# Name: Vasantha Pallavi Narla

# Student ID:34405596


# PA 3.2

## Design Document: Fault-Tolerant Database Server with Zookeeper

### 1. Introduction

This design document outlines the architecture and implementation of a fault-tolerant database server utilizing Zookeeper for consensus protocols. The server is designed to handle client requests, maintain fault tolerance through consensus, and utilize Cassandra as the backend datastore.

### 2. System Overview

#### 2.1 Components

i). MyDBFaultTolerantServerZK:

- Represents the main server class implementing fault-tolerant features.
- Extends 'MyDBSingleServer' and uses Zookeeper for consensus.
- Manages communication between servers using 'MessageNIOTransport'.

ii). Zookeeper: - Provides the consensus mechanism for leader election and coordination.

iii). Cassandra: - Serves as the backend datastore for storing and retrieving data.

#### 2.2 Consensus Protocol

The consensus protocol relies on Zookeeper for leader election and coordination of client requests. The elected leader is responsible for ordering and broadcasting requests to ensure consistency.

### 3. Key Features

### 3.1 Fault Tolerance

- The system handles faults and failures through leader election and acknowledgment-based message delivery.

- Checkpointing and recovery mechanisms ensure state persistence and recovery in the event of crashes.

### 3.2 Communication

- Utilizes ‘MessageNIOTransport‘ for efficient and asynchronous communication between servers.

- Implements a message queue to track and manage the state of messages.

### 3.3 Request Processing

- Client requests are forwarded to the leader as proposals.

- Leader broadcasts proposals to all nodes for consensus.

- Acknowledgments are collected, and the leader processes them to determine when to send the next request.

## 4. Class Structure

### 4.1 MyDBFaultTolerantServerZK

#### 4.1.1 Attributes

- ‘session‘: Cassandra session for database interaction.
- ‘cluster‘: Cassandra cluster connection.

- ‘myID‘: Unique identifier for the server.

- ‘serverMessenger‘: Message transport for server-server communication.

- ‘leader‘: Identifier for the elected leader.

- ‘queue‘: ConcurrentHashMap for tracking messages.

- ‘notAcked‘: List to track acknowledgments.

- ‘reqnum‘ and ‘expected‘: Sequencers for request numbering.

- Constants for sleep duration, table dropping, and maximum log size.

#### 4.1.2 Methods

- ‘handleMessageFromClient(byte[] bytes, NIOHeader header)‘: Processes client messages, forwards to the leader, and sends responses back to clients.

- ‘handleMessageFromServer(byte[] bytes, NIOHeader header)‘: Processes messages received from other servers.

- Checkpointing and recovery methods (‘checkpoint()‘, ‘recoverFromCheckpoint()‘).

- Consensus-related methods (‘broadcastRequest()‘, ‘enqueue()‘, 'dequeue()‘).

## 5. Interaction Flow

i). Client Request: - Client sends a request to the server. - The server processes the request and forwardst to the leader.

ii). Leader Election: - Zookeeper handles leader election among the s ervers.

iii). Consensus: - The leader broadcasts the request to all nodes for consensus. Nodes send acknowledgments back to the leader.

iv). Request Execution:

- The leader processes acknowledgments and executes the request.

- An acknowledgment is sent back to the client.

## 6. Conclusion

This design provides a robust fault-tolerant database server that ensures consistency through consensus protocols implemented with Zookeeper. The system utilizes asynchronous communication, maintains state persistence, and handles failures gracefully. The interaction flow ensures that client requests are processed reliably and consistently across all nodes.

**Tests Passed:**

- Passed all tests except test34
- Explanation: The test fails with message AssertionError. Below is the screeshot of the error.

```
server2:{-1959008799=[]}
server0:{-1959008799=[]}
 FAILED!!!!!!!!!!!! java.lang.AssertionError: nonEmpty=false
java.lang.AssertionError: nonEmpty=false
        at org.junit.Assert.fail(Assert.java:88)
        at org.junit.Assert.assertTrue(Assert.java:41)
        at GraderCommonSetup.verifyOrderConsistent(GraderCommonSetup.java:315)
        at GraderCommonSetup.verifyOrderConsistent(GraderCommonSetup.java:228)
        at GraderFaultTolerance.test34_SingleServerCrash(GraderFaultTolerance.java:268)
        at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:77)
        at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.base/java.lang.reflect.Method.invoke(Method.java:568)
        at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:50)
        at org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:12)
        at org.junit.runners.model.FrameworkMethod.invokeExplosively(FrameworkMethod.java:47)
        at org.junit.internal.runners.statements.InvokeMethod.evaluate(InvokeMethod.java:17)
        at org.junit.internal.runners.statements.RunBefores.evaluate(RunBefores.java:26)
        at org.junit.rules.TestWatcher$1.evaluate(TestWatcher.java:55)
        at org.junit.rules.TestWatcher$1.evaluate(TestWatcher.java:55)
        at org.junit.rules.RunRules.evaluate(RunRules.java:20)
        at org.junit.runners.ParentRunner.runLeaf(ParentRunner.java:325)
        at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:78)
        at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:57)
        at org.junit.runners.ParentRunner$3.run(ParentRunner.java:290)
        at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:71)
        at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:288)
        at org.junit.runners.ParentRunner.access$000(ParentRunner.java:58)
        at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:268)
        at org.junit.internal.runners.statements.RunBefores.evaluate(RunBefores.java:26)
        at org.junit.internal.runners.statements.RunAfters.evaluate(RunAfters.java:27)
        at org.junit.runners.ParentRunner.run(ParentRunner.java:363)
        at org.eclipse.jdt.internal.junit4.runner.JUnit4TestReference.run(JUnit4TestReference.java:93)
        at org.eclipse.jdt.internal.junit.runner.TestExecution.run(TestExecution.java:40)
        at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:529)
        at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:756)
        at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.run(RemoteTestRunner.java:452)
        at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.main(RemoteTestRunner.java:210)
```

- The test is failing as the nonempty variable is false, which is causing the assertion to fail. The condition **(possiblyEmpty || nonEmpty) && match** is not evaluating to true as expected.
- After debugging the code, I found that both **possiblyEmpty and nonEmpty** are false.
- Further testing has revealed **server1** has an empty list of values (**{}**), while **server0** is also empty. The failure is due to the assertion **nonEmpty=false**, indicating that the expected non-empty condition is not met.
- Correct implementation of this part would ensure the condition to be true and pass the test successfully.