Distributed Systems
Name: Shashank Ramgude
BUMail: sramgud1@binghamton.edu
Project 4

(a) The structure of code:

- **Handler.java** is the handler for both Server(Participant) as well as Coordinator.
- **Coordinator.java** plays the role of coordinator.
- **Client.java** runs clients and connects to the coordinator and then coordinator connects to all the servers.
- **Server.java** is the participant.

(b) The RPC interface participants expose to the coordinator:

- canCommitWriteFile(transaction)

  It gets vote from participant about committing or aborting for writing the file.

- canCommitDeleteFile(transaction)

  It gets vote from participant about committing or aborting for deleting the file.

- participantReadFile(transaction)

  It reads the contents of the specified file from any 1 participant.

- doCommitOrAbortWriteFile(transaction)

  If vote form participant is Yes, it will ask for commit(write file) else abort.

- doCommitOrAbortDeleteFile(transaction)

  If vote form participant is Yes, it will ask for commit(delete file) else abort.

(c) If failures occur, it is reported to client by the return types of the RPC function which client has called, i.e. readFile(fileName), writeFile(rFIle) and deleteFile(fileName).

When participant or coordinator abruptly shuts down, after restart it reads the logs stored during their operation, and if it finds any unhealthy transaction then it processes it as desired.

(d) Test cases:

Test Case 1: Read(RFile)
Returns the content if file found, else returns “File not found message”.
Transaction is logged.

Test Case 2: Write(file, contents)
If commit is successful it returns “Commit successful” else “Commit successful”

Test Case 3: Delete(file)
If commit is successful it returns “Commit successful” else “Commit successful”

Test Case 4: When any participant is down
The transaction aborts and return to the client. Here “java.net.SocketTimeoutException” is

caught.
Transaction is logged.
When it is recovered, it scans for any unfinished transactions and if found completes it and logs th
transaction in log file.

Test Case 5: Write or delete and then read on same file.
First writes the file and then reads the contents.

Test Case 6: Write or delete and then read on different file.
Writes or deletes the file and at the same time reads the contents of the other file.

Test Case 7: Coordinator down after sending do commit request to client or receiving votes.
After restart coordinator recovers unprocessed data from log file and completes them.

Test Case 8: Server down after receiving can commit message
After restart coordinator recovers unprocessed data from log file and completes them.

Test Case 9 : Server down after receiving can commit message and before or after sending reply
After restart coordinator recovers unprocessed data from log file and completes them.

Test Case 10 : Server down after receiving commit or abort message and before or after actually committing
After restart coordinator recovers unprocessed data from log file and completes them.