Project report: Stage 1
COMP3100 Distributed Systems, S2, 2022
SID: 45339201, Name: Victor Chan

1 Introduction

For this Distributed Systems Project, the desired outcome is to create a fully functional Client Server Simulator using Oracle's Virtual-Box [1] running on the Ubuntu [2] System. The goals involving ds-sims will be outlined below, these goals are reflected in Stage 1 of the COMP3100 Project. In addition, all aspects of this project can be accessed via Github[3].

In this report, the aim to create a client-side simulator using a job dispatcher which will connect to the server-side simulator, enabling it to receive jobs and sends all jobs to the largest server type. This assignment of jobs will be performed using Largest-Round-Robin (LRR), scheduling largest jobs first based on the number of cores. If there are more than one job with the same amount of cores detected, whichever job was detected first, would be used.

The report will be formatted in this following order:

The rest of this report is organised as follows. Section 2 to Section 4 gives an overview of:

- **Section 2**: System Overview: Depicting the interaction between the server-side and client-side system, outlining the process on how the jobs are to be ran.
- **Section 3**: Design: Exploring the client-side simulator, additionally the functionalities and constraints within the project.
- **Section 4**: Implementation: Explore how the software libraries, data structures, software and strategies that are applied and implemented to the client-side simulator.

2 System Overview

The functionality of the ds-sim displays the process on how the server-side simulator is utilised to dispatch jobs.

Firstly, the server must already be running before the client can connect, the sockets of the client will be running on the localhost (127.0.0.1) and port 50000. Once this connection is made, the client will initiate its communication with the server. The Handshake Protocol of ds-sim will start by sending the "HELO" greeting message, enabling the server to reply with a response message "OK" as acknowledgement. Following this, the client will send a request for authentication wih the "AUTH[user.name]" message, the server will reply with "OK" and allow the user access to the system. Client will then read through the available data and send the "REDY" message to initiate the process of the server in receiving the jobs.

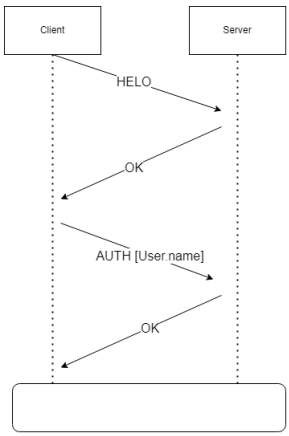

Figure 1: Handshake Protocol WorkFlow

Jobs will be written as Server[Type][State][BootupTime][HourlyTime][Core][Memory][Disk]. An example of a Job format is: 37 0 653 3 700 3800. Through the GETS command, The client will use the ”GETS ALL” or ”GETS Capable [Job Format], enabling the server to send the amount of available jobs to the client. The client will reply with ”OK” as an acknowledge that it has received the job lists from the server. This will continue until the server has sent a ”.” to depict that there are no more available information left to send to the client. The client will now begin the job scheduling using this command: SCHD[JobID]Server.Type][Server.ID], an example is SCHD 0 large 0. Once the job has been scheduled, the server will inform the client. Enabling for a loop until all the available jobs have been read and there are no more available.

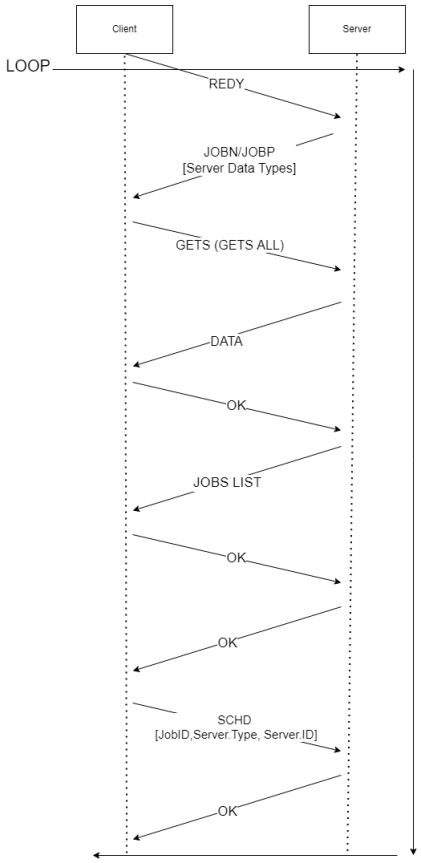

Figure 2: Job Scheduling Loop Workflow

Once all the server files have been read and the ”.” has been detected. No more new data is required to be read by the Client.

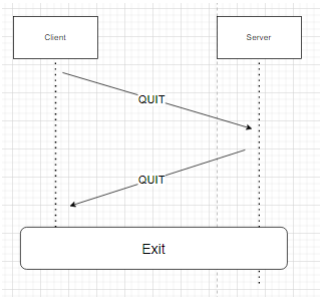

Figure 3: Quit Command Workflow

3 Design

In this section, The client-side simulators and the functions that it uses to connect to the server-side simulator will be described, its use of the handshake protocol and the receiving and scheduling of jobs. This will be ordered accordingly in the client’s order of operation.

