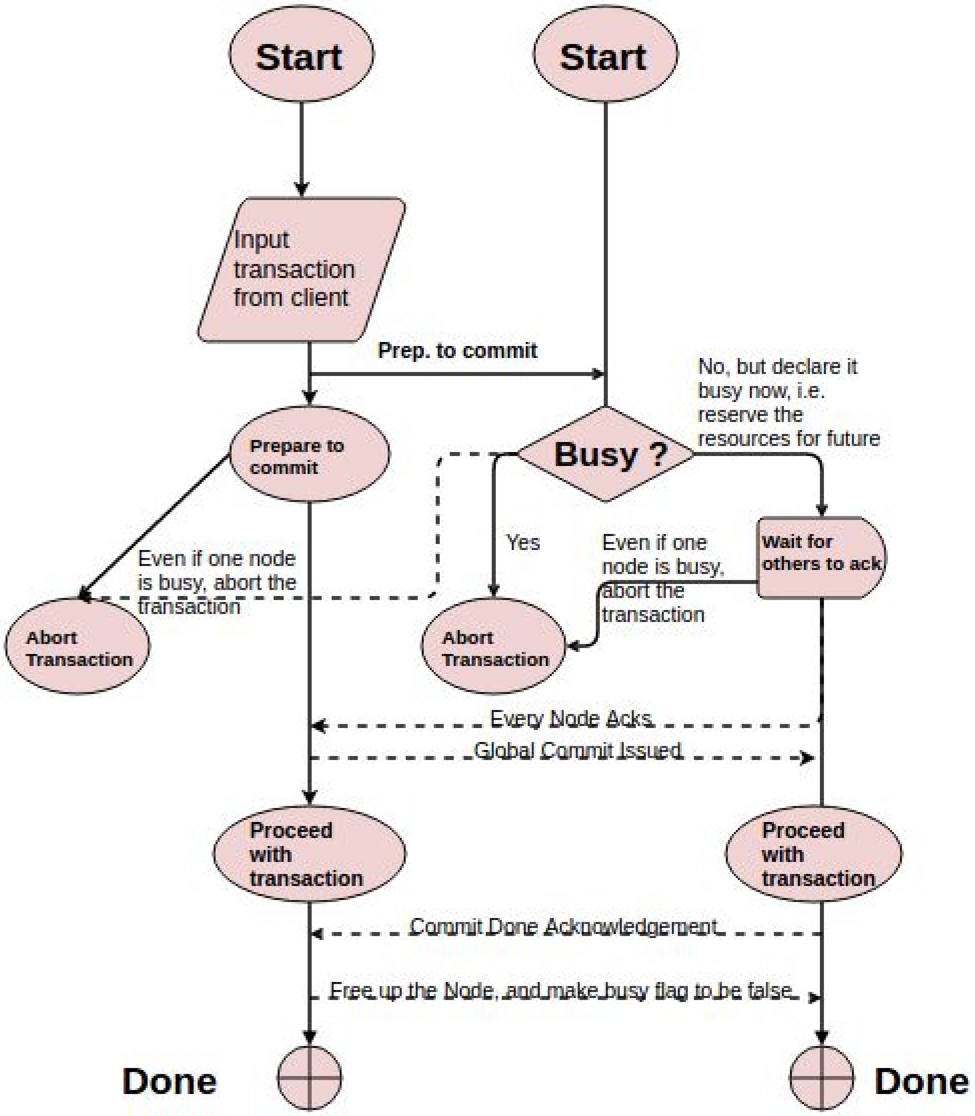
REPORT ADDENDUM

IMPLEMENTATION OF TRANSACTIONS VIA 2 PHASE COMMIT

We have implemented transactions using the following modified version of traditional 2 phase commit. The Coordinating node, as a first step determines the leaders of the keyspace requested by various commands in the transaction. The coordinator node then pings the determined leaders to check whether they are busy or not. Even if one node is busy, the transaction is aborted and client informed about resources being busy.

If all leaders are free, the busy flag is redefined to be true now, since they will be participating on the process, and thus will be busy. The coordinating node now executes the commands of transaction one at a time, and after completion, frees up the resources by resetting busy_flag to be false again.


Start
Start
Input transaction from client
Prep. to commit
No, but declare it busy now, i.e. reserve the resources for future
Prepare to commit
Busy ?
Yes
Even if one node is busy, abort the transaction
Even if one node is busy, abort the transaction
Wait for others to ack
Abort Transaction
Abort Transaction
Every Node Acks
Global Commit Issued
Proceed with transaction
Proceed with transaction
Commit Done Acknowledgement
Free up the Node, and make busy flag to be false
Done
Done

TESTING ON REMOTE MACHINES

We tested our code on 4 remote machines. The times observed to run the "**read tests"** (i.e. read 100 keys 10 times by randomly choosing the known node in the ring) was found to be minimum when there was only one machine. Upon increasing the number of hosts by calling join on another machine, the time of execution of the testing program increased, which maybe was due to network overheads. As we increased the hosts further, the time of execution didn't increase much, and tried to saturate, indicating scalability of the design.
Time taken to run on <u>1 machine</u> : **15.379 seconds** <u>2 machines</u>: **41.500 seconds**
<u>3 machines</u>: **81.405 seconds** <u>4 machines</u>: **66.657 seconds**