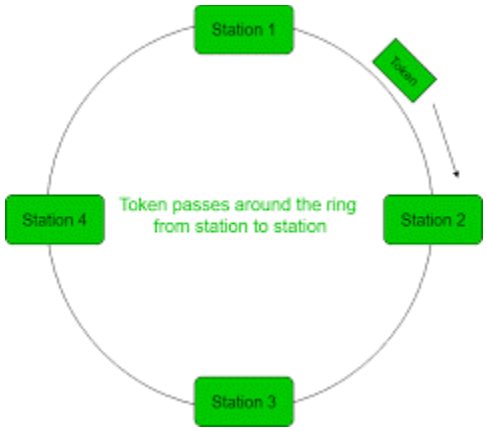
Assignment 3- Report

- Mettukuru, Avinash Reddy

Overview

This assignment is an implementation of Distributed Sequencer. It runs several sequencers whose output ID's are monotonically increasing.

Design



- A controller starts all the sequencers to online, while setting up the sequencer connect to all the other neighboring nodes(sequencers).
- Each sequencer starts by making the ring formation, and the initial sequencer passes the initial token and thus starting the token passing process.
- There is a separate thread in the background for each sequencer to listen for an incoming connection.
- A client can connect to any of the available sequencers to get a monotonically increasing ID.
- Whenever a client makes a TCP connection to the sequencer it spawns a separate thread and enqueue' s all the "get-id" requests from that client, utilized locking techniques on critical resources i.e., "requestQueue" to ensure consistency.
- When a sequencer receives a token, it awaits for the lock on critical resource i.e. "requestQueue" where it responses for all the awaiting clients.
- While passing the queue the sequencer places the latest value of ID and passes the token, thus consistency is maintained.
- To make the testing I have generated ClientDriver.java to make the testing easy with multiple clients(as a separate thread), as to check the correctness of the algorithm.

Implementation

TCP Socket Server

Each sequencer maintains two sockets one to send token to next sequencer and one accepts receiving TCP connections from clients, this particular socket is also used to connect from previous sequencer in the ring topology at the initialization phase.

Assignment 3- Report

- Mettukuru, Avinash Reddy

Concurrency

I have used *Java ExecutorService* to handle all the multi-threading which allocates a fixed pool of threads and is very handy to safely close all the thread at once when a kill signal is issued other than *-9* in Unix, similar signal in windows.

Build & Run

Project can be built and tested on Luddy servers as they have java compiler and JVM preinstalled. No additional libraries are required.

Makefile

$make compileSequencer: this command compiles the Sequencer java files into a class file for the JVM.
$make runSequencer: Starts the sequencers with the number of sequencers to start as a CLI argument.
$make compileClient: this command compiles the Client java files into a class file for the JVM.
$make runClient: Starts the client handler with the number of clients, and ports of sequencers as a CLI argument.
$make compileClientTemplate: this command compiles the Client Template java files into a class file for the JVM.
$make runClientTemplate: Starts the client template i.e., only a single client when the ports of sequencers as a CLI argument are provided.
$make clean: cleans all the binaries and flushes the data (key, value) on files.

Code Structure

- ./src
  - ./sequencer
    - ./Sequencer.java
    - ./Reqeust.java
    - ./ClientHandler.java
  - ./client
    - ./ClientDriver.java
    - ./ClientTemplate.java
    - ./StandaloneClient.java
  - ./Controller.java
  - ./Makefile

How to run

- Be sure to in the “src” directory.
- To start the sequencers, please follow the following steps:
  - Enter the following commands in the given order *$make compileSequencer* and then *$make runSequencer args=”3”* this will start 3 sequencers. When all the sequencers are up and running they will print out their IP address and port numbers as well.
- To start the clients, please follow the following steps:
  - Enter the following commands in the given order *$make compileClient* and then *$make runClient args=”2 port1 port2 port3….”* this will start 2 clients and in the place of port1,

Assignment 3- Report

- Mettukuru, Avinash Reddy

port2 etc., enter the port numbers of all the available sequencers. You could replace *compileClient, runClient* with *compileClientTemplate, runClientTemplate* and *args="port1 port2"* only if you want to choose which sequencer you want to connect to.

Tests

- To make the testing easier with multiple clients I have created a ClientDriver.java, a program that accepts number of clients and port numbers of sequencers.
- The client driver then spawns a thread for each client and randomly sends request to any of the sequencer.
- When a client receives its ID, it logs into the console along with Client ID, the sequence it contacted and a global counter which counts the order in which a client received its response.
- I have also written a "ClientTemplate.java" program which only creates a single client but the user can select which sequencer they would like to request for.

Following are the tests that are performed on the program.

Basic Tests

- Basic test case is in which there are multiple sequencers and multiple clients requesting an ID.
- After observing the console output, we can say that the ID are monotonically increasing and unique.

Edge Cases

- When only one sequencer exits, how does the ring topology hold.
- When one sequencer exits it passes the token between itself and processes all the request.

Performance

- Since a sequencer has to wait around for the token, only one sequencer will be active any point of time, this could be a huge down side for our algorithm.

Limitations

- Whenever a sequencer received a token, it will not pass it to the next sequencer until it has processed all the request in its queue, note that this queue contains all the requests from all the clients this sequencer has connected to, so when the incoming requests are more than the outgoing responses all the other sequencers might have to wait.
- As we know any client can issue a request for an Id, this could be resource heavy when considering the fact that we are spawning a new thread for each TCP connection.
- This system of network inherits the cons of Ring topology as well. When a sequencer has crashed then there is no token passing in the system. Thus entire system has failed.

Improvements

- To conquer the first limitation we can implement a separate timer hook in each sequencer to force pass the token the next sequencer without processing all the requests in the queue.
- We could use a message broker such as JSM, Apache Kafka, RabbitMQ etc., in our system design. Instead of spawning a new thread for each TCP connection now we can spawn a single thread which receives all the requests from the message broker.

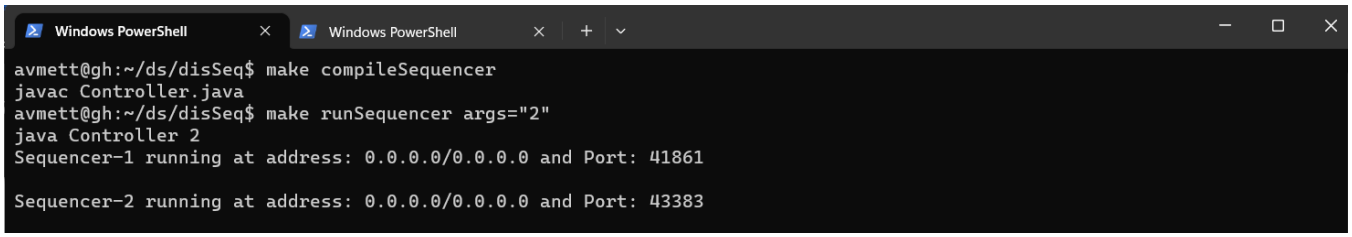
Assignment 3- Report

- Mettukuru, Avinash Reddy

- Fault tolerance should be possible, such as using *ExecutorService* handy feature we can see how many threads(Sequencers) are active so when a sequencer has failed we have to rearrange the connections in the Ring topology.

Output

Sequencer:



Client: