Jim Crowell
Jackson Morton
CS 455: Distributed Systems (“Systro”)
P3 White Paper
4/7/22

Engagement Model

Our intent is to keep to the one-shot client/server model, partially because it helps to compartmentalize how functions work (that is, since a client spins up, sends a request, and shuts down, we don’t have to worry about multitudinous states that the clients could be in and performing further requests from), and partially because it’s fewer alterations to make, and therefore fewer opportunities to introduce catastrophic failures to the system.

Coordination

For coordination, we will elect a coordinator that is elected through the bully algorithm. When a process in our system notices that the coordinator is not responding it will begin an election process that will use the bully algorithm to elect the next coordinator. For communication between the servers, we plan on using a daemon on each server that will essentially periodically ping other servers and also receive pings from other servers in order to determine who is alive and who is down.

Consistency

Tentatively: Causal Consistency. Easier to compartmentalize, similar to the engagement model, in that each process, if not “maintaining” consistency in regards to the others, effectively sets the order of events, which other processes must follow. The timestamps, obviously, will help maintain this consistency.

Synchronization and Granularity

Lamport timestamps, for sure, because LT’s are another example of keeping the “work” of maintaining them (the timestamps) with the client, which it will communicate to the server, rather than having a Vector

Clock that’s centralized on the server and tracking the clients. The saying “many hands make for light work” is in full effect, and distributing the work seems more in line with the spirit of the class. As for granularity, we’re interested in logging and checkpointing server activity from time to time and adding the new events to the master backup, rather than writing the entire backup every time. Smaller individual writes should make for less overhead for maintaining the record; while writing is Idempotent (TM), sending writes including “old” data would still require the server to check what it’s being given to write vs what’s already there OR rewrite data that’s already extant, which we see as wasted effort.