# COMP3100 - Stage 1

Benjamin Sumners - 45878803

April 2022

## Contents

# 1 Introduction

This section of the document will provide the reader with a high-level overview of this projects details, where the code to test the job-dispatching can be found and the overall outcome of the project.

## 1.1 Purpose and Goal

The purpose of this document is to provide a detail understanding of the client-server ds-sim relationship and accurately describe how to successfully accomplish a job scheduling task, in line with the assignment specifications. This project has two main goals which will be explored further within this document:

- Design a plain/vanilla version of client-side simulator,
- Implement a plain/vanilla version of client-side simulator.

Designing and Implementing this client-side simulator must achieve the goal of scheduling a job to whichever server has the largest number of CPU cores.

## 1.2 ds-sim Overview

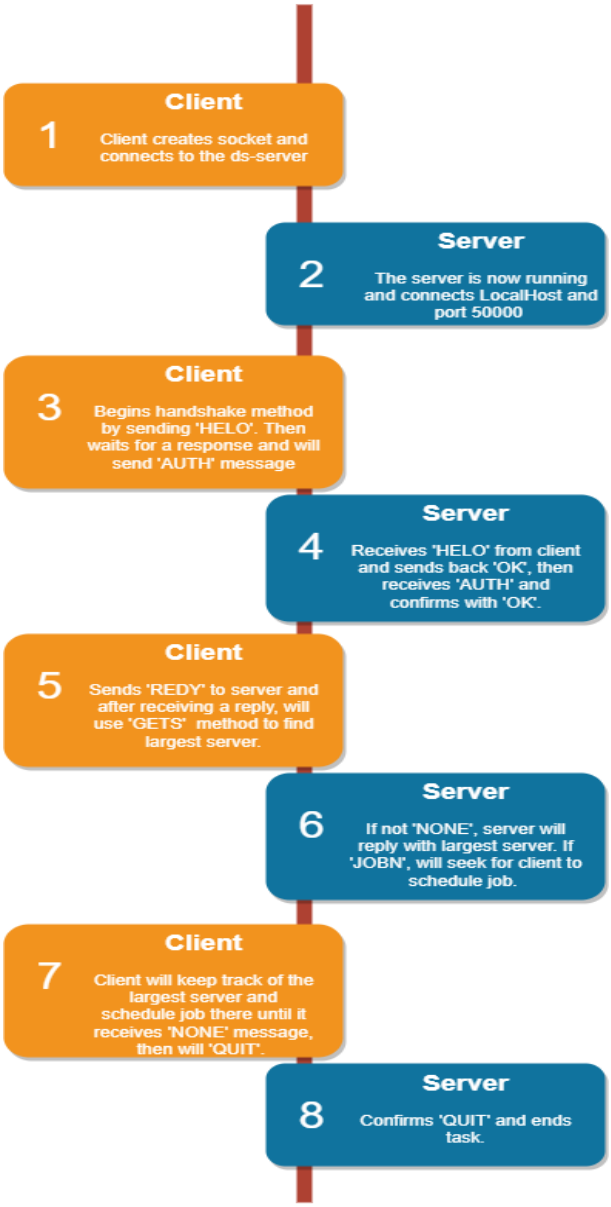
Before continuing with this document, it is important to understand what a ds-sim is, how it's used and why it's a critical element in client/server communication. The majority of this information can be found in the User Guide, developed by Young Choon Lee. Lee describes ds-sim as an "open source, language-independent and configurable distributed systems simulator" [3]. The term 'open-source' means this simulator is open to the public to see, modify and distribute code. Language-independent, which means programmers are able to utilise their preferred OS/programming language without major disruptions to the program [2]. Finally, the meaning behind "distributed systems simulator" can refer to the connection of multiple processes on multiple machines performing a single action[1]. The efficient factor of all three of those components allows for a smooth connection between a server-side simulator, receive jobs and schedules them. This project will focus on the Largest Round Robing scheduling algorithm, which will be discuss later in this document.

## 1.3 Project Location

The project has been developed and run using Ubuntu, an open source Linux operating system. See the following GitHub Repository: github.com/BenjaminSum/COMP3100—45878803 for the Client-Side code used to run this project. [5]

## 2 System Overview

The practical coding aspect of this simulation must begin with the 'handshake' technique. This involves back and forth communication from the server and client in order to establish a working connection. See below for a high-level overview of the project:



# 3 Client-side Design

Design philosophy Considerations and constraints Complex structures within servers sometime cause confusion for the client, as well as server functions. If the client code was too complex, and vise versa, the likely hood of running into errors greatly increases. This is why it's important that this implementation of the code must understand its own functions in a clear manner. Following a basic and logical steps, similar to the flowchart above, will allow for the Largest Round Robin (LRR) algorithm to run smoothly [4].

# 4 Implementation

Implementation of the client-side code involved four main steps:

- Handshake: Establish a connection between the client and server (HELO - OK - AUTH username - OK - REDY - GETS ALL...). This is the most essential part of the design as without it, the program cannot schedule jobs.
- Get Server Information: Using GETS ALL from the client will show all the server information. This is important as we need to determine the largest server to assign all jobs to. The client will then send 'OK' to acknowledge it received the information.
- Find largest server and keep track of it: Store in an ArrayList all of the servers information (server name, server ID, cores, etc), then loop over the ArrayList to determine the largest core value and store this value in a separate list.
- Assign Jobs: Now that we know which server is the largest, until the server sends 'NONE', continuously use 'SCHD' the job, JOBN, to this server until there are no more jobs, then submit 'QUIT'.

# 5 References

## References


[1] Jakob Engblom Daniel Aarno. Distributed simulation, 2015.

[2] LanguageOrientedProgramming. Language independent programming.

[3] Young Choon Lee. ds-sim: an open-source and language-independent distributed systems simulator, 2021.

[4] Uni of Minnesota Duluth. Client-server thinking.

[5] Benjamin Sumners. Comp3100—45878803, 2022.