Project report: Stage 1
COMP3100 Distributed Systems, S1, 2022
SID: 46133194, Name: Man Chun Kwok

1 Introduction

This report is to introduce the COMP3100 Assignment Stage 1 project. Stage 1 is design and implement a client simulator that can schedule the jobs by different server type to server. The simulator can connect to server, receive the jobs and schedule the jobs [1].

To schedule the jobs in this stage, it should use Largest-Round-Robin (LRR) to identify the largest server type should be used in schedule. The largest server type is defined to be the one with the **largest number of cores** and the other information is useless for identify. If there are more than or equal to two cores, we apply the first one, and then the second one.

For example (in LRR):

job 0 to largest 0, job 1 to largest 1, job 2 to largest 2, job 3 to largest 0, job 4 to largest 1, ...

In this report, I aim to introduce how to implement the vanilla version of client-side simulator. The rest of this report is organised as follows. Section 2 gives an overview of client simulator. Section 3 gives the design philosophy of client simulator. Section 4 gives the implementation of client simulator.

2 System Overview

In this Section, I will explain the communication between client and server. For the following description, I will use C for Client and S for Server and use Client view to describe. There is also a diagram (figure 1) describing the overview.

1. C sends HELO to S
2. C receives OK from S
3. C sends AUTH with username to S
4. C receives OK from S
5. C sends REDY to S
6. C receives one of the following:
    - JOBN: when receives JOBN, go to step 7
    - JCPL: when receives JCPL, go to step 5
    - NONE: when receives NONE, go to step 16
7. C sends GETS Capable with core, memory and disk to S
8. C receives DATA from S
9. C sends OK to S
10. C receives a list of server types from S
11. C sends OK to S
12. C receives a '.' from S
13. C sends SCHD with job id, server type and cores id to S
14. C receives OK from S
15. Go to Step 5
16. C sends QUIT to S
17. C receives QUIT from S and exit

3 Design

In this section, I will describe the design philosophy, functionalities and considerations in the client simulator.

3.1 Design Philosophy & Functionalities

The Single Responsibility Principle is the most important concept in Java programs. In the code [2], each class has only one responsibility. They are **Client**, **TCPService**, **JOBNcmd**, **ServerType** and **Record** class.

- Client Class
  - The responsibility of the **Client** class is to communicate with the server and schedule the jobs. Client need to send specific messages to communicate with the server. Therefore, the responsibility of the client only needs to modify the messages and communicate with the server according to the process (figure 1).
- TCPService Class
  - The responsibility of the **TCPService** class is to complete the **Transmission Control Protocol** (**TCP**). TCPService is responsible for establishing a connection to the server, providing an interface for communication and closing the connection.
- Data Structure Class
  - The responsibility of the data structure classes are to record the data. When receive the data from the server, data structure classes should record the data in the memory. **JOBNcmd** class record the important data from JOBN request. **ServerType** class record the list of server types data. **Record** class record the distinct server type.

3.2 Considerations

To identify the largest server type, the list should be received from the server and recorded in an **ArrayList**. However, there are different types of server which cannot be recorded in an **ArrayList**, but in multiple **ArrayLists**. That will be easy to record the server type and the number of server.

For example:

Server Type 1	Server Type 2	Server Type 3	Server Type 4	...
small 0 (2 core)	medium 0 (4 core)	large 0 (8 core)	super large 0 (8 core)	
small 1 (2 core)	medium 1 (4 core)	large 1 (8 core)	super large 1 (8 core)	
small 2 (2 core)	medium 2 (4 core)		super large 2 (8 core)	
	medium 3 (4 core)			

Table 1: Server type record in multiple list.

According to table 1, record the server types in multiple list can be easier than a list to identify the largest server type. It can be easy to identify how many servers in a server type and easy to filter the largest server type in cores. To improve the table 1, **Record** class to be used to optimizing it.

For example:

Server Type	Core	number of server	next order number
small	2	3	0
medium	4	4	0
large	8	2	0
super large	8	3	0

Table 2: Server type record in multiple list.

Therefore, the single list can be improved by two steps.

$$SingleList \rightarrow MultipleList \rightarrow RecordList$$

In conclusion, to identify the first largest server type, get the first maximum cores server type record from the Record List.

4 Implementation

There are many functions and a few classes in the code [2]. Although there are different functions and classes, each function and class in the code has their own job. Combining different parts of function can complete the scheduled jobs and executed sequentially. Therefore, the client can complete a large number of jobs in single command. In this section, I will explain the implementation of client in coding language, classes logic and algorithm.

4.1 Coding Language

The Code is written in **Java language** and **JDK version is 13**

4.2 Classes Logic

The **Client Class** is responsible for the main process work. There are **five** main process need to complete :

1. Welcome Message (HELO and AUTH)

```
Send HELO
Receive OK
Send Auth and username
Receive OK
```

2. Get a job from server (REDY)

```
Send REDY
Receive JOBN command
```

3. Get the Capable of a list of server type (GETS)

```
Send GETS command
Receive DATA
Send OK // This OK is ready for receive all server types
While the last message from server is not '.' do
    record all server type
Send OK
Receive '.'
```

4. Scheduled the job (SCHD)

```
Identify the largest server type
Send SCHD
While the last message from server is not 'OK' do
    Continue looping
```

