Project report: Stage 1
COMP3100 Distributed Systems, S2, 2022
SID: 45898405, Name: Benjamin_ van_de_Vorstenbosch

1 Introduction

Given a discrete-event simulator ds-sim [2], a corresponding client needs to be designed and implemented that can communicate with ds-sim in a client-server model. The role of the client is to schedule and dispatch jobs provided to it by the server using various scheduling algorithms.

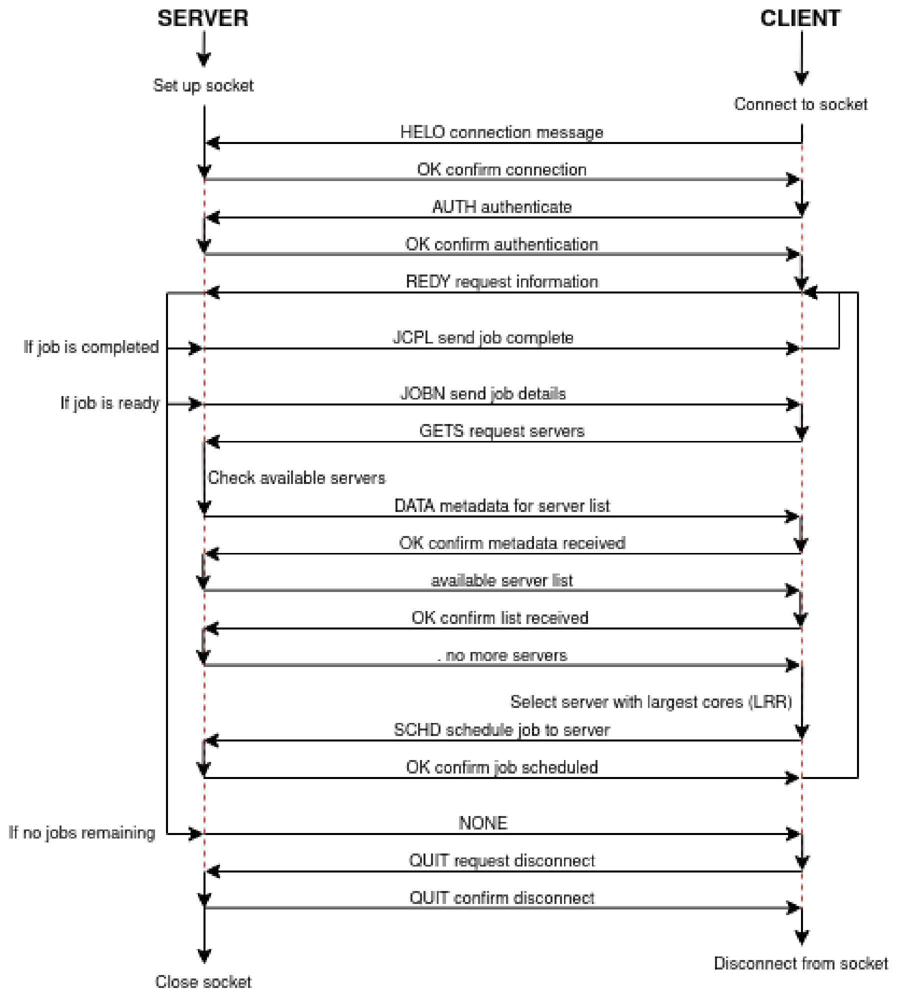
The aim of this report is to describe and explain my implementation of the client, as well how the ds-sim protocol is used for communication. It will provide a system overview, design description, and implementation of the design in my client. The client project can be found on Github [4].

2 System Overview

In this system the ds-server simulates job submissions and job executions by servers, whilst the client handles the job scheduling logic. First ds-server needs to open a socket for the client to connect to (defaulting to localhost and port 50000), and then the client must connect and authenticate before any scheduling of jobs can begin.

For stage 1 of this project, the client must implement the Largest Round-Robin (LRR) scheduling algorithm. After receiving a job, the client must decide which server it needs to be scheduled to. In LRR, the server chosen is the one with the most CPU cores total. If there are multiple equal servers, then the client needs to pick the first one that is provided.

The flow of communication between the two whilst using LRR is outlined in the flowchart below.



3 Design

3.1 Design Philosophy

This project is designed to successfully connect to the ds-sim server and schedule jobs according to the requested scheduling algorithm. To make this project meet these requirements and remain maintainable, it must follow the SOLID design principles [3]. Each principle is outlined below, with a brief description of what they entail and why it is necessary.

Single Responsibility Principle - Every module in this project must be designed for a single piece of functionality, and all modules should be relevant to the class they are in. This properly separates out logic in the program, making it more scale-able and easier to maintain over time.

