Cost and Resource Efficient Job Allocator for Distributed Systems

Adam Fulton
45634092

Introduction

The main objective of this assessment was to develop an efficient job scheduling algorithm that optimises one or more key objectives such as minimising the average turnaround time on all servers, Maximising the average resources used on all servers and minimising the total rental cost of servers used. The goal of my assessment was to develop a cost and resource-efficient algorithm that aims to both maximise the utilisation of resources used on the servers and minimise the total cost associated with using them.

Problem Definition

The problem for this assessment was to develop an efficient algorithm that optimises one or more key objectives such as turnaround time, resource utilisation and total server rental cost. With turnaround time being the time taken between submission of a process and the completion of it [1]. With Resource utilisation being a metric to describe what percentage of server resources the job is running has been utilised. and cost being the rental cost of using the server to complete the job. The three-performance metrics present a problem as they are conflicting in nature and would be extremely difficult to develop an algorithm to optimise all three objectives, for example, optimising cost would require you to use as few servers as possible which would have an impact on turnaround time [2]. The objective of my scheduling algorithm is to improve upon cost and resource utilisation while trying to achieve a minimal increase in turnaround time as much as possible depending on different workloads. I decided to optimise for cost and resource utilisation as both performance metrics complement each other as if your objective is to reduce costs you would also want to ensure the resources on the servers you are using are utilised to their fullest.

Algorithm Description

My scheduling algorithm works in the following way, and I will also give an example scheduling scenario to show how it works.

1. Client asks the server for the available servers that can run the current job the client needs the schedule. If the client keeps sending jobs back to the queue after a certain period of time the client instead asks the server for a list of all capable servers to schedule a job on.

2 The client than calculates the server with the smallest core count that is capable of running the job. Go to step 3. If the client receives no data on the available servers to run the current job. The client will push the current job back into the job queue and receive a new job to schedule from the server and will go back to step 1 until it finds a job that has available servers for it to run on. If the client can't find a server after a period of time it will go back to step 1 and instead of asking the server for a list of the available servers, it will ask the server for a list of capable servers in order to schedule the job.
3 The client schedules the job to the server with the smallest core count that is capable of the receiving the job as calculated in step 2.
4 When the client receives a command from the server that a previous scheduled job has completed. If will ask the server if the server that has just completed the job has any more jobs currently running on it. If it does not the client will tell the server to terminate it. If the server does have jobs running on it the client will not tell the server to terminate that particular server.
5 The above steps repeat in a loop until the server sends a command telling it there is no more jobs to be scheduled and the client will close the socket connection with the server.

Implementation

The details of the data structures that I have used in my scheduling are as follows:

- **Communicating with server.**
  To communicate with the server, I am using the Socket class from java.net to establish a socket connection with the server on port 5000. To receive input and parse commands from the server I am using BufferedReader and InputStreamReader classes from java.io, and to send data and commands to the server I am using the PrintStream class from java.io as well.

- **Control state of client.**

  To handle and control the state of the client and different stages of my scheduling algorithm I am using primitive Boolean variables.

- **Store Data Received from client and calculations made by client**

  To store data my client received from the server and subsequent calculates made by my client I used the ArrayList class from java.util.

Evaluation

To evaluate my scheduling algorithm, I can use the testing script provided that runs my scheduling algorithm alongside three baseline algorithms called best-fit, worst-fit and first-fit as well as the scheduling algorithm all to largest that I developed in stage one of this assessment. The testing script runs all algorithms on different supplied configuration files

and gives a comparison on how well each algorithm performed respective to each other on the performance objectives, turnaround time, resource utilisation and cost. The results of these tests as well as a discussion of the results can be seen below.

