# Fault-Tolerant-DB

Team: Snigdha Srivatsava, Rashmi Vagha

## **Replicated State Machine- RSM** (GigaPaxos) Approach

Implementation for fault-tolerant server functionality using GigaPaxos/RSM, which is a coordination service.

Here's a breakdown of the steps and code components:

**Step 1: Establish Cassandra connection**

MyDBReplicableAppGP Class: We use the hostname, post number and the keyspace name to establish the connection with the Cassandra DB.

**Step 2: Checkpointing**

MyDBReplicableAppGP Class: The function checkpoint is called automatically at certain intervals to ensure that the state of the servers is saved. In our implementation, we have added a function called saveSnapshot() where we create a file backup.csv to store all the data from the keyspace and tables.

**Step 3: Restoring**

MyDBReplicableAppGP Class: The function restore is called automatically whenever a particular server crashes and is up again. In our implementation, we have added a function called recoverSnapshot() where we use the backup file previously created to restore the state of that server before it had crashed.

Number of test cases passing: All the test cases passed for GigaPaxos.


- **Coordination server** (Zookeeper) Approach

Implementation for fault-tolerant server functionality using Apache ZooKeeper, which is a coordination service.

Here's a breakdown of the steps and code components:

**Step 1: Zookeeper Connection set up**

MyDBFaultTolerantServerZK Class: This class initializes a connection to ZooKeeper upon object creation.

connectToZookeeper() Function: This private function establishes the connection to ZooKeeper. It's responsible for managing connection events and creating necessary znodes (ZooKeeper nodes).

**Step 2: Coordination Logic Implementation**

registerInZookeeper() Function: This function is responsible for registering the server in ZooKeeper by creating a short-lived znode for this server. The server's status and presence are maintained through this znode.

**Step 3: Zookeeper for Consensus and Fault Tolerance Utilization**

process() Method: This method, required by the Watcher interface, handles events generated by ZooKeeper. It distinguishes between connection-related events and znode-related events.

handleConnectionEvent() and handleZnodeEvent() Methods: These methods handle different types of events. handleConnectionEvent() manages connection-related events, while handleZnodeEvent() manages znode-related events like creation, deletion, data changes, etc. They are crucial for maintaining consensus and fault tolerance mechanisms.

**Step 4: Datastore Logic Integration**

connectToDatastore() Function: This function connects to a backend datastore (e.g., Cassandra) using the Cassandra Java driver. It sets up the necessary keyspace and tables if they don't exist.

start() Method: This method initiates the server's functionalities. It connects to the datastore, registers the server in ZooKeeper, and starts the server logic to handle incoming requests. The server logic might involve setting up communication channels, handling requests, and maintaining server state.

The code combines ZooKeeper for coordination, fault tolerance, and consensus, along with integration with a backend datastore. It's structured to handle different aspects of the distributed system, ensuring fault tolerance by utilizing ZooKeeper for coordination among multiple servers and managing events related to both connections and znodes. Additionally, it integrates with a backend datastore for data storage and retrieval.