Open-Closed Principle - This project should be easily added to and scaled up, without needing to modify the existing code. This results in the project always running as expected, even as more functionality is added over time.

Liskov Substitution Principle Similar to the previous principle, the Liskov Substitution Principle requires any derived classes to maintain the expected base functionality of the program. This enables the project to always runs as expected.

Interface Segregation Principle - Interfaces should have specific purposes, rather than having large all-purpose interfaces. This assists with separation of logic, similar to the Single Responsibility Principle.

Dependency Inversion Principle - Lower level modules should have a level of abstraction for higher level modules to use. This will result in fewer mistakes made during development, and consistent behaviour between higher level modules.

In addition to these principles, the code must be modular, understandable, and easy to modify. This requires commenting, appropriate variable and class names, and proper separation of logic.

3.2 Considerations and Constraints

Since DS-Sim has only been confirmed to work on Ubuntu, the client must also be run on Ubuntu. A new compilation of ds-sim must be provided to work on other systems, however this client won't be tested on them and therefore may not work since the project specifications only require it to function on the one system.

There are also some limitations around ports that this client is able to connect to. Valid port numbers range from 0 to 65535, however ports 0-1023 are reserved system ports. Therefore the client cannot be allowed to connect to them.

Finally, missing resources for scheduling or failure of jobs usually needs to be considered. However this project is designed with the assumption that, provided the client schedules a job correctly, neither of these issues will appear (as mentioned in the project specifications).

3.3 Functionalities of Simulator Components

The client must be able to handle various pieces of functionality to run successfully. This includes:

- Connecting to different hosts and ports
- Communicating with ds-server by both sending and receiving messages.
- Interpreting incoming messages and sending appropriate responses as outlined by the ds-sim protocol.
- Authentication with the ds-server
- Handling of different servers and jobs so that it can schedule them appropriately
- Implementation of the LRR algorithm (Stage 1). For future stages, more algorithms need to be implemented.

4 Implementation

The implementation of this program prioritises scale-ability and maintenance. This meant comments, logic separated out into classes, and appropriate data structures or design patterns used.

As a result, there are 7 different classes in this program.

Class	**Description**
Client	The main class of the program that controls the overall flow of the system.
Connection	An abstract class for communication with the ds-sim server from anywhere in the program
Algorithm	An interface for all algorithms
AlgorithmFactory	Where algorithms are requested and created from
LRRAlgorithm	A class providing the functionality of the LRR algorithm job scheduling
Server	Stores the base and current details of servers, based on the data provided by the ds-server
Job	Stores the base and current details of jobs, based on the data provided by the ds-server

4.1 Arguments

To connect to the ds-sim server the correct host and ports needs to be used. To make use of the ds-sim server, the client also needs to be able to run an algorithm to handle the job scheduling. Therefore the client has required arguments to determine these variables, provided in the order ‘hostname‘ ‘port‘ ‘algorithm‘. If any of these are missing then the client won’t run. However if the algorithm doesn’t match to any in the program then it defaults to LRR, since this was the first (and only) implemented algorithm. The suggested host is localhost and the suggested port is 50000, since these are the defaults for ds-sim.

4.2 Connection and Communication

There is a single class in charge of all connection and communication between the client and the server, called Connection. This class was made static so that any part of the program could use a unified method for communication, guaranteeing that they use the same socket and message format (a byte array ending in a newline). The Socket comes from the java.net library, and messages sent or received use the DataOutputStream and BufferedReader classes from the java.io library.

Once the initial arguments have been verified they are given to Connection. Connection then proceeds to connect to the socket and initialise communication. However this is *not* where authentication is done, since that is logic determined by the main class. Only after the connection has been initialised will the Client authenticate the connection by sending the HELO and AUTH commands to ds-server.

