COMP3100 Assignment 2: Improved-Job-Scheduler

By: Hubert Hartan (45543003)

Introduction

Job scheduling is a process that is important for running operations in Distributed Systems, this importance is the reason why efficiency is required from the scheduling algorithm. In order to achieve this, the project will build upon Stage 1's code base and improve on it in order to achieve a better result.

Overall, Stage 2 of this project is to design a new scheduling algorithm that fits the efficiency requirement and other criteria. To order to achieve this it is required to improve on at least one of the following performance metrics:

- **Minimisation of total rental cost**
  (the total cost required to run a server to schedule jobs)
- **Minimisation of average turnaround time**
  (the time taken to complete a job, from start to finish)
- **Maximisation of average resource utilisation**
  (the average number of cores being used in servers due to scheduling jobs)

Problem Definition

The main overall focus and goal of the project is to create a new scheduling algorithm that fulfills the efficiency requirement. In order to achieve this efficiency measuring the above performance metrics is required. This new algorithm is then expected to outperform at least one of the three baseline algorithms (First Fit, Best Fit, Worst Fit) in said performance metrics. However, these performance metrics can clash with one another as improving one my cause another to become worse.

The second problem is the fact that the new algorithm should outperform allToLargest (ATL) in terms of turnaround time and it is not required for the new algorithm to outperform ATL in any other metric. This creates a path for us to take, by focusing on outperforming ATL simply on turnaround time but outperforming the baseline algorithms in the other metrics.

Algorithm Description

```java
//The new scheduling algorithm
//Based on finding the minimum wait time of a server
//and prioritising the jobs there
public String newAlgo(ArrayList<Server> servers, Job job){
    Server minTime = servers.get(0);

        for (int i = 0; i < servers.size(); i++) {
            if (minTime.getwait()> servers.get(i).getwait()) {
                minTime = servers.get(i);
            }
        }

    return "SCHD "+ job.getId()+" "+minTime.getType()+" "+minTime.getId();
}
```

Figure 1: The new algorithm's code

As shown above, the new algorithm is simple and straightforward. It simply iterates through the list of all servers and select the one with the smallest waiting time. Doing so would mean that the jobs allocated on the selected servers would have a smaller run time as they do not have to wait for other jobs for too long. Compared to the baseline algorithms the new algorithm reduces the average rental cost as less time is taken meaning less cost per time unit, and it also improves resource utilisation as the cores in each server are used more effectively.

However, in terms of turnaround time it only beats worst fit, but it does fit the criteria of improving on the turnaround time when compared to ATL. Shown below is a test run of the new algorithm using a configuration file from Stage 1 of the project. I selected this particular configuration file since it was previously used to test the old algorithm (ATL) and I believe it is a good benchmark.

```
# -----------------------------------------------------------------------------
# 20 tiny servers used with a utilisation of 91.83 at the cost of $59.81
# 20 tiny2 servers used with a utilisation of 57.07 at the cost of $43.41
# 20 small servers used with a utilisation of 86.53 at the cost of $112.43
# 14 small2 servers used with a utilisation of 55.15 at the cost of $54.56
# 20 medium servers used with a utilisation of 88.91 at the cost of $223.63
# 6 medium2 servers used with a utilisation of 28.66 at the cost of $45.51
# 20 large servers used with a utilisation of 80.24 at the cost of $481.36
# 9 large2 servers used with a utilisation of 51.99 at the cost of $156.20
# 20 xlarge servers used with a utilisation of 80.08 at the cost of $903.86
# 14 xlarge2 servers used with a utilisation of 56.83 at the cost of $555.71
# ----------------------------------- [ Summary ] -----------------------------------
# actual simulation end time: 165147, #jobs: 2620 (failed 0 times)
# total #servers used: 163, avg util: 73.01% (ef. usage: 77.18%), total cost: $2636.47
# avg waiting time: 4003, avg exec time: 4506, avg turnaround time: 8509
```

Figure 2: Running the new Algorithm with "ds-S1-config06--wk6.xml"

Implementation

In the previous stage of the project, my group and I went with a 2D array of integers in order to store server information. However, it became apparent to me that maintaining and continuing with that method is not optimal. There will be difficulties of arranging the code and changing data structures.

