Project report: Stage 1
COMP3100 Distributed Systems, S2, 2022
SID: 46225854, Name: Kris Paul Babu

Section 1: Introduction

In this report I will go into detail about my solution to stage 2 of the Major Assessment. In stage 2, a connection is created between a simulated DS server and a client server. Which involves sending handshakes, server scouting and job scheduling. The purpose of this is to demonstrate how a real life client-server job scheduling would take place. On top of enabling a handshake, I will also be using my very own job scheduling algorithm and compare its efficieny / baseline performance with various other job scheduling algorithm like best-fit, worst-fit, first-fit, largest-round-robin etc., The goal is to design an algorithm that performs well or at the very least averages the following criterions: turnaround time, resource utilisation, execution costs etc.,

Index

Section 2: Problem definition

The ds-server is a simulated version of a list of servers in a data center. The goal of the job client is to schedule jobs efficiently so that it's use of various resources and turnaround time is minimised. The server and client establish a connection with each other through use of custom commands, and there is a back and forth between both the client and the server with the client requesting information like the list of servers present, jobs to be scheduled etc., To maximise efficiency, there are various forms of scheduling algorithms that could potentially be used, like the best-fit algorithm which assigns jobs to servers based on a minimising function, the servers resource capacity and currently waiting or running jobs. The best-fit is just one of the algorithms that can be used to schedule jobs efficiently, other algorithms that schedule jobs with varying efficiency are:

- First Fit
- Worst Fit
- Largest Round Robin
- Best Fit

For this project I will be implementing an algorithm of my own, the performance of which will be compared to the baseline performance of the above mentioned algorithms. The algorithm is designed in such a way that it performs well enough or just good enough so that the factors like average turnaround time, resource utilisation and cost of use are all minimised.

When running the above scheduling algorithm it was discovered that the best-fit algorithm seemed to perform just good enough in the required criterions. So naturally, I designed my algorithm to operate / look more or less like the best-fit algorithm, with a different approach in how the calling / requesting of information was handled.

Specifically, the variations are clear when looking at how server informations were requested. In the best-fit algorithm, server information was requested periodically using the **GETS Capable** command. And if a server had running or working job, the **LSTJ** command was used to query the server for the jobs that it was currently busy with. This information is then used to conditionally select the best possible server present in the list of

servers available.

In contrast, in my own algorithm, I used the **GETS All** command, which returned the entire list of servers present in a given config file. After this, The GETS Avail or GETS Capable commands are used periodically to query the server for servers that were either available or capable of running the given jobs.

