COMP3100 STAGE 2

Creating a cost-efficient job allocator

Anthony Halteh: 45650225

Introduction

The purpose of this project is to improve one or more aspects of the Job scheduler program created in Stage 1. These aspects are as follows; average turnaround time, average resources and server rental cost. The main focus of stage 2 is on minimising server costs however, as will be mentioned later in the report, the average resource utilisation was also affected due to minimising costs. Although, as a consequence of reducing the server costs, turnaround time was significantly affected in a negative manner.

Problem Definition

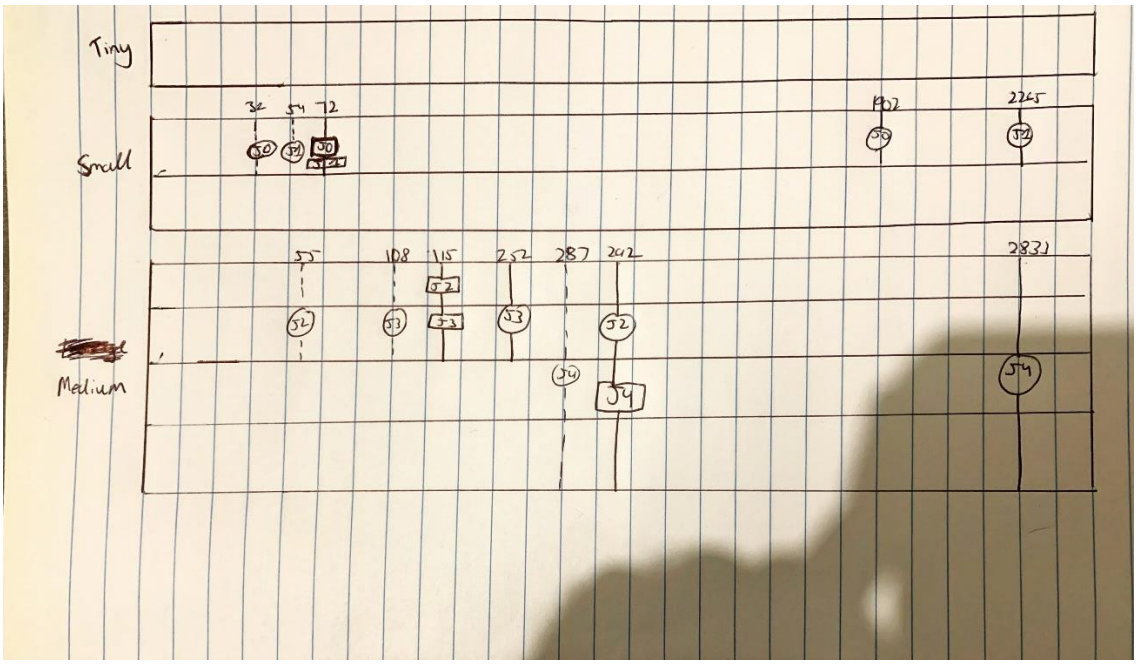
In stage one, the objective was to create an algorithm that take job requests and schedule said jobs to a server. The problem that arises with this algorithm is that it disregards any critical sorting as jobs were only scheduled to any largest server possible. There was no consideration about server cost in the original algorithm. The new algorithm for stage 2 mainly addresses the issue of relatively high server costs. This is the best aspect to improve on as companies are always looking to minimising costs at any place necessary no matter the consequence. The aim of this algorithm is to achieve the goal of reducing cost as it is a real possibility a large company may ask.

Algorithm Description

The main focus of this algorithm is to reduce server costs when scheduling jobs to servers. To achieve this goal, all jobs should be sent to the smallest server that is capable of running the job, which can be seen in *Figure 1.* As illustrated in the drawing based on the week 9 scheduling task, first two jobs are sent to the small server where they run simultaneously since both jobs have 1 core and the small server has 2 cores available. Jobs 2, 3 runs on the medium server as there are insufficient cores on the small server since both jobs require 2

cores. Job 4 is required to go on the medium server due to the larger core size of 4. Hence, all jobs in the diagram are sent to the smallest server possible therefore, reducing overall cost.

