**Executive Summary**

Sachin Mohan, Chris(Letian) Sun, Navjot Singh

In Project 3, Paxos implementation, we got to learn and build on what we did in Project 2. In Project 2, we used a technique called two phase commit to keep data consistent between machines. Two phase commit would basically have a process of sending acknowledgements to ensure that machines are up and then using go messages to commit a transaction. The problem with two phase commit is that it is not fault tolerant. If a particular machine goes down and fails to respond to an acknowledge message, two phase commit will not continue and the entire system will comes to a halt. This is not ideal or the best thing to happen because in the real world, machines do go down. Paxos handles this fault tolerance problem. Compared to two phase commit in which when one machine goes down, the whole process goes down, in Paxos, this problem doesn't exist because it relies on quorum, i.e. if majority of machines are up then the system can function without any issues. In our assignment of 5 replicated servers, that majority or size of quorum is 3. Another disadvantage for two phase commit is that when the leader or the coordinator goes down, the process can't continue (no coordinator to send acknowledge messages to). However with Paxos it is flexible in that any machine can assume the leadership role when the main Proposer goes down. The assignment was clear for the most part. What could clarified this assignment more would be provided test cases like the kvp operations that we can use to determine if our system works in case of live lock replicas.(something that tests all the edge cases).

The main challenges faced during the execution of this project was that of handling multiple threads and maintaining consistency. As each server had different processes such as Proposer, Acceptor and Learner doing operations simultaneously on shared resources maintaining consistency was a major challenge. Another challenge was forcefully making certain nodes to fail. This was achieved by calling Thread.sleep() on Acceptor thread when the system time matched a specific criteria. The criteria used for this was to divide the minute of the hour by the server no and when the remainder is 0 the thread would sleep for 10 seconds. Since the server number was different for all the servers this ensured that at any given time the majority of threads are awake.

**Performance Analysis**

Please find below the operation to time graph for Two Phase Commit and Paxos below

We could see that the time taken for each operation by Paxos is much higher than that of Two Phase Commit. This is due to the fact that for every operation there are a lot communication in the form of Proposal, Promise, Accept, Acknowledgement, Commit, happening between the servers in case of Paxos. This trade off in time has to be born in order to achieve fault tolerance as machines in distributed systems are prone to failure. Also find attached the spreadsheet containing the time taken by each operation.

**Reason for Delay in Submission**

Unlike the rest of the teams who use single thread for communication by a proposer to all the acceptors we had used multiple threads, that is one thread for an acceptor, to communicate with the acceptor. This was causing some deadlock which we couldn’t identify for a very long time. This caused the delay in submitting the project. If you take a look at the code you could see that there are individual threads handling each connection coming from a proposer and similarly individual threads in proposer to talk to each acceptor. On discussing with other classmates we came to know that this was not done by others but we tried to achieve this as this was the realistic way of implementation.