Section 3: Algorithm description

Below is an example usage log of my algorithm with the config file present in the ds-sim package called **ds-sample-config01.xml**.

```
    # ds-sim server 28-Feb, 2023 @ MQ - client-server
# Server-side simulator started with './ds-server -c /media/shared/majorAssessmentStage2/ds-sim/configs/sample-configs/ds-sample-config01.xml -v all -n'
# Waiting for connection to port 50000 of IP address 127.0.0.1
RCVD HELO
SENT OK
RCVD AUTH kris
# Welcome  kris!
# The system information can be read from 'ds-system.xml'
SENT OK
RCVD REDY
SENT JOBN 37 0 653 3 700 3800
RCVD GETS All
SENT DATA 5 124
RCVD OK
SENT juju 0 inactive -1 2 4000 16000 0 0
juju 1 inactive -1 2 4000 16000 0 0
joon 0 inactive -1 4 16000 64000 0 0
joon 1 inactive -1 4 16000 64000 0 0
super-silk 0 inactive -1 16 64000 512000 0 0
RCVD OK
SENT .
RCVD REDY
SENT JOBN 37 0 653 3 700 3800
RCVD GETS Avail 3 700 3800
SENT DATA 3 124
RCVD OK
SENT joon 0 inactive -1 4 16000 64000 0 0
joon 1 inactive -1 4 16000 64000 0 0
super-silk 0 inactive -1 16 64000 512000 0 0
RCVD OK
SENT .
RCVD SCHD 0 joon 0
t:         37 job     0 (waiting) on # 0 of server joon (booting) SCHEDULED
SENT OK
RCVD REDY
SENT JOBN 60 1 2025 2 1500 2900
RCVD GETS Avail 2 1500 2900
SENT DATA 4 124
RCVD OK
SENT juju 0 inactive -1 2 4000 16000 0 0
juju 1 inactive -1 2 4000 16000 0 0
joon 1 inactive -1 4 16000 64000 0 0
super-silk 0 inactive -1 16 64000 512000 0 0
RCVD OK
SENT .
RCVD SCHD 1 juju 0
t:         60 job     1 (waiting) on # 0 of server juju (booting) SCHEDULED
SENT OK
RCVD REDY
SENT JOBN 96 2 343 2 1500 2100
RCVD GETS Avail 2 1500 2100
SENT DATA 3 124
RCVD OK
SENT juju 1 inactive -1 2 4000 16000 0 0
joon 1 inactive -1 4 16000 64000 0 0
super-silk 0 inactive -1 16 64000 512000 0 0
RCVD OK
SENT .
RCVD SCHD 2 juju 1
t:         96 job     2 (waiting) on # 1 of server juju (booting) SCHEDULED
SENT OK
RCVD REDY
t:         97 job     0 on # 0 of server joon RUNNING
SENT JOBN 101 3 380 2 900 2500
RCVD GETS Avail 2 900 2500
SENT DATA 2 124
RCVD OK
SENT joon 1 inactive -1 4 16000 64000 0 0
super-silk 0 inactive -1 16 64000 512000 0 0
RCVD OK
SENT .
RCVD SCHD 3 joon 1
t:        101 job     3 (waiting) on # 1 of server joon (booting) SCHEDULED
```

```
SENT OK
RCVD REDY
t:        120 job     1 on # 0 of server juju RUNNING
SENT JOBN 137 4 111 1 100 2000
RCVD GETS Avail 1 100 2000
SENT DATA 2 124
RCVD OK
SENT joon 0 active 97 1 15300 60200 0 1
super-silk 0 inactive -1 16 64000 512000 0 0
RCVD OK
SENT .
RCVD SCHD 4 joon 0
t:        137 job     4 (running) on # 0 of server joon (active) SCHEDULED
t:        137 job     4 on # 0 of server joon RUNNING
SENT OK
RCVD REDY
t:        156 job     2 on # 1 of server juju RUNNING
SENT JOBN 156 5 8 3 2700 2600
RCVD GETS Avail 3 2700 2600
SENT DATA 1 124
RCVD OK
SENT super-silk 0 inactive -1 16 64000 512000 0 0
RCVD OK
SENT .
RCVD SCHD 5 super-silk 0
t:        156 job     5 (waiting) on # 0 of server super-silk (booting) SCHEDULED
SENT OK
RCVD REDY
t:        161 job     3 on # 1 of server joon RUNNING
SENT JOBN 198 6 1074 4 4000 7600
RCVD GETS Avail 4 4000 7600
SENT DATA 0 124
RCVD OK
SENT .
RCVD GETS Capable 4 4000 7600
SENT DATA 3 124
RCVD OK
SENT joon 0 active 97 0 15200 58200 0 2
joon 1 active 161 2 15100 61500 0 1
super-silk 0 booting 236 13 61300 509400 1 0
RCVD OK
SENT .
RCVD SCHD 6 joon 0
t:        198 job     6 (waiting) on # 0 of server joon (active) SCHEDULED
SENT OK
RCVD REDY
SENT JOBN 225 7 442 2 500 2100
RCVD GETS Avail 2 500 2100
SENT DATA 1 124
RCVD OK
SENT joon 1 active 161 2 15100 61500 0 1
RCVD OK
SENT .
RCVD SCHD 7 joon 1
t:        225 job     7 (running) on # 1 of server joon (active) SCHEDULED
t:        225 job     7 on # 1 of server joon RUNNING
SENT OK
RCVD REDY
t:        236 job     5 on # 0 of server super-silk RUNNING
SENT JOBN 249 8 926 1 100 800
RCVD GETS Avail 1 100 800
SENT DATA 1 124
RCVD OK
SENT super-silk 0 active 236 13 61300 509400 0 1
RCVD OK
SENT .
RCVD SCHD 8 super-silk 0
t:        249 job     8 (running) on # 0 of server super-silk (active) SCHEDULED
t:        249 job     8 on # 0 of server super-silk RUNNING
SENT OK
RCVD REDY
t:        257 job     5 on # 0 of server super-silk COMPLETED
SENT JCPL 257 5 super-silk 0
RCVD REDY
t:        303 job     4 on # 0 of server joon COMPLETED
SENT JCPL 303 4 joon 0
RCVD REDY
SENT JOBN 308 9 2010 2 600 1500
RCVD GETS Avail 2 600 1500
SENT DATA 1 124
RCVD OK
SENT super-silk 0 active 236 15 63900 511200 0 1
RCVD OK
SENT .
RCVD SCHD 9 super-silk 0
t:        308 job     9 (running) on # 0 of server super-silk (active) SCHEDULED
t:        308 job     9 on # 0 of server super-silk RUNNING
SENT OK
RCVD REDY
t:        575 job     7 on # 1 of server joon COMPLETED
SENT JCPL 575 7 joon 1
RCVD REDY
```

```
t:        642 job     2 on # 1 of server juju COMPLETED
SENT JCPL 642 2 juju 1
RCVD REDY
t:       1073 job     8 on # 0 of server super-silk COMPLETED
SENT JCPL 1073 8 super-silk 0
RCVD REDY
t:       1215 job     1 on # 0 of server juju COMPLETED
SENT JCPL 1215 1 juju 0
RCVD REDY
t:       1232 job     3 on # 1 of server joon COMPLETED
SENT JCPL 1232 3 joon 1
RCVD REDY
t:       1337 job     0 on # 0 of server joon COMPLETED
t:       1337 job     6 on # 0 of server joon RUNNING
SENT JCPL 1337 0 joon 0
RCVD REDY
t:       1778 job     9 on # 0 of server super-silk COMPLETED
SENT JCPL 1778 9 super-silk 0
RCVD REDY
t:       1897 job     6 on # 0 of server joon COMPLETED
SENT JCPL 1897 6 joon 0
RCVD REDY
SENT NONE
RCVD QUIT
SENT QUIT
# -------------------------------------------------------------------------------------
# 2 juju servers used with a utilisation of 100.00 at the cost of $0.09
# 2 joon servers used with a utilisation of 100.00 at the cost of $0.32
# 1 super-silk servers used with a utilisation of 100.00 at the cost of $0.34
# ==================================== [ Summary ] ====================================
# actual simulation end time: 1897, #jobs: 10 (failed 0 times)
# total #servers used: 5, avg util: 100.00% (ef. usage: 100.00%), total cost: $0.75
```

