Team Members:

Neel Yadav (34064006)
Aneri Rana (33157215)

Design approach for Zookeeper:

In the init method, we create a new zookeeper session and cassandra cluster. To use zookeeper's coordination services for our fault tolerant replicated database we use a combination of sequential nodes and watcher in the following manner:

1. In handleMessageFromClient(), any server that receives a new request from the client will add a new znode of persistent sequential type with the request value under '/requests' znode.
2. Zookeeper coordinates the auto-increment of request numbers for us as multiple servers try to add their requests.
3. Every server registers a watcher on children of '/requests' during initialization, so that each server is notified when any server receives and adds a request.
4. The watcher triggers rollForward() method, which takes the last executed request number for the server and then executes all the remaining requests in the /requests starting from the last executed one (this is done so that if any request triggers were missed they will still be executed in a sequential manner).

Checkpoint and recovery
We decided to select a checkpoint threshold of 50 requests to find a balance between decent frequency of backups while not creating too much backup overhead. When the last executed request number reaches a multiple of 50, the checkpoint() method is called. The checkpoint() method triggers the creation of a snapshot by calling takeSnapshot. The takeSnapshot method creates a snapshot of the Cassandra keyspace by fetching all rows from the table and serializing the data to a .ser file.

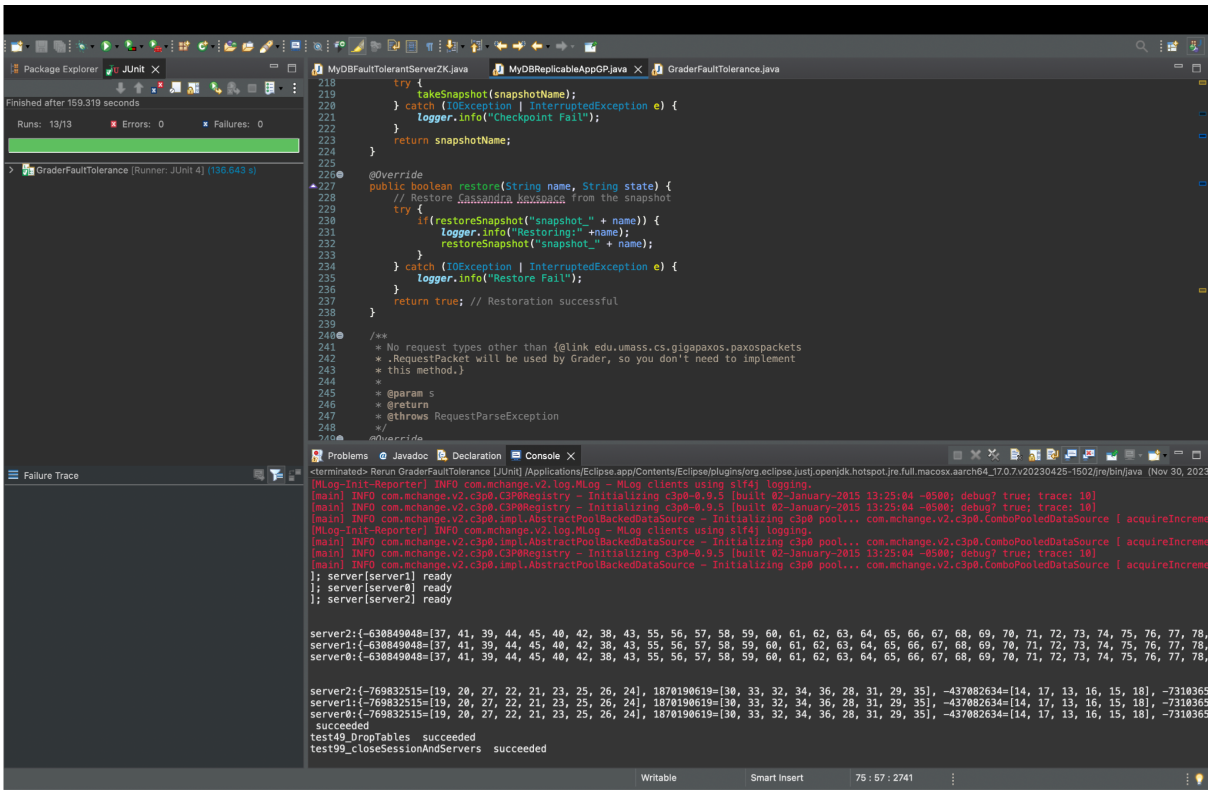
We do crash recovery during initialization. If a checkpoint exists for a server, then we call the restoreSnapshot() and rollForward() methods to bring the server back upto speed if it is recovering from a crash. The restoreSnapshot method reads the latest snapshot file and restores the data back into the keyspace table by creating insert queries for each id. If there is no checkpoint stored but there are pending requests logged, then we execute all the requests from /requests to get the server upto speed.

