Stage 1: Simple Job Dispatcher

Group Members:

1) Isaac Zhuan Jian Lee (45526249)

2) Rhyne Fong (45912491)

3) Maham Naqvi (45559708)

Introduction

The main goal of this Distributed Systems project is to be able to implement a simple job dispatcher as a client that can understand and obtain information from servers and then allocate them jobs depending on the requirements and the resources available. During Stage 1, we are aimed to provide a vanilla (plain) simulator design and implementation that works in compliance to the ds-sim simulation protocol described in ds-sim user guide. In this version of client-side simulator, the client connects to the server-side simulator, receives jobs and then schedules them to the first one of the largest* server type available. The largest server type is defined to be the server with the largest number of CPU cores and the function responsible for determining the largest server type is referred to as “allToLargest" in the Java code developed. This project is mainly focused on developing the client side of the ds-simulator with the server side already developed.

System Overview

To enable the development of an efficient client-side simulator, learning comprehensively and fully understanding how servers and clients communicate to each other was crucial for the success of this project. Therefore, during this stage, we devoted a large amount of time to conduct a detailed analysis of the requirements of the server and how we should be able to interact with it as a “client”. To begin with, we assigned the server the localhost 127.0.0.1 and port number 50000 in order to set up the server.

The main criteria of the project we needed to brainstorm on during our Client development process was determining how a Java program will interact with the C-program environment. In order for Java and C platforms to be able to successfully communicate with each other, we needed to understand the provided source documents from iLearn and Github as a reference guide and then break down the main functionalities and requirements of the server. Despite being challenging, through constant trial and error approach, we were finally able to successfully solve the problem and reverse engineer the server source code which was hidden from us as part of the assignment to solve. We were able to verify the correct functioning of the code by being able to display the exact same output as the expected output from the given XML files.

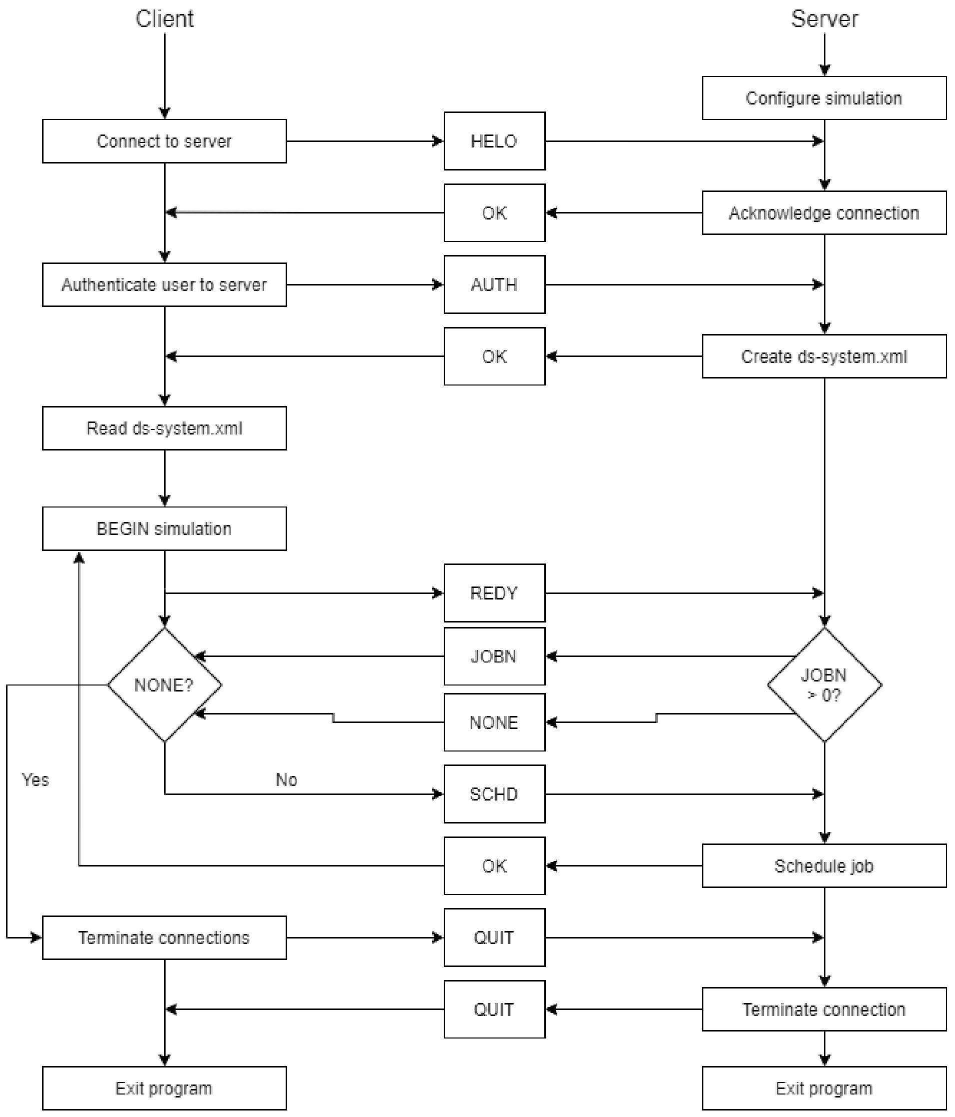

Figure 1. Flowchart of our design and implementation to dsClient.java

Throughout the development process, we faced different challenges and bugs that we used as potential opportunities to learn from and improve our communication of the client program with the server. As a team, we were able to break down the different requirements and functionalities and implement various methodologies and APIs that resulted in the success of the project. By equally distributing the workload between the three team members and devoting our time and efforts completely, we were able to achieve the goals of the project and enable a working client side that could understand the various inputs and then produce the correct output against the expected one as provided in the User guide.

