Project report: Stage One
COMP3100 Distributed Systems, S1, 2022
SID: 46410961, Name: Ava Gardiner

1 Introduction

The objective of Stage One is to design and implement a Client-Side Simulator that functions as a simple job dispatcher, similar to the reference implementation available in the ds-sim repository on GitHub.

A simple job dispatcher is a component of a computer system that assigns incoming jobs to available resources for processing and is used to manage the allocation of resources and schedule the execution of jobs based on different criteria, such as priority, resource requirements, and availability. The job dispatcher plays a critical role in ensuring the efficient utilisation of resources and maximising system throughput. In the context of distributed systems, a job dispatcher is often implemented as part of a job scheduler that operates on multiple servers or nodes, aiming to achieve efficient workload distribution and optimal resource utilisation.

To achieve Stage One, I will follow the guidelines outlined in the ds-sim user guide. I will implement the ds-simulation communication protocol, which enables me to connect to the Server-Side Simulator (ds-server), receive one job at a time, and schedule them. The Largest-Round-Robin (LRR) will be implemented to schedule the jobs and sends each job to the server with the largest type in a round-robin fashion, where the server's size is determined by the number of CPU cores. If multiple server types have the same number of CPU cores, the first one listed in the configuration file or ds-system.xml will be selected. The client-Side Simulator will be created to behave identically to the reference implementation and will generate simulator logs that match the reference implementation.

Overall, Stage One encompasses the creation and execution of a Java-based Client-Side Simulator. The Client-Side Simulator will be responsible for connecting to the Client-Side Simulator (ds-server), receiving jobs, and scheduling them utilising the LRR algorithm. Furthermore, Stage One will provide the foundation for Stage Two's completion, which will require the design and implementation of one or more new scheduling algorithms. These algorithms will be evaluated in comparison to a set of established baseline algorithms, such as First-Fit (FF), Best-Fit (BF), and Worst-Fit (WF), to develop a new job scheduler in a simulated distributed system.

2 System Overview

The system consists of two main components: the Client-Side Simulator and the Server-Side Simulator. The Client-Side Simulator is responsible for simulating the job scheduling process and sending job requests to the Server-Side Simulator. The Server-Side Simulator is responsible for managing the resources of the simulated distributed system and scheduling the received job requests.

The Client-Side Simulator interacts with the Server-Side Simulator through a socket connection. The client first sends a "HELO" message to the server to establish the connection and then sends an "AUTH" message to authenticate the user running the Client-Side Simulator.

Once the connection is established, the client receives information about the available servers from the Server-Side Simulator using the "GETS" command. The Server-Side Simulator sends information about the type, ID, limit, hourlyRate, cores, memory and disk attributes of each available server.

The Client-Side Simulator then waits for job requests from the user. When a job request is received, the Client-Side Simulator checks the available servers to find the largest server that meets the requirements of the job. If a suitable server is found, the job is scheduled on that server and the server's availability status is updated. If no suitable server is found, the job request is rejected.

The Client-Side-Simulator sends a "QUIT" message to the Server-Side Simulator to terminate the connection when the user is finished sending job requests or there are no more jobs to schedule.