3.1 Connection and Communication with Server

The client initally has an empty TCP socket (null), a TCP socket will be created and connected with an IP address of ”localhost” and using port 50000. The purpose of the TCP Socket is to enable communication between the input and output streams of the server-side and client-side simulators. The BufferReader[4] is used to read the data sent from the server with the DataOutputStream doing the contrary, its purpose being to send data by the Client. Lastly, to end the connection to the server if no data is left to be received by the client or server, a conditional statement ”QUIT” be received.

3.2 Handshake Protocol

The Client needs to perform a handshake protocol [5] before it can connect to the Server. First step, the Client will initiate the handshake protocol by written a ”HELO” message, then converting this message into bytes. Once converted and sent to the Server, the Server will respond by sending in bytes that correlate to an ”OK” message.

A similar process will occur in the next step, The client will continue the handshake protocol by writting the ”AUTH[User.name]” message for user authentication purposes, then converting the message to bytes. Once converted and sent, the Server will respond by sending an ”OK message.

The BufferReader and DataOutputStream are the two vital components used to make request to the Server.

3.3 Server Information and Receiving Jobs

The BufferReader and DataOutputStream are used to receive jobs and server information. When the ”REDY” and ”GETS” Command are inputted by the client and sent to the server, once the server receives the bytes that correlate to these commands. The server will send the client’s next available job as well as respective server information. This will be an incremental loop that loops and gathers through the server information, allowing for the BufferReader to read multiple lines that is send by the Server. DataOutputStream however is used to confirm that the data send from the server before the client can initiate the process of job scheduling.

3.4 Job Scheduling

The DataOutputStream will run the SCHD command as well with its respective extracted data elements. A sorting function is implemented to sort the Server with largest amount of cores first, ensuring the correct information is shared. After the jobs are scheduled, client the client will either continue to receive and schedule for the next job or exit the client simulator.

4 Implementation

4.1 Software Libraries

Software libraries are imported for various functions for the client side simulator. ”java.net.*” was imported for the use of the sockets that would be used by the BufferReader and DataOutputStream.

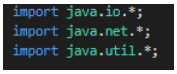

Figure 4: Software Libraries

4.2 Software

Oracle VM VirtualBox running on the Linux Ubuntu Virtual Machine (VM) will be used for this project, for the purpose of depicting a distributing systems simulation (ds-sim). Visual Studio Code[6] will be used within the VM to create, edit and debug the source code within the simulation. Ubuntu’s terminal will be used to compile the client-side simulator, enabling tests to be ran with the server.

4.3 Data Structures

ArrayLists are used to store server data to be extracted for information. It will be extracted from the parser into a list, where it could be used later on if required.

4.4 Implementation of Code

4.4.1 Software Libraries

”Java.util.*” is used to import ArrayLists and Lists that stored the parsed Server Data. ”Java.net.*” is used to create and store all the required classes and other relevant functions required. For further information, refer to Section 4.1.

4.4.2 Connecting to Server

An empty socket will be created initially. To connect to the server, it will initialise to ”localhost” IP address and port 50000. For Client and Server to communicate with one another, BufferReader and DataOutputStream variables have been created to set and initialise with the socket.

```
public class myClient {
    Run | Debug
    public static void main(String[] args) {
        Socket s = null; //Create a new socket
        try {
            // Establish Sockets,
            int port = 50000; //Set Server Port == 50000
            s = new Socket("localhost", port); //Initialise Socket
            BufferedReader in = new BufferedReader(new InputStreamReader(s.getInputStream())); //Data will be read from Server with Input Stream
            DataOutputStream out = new DataOutputStream(s.getOutputStream()); //Output written to Server with Input Stream
```

Figure 5: Server Connection Establishment

4.4.3 HandShake Protocol

The purpose of the ”out.write(([Command].getBytes))” is to send and convert messages into bytes to the Server. ”System.out.println()” is a java command used to print to the console what the server responds with. The ”in.Readline();” purpose is to read that messages being received from the Server. It will initiate the Handshake with the ”HELO” message.