Resource utilisation

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

Figure 1: total rental cost results

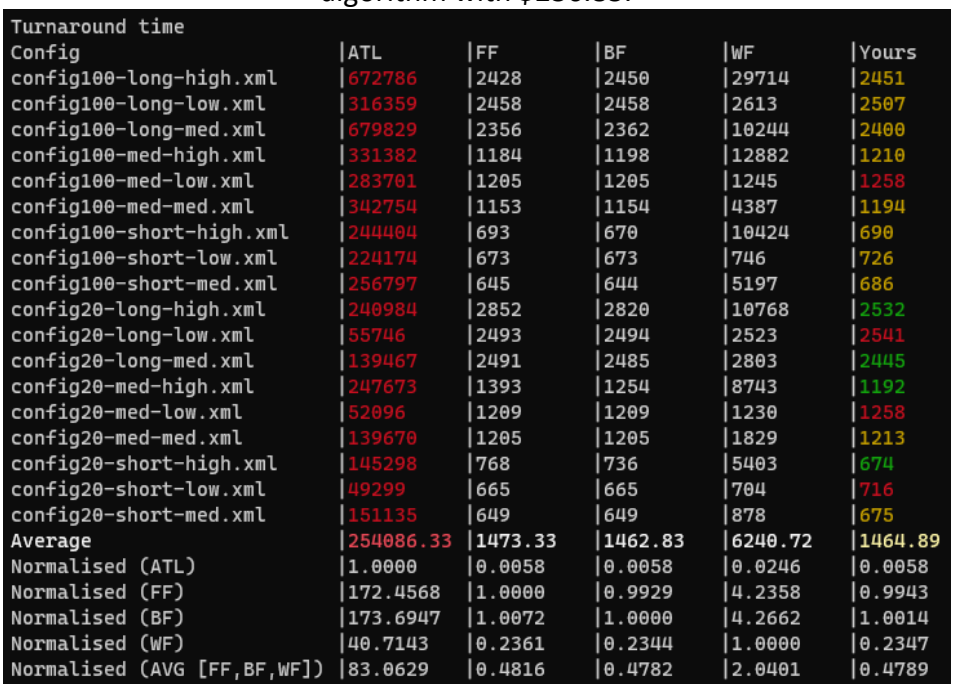
Total rental cost

Config	ATL	FF	BF	WF	Yours
config100-long-high.xml	620.01	776.34	784.3	886.06	614.71
config100-long-low.xml	324.81	724.66	713.42	882.02	303.78
config100-long-med.xml	625.5	1095.22	1099.21	1097.78	609.36
config100-med-high.xml	319.7	373.0	371.74	410.09	300.34
config100-med-low.xml	295.86	810.53	778.18	815.88	270.29
config100-med-med.xml	308.7	493.64	510.13	498.65	286.04
config100-short-high.xml	228.75	213.1	210.25	245.96	174.24
config100-short-low.xml	225.85	498.18	474.11	533.92	159.91
config100-short-med.xml	228.07	275.9	272.29	310.88	168.46
config20-long-high.xml	254.81	306.43	307.37	351.72	270.42
config20-long-low.xml	88.06	208.94	211.23	203.32	93.27
config20-long-med.xml	167.04	281.35	283.34	250.3	181.52
config20-med-high.xml	255.58	299.93	297.11	342.98	260.45
config20-med-low.xml	86.62	232.07	232.08	210.08	82.02
config20-med-med.xml	164.01	295.13	276.4	267.84	173.77
config20-short-high.xml	163.69	168.7	168.0	203.66	145.51
config20-short-low.xml	85.52	214.16	212.71	231.67	69.05
config20-short-med.xml	166.24	254.85	257.62	231.69	147.23
Average	256.05	417.90	414.42	443.03	239.47
Normalised (ATL)	1.0000	1.6321	1.6185	1.7303	0.9352
Normalised (FF)	0.6127	1.0000	0.9917	1.0601	0.5730
Normalised (BF)	0.6178	1.0084	1.0000	1.0690	0.5778
Normalised (WF)	0.5779	0.9433	0.9354	1.0000	0.5405
Normalised (AVG [FF,BF,WF])	0.6023	0.9830	0.9748	1.0421	0.5633

Figure 2: Resource utilisation

After running the test script for my scheduling algorithm the results in figure 1 above show that my scheduling algorithm outperforms the three baselines and ATL(all to largest) algorithms on the supplied test configurations for the performance metric minimising total server cost. As my results displayed in the column called “yours” have a lower cost average of 239.47 compared to the other algorithms. My scheduling algorithm also outperforms the three baseline and ATL algorithms for the performance metric of maximised resource utilisation as shown in figure 2. As my scheduling algorithms average resource utilisation shown in the same column as figure 1 is 100% compared to three baseline algorithms which average less than 75% utilisation. The ATL algorithm has the same resource utilisation average as my scheduling algorithm, however, if you look at the rental costs in figure 1 my scheduling algorithm achieves 100% resource utilisation with an average cost of $239.47 whereas ATL achieves 100% resource utilisation at a higher cost than my scheduling

algorithm with $256.85.



Turnaround time

Config	ATL	FF	BF	WF	Yours
config100-long-high.xml	672786	2428	2450	29714	2451
config100-long-low.xml	316359	2458	2458	2613	2507
config100-long-med.xml	679829	2356	2362	10244	2400
config100-med-high.xml	331382	1184	1198	12882	1210
config100-med-low.xml	283701	1205	1205	1245	1258
config100-med-med.xml	342754	1153	1154	4387	1194
config100-short-high.xml	244404	693	670	10424	690
config100-short-low.xml	224174	673	673	746	726
config100-short-med.xml	256797	645	644	5197	686
config20-long-high.xml	240984	2852	2820	10768	2532
config20-long-low.xml	55746	2493	2494	2523	2541
config20-long-med.xml	139467	2491	2485	2803	2445
config20-med-high.xml	247673	1393	1254	8743	1192
config20-med-low.xml	52096	1209	1209	1230	1258
config20-med-med.xml	139670	1205	1205	1829	1213
config20-short-high.xml	145298	768	736	5403	674
config20-short-low.xml	49299	665	665	704	716
config20-short-med.xml	151135	649	649	878	675
Average	254086.33	1473.33	1462.83	6240.72	1464.89
Normalised (ATL)	1.0000	0.0058	0.0058	0.0246	0.0058
Normalised (FF)	172.4568	1.0000	0.9929	4.2358	0.9943
Normalised (BF)	173.6947	1.0072	1.0000	4.2662	1.0014
Normalised (WF)	40.7143	0.2361	0.2344	1.0000	0.2347
Normalised (AVG [FF,BF,WF])	83.0629	0.4816	0.4782	2.0401	0.4789

Figure 3: Turnaround Time

As stated above, it is very difficult to develop a scheduling algorithm that optimises all three performance objectives. As you can see in figure 3 above my scheduling algorithm results for turnaround time are only slighter better or worse compared to other baseline and ATL algorithms. This due to the fact that the goal of my scheduling algorithm is to optimise resource utilisation and cost minimisation as much as possible. In doing so turnaround time will be longer due to the way I developed the algorithm to use as few servers as possible to reduce cost. I will now discuss the pros and cons of my scheduling algorithm.

Pros:

- Reduces cost by making sure all the servers being used are utilised to their capacity.
- Reduces cost by checking every time the server tells the client a job has been completed if there are any jobs on that server, if not the client tells the server to terminate that specific server.
- Improves resource utilisation by sorting available servers by their core count from smallest to highest as those servers would have a lower rental cost and schedules the current job to the server with lowest core count capable of running the job.
- Improves resource utilisation by sending jobs received from the server back into the job queue if it does not satisfy the above requirement. Or there are no servers available to run the job.
- Helps minimise the impact on turnaround time by having a counter that checks how many times as job has been send back to the queue. After a certain period of time the client will schedule the job to any capable server.

Cons:

- Increase in turnaround time, as the client can terminate a server if there are not jobs running on it. But if the server that was terminated is now needed to complete another job it will need to boot up thus increasing turnaround time.
- Jobs can get resent back to the job queue if there is no available server which will increase wait and turnaround time.

Conclusion

The goal of this assessment was to develop a scheduling algorithm that optimises one or more performance metrics such as resource utilisation, turnaround time and total rental cost. The goal of my algorithm was to optimise both resource utilisation and total rental cost. My algorithm works by scheduling jobs to the smallest available server based on core count that can run the job. My Algorithm is also capable of sending jobs back to the job queue until either a job has an available server or after some time a job has a capable server to be scheduled to. The algorithm also checks after every job completion if the server that has just completed the job has any more jobs running on it. If not, the algorithm will tell the server to terminate that particular server to save costs. In the evaluation section of this report, I compared my scheduling algorithm to three baseline algorithms and ATL. As shown in the figures my scheduling algorithm outperformed the other algorithms on average across all test configurations for the performance metrics, resource utilisation and total rental cost. Which is the goal of my algorithm. My scheduling algorithm was only outperformed by the other algorithms on the performance metric turnaround time. This is expected as my scheduling algorithm is designed to optimise resource utilisation and cost at the expense of turnaround time which is the only drawback of my algorithm overall.

My Github

https://github.com/AdamFulton/COMP3100StageTwo

References

[1]"What is Turnaround Time (TAT)? - Definition from Techopedia", *Techopedia.com*, 2021. [Online]. Available: https://www.techopedia.com/definition/23798/turnaround-time-tat. [Accessed: 31- May- 2021].

[2]"Stage 2:Design andimplementation of a new scheduling algorithm", *Content.ilearn.mq.edu.au*, 2021. [Online]. Available: Ilearn COMP3100 Distributed Systems [Accessed: 31- May- 2021].