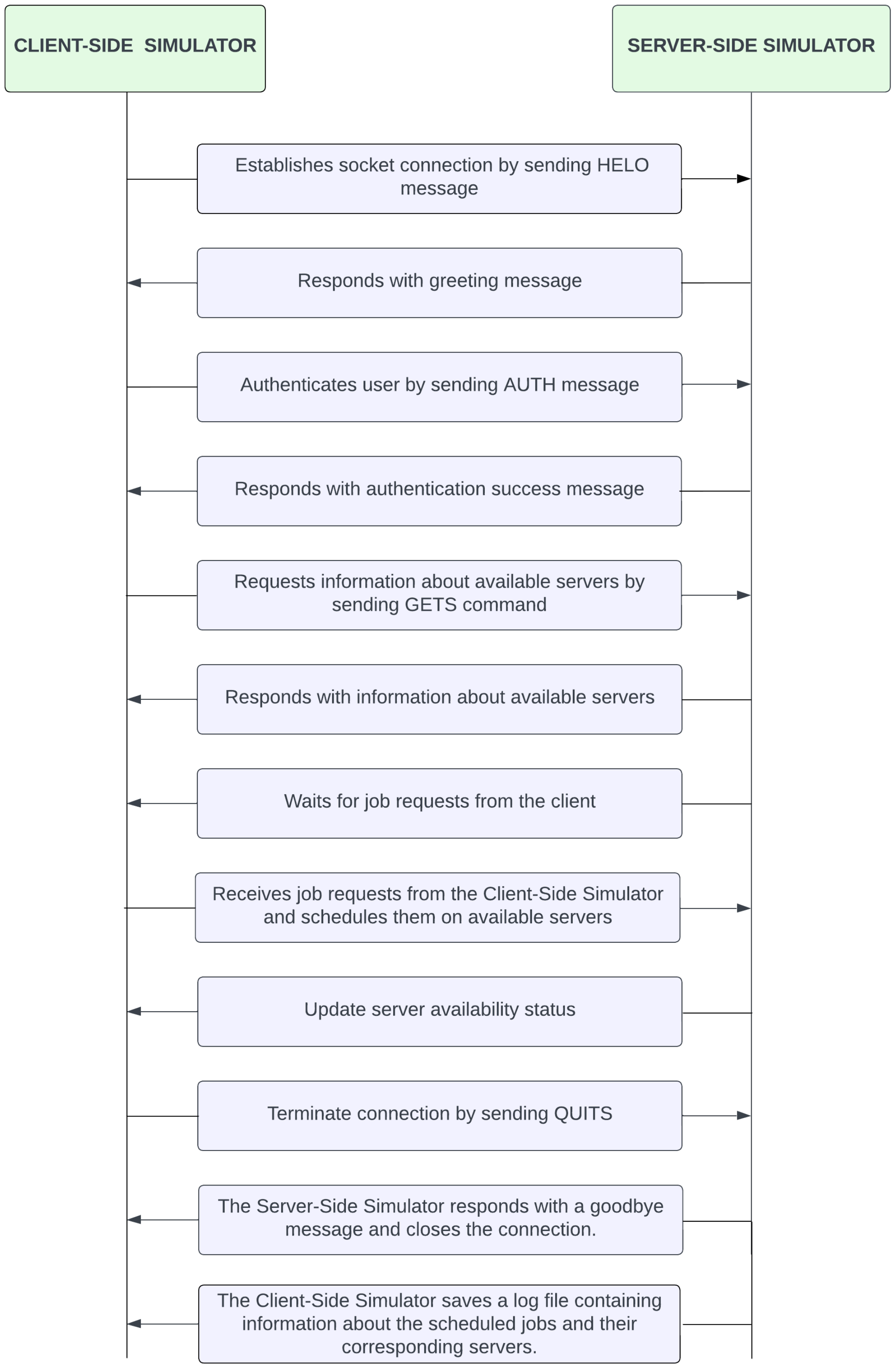
CLIENT-SIDE SIMULATOR
SERVER-SIDE SIMULATOR
Establishes socket connection by sending HELO message
Responds with greeting message
Authenticates user by sending AUTH message
Responds with authentication success message
Requests information about available servers by sending GETS command
Responds with information about available servers
Waits for job requests from the client
Receives job requests from the Client-Side Simulator and schedules them on available servers
Update server availability status
Terminate connection by sending QUITS
The Server-Side Simulator responds with a goodbye message and closes the connection.
The Client-Side Simulator saves a log file containing information about the scheduled jobs and their corresponding servers.

Figure 1: Workflow/working of both the Client-Side Simulator and Server-Side Simulator

Figure 1 provides a visual representation of the workflow of both the Client-Side Simulator and the Server-Side Simulator (ds-server). The diagram illustrates the steps involved in establishing a connection between two simulators, sending job requests, receiving information about available servers, and terminating the connection. The visual representation can aid in understanding the system's overall architecture and how different components interact with each other.

3 Design

3.1 Design Philosophy

The design philosophy for Stage One is to create a simple job dispatcher that implements the ds-sim communication protocol, as described in the ds-sim user guide. The main focus of the simulator is to receive jobs from the Server-Side Simulator (ds-server) and schedule them in a circular fashion to the server, starting from the server with the largest number of CPU cores, which is achieved through the implementation of the Largest-Round-Robin (LRR). The LRR algorithm ensures that each job is assigned to the server with the largest number of CPU cores, which helps to optimise performance by utilising the processing power of the most capable server. Additionally, the LRR algorithm ensures that each server is given an equal opportunity to process incoming jobs. The design of the Client-Side Simulator assumes that no job requires more resources than those of the largest server. The implementation of the Client-Side Simulator will be designed the same way as the reference implementation, with the simulation log generating the same results when the '-v brief' option is used.

