Programming Assignment 3 Design Documentation

Samika Kashyap : 33151952
Kinnri Sinha : 32600488

December 4, 2021

1 Introduction

1.1 Purpose

To build a distributed data storage system with consistency and fault tolerance requirements.

1.2 GigaPaxos Overview

GigaPaxos is a group-scalable replicated state machine (RSM) system, i.e., it allows applications to easily create and manage a very large number of separate RSMs. Clients can associate each service with a separate RSM of its own from a subset of a pool of server machines. Thus, different services may be replicated on different sets of machines in accordance with their fault tolerance or performance requirements. The underlying consensus protocol for each RSM is Paxos, however it is carefully engineered so as to be extremely lightweight and fast. For example, each RSM uses only a few hundred bytes of memory when it is idle (i.e., not actively processing requests), so commodity machines can participate in millions of different RSMs. When actively processing requests, the message overhead per request is similar to Paxos, but automatic batching of requests and Paxos messages significantly improves the throughput by reducing message overhead, especially when the number of different RSM groups is small. The lightweight API for creating and interacting with different RSMs allows applications to "carelessly" create consensus groups on the fly for even small shared objects, e.g. a simple counter or a lightweight stateful servlet. GigaPaxos also has extensive support for reconfiguration, i.e., the membership of different RSMs can be programmatically changed by application developers by writing their own reconfiguration policies.

1.3 High Level Design

We have implemented the class MyDBReplicableAppGP to implement a client-server model that connects to Cassandra and executes queries based on the request being sent by the grader.

MyDBReplicableAppGP Design: We create a cluster with the contact point as the broadcast_address from system.local. In our case this was 127.0.0.1. On receiving a client request, the server creates a session to connect to the keyspace 'demo' and begins executing the commands in the request from the client.

gigapaxos.properties: We specify the application we want to run on each of the active servers in this file using:

APPLICATION:server.faulttolerance.MyDBReplicableAppGP

The server host and port is defined using the following lines:
active.server0=127.0.0.1:1900
active.server1=127.0.0.1:1901
active.server2=127.0.0.1:1902

Functions implemented:

execute(): The execute function casts the received request into RequestPacket() and extracts the query to be executed on the database. The query is executed using session.execute() in this method.

checkpoint(): We use this method to checkpoint() the database after the max requests have been logged. We have implemented a method to select all values from the database and store it in the return value using delimiters ('—' for different rows and ':' for different values in a given row). We append these values to a string with the delimiters to make it easy while restoring the database.

restore(): In this method, we first create the table again if it doesn't already exist. We then use the previous checkpoint value and the delimiters we specified during execution of checkpoint to create insert and update queries to restore the database. The insert query is used to insert an empty list into the given key value. The update query is used to iteratively add the list values to the events list corresponding to the given key.

1.4 Challenges faced

We were unable to pass the test cases on our local system. We have however included our code and logic in the design document as well as some commented code in the MyDBReplicableAppGP.java file which indicates the approach we attempted but could not succeed with. Following are the screenshots of the failing tests:

