Lab3 Report

How to run:

1. Update the config\_file.json to change the ip\_address and allow it to available port.
2. Run the coordinator “***python node\_ coordinator.py*** “
3. Run the participant account A “***python node\_participant.py node2”***
4. Run the participant account B “***python node\_participant.py node3***”
5. Finally run the client to process the transaction “***python client.py***”

Simulate the basic 2PC protocol as discussed. There are 3 scenarios to be simulated.

a. A has 200 and B has 300 in their accounts initially. Everything works as expected, with  
no failures.  
i. Simulate all scenarios

b. A has 90 and B has 50 in their accounts initially. Everything works as expected, with no  
failures.  
i. Simulate all scenarios;

The first scenario returns True and replicates the changes to both account while in the 2nd and 3rd it returns False and as a result it does not propagates the new balances in the accounts.

A screenshot of a computer program

Description automatically generated

Code Explanation:

Coordinator code consists of CoordinatorNode, which is responsible calling the get balance (participant function) function for each participant before starting the transaction and based on that it will initiate it transaction method, which include the prepare and commit message

A screen shot of a computer code

Description automatically generated

Once, the prepare message from all the participants return True, it will broadcast a message ready for commit. We have a custom call for initiating the commit fail for coordinator which stimulates a coordinator crash scenario.

A screenshot of a computer code

Description automatically generated

Once, a commit message has been broadcasted to the participants through coordinator, it will updated the balances in respective accounts and update the coordinator transaction log also.

Incase, prepare message return fail as in case of 3rd scenario, it will abort the transaction on all participants.

A screenshot of a computer code

Description automatically generated

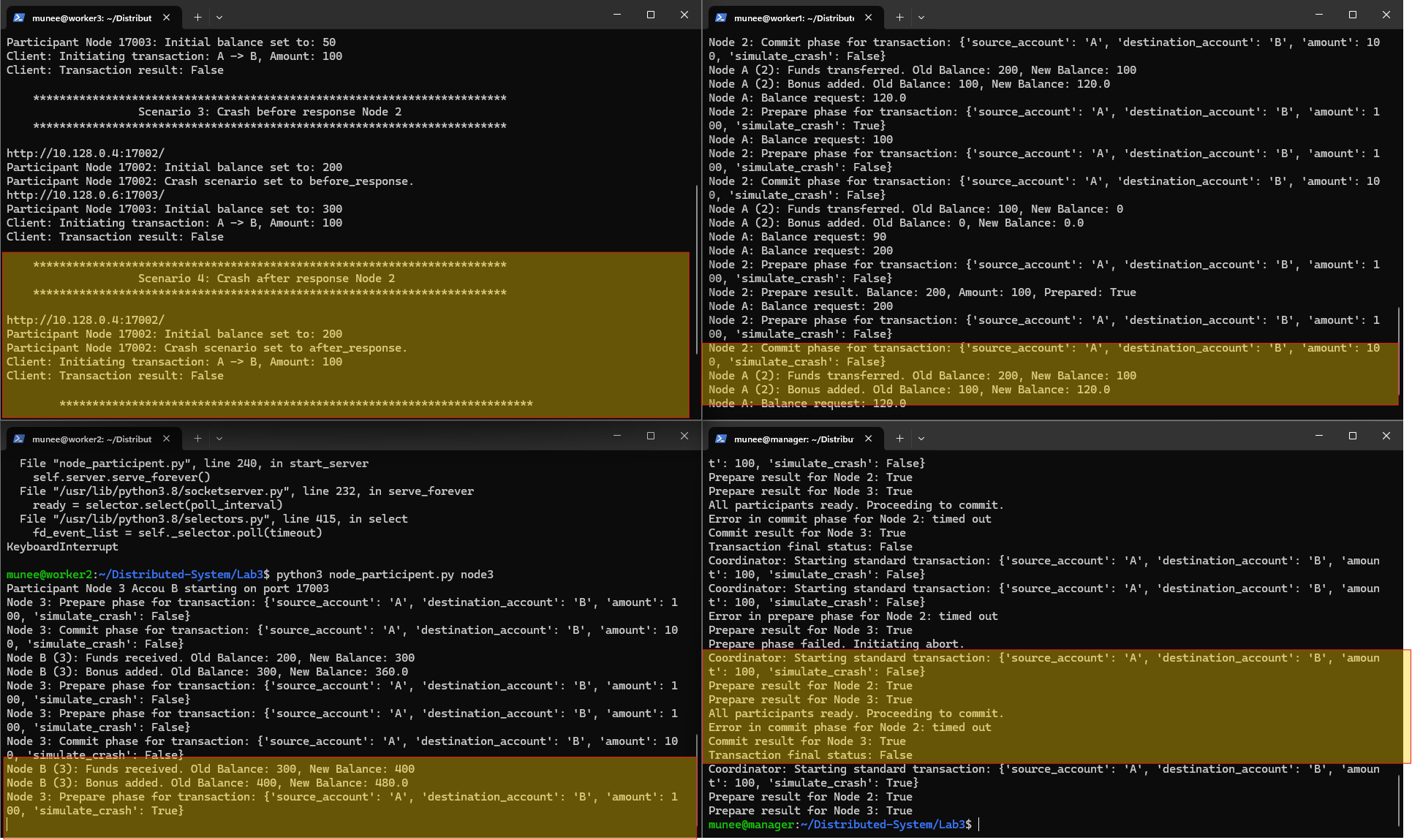
c. A has 200 and B has 300 in their accounts initially.  
i. Node-2 crashed (e.g., represented by a long sleep time) before responding to the  
coordinator.  
As the node fails before prepare, so it will receive a abort message form coordinator and that’s why we have final transaction False on client side and no balances were updated on respective accounts

