Project Report: Stage 2
COMP3100 Distributed Systems, S2, 2022
SID: 45956987, Name: Dylan Vongsouvanh

1 Introduction

This report aims to outline the goals of Stage 2 of the project involving ds-simulators. The project (i.e., client source code, report, etc.) can be accessed via GitHub [1]. The goal of Stage 2 is to design and implement a new scheduling algorithm that optimises the average turnaround time when scheduling jobs. The algorithm also aims to optimise the performance of the client such as resource utilisation and server rental cost.

This report will contain the following sections:

- Problem Definition
- Algorithm Description
- Implementation
- Evaluation

The Problem Definition will describe the scheduling problem encountered in this stage and what scheduling algorithm will be implemented including a definition of the objective function and justification of why it is the optimal choice.

The Algorithm Description will contain an example of a simple scheduling scenario with a sample configuration, schedule, description and discussion to visualise the algorithm.

Implementation will contain details of what technologies, techniques, data structures, and software libraries will be used.

Evaluation will contain the simulation with test cases/configurations, results, and comparison to the four baseline algorithms. This section will also outline the pros and cons of the implemented algorithm.

2 Problem Definition

The scheduling problem presented in this stage is that by optimising one factor (i.e., turnaround time). Turnaround time is defined as the time taken to execute (end time - start time) a job plus the waiting time. While the algorithm attempts to optimise turnaround time other factors may experience a sacrifice in performance (i.e., higher server rental cost). An example is while attempting to lower the average turnaround time, more resources are being used to guarantee the lower times. This reveals that there is a negative correlation between the two metrics which poses an issue as the objective of the stage is to be as optimal as possible while trying not to sacrifice any metrics and maintaining low average turnaround times.

The other performance metrics mentioned include: waiting time, rental cost, and resource utilisation. Waiting time is the time in-between the submission time and start time. Rental cost is the total resource usage (in seconds)*rental cost per second[1]. Resource utilisation is the actual resource usage/total resource usage[2].

The scheduling algorithm that will be implemented into the client to address this issue will be a modified version of the First Capable algorithm that takes inspiration from the worst fit algorithm. This algorithm works by assigning a job to the last server capable (or the most capable server) that also contains sufficient resources. The algorithm will also use "GETS Capable" to ensure that servers that have sufficient resources and are capable of running the jobs are selected. By doing this the average turnaround costs will be reduced, resource utilisation will be exceptional, and the total rental costs will be improved. The proposed algorithm is the optimal choice for the client as it can schedule all jobs with a reduced average turnaround time. As the objective of this stage is to aim for the lowest turnaround time, this algorithm would be able to satisfy this condition. This algorithm also does not sacrifice any other performance metrics and instead improves them.

[1] Rental cost per second is calculated by the hourly rental rate/3600
[2] Resource usage is the time in-between from when a server starts a job and it completes the last job.

3 Algorithm Description

3.1 Sample Configuration

```
<config randomSeed="4096">
<!-- Author Dylan Vongsouvanh -->

  <servers>
        <server type="tiny" limit="1" bootupTime="10" hourlyRate="0.1" cores="1" memory="2000" disk="2000" />
        <server type="small" limit="1" bootupTime="20" hourlyRate="0.2" cores="2" memory="2000" disk="8000" />
        <server type="medium" limit="1" bootupTime="40" hourlyRate="0.4" cores="4" memory="16000" disk="16000" />
        <server type="large" limit="1" bootupTime="80" hourlyRate="0.8" cores="8" memory="32000" disk="64000" />
        <server type="xlarge" limit="1" bootupTime="100" hourlyRate="1.0" cores="16" memory="64000" disk="128000" />
  </servers>

  <jobs>
        <job type="instant" minRunTime="1" maxRunTime="30" populationRate="5" />
        <job type="short" minRunTime="50" maxRunTime="180" populationRate="20" />
        <job type="medium" minRunTime="400" maxRunTime="900" populationRate="30" />
        <job type="long" minRunTime="2000" maxRunTime="20000" populationRate="30" />
        <job type="verylong" minRunTime="4000" maxRunTime="50000" populationRate="15" />
  </jobs>
  <workload type="unknown" minLoad="10" maxLoad="100" />
  <termination>

        <condition type="endtime" value="86400" />
        <condition type="jobcount" value="10" />
  </termination>
</config>
```

Figure 1: Sample Configuration [2]

The following configuration will be used as a sample scenario that will be used to test the ds client. This scenario contains 10 jobs and 5 different servers (tiny, small, medium, large, and xlarge) and there is only one server available per type. There are also 5 types of jobs of varying speeds available ranging from instant, short, medium, long, and very long).

3.2 Schedule

JOB ID	Server	Submission Time	Waiting Time	Start Time	End Time	Turnaround Time
0	xlarge	25	100	125	45201	45176
1	xlarge	64	61	125	1009	945
2	xlarge	88	37	125	214	126
3	xlarge	147	0	147	250	103
4	xlarge	198	0	198	1088	890
5	xlarge	254	0	254	10118	9864
6	xlarge	279	730	1009	1084	805
7	xlarge	289	720	1009	1136	847
8	xlarge	345	739	1084	1261	916
9	xlarge	428	660	1088	1108	680

Table 1: Algorithm Schedule

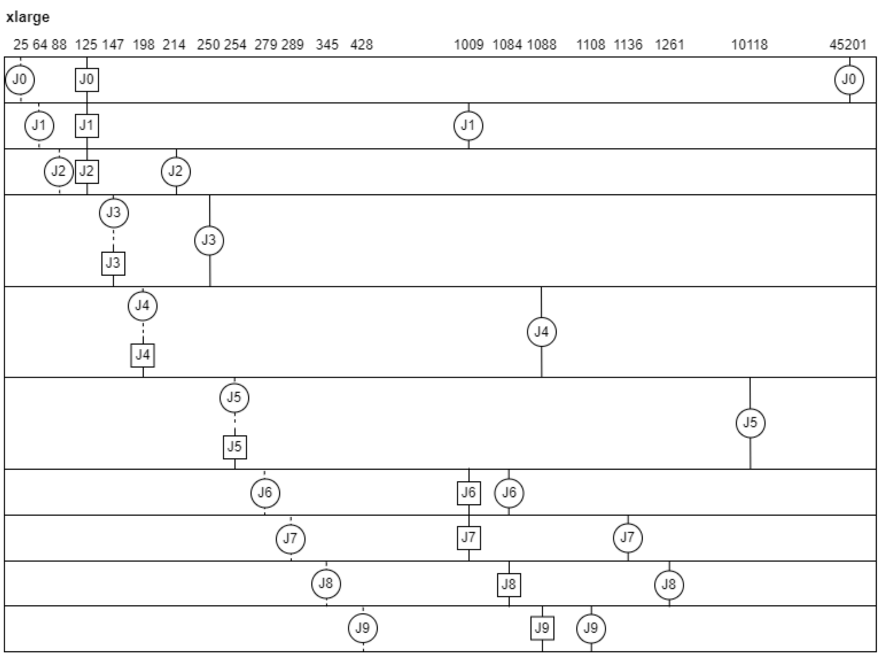

Figure 2: Timeline of Jobs Ran using the Algorithm

The table above outlines what the servers will be allocated to a job, submission time (schedule job), waiting time, start time (run job), end time (job completion), and turnaround time. The following schedule reveals what time (in milliseconds) each job will be scheduled, ran, and completed. This is used to visualise the scheduling algorithm and see the timeline of the jobs run. As the algorithm selected one server to run all jobs, some jobs will immediately run after being scheduled as there would be no waiting time.

4 Implementation

4.1 Technologies

Like Stage 1 of the project, Visual Studio Code [3] will be used to implement the baseline algorithm discussed and to complete the remainder of the client. The project will continue to be run using Ubuntu [4] via Virtual Box [5] and GitHub will again be utilised to house the project. The terminal in Ubuntu will be used again to run the simulation and perform testing and evaluation of the client.

4.2 Techniques

On top of the techniques used in the previous stage (i.e., variables of various data types, loops, and conditional statements), a parser will be used to extract the data sent by the server. Loops will also be used again to iterate through every server in the list and extract the data into the arraylist. The string split method will be used to extract certain job and server details such as job id, server name and type which will be utilised later when using the “GETS” and “SCHD” commands.

4.3 Data Structures

Similar to the first stage various date structures will be used throughout this stage. Data structures implemented in this stage include arraylists and loops. Arraylists will be used to store the parsed data while loops will be used to set conditions throughout the client (i.e., check whether there are no more jobs available and if the current reply is a valid job).

4.4 Software Libraries

Unlike the previous stage, some software libraries will be removed as there is no need for them such as the xml.parser library. This is due to the client not needing to parse from an xml file and instead will parse the data from the response of the server. Any other software libraries used in the previous stage that interacts with xml files and documents will also be removed.

5 Evaluation

5.1 Test Cases/Configuration

Many test cases were used to evaluate the performance of the algorithm including the sample scheduling scenario. Other test cases such as the Stage 1 test cases were used to test the algorithm to ensure the performance can properly be evaluated under different loads. The Stage 2 Test Suite was used to give a score in comparison to the baseline algorithms to determine how effective it is.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- generated by: Y. C. Lee -->
<config randomSeed="2000">
  <servers>
        <server type="tiny" limit="20" bootupTime="60" hourlyRate="0.1" cores="1" memory="1000" disk="4000" />
        <server type="small" limit="20" bootupTime="60" hourlyRate="0.2" cores="2" memory="4000" disk="16000" />
        <server type="medium" limit="20" bootupTime="60" hourlyRate="0.4" cores="4" memory="16000" disk="64000" />
        <server type="large" limit="20" bootupTime="60" hourlyRate="0.8" cores="8" memory="32000" disk="256000" />
        <server type="xlarge" limit="20" bootupTime="60" hourlyRate="1.6" cores="16" memory="64000" disk="512000" />
  </servers>
  <jobs>
        <job type="instant" minRunTime="1" maxRunTime="30" populationRate="15" />
        <job type="short" minRunTime="11" maxRunTime="180" populationRate="40" />
        <job type="medium" minRunTime="301" maxRunTime="900" populationRate="30" />
        <job type="long" minRunTime="1801" maxRunTime="43200" populationRate="10" />
        <job type="verylong" minRunTime="3601" maxRunTime="100000" populationRate="5" />
  </jobs>
  <workload type="light" minLoad="10" maxLoad="30" />
  <termination>
        <!-- simulation terminates whichever condition meets first -->
        <!-- 60 seconds * 60 minutes * 24 hours * 3 days = 129600 seconds -->
        <condition type="endtime" value="129600" />
        <condition type="jobcount" value="500" />
  </termination>
</config>
```

Figure 3: Test Case [6]

5.2 Results

For the implemented algorithm, all jobs ran according to the sample configuration schedule (see Table 1) and had completed all jobs successfully. The total cost to run all jobs was $12.52, the average waiting time was 304, the average execution time was 5730, the average turnaround time was 6034, and the average resource utilisation was 100%.

When running the Stage 2 Test Suite the average turnaround time was 226351.22, the average resource utilisation was 100%, and the average rental cost was $405.00. These results were formed by the total of each metric from all the test cases (18 test cases) and calculating the averages of them.

5.3 Comparison

The following comparisons of the four baseline algorithms are based on the results of the sample configuration discussed earlier. The jobs were run using the command ./ds-client -a [algorithm name].

JOB ID	Server	Submission Time	Waiting Time	Start Time	End Time	Turnaround Time
0	large	25	80	105	45181	45156
1	xlarge	64	100	164	1048	984
2	tiny	88	10	98	187	99
3	small	147	20	167	270	123
4	tiny	198	0	198	1088	890
5	medium	254	40	294	10158	9904
6	xlarge	279	0	279	354	75
7	xlarge	289	0	289	416	127
8	medium	345	9813	10158	10335	9990
9	small	428	0	428	448	20

Table 2: FF Schedule

For the First Fit algorithm, the total cost to run all jobs was $11.42, the average waiting time was 1006, the average execution time was 5730, the average turnaround time was 6736, and the average resource utilisation was 88.53%.

JOB ID	Server	Submission Time	Waiting Time	Start Time	End Time	Turnaround Time
0	large	25	80	105	45181	45156
1	xlarge	64	100	164	1048	984
2	tiny	88	10	98	187	99
3	small	147	20	167	270	123
4	tiny	198	0	198	1088	890
5	large	254	0	254	10118	9864
6	medium	279	40	319	394	115
7	xlarge	289	0	289	416	127
8	xlarge	345	0	345	522	177
9	large	428	0	428	448	20

Table 3: BF Schedule

For the Best Fit algorithm, the total cost to run all jobs was $10.30, the average waiting time was 25, the average execution time was 5730, the average turnaround time was 5755, and the average resource utilisation was 99.78%.

JOB ID	Server	Submission Time	Waiting Time	Start Time	End Time	Turnaround Time
0	xlarge	25	100	125	45201	45176
1	xlarge	64	61	125	1009	945
2	large	88	80	168	257	169
3	large	147	21	168	271	124
4	large	198	0	198	1088	890
5	large	254	0	254	10118	9864
6	large	279	0	279	354	75
7	medium	289	40	329	456	167
8	xlarge	345	664	1009	1186	841
9	large	428	0	428	448	20

Table 4: WF Schedule

For the Worst Fit algorithm, the total cost to run all jobs was $14.75, the average waiting time was 96, the average execution time was 5730, the average turnaround time was 5826, and the average resource utilisation was 100%.

JOB ID	Server	Submission Time	Waiting Time	Start Time	End Time	Turnaround Time
0	large	25	80	105	45151	45126
1	large	64	45117	45181	46065	46001
2	tiny	88	10	98	187	99
3	tiny	147	40	187	290	143
4	tiny	198	93	290	1180	983
5	small	254	20	274	10138	9884