3.2 Considerations and Constraints

To ensure the successful completion of Stage One, which is to design and implement a simple job dispatcher/scheduler for a Client-Side Simulator, there are several factors that need to be considered in its design and implementation. Firstly, the Client-Side Simulator should adhere to the ds-sim communication protocol, as described in the ds-sim user guide. The Client-Side Simulator should exchange information with the Server-Side Simulator (ds-server) and manage the simulation of jobs in a distributed system. Additionally, the Client-Side Simulator should correctly identify the largest server type based on the number of CPU cores and dispatch jobs to servers in a round-robin fashion. The LRR algorithm in the Client-Side Simulator should be able to successfully allocate equal time slices to each job in a circular order, allowing each job to run for a certain period before moving on to the next job in the queue. Furthermore, the implementation should account for situations where there are multiple server types with the same number of CPU cores and choose the first one listed in the configuration file or ds-system.xml. The design must also consider the potential for scaling up the system in the future and accommodate future modifications or updates to the protocol or system. The implementation of the Client-Side Simulator should be efficient to ensure fast and accurate job scheduling and dispatching. Finally, the design of the Client-Side Simulator should produce a simulation log that matches the output generated by the reference implementation when the 'v-brief' option is used.

3.3 Functionality

To create a successful Client-Side Simulator that functions as a simple job dispatcher/scheduler for Stage One, several functionalities must be implemented.

The following functionalities are:

- Implementing the ds-sim communication protocol: The Client-Side Simulator must implement the ds-sim communication protocol, which specifies how the client and server should communicate with each other. The communication protocol involves several steps, such as sending "HELO" message to the server, which responds with an "OK" message. The client then sends an "AUTH" message with authentication information to the server. The server replies with an "OK" message and a welcome message along with server information. The client then sends a "REDY" message to the server to signal its readiness to receive jobs. The server responds with one of several messages, depending on the state of the system. The client also responds by sending one of several commands, such as "SCHD" to schedule jobs or "GETS" to request server state information. The server responds with corresponding messages, such as "OK" for successful actions or "ERR" for invalid requests. The protocol continues until the client sends a QUIT message to the server, which responds with a QUIT message before terminating.
- Scheduling jobs using the Largest-Round-Robin (LRR): The Client-Side Simulator must schedule jobs using the LRR algorithm and should function continuously by updating the processing time estimates

and selecting the job with the longest expected processing time. If a new job arrives while the LRR algorithm is executing, the LRR algorithm should update the remaining processing time estimates for all jobs in the queue and then select the job with the longest remaining processing time. Additionally, the LRR algorithm should also handle situations where a job completes, a job is preempted, or a job is killed.

- Handling errors and unexpected events: The Client-Side Simulator must have appropriate error-handling mechanisms in place to handle errors or unexpected events, such as server failures or disconnections.
- Interacting with the Server-Side Simulator (ds-server): The Client-Side Simulator must connect to the Server-Side Simulator and receive job requests one at a time, scheduling them to servers of the largest type in a round-robin fashion. It must correctly interpret and respond to the messages sent by the Server-Side Simulator (ds-server).
- Maintaining an accurate simulation log: The Client-Side Simulator must maintain an accurate log of the simulation, which should match exactly the output generated by the reference implementation. Ensuring that the Client-Side-Simulator generates the same simulation log as the reference, using the 'v brief' option, helps to verify the correct operation of the Client-Side-Simulator.

4 Implementation

The system utilises a discrete event simulator to provide stimulation to the system. The Server-Side Simulator was designed and developed prior by the computing department to user-specified configurations. The Server-Side Simulator (ds-server) oversees the overall simulation of the distributed system. It simultaneously works alongside the Client-Side Simulator.

For Stage One, I was required to design and implement my own Client-Side Simulator that works the same way as the reference Client-Side Simulator (ds-client) that can communicate with the Server-Side Simulator (ds-server). To create my Client-Side Simulator, I used Java and three built-in Java libraries, which handle input/output and network communication.

