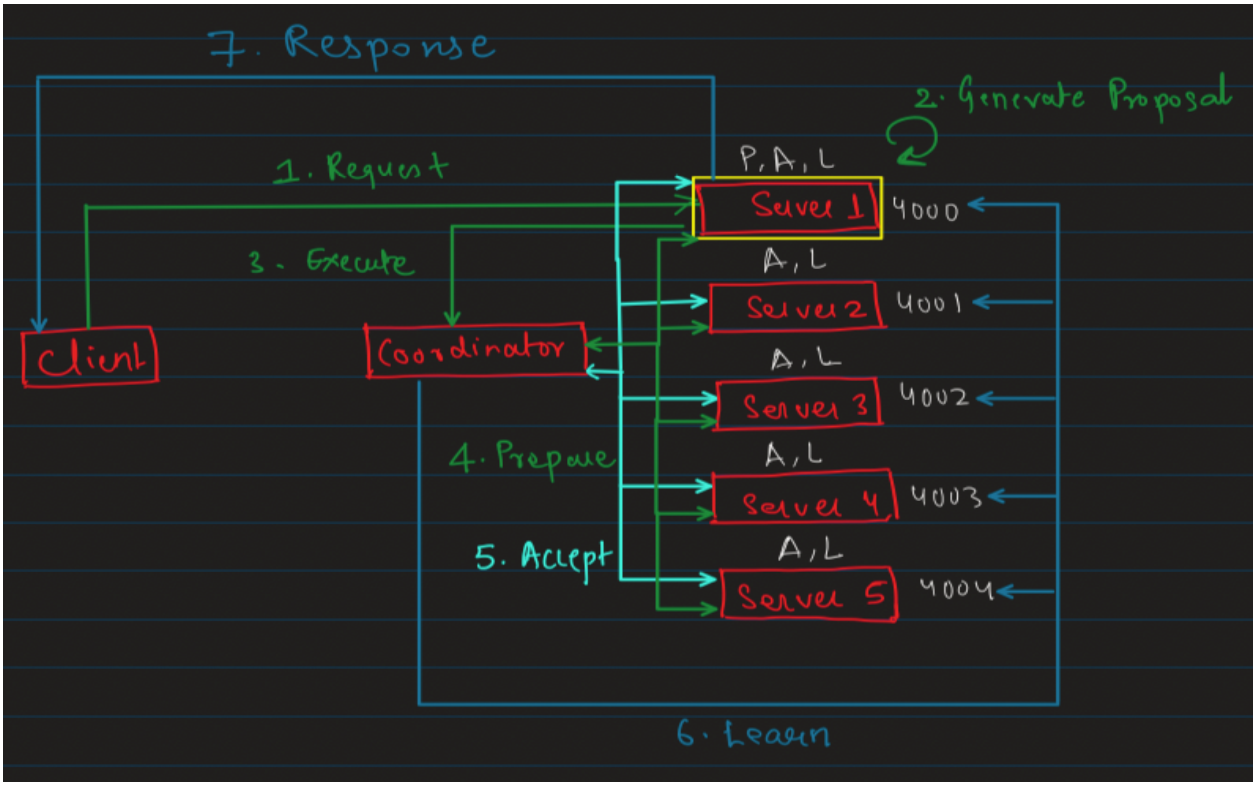
Project 4 Executive summary

Assignment Overview

The purpose of this assignment was to modify Project 3 by replacing the already implemented 2PC algorithm with Paxos algorithm. Using Paxos algorithm ensures greater fault-tolerance. This helps in continual operation of the key value store despite some replicas going down. The design had to be based on the paper “Paxos made simple” and had to consist of proposers, accepters, and learners. We also had to simulate failures by making the acceptors fail randomly and then restart them to resume functions. This would help demonstrate, how Paxos ensures reliability even if some servers are unavailable. We had to use RPC as the underlying communication protocol for the entire application. Simulating failure for other components like proposer and learner was optional. However, I decided to simulate proposer failures as well. The client would make a request to the server which would run Paxos before performing the operation requested by the client.
As defined in earlier projects, we had to pre-populate the server with seed values and then perform 5 GET, PUT, and DELETE operations on the server.

Technical impression

Designing Paxos form scratch seemed complex initially due to the number of components involved and interactions that had to be performed between components to reach consensus. To make the implementation easier, I decided to draw a high-level diagram that illustrates the interactions between all the components. The HLD made it easier to replicate the design in the code.



I also decided to use a coordinator for orchestrating Paxos. The application starts at client that sends a request to the server. I designated one of the servers as a proposer which would accept client requests. After receiving the client request, the proposer generates a proposal with a monotonically increasing unique Id. The proposer then invokes the execute method of the coordinator with the generated proposal. The coordinator upon receiving the proposal, first sends a prepare request to all the Acceptors. If the acceptors accept the proposal, they send an acknowledgement back. If a prepare majority is obtained, the coordinator moves on to the accept stage and sends an accept request to all the Acceptors. If an accept majority is obtained, the coordinator sends a learn message to all the learners which execute the operation requested by the client. If a majority cannot be obtained during the prepare or the accept phase (due to acceptor failure or if n < maxId), then the algorithm is terminated, and an error response is sent to the client. I also decided to add a randomly generated delay (>= 0 and <1s) during the prepare stage to simulate delay caused by real-life network calls. The coordinator moves on with the next stage only when it receives responses from all the acceptors.

One of the challenges that I faced during implementation was to establish a means to perform 2-way communication between the coordinator and the acceptors/proposer. My prior project implementations used Apache Thrift for RPC communication. However, Apache Thrift does not provide 2-way communication capability. So, I had to replace my apache thrift setup with a java RMI setup.

During testing, I found that the setup is robust and ensures consistency of data across all servers even on failures.

This project was really helpful in getting a practical understanding of the Paxos algorithm. Reading the “Paxos made simple” paper was helpful, but I still had questions about its working which were answered during the project implementation.