Figure 1 – Drawing of schedule



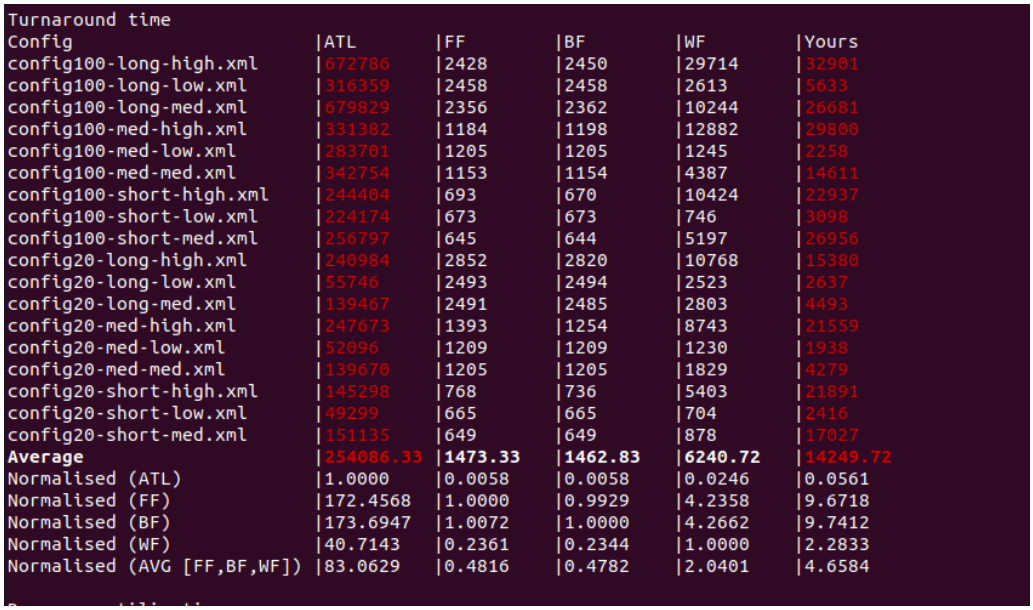
This contradicts with the algorithm in stage one where all jobs were scheduled to the largest server that was present causing relative high costs.

Implementation

To achieve this, Gets All was replaced with Gets Capable. Gets Capable sends all servers that are available for the JOBN sent as apposed to Gets All that returns all servers. Since Gets Capable needs to sent every time a Job is received, a *getsCapable* method was established inside the main while loop. This method sends the Gets command and stores all available servers it receives. It also stores the core, memory and disk numbers of the job which is used to call Gets Capable. Another addition to the algorithm was made in the *createServerInfo* method that differs from the algorithm in stage 1. Instead of gathering and storing the largest server, the *createServerInfo* now obtains the smallest server that's capable of running the job and stores it. This information gets updated with every job that is sent.

