Stage 1: 'vallina' client-side simulator with a simple job dispatcher

Project Title: DS (Distributed Systems) Job Scheduler

Group Members: Lakshmi Priya Bhuphatiraju (45431957), Tonmoy Ahmed Jitu (4266278)

Introduction

The project is to develop a Client-side program which can send command to the Server and comprehend the information from ds-server. The Client should also be able to schedule the jobs to a specific server depending on various factors such as but not limited to available resources and the largest server.

The implementation of the "vanilla" client-side simulator is done in Stage One where the client receives jobs and schedules them to the largest server available. Our design is to implement methods which will allow us to send, receive and comprehend information from the server. The Client-side is a simple simulator which runs by allocating jobs to the largest server available.

System Overview

In order to design and implement the Client-side, our team had to understand the Server-side in detail, including how to interact with the server (which protocols to use), how to connect to the server and how to send or receive requests or responses from the server.

Our main design philosophy was to keep our code as readable as possible and to have separate classes which had its own unique features. This allowed us to not have multiple garbage code running in the background and turn into a bug in the later stage. We also made sure we understood how the Client and Server side communicated so that we can start to develop the methods we may need to write in our program. We also made sure that every single commit had a specific description just in case we can revert to previous versions if anything went wrong.

We had to keep a lot of things under consideration during the design stage. First, we are only two members in the group, hence we had to divide the work accordingly and find out the best way to utilize our time during the new form of online study. We also had to ensure there was proper communication so that we can help one another out if the concept was not clear. We made sure each method was discussed and made clear so that we can understand and come up with quick bug fixes.

While developing the program a lot of constraints have to be dealt with. Firstly, we had to understand how the Server-side program was coded. Hence, we divided the work with a lot of time being allocated for research and reading so that we can clarify our concepts. We also had to dissect the Server-side code and find out how it was working and using different inputs to run its side of the program. This enabled us to better get a grip of what the Client-side had to communicate with the Server-side. Secondly, we had to find out a way to separate data from the XML file and store it accordingly so that we can use the attributes later to find the largest server. This took a fair bit

of work but was fairly easy once we used the Javax library to get most of the work done. Last but not the least was getting to run it in the linux environment. There were constraints like not having the proper library installed which was resolved by looking up some quick fixes.

System Architecture

The system consists of methods and functions which are used to send and receive commands from both the Client-side and the Server-side. There are three main classes Electrode3100, Client and Server. Whereas Electrode3100 has all the methods and functions implemented for communication with the server, the Server stores attributes from the jobs, and the Client class simply starts the simulation.

From the class Electrode3100 some of the components are explained below:

- Send: This method sends commands to the Server-side using byte codes. The function flush is used within the method which clears the "output" string of any garbage value after any message has been sent.
- Receive: This method receives commands from the server. The commands are stored in a string format. Since the Server-side sends commands in bytecode, the method needs to receive it accordingly.
- Quit: If both the Client and Server passes the QUIT command the program initiates the close protocol.
- Run: The run method initiates all other methods and actually schedules the job.
- ParseXML: The parseXML method is used with the help of the Java DOM Parser library. This uses a special document builder which reads through the attributes of information from the XML file and sorts it according to the tags. The method also uses the NodeList to store the information. All the tags are sorted using the Server Class. A node list is used as it allows easy extraction of meta datas.
- SetLargest: This method is a setter, which sets the largest server ID to allToLargest.
- FindLargestServer: This method goes to the ServerList and finds the largest server by comparing one server with another.

Figure 1.0 gives a rough idea of how the program communicates with the Server-side and what is the expected output for specific commands.