6	medium	279	40	319	394	115
7	medium	289	105	394	521	232
8	medium	345	176	521	698	353
9	tiny	428	752	1180	1200	772

Table 5: FC Schedule

For the First Capable algorithm, the total cost to run all jobs was $10.83, the average waiting time was 4643, the average execution time was 5730, the average turnaround time was 10373, and the average resource utilisation was 100%.

When comparing the results of the sample configuration and the test suite, the implemented algorithm does not excel at providing the best average turnaround time only successfully beating First Capable, however, it does provide better performance across all other metrics particularly resource utilisation at 100% in every test. Total rental cost outperforms three out of the four baseline algorithms, only being surpassed by First Capable by a small margin.

Total rental cost

Config	FC	FF	BF	WF	Yours
config20-long-low.xml	104.68	206.54	208.86	237.31	88.06
config20-med-med.xml	150.99	287.25	268.48	273.50	164.01
config20-short-high.xml	133.57	181.77	185.17	220.23	163.69
config32-long-high.xml	199.63	239.41	240.58	259.97	199.96
config32-long-med.xml	138.35	226.20	226.04	206.52	131.75
config32-med-high.xml	177.25	222.35	217.29	243.56	178.34
thmbox 2-med-low.xml	87.90	194.82	194.11	141.35	60.96
config32-short-low.xml	83.27	192.14	192.14	134.86	65.50
config32-short-med.xml	110.87	183.44	184.59	188.09	121.32
config50-long-high.xml	1155.19	1501.51	1499.19	1614.23	1172.19
config50-long-low.xml	365.60	824.02	779.63	778.55	359.54
config50-long-med.xml	735.52	1140.81	1163.05	1267.73	725.93
config50-med-high.xml	995.57	1323.38	1313.19	1512.19	1052.12
config50-med-low.xml	301.83	876.88	841.25	804.84	318.20
config50-med-med.xml	728.25	1246.19	1288.20	1271.69	769.45
config50-short-high.xml	537.72	692.03	710.52	817.12	687.44
config50-short-low.xml	251.75	890.16	868.14	819.33	332.81
config50-short-med.xml	567.65	1058.34	1083.68	1180.50	698.70
Average	**379.20**	**638.18**	**636.89**	**665.09**	**405.00**
Normalised (FC)	1.0000	1.6830	1.6796	1.7539	1.0680
Normalised (FF)	0.5942	1.0000	0.9980	1.0422	0.6346
Normalised (BF)	0.5954	1.0020	1.0000	1.0443	0.6359
Normalised (WF)	0.5701	0.9595	0.9576	1.0000	0.6089
Normalised (AVG [FF,BF,WF])	0.5863	0.9868	0.9848	1.0284	0.6262

Figure 4: Stage 2 Test Suite Total Rental Cost Results

5.4 Pros and Cons

There are advantages of implementing the proposed algorithm over the other baseline algorithms. One of the advantages is that there likely will only be one server required to complete all jobs as the algorithm assigns the last capable server which would be the largest server capable enough to run jobs. By starting one server, the total rental costs will be reduced and the resources will be utilised efficiently. Another advantage to the algorithm is related to the previous advantage which is leftover resources still being available after each job is complete. With leftover resources available, partitions are created to allow for other jobs to still be run on the same server.

There are also drawbacks associated with the implemented algorithm. One of the weaknesses of the algorithm is that it is not as efficient at memory allocation when assigning jobs to servers. As the algorithm automatically selects the most capable server, it ignores the other capable servers that may instead be a better fit thus wasting/leaving behind memory. Another drawback to the algorithm is that since the assigned server would be the last capable one in the list there would be multiple jobs being scheduled and run on the one server thus increasing the wait time and therefore, the turnaround time.

6 Conclusion

In conclusion, the proposed and implemented algorithm does offer better performance over the four baseline algorithms providing slightly better turnaround times than the First Capable algorithm but failing at outperforming the First Fit, Best Fit and Worst Fit algorithms. The algorithm excels at optimising resource utilisation beating all other algorithms and is excellent at reducing the total rental cost coming second, slightly behind First Capable. What could be done to improve the algorithm is to implement a method to only assign jobs to servers using “GETS Avail” rather than “GETS Capable” so that jobs are only assigned to servers that are free and do not have any other job scheduled and/or running. The algorithm could also opt for inactive servers first, avoiding servers with an “active” and/or “booting” status to reduce the waiting time in-between jobs.

References

[1] D. Vongsouvanh, “Github Repository.” https://github.com/Dylan-vong/COMP3100Project.

[2] Dylan Vongsouvanh, “ds-sample-config.xml.”

[3] Microsoft, “Visual Studio Code.” https://code.visualstudio.com/.

[4] Canonical Ltd, “Ubuntu.” https://ubuntu.com/.

[5] Oracle, “VM VirtualBox.” https://www.virtualbox.org/.

[6] Y. C. Lee, “ds-s1-config07-wk.xml.”