Project report: Stage 1
COMP3100 Distributed Systems, S1, 2023
SID: 45156417, Name: Justin Ling

1 Introduction

The purpose of stage 2 is to design and implement a new scheduling algorithm in a distributed system. The algorithm in particular will schedule jobs to the server to minimise turnaround time as well as minimise the impact of other performance metrics[1] such as:

- Server Rental Costs
- Server Resource Utilisation

Through our implementation, we've analyzed the server's information such as cores, memory and disk requirements and have developed an algorithm to determine which is the most suitable to schedule a job. Running this algorithm will allow us to compare against the provided baseline algorithms such as:

- First Fit (FF) and (FFQ)
- Best Fit (BF) and (BFQ)
- Worst Fit Queue (WFQ)

Each baseline algorithm has its own performance as it's designed to achieve different results and therefore the performance metrics of each algorithm will be compared against ours. While it's difficult to ensure all performance metrics are met, the focus of our implementation handles most of the objectives in a fair manner with a main focus on turnaround time. Thus, this report will evaluate the performance of our algorithm and discuss any potential recommendations or flaws in our design.

2 Problem Definition

In a distributed system ensuring that the scheduling and assigning of jobs in an efficient way is the main objective. This type of objective creates a clear understanding of the problem definition however few problems arise while aiming for a more efficient system such as conflicting performance metrics, availability of resources and algorithm flexibility.

- Conflicting Performance Objectives - While the main objective is to minimise turnaround time, when we focus on improving this single metric it may lead to other performance metrics worsening, such as increased rental costs or resource utilisation. The key would be to decide on which specific trade-off would be appropriate for the problem and balance the performance of the algorithm.
- Availability of Resources - Every job in a server has a limited number of resources such as the number of cores, memory size and disk capacity. The algorithm will need to target each specific resource and ensure jobs are assigned to the right server.
- Algorithm Flexibility - Due to the vast number of possible configurations, any implemented algorithm should be flexible enough to work with different configurations. This ensures the algorithm works in many different environments and performs at a similar level.

By focusing on these challenges, our implemented algorithm is able to achieve the main objective which is to minimise turnaround time without heavily impacting other performance metrics. In stage 2 I've developed a brand new algorithm to target these problems which aims to reduce turnaround time.

3 Algorithm Description

This section provides an overview of the scheduling algorithm. We will explore how the algorithm will operate in a scheduling environment with a sample configuration to illustrate its functionality.

```
<config randomSeed="123456">
    <servers>
        <server type="small" limit="4" bootupTime="60" hourlyRate="0.2" cores="2" memory="4000" disk="16000" />
        <server type="medium" limit="4" bootupTime="60" hourlyRate="0.4" cores="4" memory="16000" disk="64000" />
        <server type="large" limit="4" bootupTime="60" hourlyRate="0.8" cores="8" memory="32000" disk="256000" />
    </servers>
    <jobs>
        <job type="long" minRunTime="1201" maxRunTime="3600" populationRate="100" />
    </jobs>
    <workload type="low" minLoad="40" maxLoad="80" />
    <termination>
        <condition type="endtime" value="86400" />
        <condition type="jobcount" value="2000" />
    </termination>
</config>
```

S2DemoConfigs - config12-long-med.xml

Consider the following sample config from Figure 1, In this demo there are three server types; small, medium and large. Each server has its own unique features such as hourly rate, number of cores, memory, and disk, with limit and bootup time the same. The execution of our algorithm will process the single job ”long” and will receive the jobs from the server and then process the capable servers based on the available resources. The algorithm will either complete all the jobs or based on the configuration the termination will either timeout after 86,400 seconds or process all the 2000 jobs.

In this configuration, our algorithm starts by establishing a handshake with the server this ensures the initial communication and authentication is confirmed by the server. After authentication, the client sends the server a ”REDY” message to request information from either ”JOBN” or ”JOBP” messages. The algorithm then enters the loop to analyse and compares the job information received with the available resources in the server. Sending a ”GETS Capable” to retrieve server information that meets the requirements of the jobs.

Once the jobs have been selected, the algorithm schedules these jobs on that server and proceeds until all jobs are completed. The main idea behind this algorithm is to consider server resource availability and ensure each job goes to the most suitable server. However, while the algorithm works efficiently, our designed implementation does not explicitly incorporate First-fit or Best-fit algorithms. Our implementation does iterate through the server records based on the first capable server based on the serverType, scheduleID and resources available. This is similar to the first fit approach where the algorithm assigns each job to the first server which can accommodate.

4 Implementation

In this section, we will explore the client-side simulator which is implemented in Java with version 11.0.18 and utilises a socket connection with input/output streams to read and write to the server. The algorithm is split into modular components with separate send, receive and quit components with the majority of the functionality working within the main function.

4.1 Initial Handshake

During the initial handshake process, the client initially sets up all the necessary variables to hold the information which will be received from the server such as ”messages” and ”currMessages”. It then establishes the initial handshake with the server sending ”HELO” and authenticating it with the system user. After ensuring the authentication is successful the client will enter into the scheduling algorithm.

4.2 sendMessage Function

The sendMessage function is responsible for sending messages to the server through the DataOutputStream. It takes the output and message as an argument and the message is then concatenated with a newline using the ’+’ operator. The string is then converted with the .getBytes() method and finally, a flush() is used to ensure the data is sent correctly.

4.3 receiveMessage Function

Similarly, the receiveMessage static function is to read the messages received from the server. It utilises a readLine method on the inputReader argument to return the input as a string.

4.4 quit Function

The quit function is called when the algorithm either completes or authentication fails in the beginning.

4.5 Gets Capable

Our implementation utilises ”GETS Capable” in order to query the selected server information based on specific requirements. For example in our code Gets Capable returns the jobCores, jobMemory and jobDisk. This is to ensure we only receive servers that can fulfil the specifications of the job.

4.6 Main

All of the job scheduling and algorithm implementations occurs in the main method. It begins after authentication is approved by the server. The algorithm enters into the initial if statement after the client sends ”REDY” and receives a response. Within the loop, the client requests the job information only if the jobs are either JOBN or JOBP else it moves to the next request from the server. If the jobs match, a GETS capable is sent to the server to find all the capable servers. Once a suitable server is found the client will schedule the job and then moves onto the next message. The algorithm occurs when we receive the server lists and then we store this information as the first server and utilise it as we’ve already ensured the server fits the requirements of the job. Finally, the loop continues until there are no more jobs to schedule.

5 Evaluation

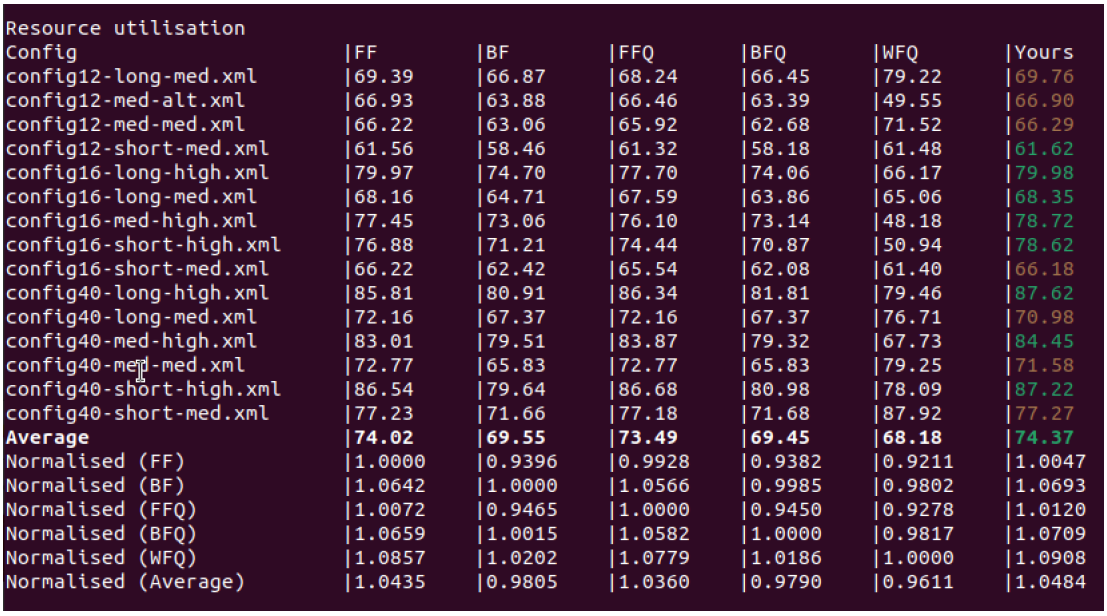
In order to effectively evaluate our algorithm, we’ve utilisesd a config file from the S2TestConfigs which will be used in this section to analyze and evaluate our algorithm. We will also explore the results of our algorithm against the other baseline algorithms such as FF, BF, FFQ, BFQ and WFQ against the key performance metrics of turnaround time, rental costs and resource utilisation.

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	3553	3553	3553	3553	5928	3562
config40-med-high.xml	1505	896	940	933	3097	906
config40-med-med.xml	941	941	941	941	1988	941
config40-short-high.xml	485	1365	373	301	2817	228
config40-short-med.xml	180	182	198	193	1945	180
Average	**1867.13**	**2417.27**	**2559.07**	**2401.80**	**9958.40**	**1397.73**
Normalised (FF)	1.0000	1.2946	1.3706	1.2864	5.3335	0.7486
Normalised (BF)	0.7724	1.0000	1.0587	0.9936	4.1197	0.5782
Normalised (FFQ)	0.7296	0.9446	1.0000	0.9385	3.8914	0.5462
Normalised (BFQ)	0.7774	1.0064	1.0655	1.0000	4.1462	0.5820
Normalised (WFQ)	0.1875	0.2427	0.2570	0.2412	1.0000	0.1404
Normalised (Average)	0.4861	0.6294	0.6663	0.6253	2.5928	0.3639

S2TestConfigs - Turnaround Time

Overall, the turnaround time for our implementation represents a fairly successful outcome across all the various configuration tests. It performed consistently better than the baseline algorithms between config16-med-high down to config-40-long-high. Between other configs our implementation compared similarly to the other algorithms but overall we’ve managed to achieve the best average results in Turnaround time.

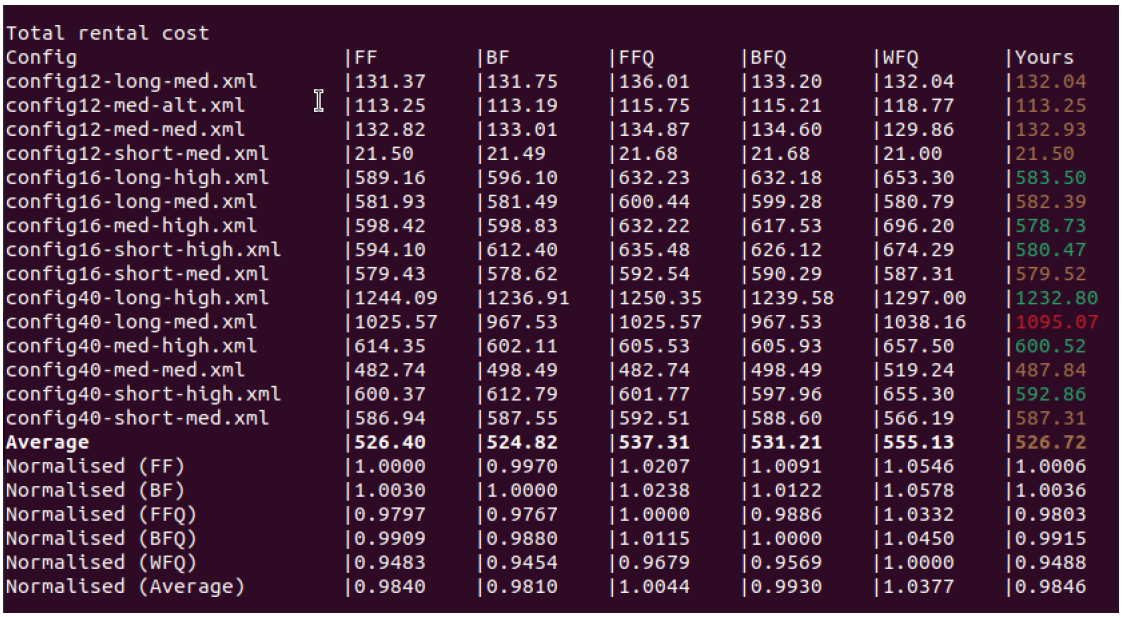


Resource utilisation

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	72.16	67.37	72.16	67.37	76.71	70.98
config40-med-high.xml	83.01	79.51	83.87	79.32	67.73	84.45
config40-med-med.xml	72.77	65.83	72.77	65.83	79.25	71.58
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	**74.02**	**69.55**	**73.49**	**69.45**	**68.18**	**74.37**
Normalised (FF)	1.0000	0.9396	0.9928	0.9382	0.9211	1.0047
Normalised (BF)	1.0642	1.0000	1.0566	0.9985	0.9802	1.0693
Normalised (FFQ)	1.0072	0.9465	1.0000	0.9450	0.9278	1.0120
Normalised (BFQ)	1.0659	1.0015	1.0582	1.0000	0.9817	1.0709
Normalised (WFQ)	1.0857	1.0202	1.0779	1.0186	1.0000	1.0908
Normalised (Average)	1.0435	0.9805	1.0360	0.9790	0.9611	1.0484

S2TestConfigs - Resource Utilisation

Moving onto resource utilisation, our implementation performed similarly with the results in the turn around time table. Our algorithm performed comparatively to most of the baseline algorithms with the baseline algorithm WFQ having one of the best performances overall compared with the worst performance in turnaround time. Our algorithm also had the highest average based on resource utilisation.



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	1025.57	967.53	1025.57	967.53	1038.16	1095.07
config40-med-high.xml	614.35	602.11	605.53	605.93	657.50	600.52
config40-med-med.xml	482.74	498.49	482.74	498.49	519.24	487.84
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	**526.40**	**524.82**	**537.31**	**531.21**	**555.13**	**526.72**
Normalised (FF)	1.0000	0.9970	1.0207	1.0091	1.0546	1.0006
Normalised (BF)	1.0030	1.0000	1.0238	1.0122	1.0578	1.0036
Normalised (FFQ)	0.9797	0.9767	1.0000	0.9886	1.0332	0.9803
Normalised (BFQ)	0.9909	0.9880	1.0115	1.0000	1.0450	0.9915
Normalised (WFQ)	0.9483	0.9454	0.9679	0.9569	1.0000	0.9488
Normalised (Average)	0.9840	0.9810	1.0044	0.9930	1.0377	0.9846

S2TestConfigs - Rental Cost

The rental costs of our algorithm showed that it performed quite similarly to most of the configs except in config40-long-med. Which performed the worst across all the baseline algorithms. Overall our implementation worked for all the configuration sizes.

5.1 Pros

- Turnaround Time: Comparing the baseline algorithms against our implementation has shown that the algorithm has achieved a competitive Turn around time.
- Job Scheduling: With a competitive turnaround time, our algorithm was able to schedule the right jobs to the suitable servers. This meant there was an efficient distribution of jobs.
- Scalability: The algorithm performed similarly across most config files. There seems to be room to improve at the lower configs.

5.2 Cons

- Total Rental Costs: Our implementation seemed to have a higher total rental cost compared with the baseline algorithms. It had the highest rental cost in config40-long-med.
- Poor performance in smaller configs: The algorithms normalised performance in the lower range of the config setups seems to be at a fairly sound performance compared to the performance in the mid to higher config ranges. This could mean the algorithm is not as suitable for smaller configs.

6 References

- Link to Justin's Github — https://github.com/Justin-zi/COMP3100
- [1]Distributed System Documentation "distsys-MQ" — https://github.com/distsys-MQ/ds-sim