Cloud Job Scheduler for Distributed Systems

|  |  |
| --- | --- |
| **Name** | **Student ID** |
| Rajiv Mehta | 45433062 |
| Chan Alex Ros | 44419775 |
| Lachlan Tripolone | 45229287 |

Table 1. List of Group Members

# Introduction

Distributed systems refers to a collection of independent machines that communicate messages towards each other in order to achieve common goals that are set by the client [1]. The main components of distributed systems are the servers. These servers are often connected together and work together to achieve the same goal. Individual servers can vary in many aspects such as size and processing power and in turn, allow for better control of resources when jobs are requested by the client. This creates a sense of ambiguity for the client-side on the number and server types that there are as the servers work concurrently perform jobs. Therefore, the goal of this project is to simulate a distributed system with a client and an unknown server as specified by the client below.



The client has assigned the team to create a client-side simulator with a basic job scheduler and dispatcher in order to simulate the exchange and processes of distributed systems. In particular the client-side simulator must be able to successfully connect to server-side simulator, receive information from the servers in the terms of ‘Jobs’ and then schedule these jobs. With these jobs, the client-server then needs to sort through the server types and send all the jobs to the largest server type, defined by the server with largest number of cores.

# System Overview

The simulation of a distributed system has two main components, the client-side simulator that the client has tasked the team to create and the server-side simulator. The server-side simulator is unknown to the team though, as previously described in section 1, in distributed systems servers can be different in different aspects such as processing power and they can all work together to complete jobs. This means that the server-side simulator is a cloud and the client-side simulator needs to be able to request information from the server about all the types of servers available, then sorts through all the server type and find the largest server type. Afterward, the client will schedule all the job to the largest server and send it accordingly to the server so that it can process all the jobs as requested.

The client-side simulator can be split up into three main functions:

* **Handshake**: The first step is to set up the connection between the client and the server. This can be done by sending and receiving messages to and from the server as seen in figure 1. Before the ‘Welcome Message’.
* **Finding the largest server type**: After the handshake has occurred the client-side then sends a message to the sever asking for information of all servers in the system, regardless of their state. The client-side then stores this information and sorts through it to find the ‘largest server type’.
* **Sending Jobs to the Largest Server type**: The last function that the client-side simulator does is the send the jobs to the largest server type. The client first sends a ‘REDY’ message to the server and the server sends the first job. This then starts a loop within the client-side simulator where the server keeps sending Jobs to the client and the client sending those jobs to the server with the largest server type. The loop ends when the server has no more jobs to send and then the client sends a ‘QUIT’ message to the server causing for the connection to be stopped between the client and server as seen in Figure 1.

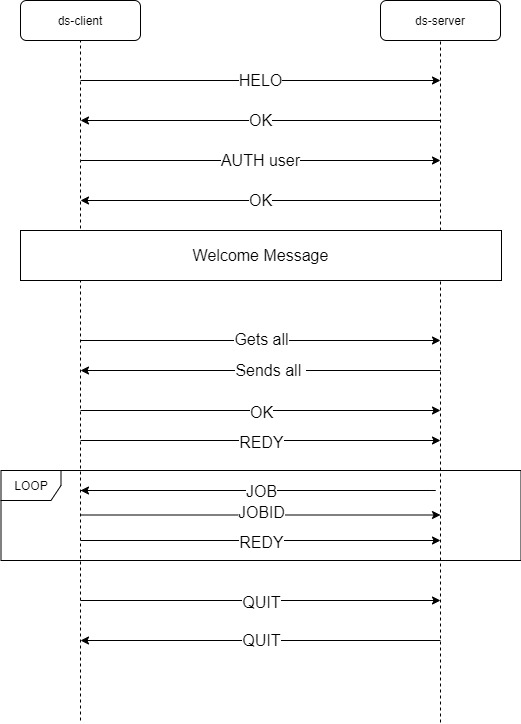


Figure 1. Communication protocol for simulation

# Design

There were many important considerations during the design process of the client-side simulator. This section covers the design philosophy, constraints, and most importantly the functionalities of the simulator.

