Project Report: Stage 1
COMP3100 Distributed Systems, S1, 2022
SID: 45956987, Name: Dylan Vongsouvanh

1 Introduction

This report aims to outline the goals of this project involving ds-simulators. The goals defined will particularly be reflective of Stage 1 of the project. The project (i.e., ds-sim, source code, report, etc.) can be access via GitHub [1].

The main goal of Stage 1 is to develop and implement a client-side simulator for the ds-sim which will connect to the sever-side sim, receive jobs, and will schedule future jobs. The jobs that will be schedules will be based off LRR (Largest-Round-Robin) which will schedule the largest job available based off the number of cores. If more than one job is found then the first one will be used.

This report will contain the following:

- System Overview
- Design
- Implementation

The System Overview will describe how the system both server-side and client-side will work outlining the process of running jobs.

Design will look in depth of what the client-side simulator will look like, mapping out the functions within whilst looking into considerations and constraints of the project.

Finally, implementation will explore what technologies, techniques, software libraries and data structures will be applied to create the client-side simulator and how they are implemented.

2 System Overview

This section will describe the workflow of the ds-sim showing the steps taken to connect to the server-side simulator and how to dispatch jobs. Firstly, the server must be running before the client can connect using sockets running on localhost and on port 50000. Once connected the client will begin communicating with the server and will begin a Handshake Protocol starting by sending “HELO” as a greeting in which the server will reply with “OK” once received. The client will then send an authentication request using “AUTH [username]” and as the server does not actually require proper authentication it will again reply with “OK” and will welcome the user to the system and will grant access to “ds-system.xml”.

The client will then read through “ds-system.xml” then send “REDY” to signal the server that it is ready to receive the next job. The server will reply with the next job written as: JOBN [submitTime] [jobID] [estRuntime] [core] [memory] [disk]. An example is: JOBN 37 0 653 3 700 3800. The client will then run the GETS command either using “GETS All” or “GETS Capable [job information]” which the server will send the number of available jobs. The client will then send “OK” and will receive a list of the jobs from the server followed by another “OK” from the client. The server will then send “.” to signify that there is no more information to send. The client then can begin scheduling a job using the command: SCHD [jobID] [serverType] [serverID]. An example is: SCHD 0 joon 0. The server will then tell the client that the job has been scheduled.

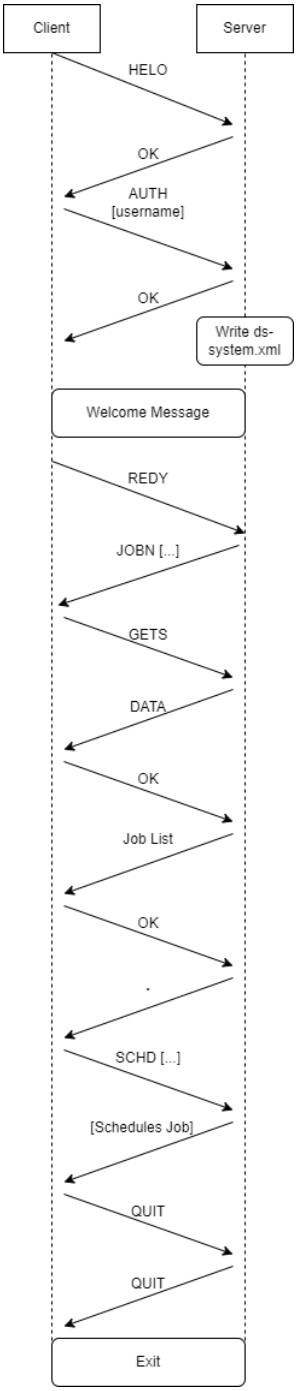

Figure 1: ds-sim workflow diagram

3 Design

In this section, the functions of the client-side simulator will be explored focusing on connecting to the server, handshake protocol, and receiving and scheduling jobs.

3.1 Connecting and Communicating with the Server

To connect to the server, the client will need to create a TCP socket and initialise it to connect to an IP address (localhost) and on port 50000. To enable communication between server and client input and output streams will be used to read the data being sent on both ends. BufferedReader will be used to read the data being sent from the server while DataOutputStream will be used to send the data being sent by the client. A conditional statement will be written to end connection to the server once there are no more inputs/data being received from the client or the server receives “QUIT”.

3.2 The Handshake Protocol

When connecting the the server, the client will need to complete a handshake protocol to prior to scheduling jobs. The client will begin by writing the message “HELO” which would then be converted into bytes in which the server will respond by sending bytes that when read will produce “OK”. This process will be repeated for authentication writing the message “AUTH [username]” instead.

3.3 Reading xml Files

Before the client can receive jobs it needs to read and extract the data from “ds-jobs.xml” which stores a list of jobs. This function will include parsing the data in the file and creating arraylists to store the data. A DOM parser will be used to extract the data. The file path for the xml file also needs to be defined for the parser to work. As the parser is reading the data, it will temporarily save each element into a list. A for loop will be used to store all the jobs available for the client to run. However, as the simulator can only run one job at a time only the first job will be presented. A job class will also be implemented to initialise the data types of each element in the arraylist. This class will also simultaneously store the data extracted by the parser into the arraylist.

For the client to read “ds-system.xml” and gather all the servers, the same functions will be applied however, the server arraylist will have its own class to initialise its own data types. However, unlike the jobs, all the servers available can be presented to the client.

3.4 Receiving Jobs

3.5 Scheduling Jobs

4 Implementation

References

[1] D. Vongsouvanh, “Github repository.” https://github.com/Dylan-vong/COMP3100Project.