To solve this, I adopted the popular method of creating a class for both servers and jobs. I used these new classes in order to store server and job information, respectively.

```
Server(String type, int id, String state, int bootupTime, int coreCount, int memory, int disk,int waitTime, int
    this.type = type;
    this.id = id;
    this.state = state;
    this.coreCount = coreCount;
    this.memory = memory;
    this.disk = disk;
    this.waitTime = waitTime;
    this.runTime = runTime;
}
```

Figure 3: The Server class constructor

Evaluation

Turnaround time					
Config	ATL	FF	BF	WF	Yours
config100-long-high.xml	672786	2428	2450	29714	4010
config100-long-low.xml	316359	2458	2458	2613	4006
config100-long-med.xml	679829	2356	2362	10244	3917
config100-med-high.xml	331382	1184	1198	12882	1955
config100-med-low.xml	283701	1205	1205	1245	1972
config100-med-med.xml	342754	1153	1154	4387	1951
config100-short-high.xml	244404	693	670	10424	1160
config100-short-low.xml	224174	673	673	746	1153
config100-short-med.xml	256797	645	644	5197	1135
config20-long-high.xml	240984	2852	2820	10768	3890
config20-long-low.xml	55746	2493	2494	2523	3978
config20-long-med.xml	139467	2491	2485	2803	3826
config20-med-high.xml	247673	1393	1254	8743	1888
config20-med-low.xml	52096	1209	1209	1230	1918
config20-med-med.xml	139670	1205	1205	1829	1932
config20-short-high.xml	145298	768	736	5403	1531
config20-short-low.xml	49299	665	665	704	1121
config20-short-med.xml	151135	649	649	878	1075
Average	**254086.33**	**1473.33**	**1462.83**	**6240.72**	**2356.56**
Normalised (ATL)	1.0000	0.0058	0.0058	0.0246	0.0093
Normalised (FF)	172.4568	1.0000	0.9929	4.2358	1.5995
Normalised (BF)	173.6947	1.0072	1.0000	4.2662	1.6110
Normalised (WF)	40.7143	0.2361	0.2344	1.0000	0.3776
Normalised (AVG [FF,BF,WF])	83.0629	0.4816	0.4782	2.0401	0.7704

Figure 4: Turnaround time for all algorithms

As shown on Figure 4, the new algorithm outperforms the worst fit algorithm and ATL algorithm in turnaround time (as lower turnaround time is better). This fits the provided criteria.

Resource utilisation

Config	ATL	FF	BF	WF	Yours
config100-long-high.xml	100.0	83.58	79.03	80.99	92.4
config100-long-low.xml	100.0	50.47	47.52	76.88	63.02
config100-long-med.xml	100.0	62.86	60.25	77.45	77.82
config100-med-high.xml	100.0	83.88	80.64	89.53	92.26
config100-med-low.xml	100.0	40.14	38.35	76.37	56.64
config100-med-med.xml	100.0	65.69	61.75	81.74	76.27
config100-short-high.xml	100.0	87.78	85.7	94.69	92.68
config100-short-low.xml	100.0	35.46	37.88	75.65	48.06
config100-short-med.xml	100.0	67.78	66.72	78.12	77.0
config20-long-high.xml	100.0	91.0	88.97	66.89	95.92
config20-long-low.xml	100.0	55.78	56.72	69.98	70.7
config20-long-med.xml	100.0	75.4	73.11	78.18	85.51
config20-med-high.xml	100.0	88.91	86.63	62.53	95.02
config20-med-low.xml	100.0	46.99	46.3	57.27	58.63
config20-med-med.xml	100.0	68.91	66.64	65.38	79.36
config20-short-high.xml	100.0	89.53	87.6	61.97	97.04
config20-short-low.xml	100.0	38.77	38.57	52.52	44.03
config20-short-med.xml	100.0	69.26	66.58	65.21	73.93
Average	**100.00**	**66.79**	**64.94**	**72.85**	**76.46**
Normalised (ATL)	1.0000	0.6679	0.6494	0.7285	0.7646
Normalised (FF)	1.4973	1.0000	0.9724	1.0908	1.1448
Normalised (BF)	1.5398	1.0284	1.0000	1.1218	1.1774
Normalised (WF)	1.3726	0.9168	0.8914	1.0000	1.0495
Normalised (AVG [FF,BF,WF])	1.4664	0.9794	0.9523	1.0683	1.1212

