project4_executive_summary_weiqing_gao

Assignment Overview

This assignment focuses on the implementation and understanding of the Paxos by introducing consensus protocols, leader election, and fault-tolerant replication. The goal is to ensure consistency and availability across five replicas, even in the presence of simulated failures or role crashes. In addition, this project introduces ring-based leader election, allowing any node to become a proposer and initiate consensus if elected as the leader. The critical learning outcomes include mastering the Paxos protocol, building supervision mechanisms for fault recovery, and integrating distributed election algorithms into a replicated state machine.

Overall, the project transitions the system from a coordinator-based atomic commit model (2PC) to a fault-tolerant consensus model (Paxos), introducing practical challenges such as distributed role supervision, leadership dynamics, and log consistency under partial failure.

Technical Impression

This assignment offered a hands-on experience in implementing a multi-role consensus system with fault tolerance. Compared to 2PC, one of the core takeaways of Paxos was that distributed consensus can be made fault-resilient not by central coordination, but through redundancy and agreement over majority.

To implement this, I designed and modularized the system into Paxos roles: Proposer, Acceptor, and Learner, each supervised via a restartable thread model. When the proposer is elected as leader through a ring-based election, it initiates proposals by contacting acceptors using Paxos prepare/accept messages. The system reaches consensus if a majority of acceptors promise and accept a given proposal. A no-op proposal mechanism is also included to enable heartbeats and liveness detection.

One technical challenge was understanding how to maintain progress despite transient role crashes. Initially, Paxos role threads would silently fail when simulating faults, causing the whole system to stall. To address this, I implemented a `RoleSupervisor` class that automatically restarts failed role threads. This introduced reliability and allowed the system to recover from intermittent failures without losing leadership or proposal state.

Another highlight was the implementation of ring-based leader election, which ensures that all nodes can participate in electing a new proposer if the current one fails. During development, I observed that if the leader was not initialized or synchronized across nodes, all proposals would be rejected with `"Not leader"` errors. To mitigate this, I ensured that each node temporarily assumes itself as leader on startup and then updates `leaderAddr` upon receiving the token. This approach minimized the impact of startup race conditions.