Project report: Stage 1
COMP3100 Distributed Systems, S1, 2022
SID: 46129448, Name: Mariel Anne Uykim

1 Introduction

In this project, the task is to simulate the DS-Sim simulator protocol. Specifically the task is to create a client side program that dispatches the jobs given to the available servers. This is implemented using Java as the programming language.

DS Sim is a discrete event simulator that schedules jobs to different servers. It contains two main components: the client and server that work together to schedule and finish jobs.

In this report, I will discuss the individual components of the DS Job Scheduler System as well as how they work together. More specifically, I will be discussing the client side aspect of it.

The GitHub repository for the final project can be viewed here (`https://github.com/mariel-uykim/DSClient.git`). The main DSClient file is titled 'DSClient.java'.

2 System Overview

DS-Sim follows a client-server model, wherein each process or terminal within a network is either a client or a server. The client being the one who initiates the communication and the server responding to the requests. The DS-Sim specifically communicates through socket programming in Java. Socket programming provides a two way communication channel between the client and server operating in a network environment. It allows the client program to communicate with the server application which is written in another language (C) due to its transparent nature [1].

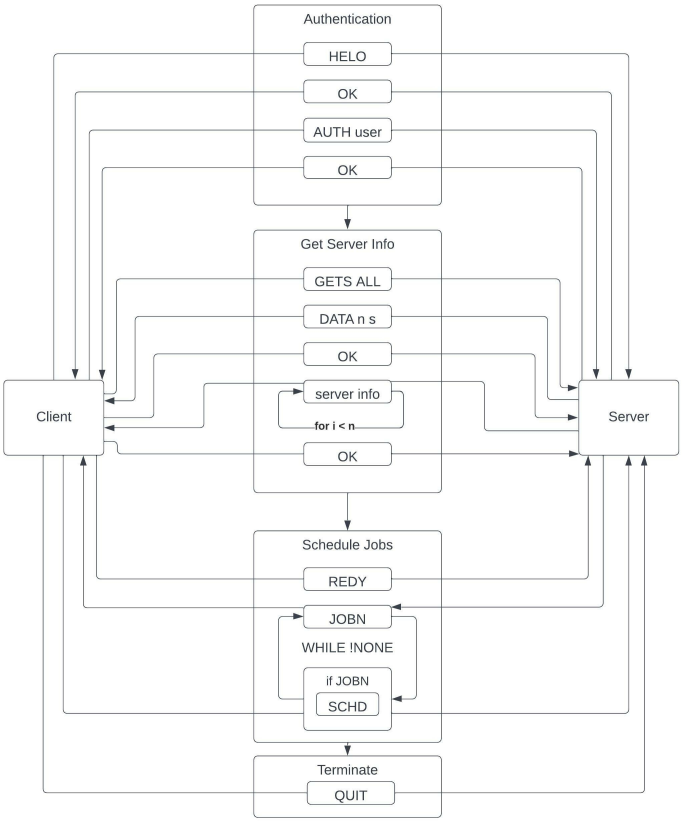

Figure 1: DS Sim workflow using LRR algorithm

As observed in Figure 1, the client and server continuously communicates throughout the process. The client mainly being the one to send commands and the server being the one who receives it and sends back information. They continuously exchange messages until the tasks are finished and the final command of QUIT is sent by the client.

2.1 DS Server

The DS Server handles the server side of the program. It simulates the server-side aspect of the client-server model. Its role includes handling the servers, executing the jobs given, and handling client requests. It reads from the user specified configuration which is written in XML file format to see all the servers available and all the jobs that need to be fulfilled. Once started, the server-side simulator listens for a request in a specified port. Once it receives a message on that port, it starts to execute the command. It communicates with the DS Client throughout the entire job scheduling and execution process using the Simple Mail Transfer Protocol (SMTP) [2].

2.2 DS Client

The DS Client represents the client side of the Job Scheduler program. The task of the DS Client is to schedule the jobs given to it by the DS Server. It decides which servers to use for each given job. There are multiple algorithms that could be implemented to achieve this task, however, this program implements the Largest Round Robin algorithm.

The Largest Round Robin algorithm is a simple algorithm that schedules the jobs with the servers having the largest server type The is defined by which server type has the highest CPU cores.

3 Design

The Distributed System Simulator (DS-Sim) has two main components: the server-side simulator and the client-side simulator. Each were designed and given several functionalities in order to perform their intended roles.

3.1 Design Philosophy and Considerations

The design of the DS-Sim is based off of the client-server model wherein the DS Client is the client side and the DS Server is the server side application. When designing the system, it was important to consider the expectations of it based on the Assignment specifications. This includes its compatibility to the existing DS Server application, scheduling jobs among the largest servers in a round robin fashion, and following the query format of the server-side application.

3.2 Functionalities

The DS Client's functionality is limited to being able to communicate to the server, requesting for data, and scheduling jobs given to it using the LRR algorithm.