Additionally, we also store the last executed request number in the snapshot filename with format "checkpoint_[reqNum].ser". This is used during recovery so that any pending requests are executed after the last executed request number only.

Trim Logs
Everytime we have reached the checkpoint threshold of 50 requests, we call trimLogs() after taking the snapshot. If the log capacity has reached 400 then the oldest 50 requests would be pruned to make sure the logs don't grow infinitely long.

Design approach for Gigapaxos:

We extend the replicable API of gigapaxos and implement the three functions, execute, checkpoint and restore to successfully implement a consistent cassandra database. We were able to pass all the test cases.



```
Finished after 159.319 seconds
Runs:  13/13     Errors:  0     Failures:  0
> GraderFaultTolerance [Runner: JUnit 4] (136.643 s)
```

```java
        try {
            takeSnapshot(snapshotName);
        } catch (IOException | InterruptedException e) {
            logger.info("Checkpoint Fail");
        }
        return snapshotName;
    }

    @Override
    public boolean restore(String name, String state) {
        // Restore Cassandra keyspace from the snapshot
        try {
            if(restoreSnapshot("snapshot_" + name)) {
                logger.info("Restoring:" +name);
                restoreSnapshot("snapshot_" + name);
            }
        } catch (IOException | InterruptedException e) {
            logger.info("Restore Fail");
        }
        return true; // Restoration successful
    }

    /**
     * No request types other than {@link edu.umass.cs.gigapaxos.paxospackets
     * .RequestPacket will be used by Grader, so you don't need to implement
     * this method.}
     *
     * @param s
     * @return
     * @throws RequestParseException
     */
    @Override
```

```
<terminated> Rerun GraderFaultTolerance [JUnit] /Applications/Eclipse.app/Contents/Eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.aarch64_17.0.7.v20230425-1502/jre/bin/java  (Nov 30, 2023
[MLog-Init-Reporter] INFO com.mchange.v2.log.MLog - MLog clients using slf4j logging.
[main] INFO com.mchange.v2.c3p0.C3P0Registry - Initializing c3p0-0.9.5 [built 02-January-2015 13:25:04 -0500; debug? true; trace: 10]
[main] INFO com.mchange.v2.c3p0.C3P0Registry - Initializing c3p0-0.9.5 [built 02-January-2015 13:25:04 -0500; debug? true; trace: 10]
[main] INFO com.mchange.v2.c3p0.impl.AbstractPoolBackedDataSource - Initializing c3p0 pool... com.mchange.v2.c3p0.ComboPooledDataSource [ acquireIncreme
[MLog-Init-Reporter] INFO com.mchange.v2.log.MLog - MLog clients using slf4j logging.
[main] INFO com.mchange.v2.c3p0.impl.AbstractPoolBackedDataSource - Initializing c3p0 pool... com.mchange.v2.c3p0.ComboPooledDataSource [ acquireIncreme
[main] INFO com.mchange.v2.c3p0.C3P0Registry - Initializing c3p0-0.9.5 [built 02-January-2015 13:25:04 -0500; debug? true; trace: 10]
[main] INFO com.mchange.v2.c3p0.impl.AbstractPoolBackedDataSource - Initializing c3p0 pool... com.mchange.v2.c3p0.ComboPooledDataSource [ acquireIncreme
]; server[server1] ready
]; server[server0] ready
]; server[server2] ready


server2:{-630849048=[37, 41, 39, 44, 45, 40, 42, 38, 43, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78,
server1:{-630849048=[37, 41, 39, 44, 45, 40, 42, 38, 43, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78,
server0:{-630849048=[37, 41, 39, 44, 45, 40, 42, 38, 43, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78,


server2:{-769832515=[19, 20, 27, 22, 21, 23, 25, 26, 24], 1870190619=[30, 33, 32, 34, 36, 28, 31, 29, 35], -437082634=[14, 17, 13, 16, 15, 18], -7310365
server1:{-769832515=[19, 20, 27, 22, 21, 23, 25, 26, 24], 1870190619=[30, 33, 32, 34, 36, 28, 31, 29, 35], -437082634=[14, 17, 13, 16, 15, 18], -7310365
server0:{-769832515=[19, 20, 27, 22, 21, 23, 25, 26, 24], 1870190619=[30, 33, 32, 34, 36, 28, 31, 29, 35], -437082634=[14, 17, 13, 16, 15, 18], -7310365
 succeeded
test49_DropTables  succeeded
test99_closeSessionAndServers  succeeded
```

The execute method handles the execution of incoming requests. It extracts the CQL query from the request and executes it using the Cassandra Session. The takeSnapshot method creates a snapshot of the Cassandra keyspace by taking all rows from the table in the keyspace and serializing the data to a ser file. The restoreSnapshot method reads a snapshot file and restores the data back into the table in the table keyspace by creating insert queries for each id. The checkpoint method triggers the creation of a snapshot by calling takeSnapshot and returns the snapshot name. The restore method restores the keyspace from a given snapshot name using restoreSnapshot.