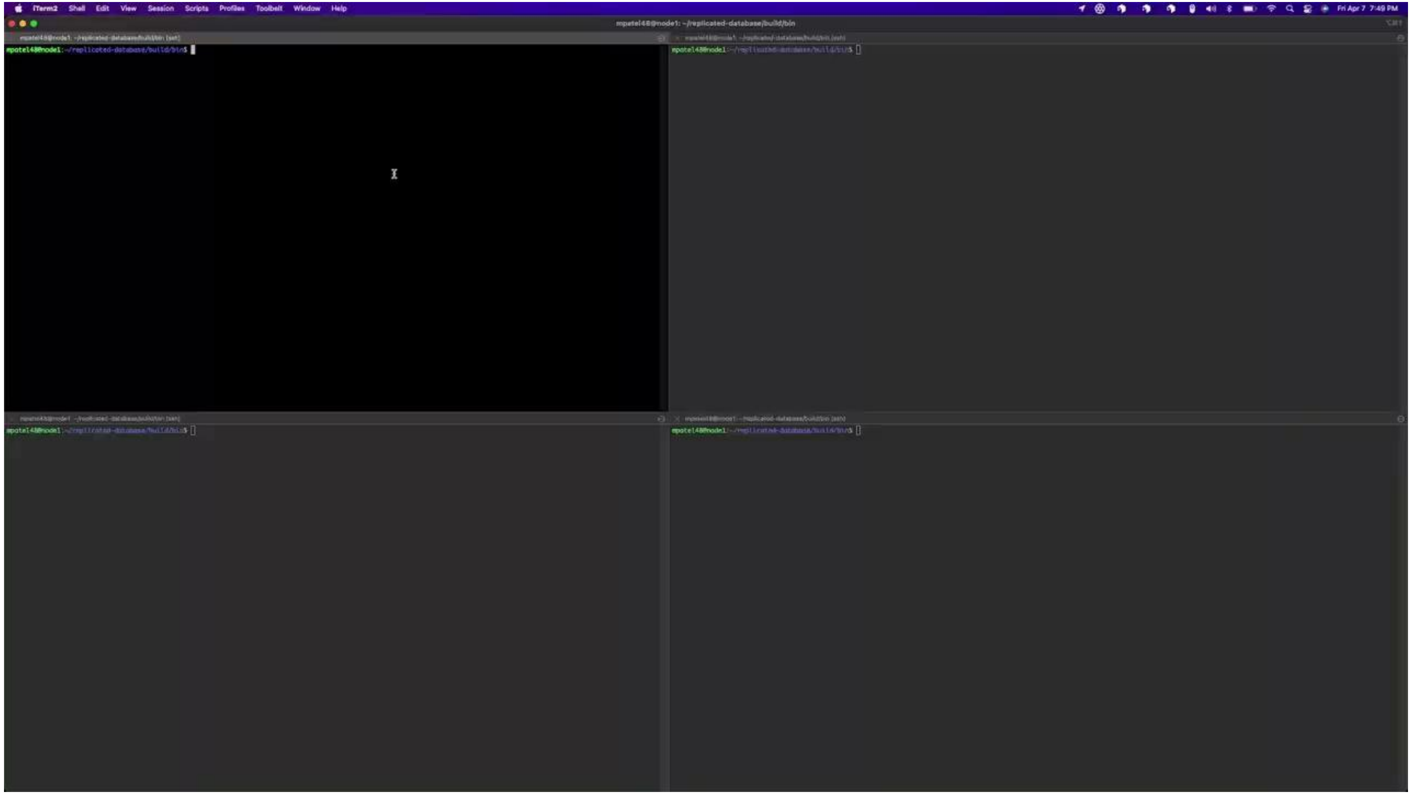
- Steps for client contacting the Raft Servers:
  1. Client sends the request to the leader.
  2. Leader acknowledges the receipt of the response (may fail if leader queue is full).
  3. Client receives +ve/-ve acknowledgement at some time in future for "put" requests.
- "get" requests are responded successfully only if leader has at least one entry of it's term committed.
- Client API provides both blocking and non-blocking put requests (blocked until ack-ed or timeout).
- Client caches leader ID after first successful request and updates it on failure to reach leader.



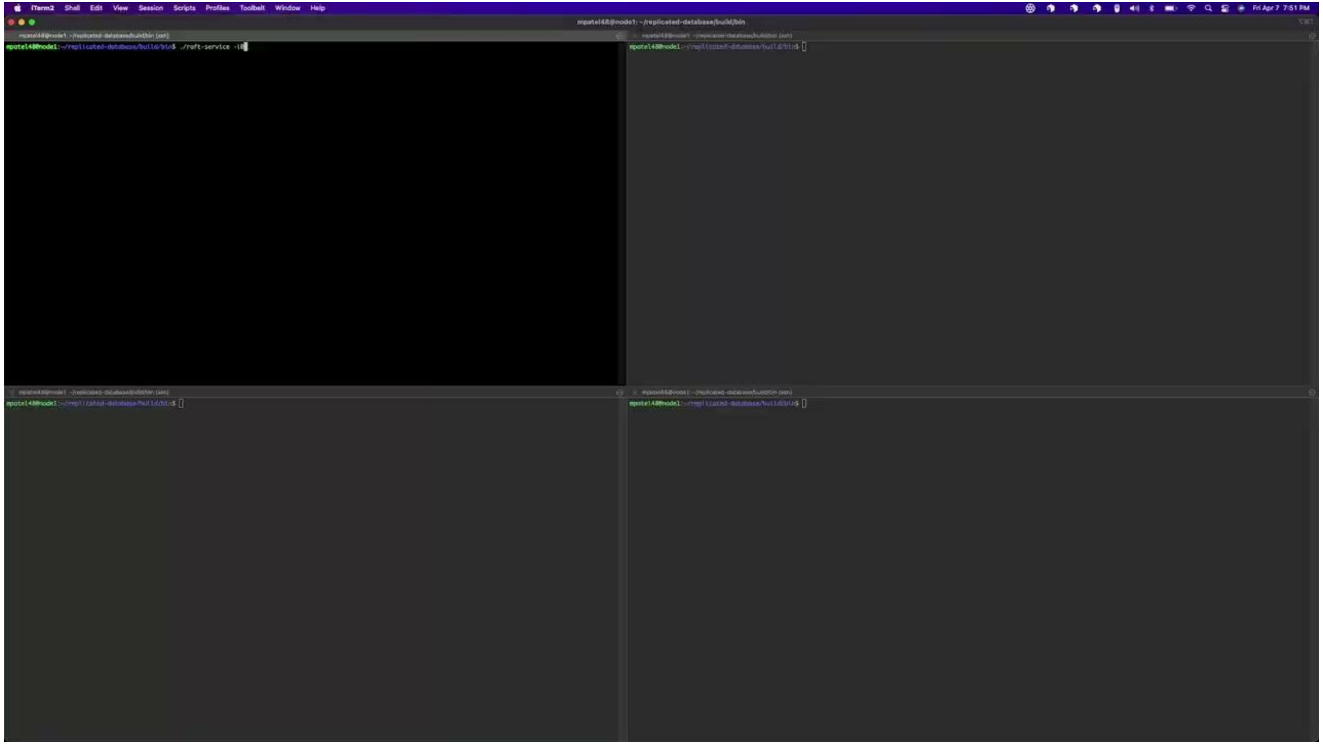
Demo - normal operation



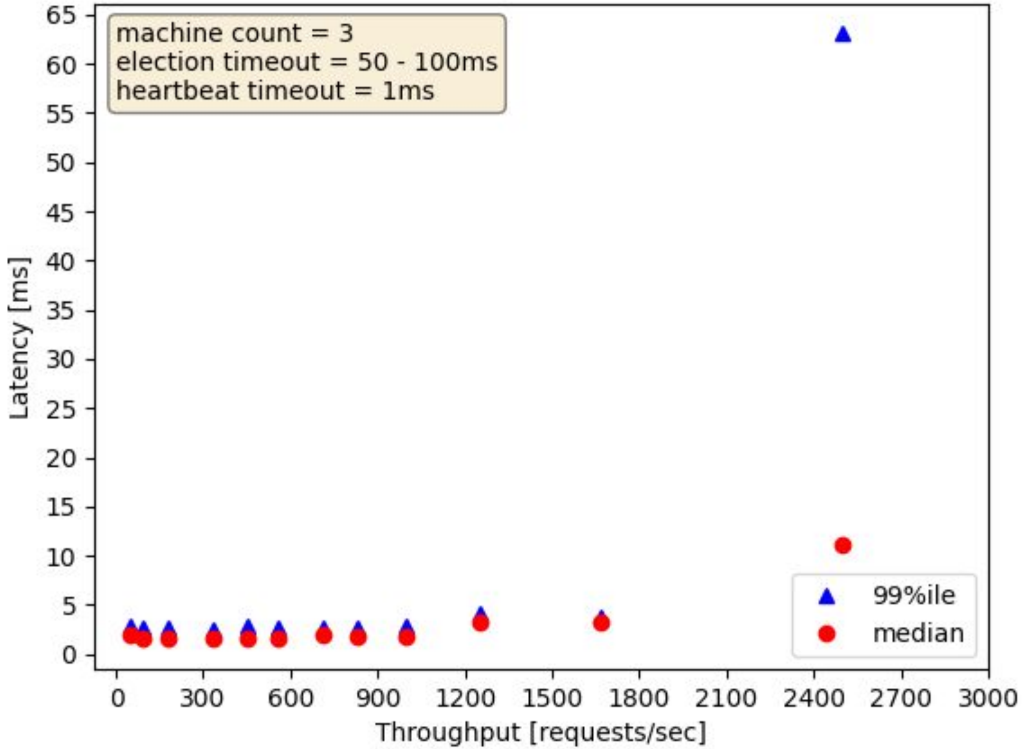
Demo - leader crash



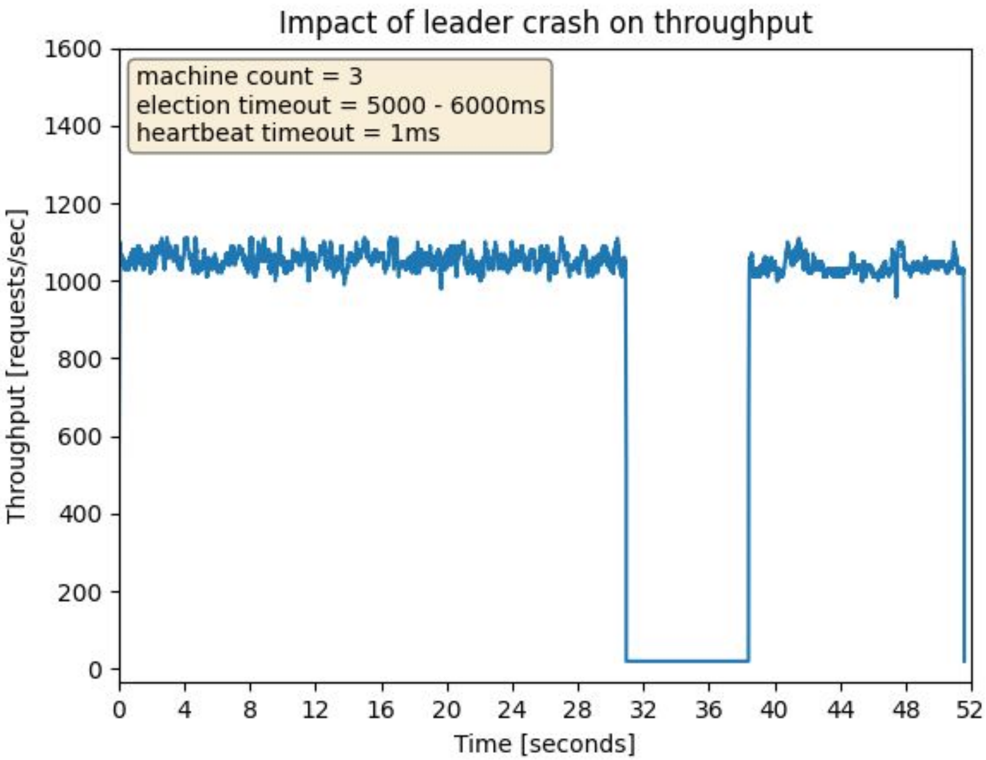
Demo - follower crash



Performance: Throughput vs Latency

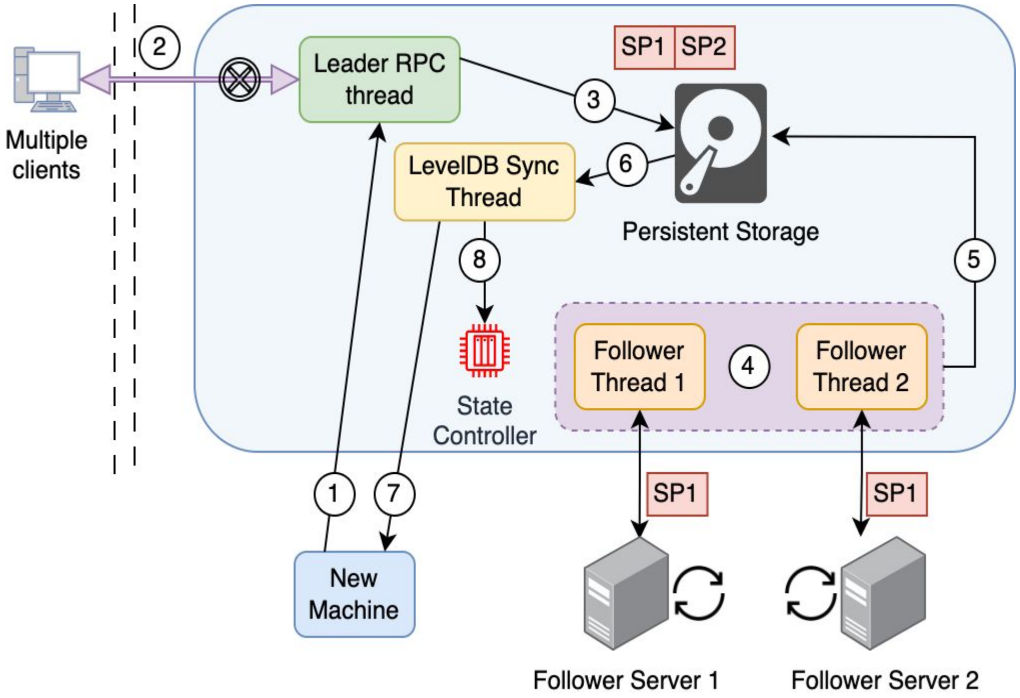


Impact of leader crash on throughput