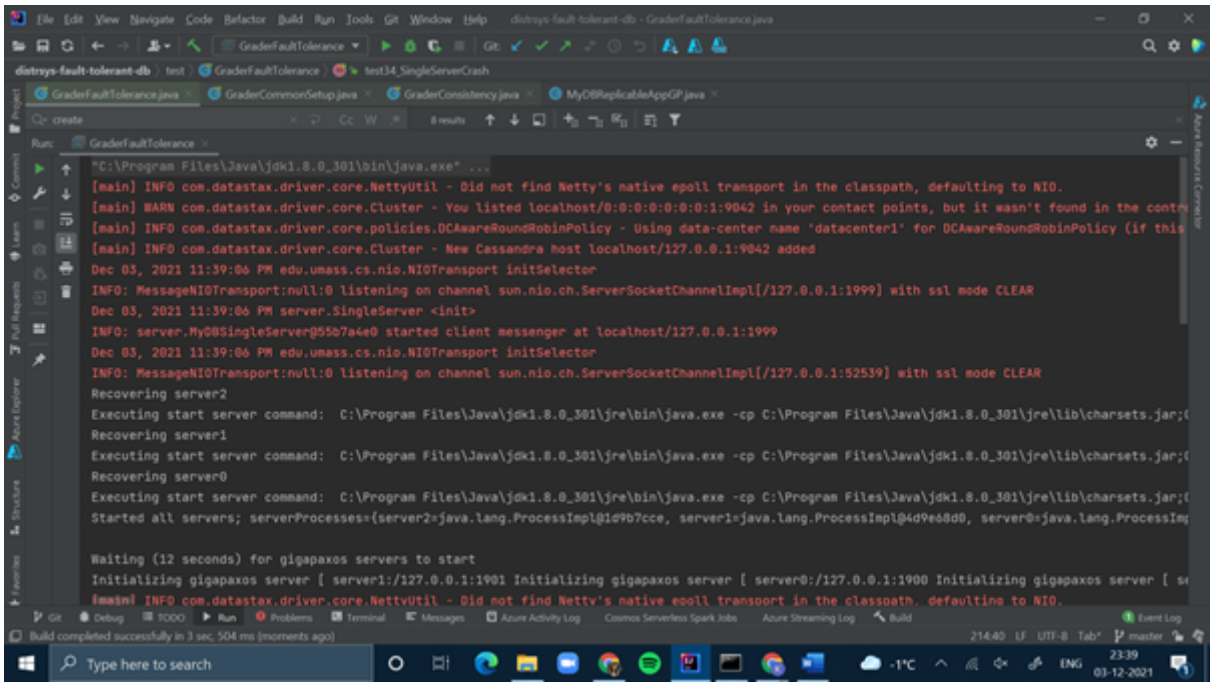

Figure 1: Picture 1

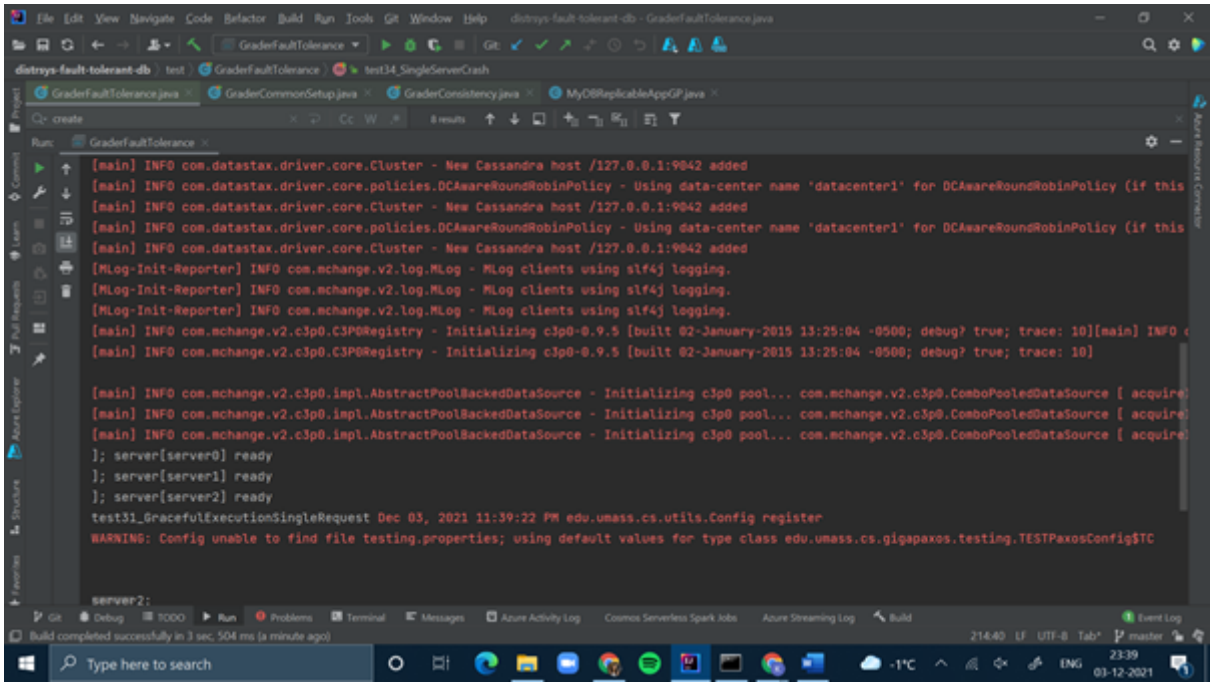

Figure 2: Picture 2

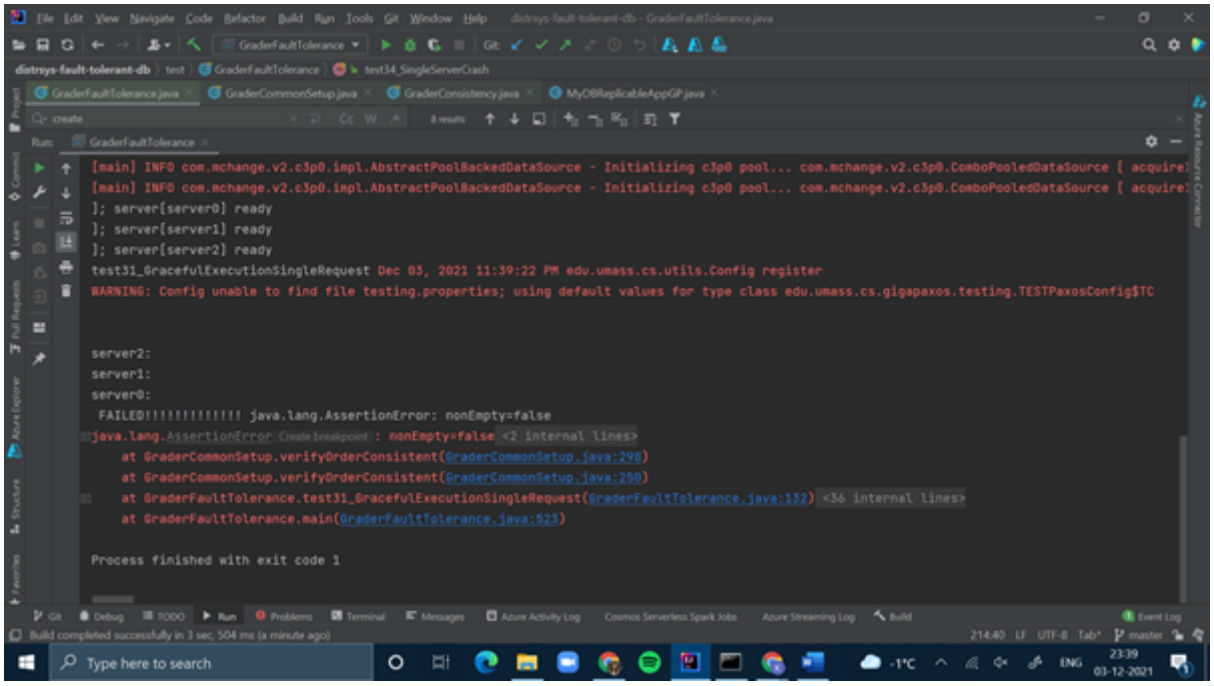

Figure 3: Picture 3