With the authentication of usernames, a username will be stored as a String in the system and inputted during authentication. However, as this server does not currently possess any authentication processes as of now, anything inputted after the ”AUTH” Command would display the same result.

```
// Start of HandShake Protocol
// Hello [HELO]
out.write(("HELO\n").getBytes()); //Client sends "HELO" to Server
System.out.println("Sent: HELO");
String str = in.readLine(); //Gather Response from Server
System.out.println("Received: " + str); //Client receives "HELO" response from Server

// Authenticate [AUTH]
String username = System.getProperty("user.name");
out.write(("AUTH " + username + "\n").getBytes()); //Client sends "AUTH" to Server to authenticate username
System.out.println("Sent: AUTH");
str = in.readLine(); //Gather Response from Server
System.out.println("Received: " + str); //Client receives "AUTH" response from Server
//End of HandShake Protocol
```

Figure 6: HandShake Protocol

4.4.4 ServerList

Client will receive server data from Server, the received information will be converted and placed into a ServerList for later use. An ”OK” message will be sent to the Server to indicate that Server Data has been received successfully. This process will continue until there is no more data available to check. Next, the servers will be sorted in ascending size where the largest server will be determined in the following step, for the purpose of job scheduling.

```
int numOfServers = Integer.parseInt(str.split(" ")[1]);
List<Server> serverList = new LinkedList<>();

for (int i = 0; i < numOfServers; i++) {
    str = in.readLine();
    String[] serverDataType = str.split(" ");
    //Server with its respective Data Type from Server Type to Disk is added to serverList
    serverList.add(new Server(serverDataType[0], serverDataType[1], serverDataType[2], serverDataType[3], serverDataType[4], serverDataType[5], serverDataType[6], serverDataType[7]));
}

out.write(("OK\n").getBytes());
System.out.println("Sent: OK");

str = in.readLine();
System.out.println("Received: " + str);
```

Figure 7: ServerList

4.4.5 Job Scheduler with Largest Round Robin (LRR)

Job Scheduler will compare if Server Type equates to JOBN/JOBP, incrementing through the ServerList. The Detected Jobs will be converted from the Server as Bytes where it will be read in the 'JobSchedule' Format through the Terminal. This process will increment until a "." has been detected, indicating that there are no more available data to be read, causing the process to terminate.

```
// Assign jobs to the largest servers with Job Scheduler (Largest-Round-Robin)
do {
    String[] ServerDataTypes = str.split(" "); //split the str into an array then save as ServerDataTypes Array

    if (ServerDataTypes[0].equals("JOBN") || ServerDataTypes[0].equals("JOBP")) { // Handling of Jobs
        Server currentServer = largestServers.get(serverIndex);
        String JobID = ServerDataTypes[2];
        String JobSchedule = "SCHD " + JobID + " " + currentServer.Type + " " + currentServer.ID + "\n"; //Sceduling a Job

        out.write(JobSchedule.getBytes());
        System.out.print("Sent: " + JobSchedule);

        str = in.readLine(); //Server sends OK
        System.out.println("Received: " + str);

        serverIndex++; //Increment Server Index
        serverIndex = serverIndex % largestServers.size();
    }

    out.write(("REDY\n").getBytes());
    System.out.println("Sent: REDY");

    str = in.readLine();
    System.out.println("Received: " + str);
} while (!str.equals("NONE"));
```

Figure 8: Largest Round Robin

4.4.6 Server Class and Parser

A server class will initialise different data types within the Server itself. Every Data Type will be different and unique within the Server class. The parse functions [7] will be used to convert any String functions to an integer, enabling them to be passed throughout the Client-side simulator.

```
public class Server{
    //Initialise different data types for Server ArrayList
    String Type; //Server[0]
    int ID; //Server[1]
    String State; //Server[2]
    int bootupTime; //Server[3]
    double hourlyRate; //Server[4]
    int coreNo; //Server[5]
    int Memory; //Server[6]
    int Disk; //Server[7]

    //Parameters taken from Section 8 of ds-sim User Guide by Young Lee, Young Kim and Jayden Kim
    public Server(String type, String id, String state, String bootup, String rate, String cores, String memory, String disk) {
        this.Type = type;
        this.ID = Integer.parseInt(id);
        this.State = state;
        this.bootupTime = Integer.parseInt(bootup);
        this.hourlyRate = Double.parseDouble(rate);
        this.coreNo = Integer.parseInt(cores);
        this.Memory = Integer.parseInt(memory);
        this.Disk = Integer.parseInt(disk);
    }
}
```

Figure 9: Server Data Types

References

[1] Oracle, "Oracle VM VirtualBox." https://www.virtualbox.org/.

[2] Canonical, “Ubuntu.” https://ubuntu.com/.

[3] V. Chan, “Victor’s COMP3100 Stage 1 Github Repository.” https://github.com/PandaPiller/VictorUbuntu/, 2022.

[4] J. Jenkov, “Java BufferedReader.” http://tutorials.jenkov.com/java-io/bufferedreader.html/.

[5] S. Classroom, “SSL/TLS handshake Protocol.” https://www.youtube.com/watch?v=sEkw8ZcxtFk&ab_channel=SunnyClassroom/.

[6] Microsoft, “Visual Studio Code.” https://code.visualstudio.com/.

[7] Oracle, “Class Integer Document.” https://docs.oracle.com/javase/7/docs/api/java/lang/Integer.html/.