Membership Change Protocol

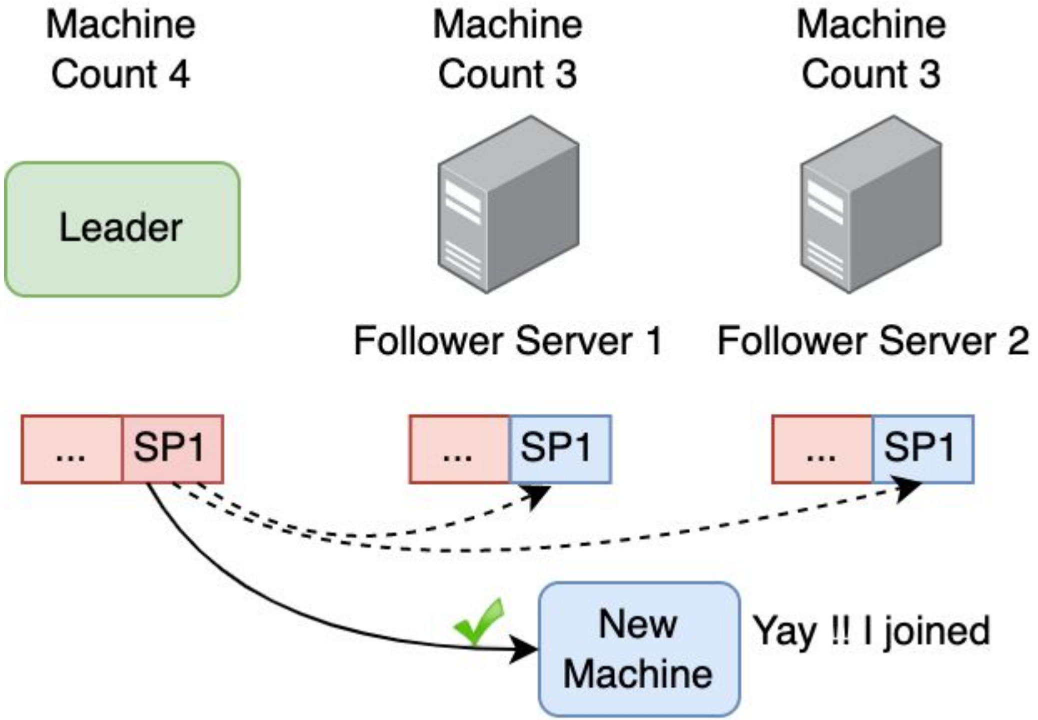
1. New machine sends a request to the leader to join the raft system.
2. Leader stops accepting new client "put" requests. "get" requests continue.
3. Leader writes two special entries in the log (SP1 & SP2) with new server count.
4. Leader tries to replicates SP1 and SP2.
5. Leader waits for majority of SP2 and notifies SP1 is committed.
6. Leader increments machine count on majority of SP2 (follower increments when they commit SP1).
7. Send +ve ack to new machine to start raft.
8. Leader steps down for re-election.



