Design

This project is an implementation of a fault-tolerant replicated database using a custom Raft consensus algorithm in which all server replicas execute client SQL requests in the same total order, maintaining consistency even through crashes and restarts. This system implements 3 core mechanisms of Raft: leader election with randomized timeouts(150-300ms) to prevent split votes, log replication using AppendEntries RPCs that also serve as heartbeats, and safety guarantees using term conflict checking before truncating the logs. We separate client communication from server-to-server Raft protocol communication using different network ports (offset by 100), preventing interference between client requests and internal consensus messages.

Persistence uses Java object serialization for metadata (currentTerm, votedFor) and log entries, but importantly stores the lastApplied index within Cassandra itself to prevent re-execution of committed transactions after crashes, ensuring exactly-once semantics. Log truncation triggers at 400 entries by snapshotting lastIncludedIndex and lastIncludedTerm, using Cassandra to store application state instead of serializing the entire database. We achieve idempotency through unique identifiers for requests that combine serverID, client socket address and request ID. Concurrency is managed through sync'd blocks protecting the raftLog. Commit index incrementing follows majority rules by tracking matchIndex. Crash recovery consists of a multi-step state restoration by loading the persistent state, querying Cassandra for lastApplied, replaying uncommitted entries and then rejoining as a follower.

Tests Passed

Tests 31-38 have been passed locally.

Issues

The code contains three critical bugs. Firstly, the sendAppendEntries method fails to update matchIndex and nextIndex when a heartbeat succeeds. This prevents the leader from acknowledging that followers are caught up for existing log entries, causing the commitIndex to stall indefinitely for past terms until a new command is issued. Secondly, the system lacks an InstallSnapshot RPC mechanism in sendHeartbeats. If a follower lags behind the leader's lastIncludedIndex (due to log truncation), the leader cannot send the missing log entries (as they are deleted) and fails to send the snapshot, causing the follower to be permanently unable to rejoin. Third, there is a memory leak in executeCommand because pendingRequests.remove(requestId) is located inside the try block after execution. If session.execute throws an exception, the request ID remains in the map forever, causing memory growth and blocking the client from retrying that specific request ID.