5. Close the Client when finished the jobs (QUIT)

```
Send QUIT
Receive QUIT
Close socket and exit program
```

The **TCPService** is responsible for the network process work. There are **four** main process need to complete :

1. Connect to server by using **Socket**

```
Use Socket to connect to an ip and port
```

2. Read data from server by using **BufferReader**

```
Use BufferReader to read the data
Transform the data to string
```

3. Send data to server by using **DataOutputStream**

```
Transform the data to bytes
Use DataOutputStream to send the data
```

4. Close the connection

```
Close BufferReader
Close DataOutputStream
Close Socket
```

The following show the brief of pseudo code for use every function in Client.

```
Call function welcome
While the last message from server is not 'NONE' do
    If JOBN then
        Call function getCapable
        Call function scheduleJob by using function getServerList
    End If
Call function close
```

The **JOBNcmd** is a data structure. It include :

submit time	job id	estRuntime
cores	memory size	disk size

The **ServerType** is a data structure. It include :

server type	system id	active state
boot up time	cores	memory size
disk size	scheduled jobs	jobs running

The **Record** is a data structure. It include :

server type	cores	loop counting	maximum counting

4.3 Algorithm

The main logic of the program is to send and receive. The algorithm is mainly used to identify the largest of server type.

4.3.1 Get Server Type List

To get the server type list, each data line should record in a **ServerType** class (**st**) and then store in an **ArrayList** (**serverTypes**).

If receive a new **st** is different server type to **serverTypes**

- **record** all the information about **serverTypes** into an **ArrayList** (**records**)
- **record** the server type has been received in an **ArrayList** (**allServerTypes**)

4.3.2 Schedule A Job

To schedule a job, it should identify the **largest server type** and use **Largest-Round-Robin** (LRR) to sends each job to server.

To Identify the **largest server type**, it can use Java Stream to filter the **first largest server type**.

- Filter the server type in **records** only exists in **allServerTypes** and store in a new **ArrayList**.
- Get the first maximum cores server type record from the new **ArrayList**.

The first maximum cores server type record is the **largest server type**

References

[1] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center," in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, (USA), p. 295–308, USENIX Association, 2011.

[2] M. C. Kwok, "Comp3100 assigment stage 1." https://github.com/8593K/COMP3100, 2022.

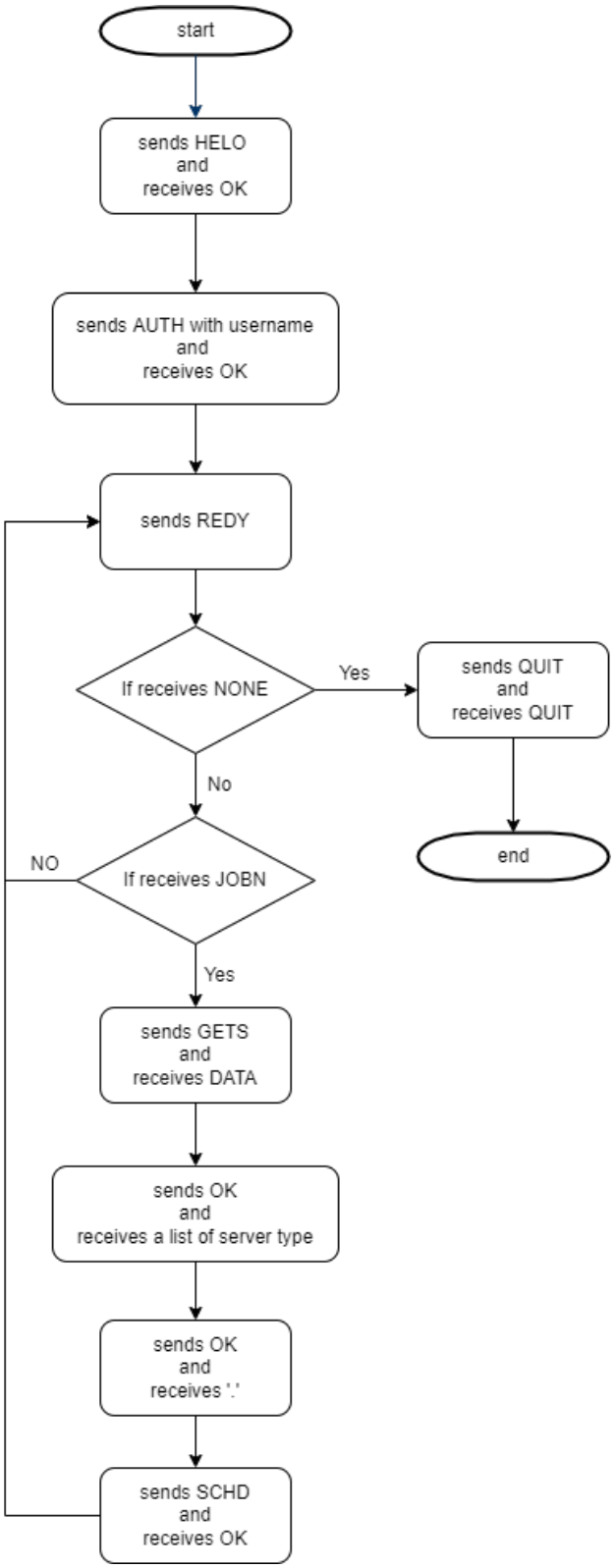

Figure 1: System Overview Steps in Client view.