Design

To commence our work on the project, we began by creating three java classes: dsClient.java, dsJob.java and dsServer.java to follow good programming practice of dividing big chunks of code into relevant classes and methods. The dsJob.java and dsServer.java classes contain relevant functions and methods that store, process and retrieve information about the different scheduled jobs and server type descriptions derived from the config .xml files. As much as it is essential that the different classes are able to effectively communicate with each other and enable the desired functioning of the code, it is also fundamental to ensure that everyone including the team members as well as any potential outside viewer is able to easily understand the algorithms of the code program. Therefore, to facilitate simplicity and improve readability of our code, we adopted commenting extensively throughout the different sections of our code program to ensure “good” coding practice so that all users can understand how the different functions work together to bring out the desired functionality of the ds client side simulator and also understand how the data is parsed in terms of transferring attributes from the XML files.

To allow uniform structure and good design of the code, we mutually agreed to do some basic prototyping in our local computers, before committing and pushing to GitHub. We analysed what classes, attributes and methods are necessary elements required and then assigned each other different sections of the code to work on individually. Whilst developing our part of code, we made sure we followed good naming conventions when it came to choosing appropriate variables and function signatures. We also ensured we all made comments for our developed part of codes so that all three team members could understand the development process and follow accordingly. We tested two terminals in Ubuntu where client and servers communicate with each other by sending messages through an assigned port, 50000. Through multiple tests after regular intervals, we were able to send messages across successfully. The diagram below shows how our client program connects with the server program with the aid of socket programming which helps two applications to communicate with each other through a common port.

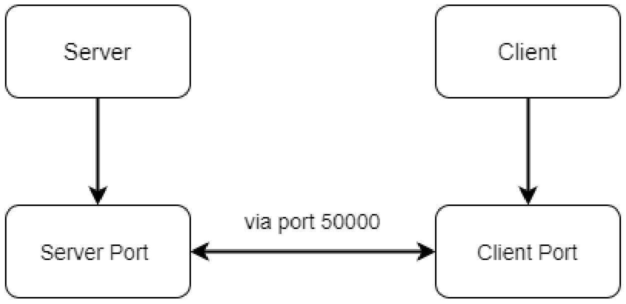

Figure 2. Socket Programming

<u>Implementation</u>

One of the biggest challenges we faced in the implementation of the Client side ds server was the fact that the server side of the system used C language and did not have String data type in its environment unlike Java. Instead, within the C language environment, it runs and stores an array of Chars to hold String values [1]. Therefore, in order to make the two sides of ds system compatible with each other, we used bytes[] array data structure to store and send String messages to the system in the Java program. In the implementation, we ensured that when we sent messages to the server using "HELO, REDY, etc." commands, we created a handshake connection between the "client-side" and the "server-side" before asking the server to process and render the scheduled jobs. After the handshake was completed, the client would locate the .xml files containing attributes and through parsing data within the files, it would successfully schedule and provide jobs to the server.

After importing a number of modules and API libraries and implementing the handshake connection setup with our server through socket programming, next we ensured that the client was to dispatch jobs to a specific server as per the requirement goals of the project. It was expected that the next largest server available with the largest number of CPU cores would be selected by the client for job scheduling. To determine which server is the largest we created an "allToLargest" function to process and identify from a list of available servers in the form of ArrayList data structure (to capture all available servers in the xml file), the particular server that contains the largest number of CPU cores. The largest server was then chosen and the client would then dispatch jobs to this server.

In order to test the communication between the dsClient and dsServer, we used two terminals to run both sides of the ds system. In one terminal, we compiled a Java file using the command *Java dsClient* whereas in the other terminal, we ran server-side using the command *./ds-server ../../configs/* with our chosen config-file as jobs to dispatch. This simulated a job dispatcher and we were able to successfully analyze that the server-side indeed produced messages that were received by the client. We were also to check and verify the status of the server's processing performance. Therefore through dedicated code development and rigorous testing, we were able to achieve the set goals of the assignment and successfully develop a vanilla version of the client side server that can connect to the server-side simulator, receive jobs and then schedule them to the largest server type available.

During this project, Maham was dedicated and responsible for establishing the socket connection programming between the server and the client, Rhyne was focused on ensuring that the data was correctly parsed and retrieved through the XML config files and for the correct implementation of the allToLargest function. Finally, Isaac was devoted to process any missing gaps or functionality in the code and to improve the code reuse, commenting and refactorisation.

References

GitHub Repository: https://github.com/IsaacZJMQ/COMP3100Stage1

[1] "Representation of Strings (The GNU Library)", [Online]. Available: "https://www.gnu.org/software/libc/manual/html_node/Representation-of-Strings.html [Accessed 17-April 2021].