The three built-in Java libraries I used were:

- Java.io: The java.io library was used for the input/output operations, such as reading from and writing to files or streams.
- Java.net: The java.net library was used for networking operations, such as creating and using sockets and sending and receiving data over the network.
- Java.util: The java.util library provided various utility classes, such as data structures, like ArrayList and HashMap, as well as sorting and searching algorithms

Additionally, the development of the Client-Side Simulator used the following techniques and data structures to communicate with the Server-Side Simulator (ds-server)

- Sockets: The client establishes a connection with the server using a Socket object. The Socket class provides a way to establish a two-way communication link between the Client-Side Simulator and the Server-Side Simulator (ds-server). The client socket initiates a connection request to the server socket with the specified IP address (127.0.0.1) and port number (50000) of the ds-server. The server socket receives the connection request and sends a reply indicating that it is ready to establish a connection.
- Input and Output Streams: The Client-Side Simulator uses input/output streams to communicate between the Client-Side-Simulator and the Server-Side-Simulator (ds-server). Additionally, these streams provide a way to read and write different data types, such as integers, strings, and bytes. The input stream is used to read the data from the ds-server, while the output stream is used to send data to the ds-server. These streams allow the client to send and receive messages in real time, which is important for communication between the client and the server. For example, when the client sends a request to the server, it writes the request message to the output stream, which sends the message to the server. The server then reads the message from its input stream and processes it. Similarly, when the server sends a response back to the client, it writes the response message to its output stream, which sends a message to the client. The client then reads the message from its input stream and processes the response.

- BufferedReader: The Client-Side Simulator uses a BufferedReader object to read data from the input stream of the socket. Specifically, it reads data from the input stream and stores it in an internal buffer, making it more efficient to read multiple characters or lines at a time, rather than reading one character at a time.
- Arrays: The Client-Side Simulator uses a variety of arrays to store and process server data. These arrays include server types, server CPU core counts, server memory sizes, server ids, server availability status, last scheduled indexes, and server type counts. When the client first establishes a connection with the server, it receives information about the available servers from the server. It then initialises these arrays with the relevant information about each server, such as its type, number of CPU cores, and memory size. The use of arrays allows the client to quickly access and manipulate information about the available servers. For example, when processing job requests, the client can easily iterate over the server data arrays to find the server that meets the requirements of the job. Additionally, the use of arrays allows for the counting and tracking of servers with a particular type, which is useful when the client needs to schedule a job on a server of a particular type and wants to choose the server with the least number of jobs already.
- String Manipulation: The Client-Side Simulator uses string manipulation techniques to extract and split the server data and job request data. The client uses the split() method of the String class to split a string into an array of strings based on a delimiter.
- Looping: To schedule a job, the Client-Side-Simulator uses for and while loops to iterate over server data and job requests to find jobs that meet the requirements and have the largest CPU core count. If multiple servers have the same CPU core count, the client selects the server with the lowest index. The server's availability status is updated, and the job is scheduled on the selected server.

To design and implement the Client-Side Simulator that acts as a simple job dispatcher/scheduler using the Largest-Round-Robin (LRR), I also followed the reference pseudo code for the Largest Round Robin (LRR). The pseudo-code provided guidance on how to use the GETS command to identify the largest server type.

Overall, the Client-Side Simulator that I have created communicates with the Server-Side Simulator (ds-server) over a network using sockets. It first establishes a connection with the server by creating a new Socket object and specifying the IP address (127.0.0.1) and port number (50000) of the ds-server. After a connection is established, the client sends a series of messages to the server, starting with a "HELO" message and then an "AUTH" message that includes the user's name. The server responds to each message with a message of its own, which the client reads using a DataInPutStream. The client then initialises some data about the servers that the client can use to schedule jobs. The program then enters a loop that reads job requests from the user and schedules them on an available server. The program checks each server to see if it meets the criteria for handling the job, for example, has enough CPU cores and memory, and if so, schedules the job on the largest available server that meets the criteria. If no server is available to handle the job, the program outputs a message saying so. Finally, the program sends a "QUIT" message to the server to end the connection and closes the socket.

References

[1] School of Computing, Macquarie University, "Stage 1: Pseudo code for Largest Round Robin (LRR)," Accessed: Mar. 15, 2023. [Online].

[2] School of Computing, Macquarie University, "Scheduling in Distributed Systems," Accessed: Mar. 15, 2023. [Online].

[3] School of Computing, Macquarie University, "Design and implementation of a client-side simulator with a simple job dispatcher," pp. 1–2, Accessed: Mar. 15, 2023. [Online].

[4] Y. Lee, Y. Kim, and J. King, "ds-sim: an open-source and language-independent distributed systems simulator User Guide," pp. 1–26, Accessed: Mar. 15, 2023. [Online].

GITHUB REPOSITORY: https://github.com/avagardiner/COMP3100Project