CS 578
Programming Assignment - 4.2

Mauli Mehulkumar Patel

Student ID: 35231636

Design:

This is the design for the creation of a fault-tolerant system using zookeeper for the coordination and cassandra for the storage.
It attempts to implement checkpointing, fault tolerance, and recovery after failure.

MyDBFaultTolerantServerZK.java :

Data outline:

- Client requests in /requests
- Server registers it address in /servers

Architecture:

- Server: It is the key-value store. With zookeeper, it manages the clients. The leader will take in all the writes and connect to others, followers.
- Client: Sends the operations like GET etc.
- Zookeeper: It is the coordination service. It will keep a track of the current leader.

Leader Election used:

- Each server creates a node under servers which is created in the folder.
- Once numbers are given, the server with the lowest number becomes the leader.
- If a leader has failed, then the succeeding server becomes the leader after everyone is notified.

In case of Failure:

- If the server fails, then the session will expire and the leader is removed.
- As the other servers come to know, a new leader is announced.
- Clients then connect to the new leader.

Concurrency:

- All servers accept clients paralleling like threading.
- A leader change will call for some reconfiguration as only the leader can perform writes.

Failing Tests:

- Total Tests:
    - test31_GracefulExecutionSingleRequest
    - test32_GracefulExecutionMultipleRequestsSingleServer
    - test33_GracefulExecutionMultipleRequestsToMultipleServers
    - test34_SingleServerCrash
    - test35_TwoServerCrash
    - test36_OneServerRecoveryMultipleRequests
    - test37_TwoServerRecoveryMultipleRequests
    - test38_EntireStateMatchCheck
    - test39_InstantaneousMassacreAndRevivalTest
    - test40_SerialKillAndRecover
    - test41_CheckpointRecoveryTest

- From 31 to 33, have been seen to pass while remaining fail.
- Test 35: ReplayPendingRequests() does not apply every missed request to Cassandra; it only updates lastAppliedZnode to the most recent child name. The two crashed servers jump their checkpoint to the most recent node when they replay after restarting, avoiding the need to reconstruct intermediate operations. Their local Cassandra state may therefore be different from the state of the surviving server.
    - Solution: Modify replayPendingRequests() to read all znodes to lastAppliedZnode.
- Test 36: Other servers keep adding nodes under /requests while the server is unavailable. My server only updates its checkpoint to the most recent

node name upon restart; it does not correctly reapply all missing operations.

    - Solution: Implement full replay from lastAppliedZnode forward.

- Test 37: With the current “checkpoint only to latest node name” behavior, both recovering servers will have incomplete state.
    - Solution: proper ordered replay, using alphabetical sorted names.
- Test 38: Different servers may apply different subsets of operations because my implementation does not actually replay every operation.
    - Solution: Same as above
- Test 39: Some servers forget operations that were logged but not fully applied because I don't properly restore state by replaying all log entries after the checkpoint.
- Test 40: Some servers skip operations that took place while they were unavailable due to the same missing replay logic.
- Test 41: My current use of the checkpoint table is limited.