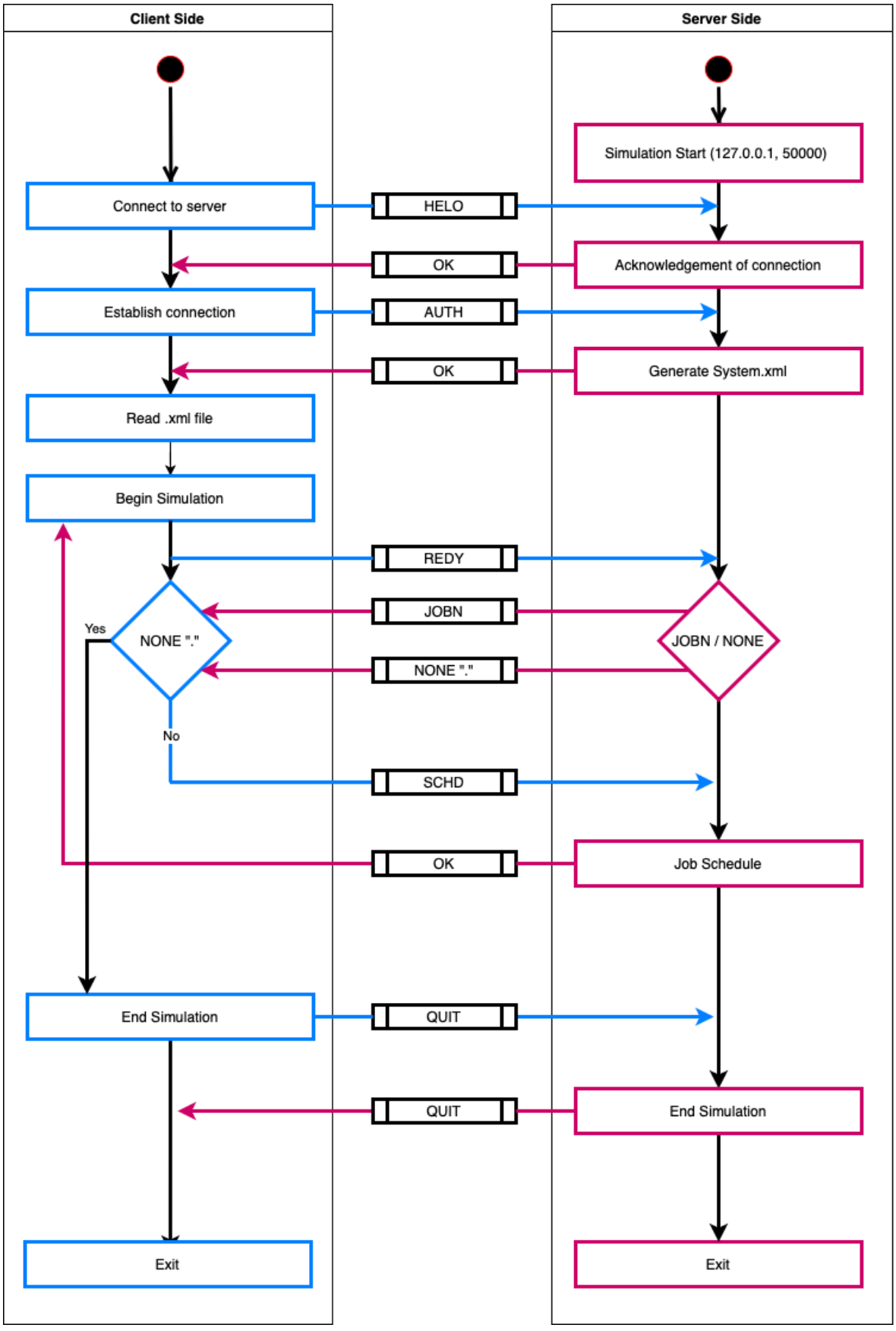

Figure 1.0 Client-side and Server-side communication

Implementation Details

The program was implemented using Java Programming Language as we're familiar with syntax. The whole job scheduler was run in a linux environment (Ubuntu), where the java programs were compiled and run along with the server.

To start off with the project we had to come up with an UML for the necessary methods we should have in our program in order for the Client-side to communicate with the Server-side. We used the concept of design patterns so that it could allow us to build on the program in the later stages. We also discussed coding practices like readability of the code, commenting out codes so that the other person can understand better and also using the same environment for development. Please refer to our detailed architecture diagram (Figure 1.0) above that helped us with our implementation strategy.

Understanding Server-side Application Programming Interface (API) from the given C code was pretty daunting. It took us a little while to understand how the server used the commands and the information. We then wrote our Java class to first try and establish a connection to the server. Socket programming needed a lot of research around how to extract information out of an ".xml" file and use that information to extract the largest server. This was a tedious exercise that consumed quite a chunk of our time. Parsing the response from the server had few challenges, we did however successfully get the expected output after rigorous testing. Having prior knowledge of C certainly helped.

We had three main important class;

- *Electrode3100* which consists of all the necessary methods needed to send and receive commands to and from the Server-side.
- *Client* which is used to initiate the simulation.
- *Server* which saves the attribute of the XML file.

The classes consisted of libraries with specific features which enabled us to implement some of the methods. Especially the Java DOM Parser library which helped to build an array of string using tags from the XML file. Then we followed through by making our own Node list by reading the attributed data from the different tags. Another method (allToLargest()) was used in conjunction with the XML parser, which goes through the core count and identifies the largest server from the list.