Evaluation

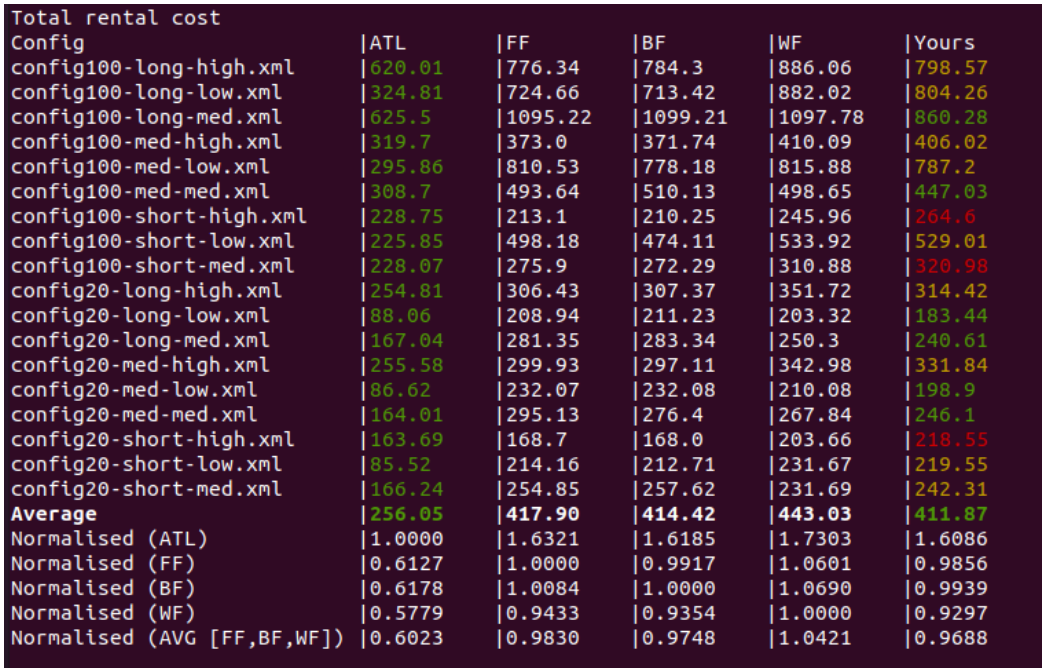
Figure 2 – Turnaround Time Results



Turnaround time

Config	ATL	FF	BF	WF	Yours
config100-long-high.xml	672786	2428	2450	29714	32901
config100-long-low.xml	316359	2458	2458	2613	5633
config100-long-med.xml	679829	2356	2362	10244	26681
config100-med-high.xml	331382	1184	1198	12882	29800
config100-med-low.xml	283701	1205	1205	1245	2258
config100-med-med.xml	342754	1153	1154	4387	14611
config100-short-high.xml	244404	693	670	10424	22937
config100-short-low.xml	224174	673	673	746	3098
config100-short-med.xml	256797	645	644	5197	26956
config20-long-high.xml	240984	2852	2820	10768	15380
config20-long-low.xml	55746	2493	2494	2523	2637
config20-long-med.xml	139467	2491	2485	2803	4493
config20-med-high.xml	247673	1393	1254	8743	21559
config20-med-low.xml	52096	1209	1209	1230	1938
config20-med-med.xml	139670	1205	1205	1829	4279
config20-short-high.xml	145298	768	736	5403	21891
config20-short-low.xml	49299	665	665	704	2416
config20-short-med.xml	151135	649	649	878	17027
Average	**254086.33**	**1473.33**	**1462.83**	**6240.72**	**14249.72**
Normalised (ATL)	1.0000	0.0058	0.0058	0.0246	0.0561
Normalised (FF)	172.4568	1.0000	0.9929	4.2358	9.6718
Normalised (BF)	173.6947	1.0072	1.0000	4.2662	9.7412
Normalised (WF)	40.7143	0.2361	0.2344	1.0000	2.2833
Normalised (AVG [FF,BF,WF])	83.0629	0.4816	0.4782	2.0401	4.6584

Figure 3 – Rental Cost Results



Total rental cost

Config	ATL	FF	BF	WF	Yours
config100-long-high.xml	620.01	776.34	784.3	886.06	798.57
config100-long-low.xml	324.81	724.66	713.42	882.02	804.26
config100-long-med.xml	625.5	1095.22	1099.21	1097.78	860.28
config100-med-high.xml	319.7	373.0	371.74	410.09	406.02
config100-med-low.xml	295.86	810.53	778.18	815.88	787.2
config100-med-med.xml	308.7	493.64	510.13	498.65	447.03
config100-short-high.xml	228.75	213.1	210.25	245.96	264.6
config100-short-low.xml	225.85	498.18	474.11	533.92	529.01
config100-short-med.xml	228.07	275.9	272.29	310.88	320.98
config20-long-high.xml	254.81	306.43	307.37	351.72	314.42
config20-long-low.xml	88.06	208.94	211.23	203.32	183.44
config20-long-med.xml	167.04	281.35	283.34	250.3	240.61
config20-med-high.xml	255.58	299.93	297.11	342.98	331.84
config20-med-low.xml	86.62	232.07	232.08	210.08	198.9
config20-med-med.xml	164.01	295.13	276.4	267.84	246.1
config20-short-high.xml	163.69	168.7	168.0	203.66	218.55
config20-short-low.xml	85.52	214.16	212.71	231.67	219.55
config20-short-med.xml	166.24	254.85	257.62	231.69	242.31
Average	**256.05**	**417.90**	**414.42**	**443.03**	**411.87**
Normalised (ATL)	1.0000	1.6321	1.6185	1.7303	1.6086
Normalised (FF)	0.6127	1.0000	0.9917	1.0601	0.9856
Normalised (BF)	0.6178	1.0084	1.0000	1.0690	0.9939
Normalised (WF)	0.5779	0.9433	0.9354	1.0000	0.9297
Normalised (AVG [FF,BF,WF])	0.6023	0.9830	0.9748	1.0421	0.9688

Figure 4 – Resource Utilisation Results

Resource utilisation

Config	ATL	FF	BF	WF	Yours
config100-long-high.xml	100.0	83.58	79.03	80.99	79.53
config100-long-low.xml	100.0	50.47	47.52	76.88	87.6
config100-long-med.xml	100.0	62.86	60.25	77.45	99.29
config100-med-high.xml	100.0	83.88	80.64	89.53	88.94
config100-med-low.xml	100.0	40.14	38.35	76.37	86.97
config100-med-med.xml	100.0	65.69	61.75	81.74	67.35
config100-short-high.xml	100.0	87.78	85.7	94.69	99.72
config100-short-low.xml	100.0	35.46	37.88	75.65	84.14
config100-short-med.xml	100.0	67.78	66.72	78.12	97.91
config20-long-high.xml	100.0	91.0	88.97	66.89	76.07
config20-long-low.xml	100.0	55.78	56.72	69.98	79.69
config20-long-med.xml	100.0	75.4	73.11	78.18	90.12
config20-med-high.xml	100.0	88.91	86.63	62.53	62.17
config20-med-low.xml	100.0	46.99	46.3	57.27	65.66
config20-med-med.xml	100.0	68.91	66.64	65.38	83.41
config20-short-high.xml	100.0	89.53	87.6	61.97	53.57
config20-short-low.xml	100.0	38.77	38.57	52.52	57.09
config20-short-med.xml	100.0	69.26	66.58	65.21	78.06
Average	**100.00**	**66.79**	**64.94**	**72.85**	**79.85**
Normalised (ATL)	1.0000	0.6679	0.6494	0.7285	0.7985
Normalised (FF)	1.4973	1.0000	0.9724	1.0908	1.1956
Normalised (BF)	1.5398	1.0284	1.0000	1.1218	1.2295
Normalised (WF)	1.3726	0.9168	0.8914	1.0000	1.0960
Normalised (AVG [FF,BF,WF])	1.4664	0.9794	0.9523	1.0683	1.1709

When running the algorithm on the latest test provided on iLearn, the conclusion can be made that the Rental Cost average is better than the First Fit, Best Fit and Worst Fit algorithms as illustrated in *Figure 3*. This is the result of scheduling all jobs to the smallest server possible capable of running the job. However, it can be noted that this clearly does not 100% of the time as there are three tests which perform worse than the other algorithms mentioned above.

One significant drawback of this algorithm is the larger turnaround times created. In *Figure 2* the average turnaround time is more than double the average Worst Fit turnaround time. This is a consequence of sending all jobs to the smallest server. After being scheduled, jobs need to wait for any available room on the server before being executed, this is illustrated in *Figure 1*. Job 4 needs to run on the medium server as job requires 4 cores however, Job 2 is running at same time Job 4 boots. Job 4 will not run until Job 2 finishes thus creating a queue. Since *Figure 1* is a small test, the turnaround time does not appear drastic however, these queues and wait times are amplified when running large configurations therefore, creating exceedingly large turnaround times. It can also be noted that Recourse Utilisation slightly affected in this algorithm but not as severe as the Turnaround Time. As illustrated in *Figure 4* majority configurations were still superior than the First Fit, Best fit and Worst Fit algorithms.

Conclusion

In conclusion, being ask to reduce the cost of servers is a real scenario that can happen when working for large companies, which is why the algorithm was modelled around doing exactly that. This was achieved by sending all jobs to smallest server capable of executing said job. However, there are consequences for doing so, such as significantly worse turnaround time and a slight decrease in resource utilisation.

References

[1] A. Halteh, "Github," [Online]. Available: https://github.com/TonyMontana873/COMP3100_stage2.git.