Below is the configuration file used in the above log, **ds-sample-config01.xml**:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- generated by: Y. C. Lee -->
<config randomSeed="1024">
  <servers>
    <server type="juju" limit="2" bootupTime="60" hourlyRate="0.2" cores="2" memory="4000" disk="16000" />
    <server type="joon" limit="2" bootupTime="60" hourlyRate="0.4" cores="4" memory="16000" disk="64000" />
    <server type="super-silk" limit="1" bootupTime="80" hourlyRate="0.8" cores="16" memory="64000" disk="512000" />
  </servers>
  <jobs>
    <job type="short" minRunTime="1" maxRunTime="300" populationRate="60" />
    <job type="medium" minRunTime="301" maxRunTime="1800" populationRate="30" />
    <job type="long" minRunTime="1801" maxRunTime="100000" populationRate="10" />
  </jobs>
  <workload type="moderate" minLoad="30" maxLoad="70" />
  <termination>
    <condition type="endtime" value="604800" />
    <condition type="jobcount" value="10" />
  </termination>
</config>
```

The algorithm starts of with the regular procedure for establishing a connection between the server and the client. After the client has established a connection with the server, the client uses the **GETS All** command (line 12) to retrieve all the servers present in the config. This information is stored to help us find the initial core count when using GETS Capable.

After the client has run the query to ask for the entire list of servers, the client then gets ready for the first job in the global queue. The client queries the server for a list of servers that can run the given job and are currently available and has enough resource capacity to handle the job using the GETS Avail command (line 24). Since this is the first job, all the servers are available at the moment so the client finds the best fit server (the one with a core count less than the other servers and at the same time has a core count enough to handle the job) and assigns that server the job.

When using the gets avail, it's possible that all servers that can run the given job are busy. In such a case, the GETS Capable command is used, this commmand returns a list of servers that can eventually run the given job. The issue with using GETS Capable is that, it doesn't return the initial core count of the server, this could lead to the servers returned by GETS Capable not having enough cores to run the job. This where we use the server list that we obtained using the GETS All command. We replace the core count with the initial core count using the list of servers. This enables the algorithm to identify what server to assign the job to (the server that is best fit).

Section 4: Implementation

```
import java.io.*;
import java.net.*;

public class MyClient {

    public static void main(String[] args) {
        try {

            Socket socket = new Socket("localhost", 50000);
            DataOutputStream dout = new DataOutputStream(socket.getOutputStream());
            BufferedReader dis = new BufferedReader(new InputStreamReader(socket.getInputStream()))
                ;

            dout.write(("HELO\n").getBytes());
            dis.readLine();

            String username = System.getProperty("user.name");
            dout.write(("AUTH " + username + "\n").getBytes());
            dis.readLine();

            dout.write(("REDY\n").getBytes());
            dis.readLine();
```

The above piece of code represents how the client establishes a connection with the server, using standard ds-sim commands like HELO, AUTH and REDY.

```
            String allData = "";
            dout.write(("GETS All\n").getBytes());
            allData = dis.readLine();
            dout.write(("OK\n").getBytes());

            String[] allServers = new String[Integer.parseInt(allData.split(" ")[1])];
            for (int i = 0; i < allServers.length; i++) {
                allServers[i] = dis.readLine();
            }

            dout.write(("OK\n").getBytes());
            dis.readLine();
```

In the above code, the client uses the GETS All command to retrieve all the servers present. This will help us later when we need to find the initial core count of a server. After this, we use a whie loop to start scheduling jobs to the server.

```

            while (true) {

                dout.write(("REDY\n").getBytes());
                String jobs = (String) dis.readLine();

                if (jobs.equals("NONE")) {
                    break;
                }

                String[] jobsIndivData = jobs.split(" ");
                int jobId = Integer.parseInt(jobsIndivData[2]);
                int neededCores = Integer.parseInt(jobsIndivData[4]);

                String data = "";
                if (!jobsIndivData[0].equals("JCPL")) {
                    dout.write(
                            ("GETS Avail " + jobsIndivData[4] + " " + jobsIndivData[5] + " " +
                                jobsIndivData[6] + "\n").getBytes());
                    data = dis.readLine();
                    dout.write(("OK\n").getBytes());
                }

                else {
                    dout.write(("REDY\n").getBytes());
                    dis.readLine();
                    continue;
                }
```

In the above code, we start an inifinite while loop with a condition within the while loop that says if the variable **jobs** is equal to **NONE**, the loop breaks. This ensures that if there are no more jobs then the client will eventually send the **QUIT** command. Or if the jobs array does not contain **JCPL** the GETS Avail command is used. If the jobs array does contain JCPL, the continue keyword is used, which skips the rest of the code in the loop and esentially restarts the loop, moving on to the next set of jobs. When the Gets Avail command is used the server responds with a list of servers that are available and can run a job with the specific resource capacity and core count.