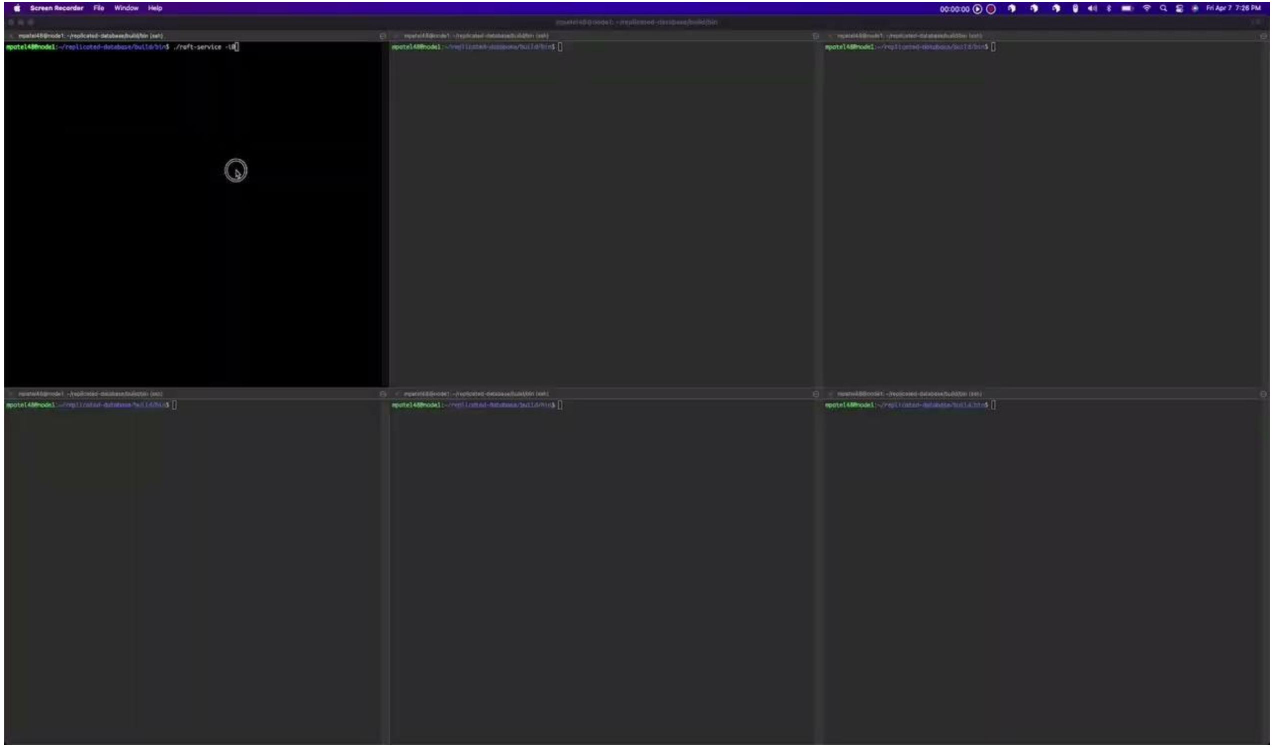
Why two special entries??

Proof by fallacy:

1. Leader replicates SP1 among majority of followers and commits SP1
2. Leader increments its own machine count.
3. Leader acknowledges new machine and the new machine starts as a follower.
4. Leader crashes or gets into network partition.
5. Any follower gets re-elected as a 3 machine system although the new machine has already joined.



