COMPSCI 578 - PA4.3

Raft Implementation

This document explains the design and current implementation of MyDBFaultTolerantServerZK, implementing a fault-tolerant replicated database server using Apache Raft (through Apache Ratis) together with Apache Cassandra. The system ensures that all client commands are executed in the same order on all replicas and remain consistent even when servers crash.

Design Overview:

Raft Group Setup

Each server is configured as a Raft peer using information from NodeConfig. All servers share the same Raft group ID so that they form a single consensus group. Communication between Raft peers uses gRPC, with ports derived from the server configuration.
Raft is configured with reasonable timeout values and automatic snapshot triggering. A maximum log size is enforced to prevent the Raft log from growing without bound.

Handling Client Requests

When a server receives a request from a client, the following steps occur:

- The request is converted into a string command.
- The command is wrapped as a Raft Message.
- The message is sent to the Raft leader using a RaftClient.
- If the request fails due to a temporary issue, it is retried a limited number of times.

This design ensures that all commands go through Raft and are totally ordered before execution.

Checkpointing:

The state machine implements the takeSnapshot() method by returning the last applied index. This allows Raft to know when it can truncate its log. However, the actual Cassandra data is not saved as part of the snapshot.

Testing Results:

Number of test cases passed : 3 test cases have passed.

- test31_GracefulExecutionSingleRequest
- test32_GracefulExecutionMultipleRequestsSingleServer
- test33_GracefulExecutionMultipleRequestsToMultipleServers

Failed Test Cases:

The crash, recovery, state consistency, and checkpointing tests fail mainly because the system does not fully handle what happens when servers go down and come back up. When a server crashes, it loses its database state, and when it restarts, there is no clear way for it to catch up with what it missed while it was offline. The server may start with an empty or outdated database, and it does not properly synchronize with the other running servers. Because of this, different servers can end up with different data, which causes the state consistency tests to fail. Even though snapshots are enabled, they do not actually save the database contents, so after a full crash and restart, the system may not restore the correct state. To fix these problems, the system would need a better way to save state and recover it so that servers can rejoin and match the rest of the system after failures.