3.3 Server communication

Designing the server communication process was important to begin the process. Using packages from Java's in-built libraries (which will be further discussed in Section 4.1) it was easier to come up with the design for the system. The first part is to establish a connection to the server-side application. When opened, the DS Server simulator actively listens to the port and the client sends the HELO message to the same port. When the server sends back an OK message, the connection is successful and the other operations can begin. The queries sent are expected to be in a certain order. The client must connect to the server first and authenticate, and then get the server information before deciding how and which to schedule and then finally scheduling the given jobs.

3.3.1 Designing LRR Implementation

When deciding how to design the implementation of the LRR algorithm, there were multiple ways that were tested to get the largest servers.

A naive approach was first implemented to find the largest servers. It was designed to get the last item sent in the list as it assumed it was the largest data type and assumed that the server id type was consecutive so the server ids were in the range of zero to the current server id. This was based off the assumption that the list was always in ascending order in terms of the number of cores and that the server id is just incremented by one.

This would cause problems later on due to the possibility of the server having multiple server types with the same amount of cores. A revision was made into the design which now loops through the entire list of servers to ensure and stores the current largest type in an array. It compares each server until it has gone through the entire loop. The servers with the first largest server type is then used when implementing the LRR.

To implement the actual LRR algorithm, the list obtained earlier is looped through. The current server in the iteration is used to schedule the current job.

3.4 Constraints

The usage of the DS Client is limited to the specifications and requirements of the project which includes its compatibility to only the DS Server application from the DS-Sim folder. Apart from compatibility, the list relies on certain data information to be in a certain order. For example, it expects the number of servers in the DATA statement returned by the server-side application to be in the second position. To handle any changes to positions, the index variables have to be adjusted accordingly.

4 Implementation

To create the DS-Sim Client-side simulator and to ensure it is able to communicate to the server properly to do its tasks, the following libraries, data structure, and functions were implemented.

4.1 Technologies and Libraries

DS Client utilises three in-built Java libraries. This includes the Java.net package to socket class which allows the application to connect to the server. The second package is the Java.io package which is used for reading inputs from the server and outputting data or queries to the server. The last package used is the Java.util package which is used to implement the ArrayList data structure (discussed further in the next sub section).

4.2 Data Structure

DS Client uses arrays and array list to organize and store server information. Since the server information that is given by the server-side simulator contains multiple data for each server, the information needed to be stored in a 2D array with the rows being each server and the columns being the data of the server. Since the number of servers that have the largest server type is uncertain, an array list was the choice of data structure used as it can expand dynamically.

4.3 Functions

DS Client has several functions that are used to implement certain features.

4.3.1 Server Connection

Using the Socket class from Java.net package, a function was implemented to connect to the server that is listening on the same port as the Client.

4.3.2 Authentication

The client authenticates itself to the DS Server by using the AUTH command followed by the name of the system. The server responds with an OK if authentication was successful.

4.3.3 Sending and Receiving Messages

The DS Client has separate functions for sending messages to the server and receiving messages from the server. Using Java.io's BufferReader and its readline method that reads from the InputStreamReader, the client is able to read the messages sent by the server. The function for sending messages uses the DataOutputStream

to write the output to be sent to the DS Server. These functions are used in order to communicate to the server by sending commands and receiving data.

4.3.4 Getting Server Information

To get information on the servers, the client has to use the GETS command which has a parameter of getting servers based on the capabilities, availability, or all the servers. The current DS Client application uses the GETS All command which responds with DATA nRec size. The nRec being the number of servers available.

After sending an OK, the DS Server starts to send the servers line by line. The DSClient implements a for loop that concatenates each server and the data that follows it into a string and separates it with "###" to make it easier to split later on.

4.3.5 Finding Largest Server Type

In order to find the largest server type the string containing all the server information is splitted twice using the .split() method. First being with "###" which is stored into an array and the second being with " " which is stored into a two-dimensional array. In the process of storing data into a two-dimensional array, the client is finding the largest server type by checking the number of cores it contains and if the current server has the same server type. If the current server has the same type as those on the 2D ArrayList, it gets added into it. If it has a larger server type, the 2D ArrayList gets cleared and it is added into the list. The 2D ArrayList, which contains all the servers with the largest server type, is returned.

4.3.6 Scheduling Jobs

After getting a list of all the servers with the largest type, the DS Client can schedule jobs in a round robin fashion using a while loop. The client tells the server it is ready using the REDY command and would get a JOBN response that contains details of the job. An index (i), which is set to 0 initially and would be incremented to the length of the list minus 1 before it gets restarted, keeps track of which server to use. The client then sends a SCHED command to the server that contains the server id, server type, and the job id. This continues to loop until there is no more jobs left available. In this case, the loop can end.

4.3.7 Closing Connection

After scheduling all jobs and successfully completing all of them, the Client sends a QUIT command to the server and closes the socket connection in the client side before terminating completely.

References

[1] R. L. R. Maata, R. Cordova, B. Sudramurthy, and A. Halibas, "Design and Implementation of Client-Server Based Application Using Socket Programming in a Distributed Computing Environment," in *2017 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*, (Coimbatore), pp. 1–4, IEEE, Dec. 2017.

[2] Y. C. Lee, "ds-sim: an open-source and language-independent distributed systems simulator," Dec. 2020.