Priya is in charge of building test cases for ensuring the methods were built properly after being written. She also took part in finding out how to extract information from an XML find using a Java DOM Parser library which was very efficient and easy to use once we got the idea and the concept behind it. Tonmoy is in charge of building class diagrams and dissecting the Server-side code which was used to understand how the Server-side communicated with the client. The following figure shows the commit rate of our github project. As you can see from figure 2.0, we made the majority of the commits on Tuesday which is our practical session day.

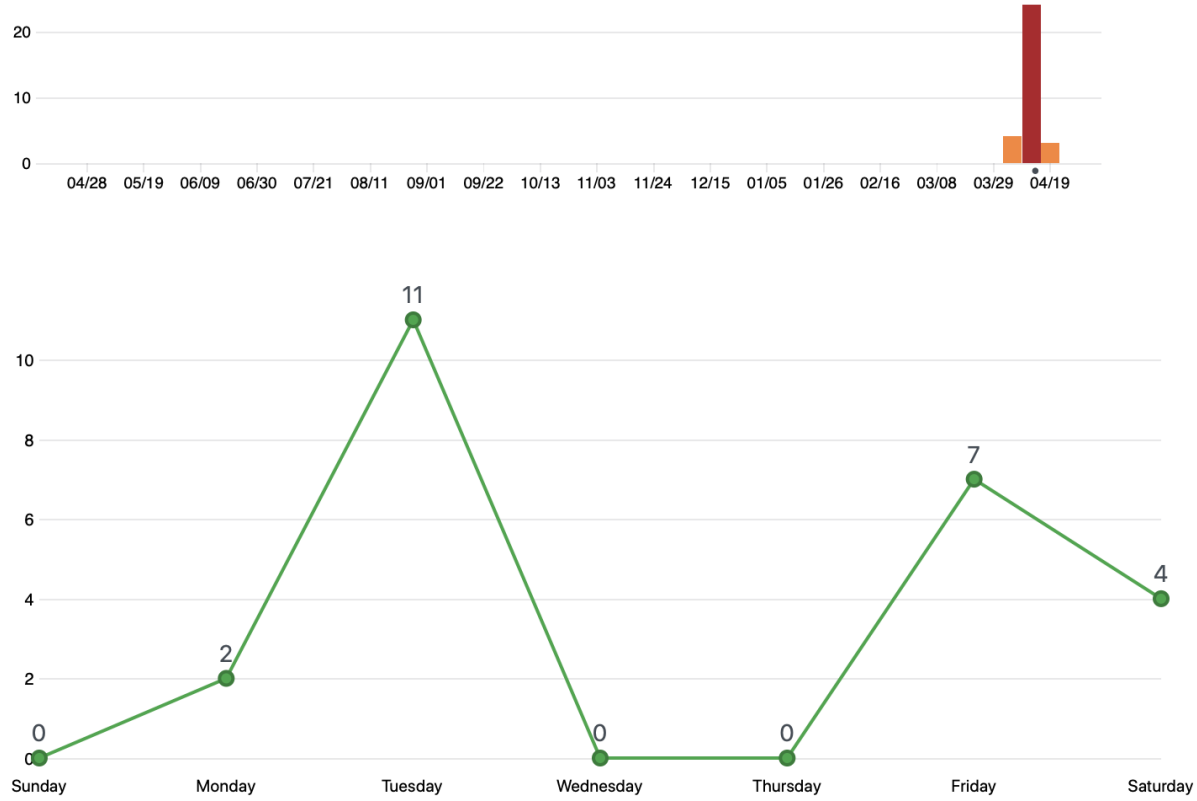

Figure 2.0 Github commit rate in a following week.

References:

[1] Davide Brugali, 2019, ”Pattern Client-Dispatcher-Server” Research Gate. <https://www.researchgate.net/figure/Pattern-Client-Dispatcher-Server-from-Buschmann-96_fig1_2835461> (accessed March 18, 2020).

[2] Neha Vaidya, 2019. ”Know all about Socket Programming in Java” Edureka. edureka.co/blog/socket-programming-in-java/ (accessed March 29, 2020).

[3] Tutorialspoint.com. 2020. *Java DOM Parser - Parse XML Document - Tutorialspoint*. [online] Available at: <https://www.tutorialspoint.com/java_xml/java_dom_parse_document.htm> [Accessed 12 April 2020].

Github: https://github.com/tonmoy0010/electrode.3100