Caleb Norton

CSCE 662

MP2.2 Design Doc

To keep all operations synchronized to the filesystem, I wrote a filesystem wrapper (sns_db) that used unix “flock” (file lock) to ensure that files are only accessed by one process at a time (server and synchronizer)

To make sure filesystems that are in the same cluster are in sync when both servers are online, I essentially forward each RPC call that the client makes to the master server on to the slave server. This way, both the master and slave server see every operation from users in the cluster.

Finally, we need to ensure that data is synchronized between clusters. In this case, the master synchronizer

To handle master server failure, the coordinator keeps track of how long ago the last received heartbeat for each master server was, then fails-over to the backup server and redirects clients to connect to it. The coordinator also informs the old master synchronizer that its server has failed and informs the slave synchronizer to take over as master along with its SNS server.

I order for messages from other clusters to show up live, the timeline handler periodically checks the database file (which could be updated by the synchronizer) and forwards new messages it has not sent to the client already as it sees them.

I had issues with race conditions in the starter code implementation of rabbit mq message passing where messages would be retrieved from the wrong queues so I had to work around it with additional field checking to ensure each message type was processed correctly.

My implementation is very inefficient since each master synchronizer essentially uploads its entire database every few seconds… however, this is fine for a proof of concept and could be improved later.

Ideally, the database synchronization would be handles by database primitives and native functions, not a synchronizer service that runs on top of it.