Aayush Joshi, Ben Hinchliff, Gabriel Rodriguez, Tymon Vu
CSC 569
Dr. Pantoja

Lab 3 - Raft

Raft is meant to provide a functionality for distributed consensus, including nodes to assume roles as followers, candidates, or leaders. Handling elections and voting for leaders. Our nodes transition between states following gossiped heartbeats and election outcomes. With each node tracking the current term, votes, and leader information. The election mechanism utilized causes nodes to become candidates after not hearing from the leader in a randomized timeout, then requesting votes from peers to elect a leader. Gossiping synchronizes states, heartbeats, and liveliness with nodes being able to detect failure if a peer hasn't sent a heartbeat within our specified timeout. Although a proper implementation of Raft should be utilizing peer-to-peer communication, we decided to stick with the client-server flow to avoid headaches and stick with the RPC implementation of our gossiping protocol, but discovered newer issues when implementing just the basic Raft consensus protocol.

Our implementation wasn’t without its struggles, as accounting for edge cases and timing the protocol led to weird deadlocking situations when voting for a leader. The limited number of nodes requires the program to achieve some finesse for the nodes to align and vote, as well as leaving a good amount of time for an election to allow for a leader to actually be elected. As well as the server-client model requiring weird workarounds for pushing information forth to other nodes, and sometimes the membership table that the server had would be outdated/have the wrong information. Resulting in situations where nodes would be marked dead that were alive, or alive but were very much dead. Given more time, improvements could be made to include more aspects of the Raft protocol, but core functionality is implemented and working.