```
                int capableCalled = 0;

                if (data.split(" ")[1].equals("0")) {
                    dis.readLine();
                    dout.write(
                            ("GETS Capable " + jobsIndivData[4] + " " + jobsIndivData[5] + " " +
                                 jobsIndivData[6] + "\n").getBytes());
                    data = dis.readLine();
                    dout.write(("OK\n").getBytes());
                    capableCalled = 1;
                }

                int serversPresentNo = Integer.parseInt(data.split(" ")[1]);

                String[] serverList = new String[serversPresentNo];
                for (int i = 0; i < serversPresentNo; i++) {
                    serverList[i] = dis.readLine();
                }

                if (capableCalled == 1) {
                    for (int i = 0; i < serverList.length; i++) {
                        String server1Name = serverList[i].split(" ")[0];
                        String server1Id = serverList[i].split(" ")[1];
                        for (int j = 0; j < allServers.length; j++) {
                            String server2Name = allServers[j].split(" ")[0];
                            String server2Id = allServers[j].split(" ")[1];

                            if (server1Name.equals(server2Name) && server1Id.equals(server2Id)) {
                                serverList[i] = allServers[j];
                            }
                        }
                    }
                }
```

In the above set of code, we initialise a boolean variable called **capableCalled**. The value of this variable is set to 1 if the command GETS Capable is called, which on the other hand, is only called if GETS Avail does not return any servers. When the client uses GETS Capable, it is known that all the servers are busy, therefore, the core count on the servers returned are not the initial core counts, which is what we want, because it might be possible that the core count for the servers returned from the GETS Capable command are not sufficient.

If capableCalled is set to 1, the client then iterates over the list of servers returned by GETS Capable and replaces it with servers that were returned using GETS Avail. Of course, the names and IDs of the servers are compared before the list of servers returned by GETS Capable is altered.

```
                String bestServerName = "";
                String bestServerId = "";
                int bestServerCores = 0;

                bestServerName = "";
                bestServerId = "";
                bestServerCores = Integer.MAX_VALUE;

                for (int i = 0; i < serversPresentNo; i++) {
                    String currentSserverName = serverList[i].split(" ")[0];
                    String currentServerId = serverList[i].split(" ")[1];
                    int currentServerCores = Integer.parseInt(serverList[i].split(" ")[4]);

                    if (currentServerCores >= neededCores && currentServerCores <= bestServerCores)
                         {
                        bestServerName = currentSserverName;
                        bestServerId = currentServerId;
                        bestServerCores = currentServerCores;
                        break;
                    }
                }
```

The above set of code finds the best possible server present in the list of servers returned by either the GETS Avail or GETS Capable command. The server with the right amount of cores (the server with cores higher or equal to the amount of cores needed for the job and lower than the core count of the rest of the servers returned) is selected. The client uses for loop to iterate over the server to compare each server core count.

```

                dout.write(("OK\n").getBytes());
                dis.readLine();

                if (jobsIndivData[0].equals("JOBN")) {
                    String schedCommand = "SCHD " + jobId + " " + bestServerName + " " +
                         bestServerId;
                    dout.write((schedCommand + "\n").getBytes());
                    dis.readLine();
                }

            }
            dout.write(("QUIT\n").getBytes());
```

```
13              dis.readLine();
14              dout.flush();
15
16              dout.close();
17              dis.close();
18              socket.close();
19          } catch (
20
21          Exception e) {
22              System.out.println(e);
23          }
24      }
25  }
```

And finally, once the most fit server for the given job is found, the client uses the **SCHD** command to assign the given job to said server. And when all the jobs have been completed and the loop is broken, the client uses the QUIT command to exit the simulation.

Section 5: Evaluation

```
Running client with config12-long-med.xml
Running client with config12-med-alt.xml
Running client with config12-med-med.xml
Running client with config12-short-med.xml
Running client with config16-long-high.xml
Running client with config16-long-med.xml
Running client with config16-med-high.xml
Running client with config16-short-high.xml
Running client with config16-short-med.xml
Running client with config40-long-high.xml
Running client with config40-long-med.xml
Running client with config40-med-high.xml
Running client with config40-med-med.xml
Running client with config40-short-high.xml
Running client with config40-short-med.xml

Turnaround time
Config                      |FF       |BF       |FFQ      |BFQ      |WFQ       |Yours
config12-long-med.xml       |2400     |2397     |2802     |2630     |6880      |2407
config12-med-alt.xml        |388      |373      |813      |698      |3804      |367
config12-med-med.xml        |654      |653      |893      |831      |4576      |653
config12-short-med.xml      |61       |60       |106      |100      |677       |60
config16-long-high.xml      |3123     |4674     |6536     |6666     |23972     |2671
config16-long-med.xml       |2548     |2562     |3997     |3778     |19538     |2556
config16-med-high.xml       |3749     |5328     |6123     |5494     |23740     |1408
config16-short-high.xml     |3215     |8260     |5517     |4369     |24814     |991
config16-short-med.xml      |699      |851      |1952     |1972     |17139     |667
config40-long-high.xml      |4506     |4164     |3642     |3568     |8461      |3369
config40-long-med.xml       |3023     |3022     |3330     |3346     |11724     |3026
config40-med-high.xml       |1505     |896      |940      |933      |3097      |906
config40-med-med.xml        |963      |972      |1237     |1232     |9123      |964
config40-short-high.xml     |485      |1365     |373      |301      |2817      |228
config40-short-med.xml      |180      |182      |198      |193      |1945      |180
Average                     |1833.27  |2383.93  |2563.93  |2407.40  |10820.47  |1363.53
Normalised (FF)             |1.0000   |1.3004   |1.3986   |1.3132   |5.9023    |0.7438
Normalised (BF)             |0.7690   |1.0000   |1.0755   |1.0098   |4.5389    |0.5720
Normalised (FFQ)            |0.7150   |0.9298   |1.0000   |0.9389   |4.2203    |0.5318
Normalised (BFQ)            |0.7615   |0.9903   |1.0650   |1.0000   |4.4947    |0.5664
Normalised (WFQ)            |0.1694   |0.2203   |0.2370   |0.2225   |1.0000    |0.1260
Normalised (Average)        |0.4581   |0.5957   |0.6407   |0.6016   |2.7039    |0.3407

Resource utilisation
Config                      |FF       |BF       |FFQ      |BFQ      |WFQ       |Yours
config12-long-med.xml       |69.39    |66.87    |68.24    |66.45    |79.22     |69.76
config12-med-alt.xml        |66.93    |63.88    |66.46    |63.39    |49.55     |66.90
config12-med-med.xml        |66.22    |63.06    |65.92    |62.68    |71.52     |66.29
config12-short-med.xml      |61.56    |58.46    |61.32    |58.18    |61.48     |61.62
config16-long-high.xml      |79.97    |74.70    |77.70    |74.06    |66.17     |79.98
config16-long-med.xml       |68.16    |64.71    |67.59    |63.86    |65.06     |68.35
config16-med-high.xml       |77.45    |73.06    |76.10    |73.14    |48.18     |78.72
config16-short-high.xml     |76.88    |71.21    |74.44    |70.87    |50.94     |78.62
config16-short-med.xml      |66.22    |62.42    |65.54    |62.08    |61.40     |66.18
config40-long-high.xml      |85.81    |80.91    |86.34    |81.81    |79.46     |87.62
config40-long-med.xml       |75.19    |69.43    |74.92    |69.00    |65.26     |75.21
config40-med-high.xml       |83.01    |79.51    |83.87    |79.32    |67.73     |84.45
config40-med-med.xml        |58.92    |54.46    |58.83    |54.53    |45.66     |58.96
config40-short-high.xml     |86.54    |79.64    |86.68    |80.98    |78.09     |87.22
config40-short-med.xml      |77.23    |71.66    |77.18    |71.68    |87.92     |77.27
Average                     |73.30    |68.93    |72.74    |68.80    |65.18     |73.81
Normalised (FF)             |1.0000   |0.9404   |0.9924   |0.9387   |0.8892    |1.0070
Normalised (BF)             |1.0633   |1.0000   |1.0553   |0.9981   |0.9455    |1.0708
Normalised (FFQ)            |1.0077   |0.9476   |1.0000   |0.9458   |0.8960    |1.0147
Normalised (BFQ)            |1.0654   |1.0019   |1.0573   |1.0000   |0.9473    |1.0728
Normalised (WFQ)            |1.1246   |1.0576   |1.1161   |1.0556   |1.0000    |1.1325
Normalised (Average)        |1.0503   |0.9877   |1.0423   |0.9858   |0.9339    |1.0576

Total rental cost
Config                      |FF       |BF       |FFQ      |BFQ      |WFQ       |Yours
config12-long-med.xml       |131.37   |131.75   |136.01   |133.20   |132.04    |132.04
config12-med-alt.xml        |113.25   |113.19   |115.75   |115.21   |118.77    |113.25
config12-med-med.xml        |132.82   |133.01   |134.87   |134.60   |129.86    |132.93
config12-short-med.xml      |21.50    |21.49    |21.68    |21.68    |21.00     |21.50
config16-long-high.xml      |589.16   |596.10   |632.23   |632.18   |653.30    |583.50
config16-long-med.xml       |581.93   |581.49   |600.44   |599.28   |580.79    |582.39
config16-med-high.xml       |598.42   |598.83   |632.22   |617.53   |696.20    |578.73
config16-short-high.xml     |594.10   |612.40   |635.48   |626.12   |674.29    |580.47
config16-short-med.xml      |579.43   |578.62   |592.54   |590.29   |587.31    |579.52
config40-long-high.xml      |1244.09  |1236.91  |1250.35  |1239.58  |1297.00   |1232.80
config40-long-med.xml       |1552.56  |1567.68  |1583.64  |1584.85  |1552.89   |1561.71
config40-med-high.xml       |614.35   |602.11   |605.53   |605.93   |657.50    |600.52
config40-med-med.xml        |1281.93  |1281.04  |1294.66  |1293.52  |1183.21   |1282.44
config40-short-high.xml     |600.37   |612.79   |601.77   |597.96   |655.30    |592.86
config40-short-med.xml      |586.94   |587.55   |592.51   |588.60   |566.19    |587.31
Average                     |614.81   |617.00   |628.65   |625.37   |633.71    |610.80
Normalised (FF)             |1.0000   |1.0036   |1.0225   |1.0172   |1.0307    |0.9935
Normalised (BF)             |0.9965   |1.0000   |1.0189   |1.0136   |1.0271    |0.9900
Normalised (FFQ)            |0.9780   |0.9815   |1.0000   |0.9948   |1.0081    |0.9716
```

```
Normalised (BFQ)          |0.9831    |0.9866    |1.0052    |1.0000    |1.0133    |0.9767
Normalised (WFQ)          |0.9702    |0.9736    |0.9920    |0.9868    |1.0000    |0.9638
Normalised (Average)      |0.9854    |0.9889    |1.0076    |1.0023    |1.0157    |0.9790

Final results:
2.1: 1/1
2.2: 1/1
2.3: 7/7
```

Algorithms	Average Turnaround Time	Average Resource Utilisation	Average Total Rental Cost
FF	1833.27	73.3	614.81
BF	2383.93	68.93	617
FFQ	2563.93	72.74	628.65
BFQ	2407.4	68.8	625.37
WFQ	10820.47	65.18	633.71
Mine	1363.53	73.81	610.8

My algorithm, for the most part has worked well enough. On average it has scored high enough in all the tests. Although there are a few yellow results in the mix, indicating that the results could've been slightly better. The good news is that there are no red results, which would've indicated that the algorithm performed poorly.

Section 6: Conclusion

Well the algorithm has, for the most part, performed well in all tests, but it has not been perfected. Although, achieving perfection in in all three criterions is an impossible and contradicting task. As perfecting one aspect of the algorithm could potentially mean having a shortcoming in another.

Section 7: References

Link to Github: https://github.com/krispaulbabu/majorAssessmentStage2