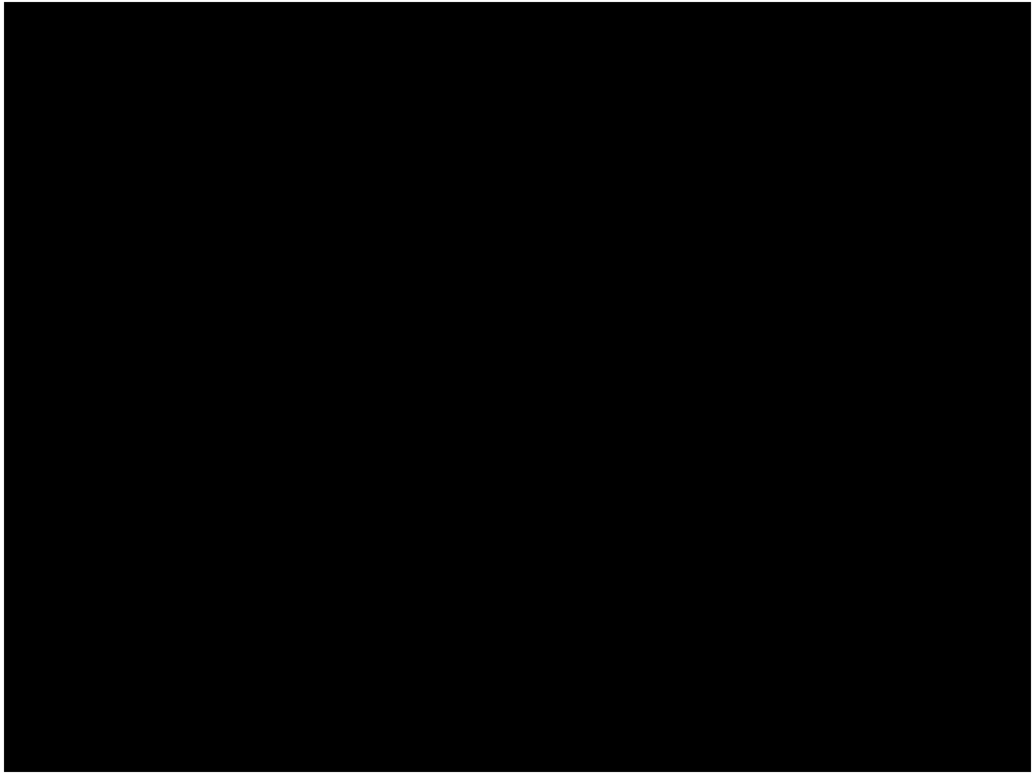
Demo - membership change



Testing Correctness : Gracious Monkey

- Designed a random network partitioning system (inspired from Netflix's Chaos Monkey).
    - Uses `iptables` linux command to create self network isolation (drops all incoming and outgoing packets), something similar to Firewall.
    - With some probability decides whether to apply/keep the partition or withdraw it.
    - Works successfully for gracious time limits (all time limits in order of seconds).
- Adopted fail-fast programming style and did manual testing for cases like:
    - Leader Crash (leading to re-election of new leader).
    - Follower Crash (can sync back with the leader when re-started).
    - Network Partitions for both the leader and followers.
        - Leader in network partition leads to re-election among followers. When back, the raft system adopts the old leader as a follower.
        - Follower in network partition can detect absence of heartbeat and enter election phase. When back, other servers increment term and participate in new election.

Testing Correctness - Demo



Takeaways!!

- Hard crashes can lead to metadata corruption:
  - Updates to multiple persistent files need to be atomic (similar to what is shown in the Alice paper). Hard crashes can lead to file corruption leave inconsistencies within metadata.
  - Inconsistent metadata can later be propagated across the raft system (similar to “Redundancy Does not imply fault tolerance” paper discussed in class).
- Polling based systems are easier to design than event-based systems:
  - Conditional variables in event-based systems can potentially lead to deadlocks.
  - Programming polling based system is easier with locks and atomic instructions.
- Testing Correctness for Raft is hard:
  - There are many edges and it is not possible to cover them all.
  - Manual testing is not possible at millisecond time scales (human responsiveness).
  - Sometimes GRPC can stall depending on when network partition happens.