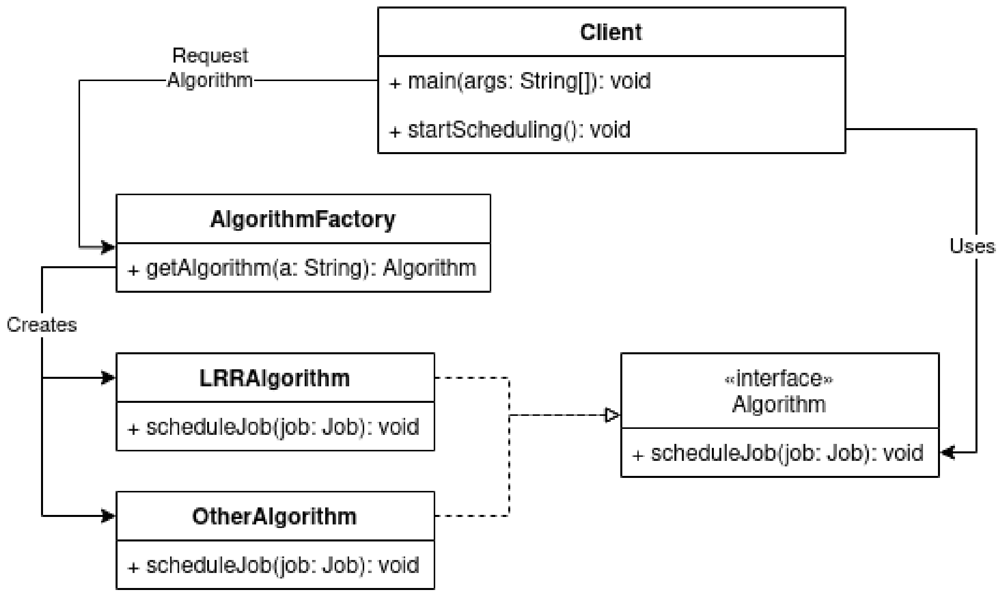
At the end of the program, once all communication with the server has been completed (i.e. the server has sent ”NONE” indicating there are no more jobs to be received), the client asks Connection to gracefully close the connection by sending a ”QUIT” message to the server. Then, the program ends.

4.3 Algorithms and Scheduling

As mentioned, the client must be able to select an algorithm based on the corresponding value provided in the args. Currently the only valid algorithm names are 'lrr' or 'LRR' - case has been ignored here. Any other algorithms requested will default to LRR.

Algorithms are created as classes designed with the Factory pattern. The client uses the abstract class AlgorithmFactory to generate the algorithm required, hidden behind the Algorithm interface. This makes the code incredibly easy to modify without adjusting existing code. When a new algorithm is added, only two things need to be done. First, a new class implementing Algorithm needs to be created and the scheduleJob function designed. Secondly, the algorithm needs to be added to the AlgorithmFactory so that it can actually be used.

The diagram below demonstrates how the Factory pattern was implemented. Note that "OtherAlgorithm" is not an existing algorithm class, and is simply here for demonstration purposes.


Client
+ main(args: String[]): void
+ startScheduling(): void
Request Algorithm
AlgorithmFactory
+ getAlgorithm(a: String): Algorithm
Uses
Creates
LRRAlgorithm
+ scheduleJob(job: Job): void
«interface» Algorithm
+ scheduleJob(job: Job): void
OtherAlgorithm
+ scheduleJob(job: Job): void

After the connection with the server has been established and authenticated, the client starts taking data from the server and scheduling jobs. At the current stage, the only messages the server will send are "JOBN" (send a job), "JCPL" (complete a job), or "NONE" (no jobs left). If "JOBN" is called, a new Job will be created using the data provided by the ds-server and it will be sent as a parameter into the Algorithm's scheduleJob function. Then the client requests more jobs by sending "REDY". If "JCPL" is sent, the client tells the stored server holding the job that it's complete so that it can be marked complete. Then the client sends "REDY", requesting data again. If "NONE" is sent, the server has no more jobs to schedule.

4.4 Largest Round Robin Algorithm

Each Algorithm takes in a Job that needs completing. In the LRR Algorithm, a request is sent to ds-server to return all servers that are capable of handling the job. A list of servers will be returned, and these are stored in the client's HashMap of types String (key) and Server (value). If the servers already exist in the hashmap, they will have variables such as "currentMemory" updated so that both base and current values can be accessed.

Finally, the largest server needs to be chosen. It is simply picked based on which of the available servers has the the largest number of CPU cores, and picking the first one if there are multiple available. However, servers are not simply identified by a name but also have an id (e.g. Server 0, Server 1, AnotherServer 0, AnotherServer 2). Once largest server identified it picks the server id that has had the least number of jobs assigned to it.

Finally, the algorithm schedules the job by sending a "SCHD" command through Connection.

References

[1] DoFactory. “C# factory method.” (), [Online]. Available: https://www.dofactory.com/net/factory-method-design-pattern.

[2] Y. C. Lee. “Ds-sim github repo.” (Mar. 9, 2022), [Online]. Available: https://github.com/distsys-MQ/ds-sim.

[3] S. Oloruntoba. “Solid: The first 5 principles of object oriented design.” (Dec. 1, 2021), [Online]. Available: https://www.digitalocean.com/community/conceptual_articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design.

[4] B. van de Vorstenbosch. “Project github repo.” (), [Online]. Available: https://github.com/BVengo/comp3100Project.