A screenshot of a computer screen

Description automatically generated

ii. Node-2 crashed (e.g., represented by a long sleep time) after responding to the  
coordinator.

In our implementation, we imagine that when the participants failed after broadcasting the commit message, it will wait for recovery and when it recovers it will append the previous transaction based on commit or abort message from the coordinator, which will be track based on prepared message from the coordinator. There is a limitation in our implementation as the balances were updated on accounts after the commit the commit message, so we might need to implement a majority commit method to update the final status/



To present the above two scenarios, we have implemented a timeout in coordinator and sleep in participants prepare and commit message response.

A screenshot of a computer program

Description automatically generated



iii. (6935-only) Node-1 crashed after sending out the request and potential solutions  
to recovery from the crash.

We implemented to track the previous states from the log, as in this case coordinator restart after the prepare fails, so it get the values prepare phase asTrue and based on that it start the commit message that’s why we have updated values in account

A screenshot of a computer program

Description automatically generated

We recover the coordinator before timeout, so it was ready to commit the transaction based on previous state

A screenshot of a computer program

Description automatically generated

In this scenario, the coordinator recovers only after all messages were unable to connect to coordinator, so only phase it get was previously committed transaction which it has committed already

6935 Required and 5597 Bonus): Each of Node-2 and Node-3 have two replicas. Therefore, the  
cluster will have 1 coordinator, 3 nodes for account A, and another 3 nodes for account B. For  
accounts A and B, we use Paxos OR Raft to achieve consensus on the values of A and B.  
a. It basically integrates your Lab-2 into this system

For this part of the lab, we have created a cluster for both account A and B. We have only updated our config\_file to get the correct ips for the cluster.

To run this code:

Update the config\_file:

Run the following command for each node in cluster A and B on respective ips

python3 node\_raft.py clusterA node1 (run for node2 and node3 also)

python3 node\_raft.py clusterB node3 (run for node4 and node5 also)

python3 node\_coordinator.py

python3 node\_participant\_raft.py node7 (Account A)

python3 node\_participant\_raft.py node7 (Account B)

A screenshot of a computer program

Description automatically generated

Code:

Raft code and the coordinator code, remain almost the same with minor changes. The major changes were introduced in participant code, as it need to find the leader in respective cluster and then submitted the values on the cluster through the leader. Find leader, submit values via leader function are similar to raft, the main part in to get the current balance of the account and then pass it to the leader write function and once its return true, it will start the transaction based on the current value and balance of the account. Once the participants have the updated balance, it will again submit to the leader to propagate in the cluster based on raft consensus.

For the simplicity and time constraints, we have only implemented it for one transaction at this point and will update for overall scenarios in github.

A screenshot of a computer program

Description automatically generated

Raft cluster up and running for each cluster

A screenshot of a computer

Description automatically generated

Values distributed on respective clusters from the account and transaction performed through coordinator

A screenshot of a computer

Description automatically generated