Figure 5: Resource Utilisation for all algorithms

Figure 5 shows that the new algorithm outperforms all the baseline algorithms in terms of average resource utilisation (as larger resource utilisation is better). We have passed the criteria as the new algorithm is not required to outperform the ATL algorithm in terms of resource utilisation.

Total rental cost

Config	ATL	FF	BF	WF	Yours
config100-long-high.xml	620.01	776.34	784.3	886.06	722.13
config100-long-low.xml	324.81	724.66	713.42	882.02	499.27
config100-long-med.xml	625.5	1095.22	1099.21	1097.78	812.46
config100-med-high.xml	319.7	373.0	371.74	410.09	358.27
config100-med-low.xml	295.86	810.53	778.18	815.88	534.17
config100-med-med.xml	308.7	493.64	510.13	498.65	385.16
config100-short-high.xml	228.75	213.1	210.25	245.96	221.15
config100-short-low.xml	225.85	498.18	474.11	533.92	362.62
config100-short-med.xml	228.07	275.9	272.29	310.88	234.61
config20-long-high.xml	254.81	306.43	307.37	351.72	311.05
config20-long-low.xml	88.06	208.94	211.23	203.32	146.08
config20-long-med.xml	167.04	281.35	283.34	250.3	237.79
config20-med-high.xml	255.58	299.93	297.11	342.98	302.62
config20-med-low.xml	86.62	232.07	232.08	210.08	161.61
config20-med-med.xml	164.01	295.13	276.4	267.84	257.86
config20-short-high.xml	163.69	168.7	168.0	203.66	170.34
config20-short-low.xml	85.52	214.16	212.71	231.67	190.84
config20-short-med.xml	166.24	254.85	257.62	231.69	248.41
Average	**256.05**	**417.90**	**414.42**	**443.03**	**342.02**
Normalised (ATL)	1.0000	1.6321	1.6185	1.7303	1.3358
Normalised (FF)	0.6127	1.0000	0.9917	1.0601	0.8184
Normalised (BF)	0.6178	1.0084	1.0000	1.0690	0.8253
Normalised (WF)	0.5779	0.9433	0.9354	1.0000	0.7720
Normalised (AVG [FF,BF,WF])	0.6023	0.9830	0.9748	1.0421	0.8045

Figure 6: Total Rental Cost for all algorithms

Shown above in figure 6, the new algorithm outperforms the baseline algorithms in terms of total rental cost (lower rental cost is better). We passed the criteria as the new algorithm is not required to outperform the ATL algorithm in this particular metric.

As such, the new algorithm has passed the evaluation and all the necessary requirements and criteria. This is proof as shown by the 3 provided figures of the result of the test script.

Conclusion

To conclude, the new algorithm that I implemented has met all the necessary criteria set out. I have found plenty of difficulty a long the way, with a main one being coming up with a new algorithm. I have also found out that it is impossible to satisfy all metrics as some metrics will clash with one another, at the end of the day it is simply a matter of perspective and what each person finds as the bigger priority. If I were to re attempt the project, I would definitely start it earlier and not underestimate it as much as I have.

References

[1]GitHub Repository of the Project: https://github.com/HubertHartan/COMP3100-Group46