Firstly, it was crucial to have a clear design philosophy during the design process because it helped to guide during the coding process especially emphasizing on how the code should be structured. The design philosophy for creating the client-side simulator focuses on three vital aspects which are the ergonomics, modularity, and functionality. For the ergonomics aspect, the use of whitespace, comments, and naming convention were followed to enhance the code readability and overall structure. As for the modularity aspect, the code was design in such a way that there were no magic numbers, and the code was easy to understand and modified to comply with future requirements. Finally, the functionality was designed with close consideration to the requirements and specifications given.

There were a few of notable constraints to the design process of the client-side simulators. The main constraint was designing an algorithm for the client-side so that it only schedules jobs to the first largest server and not any other types of server. This was a constraint because it limited the client from just sending jobs to any servers available which would have been much easier to do. Another constraint was the restriction to only using Java to design the client. It made an impact during the communication process between the client and the server because the server was written in C language. The time constraint was also a factor to considered because the project should be completed before the deadline and the client’s functionalities should be fully implemented.

The functionalities of the client-side simulator were designed in accordance with the requirements and specifications given. The simulator utilized socket programming for client and server to communication and send messages so that job scheduling can be implemented. When client was able to communicate with the server, the client-side also needed to be able to request and read multiple lines for information from the server. Finally, the client’s main function is ‘allToLargest’ which analyse the information about the servers available and finding the first largest server type so that it can later schedule all the incoming jobs to that largest server. These were the functionalities that the client was design to do so that it meets with the requirements and specification.

Overall, the design process was important as it helped to guide the implementation process of the client-side simulator. There were three focuses for design philosophy which helped to enhance the code readability and overall structure of the code. The constraints also played a huge role in the design process as there were obstacles to overcome. Finally, by following the requirements and specifications the client’s functionalities were developed and implemented.

# Implementation

As mentioned above, the implementation process follows closely with the design process as it helps to have a clear understanding the goal of the project and how it should implement. This section will explain the techniques and software libraries used to successful develop the client-side simulator. Moreover, it will discuss about the group members and the functions they were responsible for. Therefore, the section will explain the three main implementations which are the handshake protocol, finding the largest server type, and the algorithm to schedule jobs to the largest server.

The handshake protocol is the first main component that needed to be implemented and was looked after by Rajiv. The handshake-protocol is a protocol that creates a connection via the client-side simulator and the sever-side simulator. It does this through the use of sending and receiving designated messages to each other defined in the DS-USER manual. The first step was to send out the ‘HELO’ message out to the server. As the servers are C-based and the team used Java, it meant that when writing out messages to the server, the messages had to be converted into bytes in order for there to be no issues when the server received messages. Then when the server received the “HELO’ message, it would send a signal back and the protocol would then send out the next message in order to establish a connection. This continues until both the client-side and server-side are in a ready state, meaning that the connection between the client and server is secure. The exchange of messages can be seen in Figure 1 on page 2.

The person in charge of the finding the largest server type was Alex. For finding the largest server type, the client-side’s code needs to import the Java Scanner library because the delimiter method was used to break up the message making it easier to read and compare. To implement this function, there were three important variables that were created and used to keep track of the largest server type, the ID of that server, and finally the number of cores. Then a loop is created to loop through the incoming lines of messages about all the servers available. The space ‘ ‘ was used as the delimiter to breaks up the messages and stored the information accordingly in the variables. The loop goes through each line of message and compare the number of cores for each server type and as a result finding the largest server core which will be the largest server type. It would then be stored correctly in the largestServerType, largestServerID, largestServerCore.

The algorithm to schedule jobs to the largest server was handled by Lachlan. A loop is created to read the jobs type and ID which will then be used to schedule to the largest server. The algorithm was heavily reliant on the completion of finding the largest server because it needs that information to send to the server. Similar to finding the largest server type, it uses delimiter to scan the message about the next jobs to extract the job ID that will be used to send to the server later. Finally, in the loop, there is a condition that check to make sure the message about the job is new/next job (JOBN) and not about job completion. Finally, it just schedules the job by using the job ID and sending it to the largest server by using the information from the implementation by Alex. Once there are no more jobs or it is completed, the client will send QUIT to the server indicating the scheduling process is finished.

# References

[1] Y. K. K. J. K. Young Choon Lee, "ds-sim: A Distributed Systems Simulator User Guide," Macquarie University, 18 02 2021. [Online]. Available: <https://github.com/distsys-MQ/ds-sim/blob/master/docs/ds-sim_user-guide.pdf>. [Accessed 17 April 2021].