Stage2 Report

Project title	Cloud job scheduler
Student name	WENMING ZHAO
Student ID	45803080

Introduction

The task of the second stage is to create an algorithm for scheduling jobs. In this stage, specific job groups will be provided, and reasonable allocation of these job groups is the ultimate goal. The algorithms that can be considered are all to the largest, first fit, best fit and worst fit, or better. Client. java will be used for testing in this stage, and wf2.java and ff.Java can only run under the condition of jobCount < 20, so they are not tested, only used to explain the algorithm. This report will introduce the algorithm Client.java used, the reasons for choosing this algorithm, and the advantages and disadvantages of this algorithm compared with FF, BF and WF.

Problem definition

In the second stage, we need to create an algorithm that can achieve the best balance of turnaround time, resource utilization and server rental cost. Because these goals conflict, for example, reducing the cost of server leasing may increase the turnaround time, so the specific algorithm should be selected according to the needs.

In client.java, the all to the largest algorithm is used because it is most easily designed, although this allocation does not take advantage in turnaround time and server lease costs, but its resource utilization is excellent. In ff.java, The algorithm used is first fit which can reduce turnaround time obviously. And worst fit was used in wf2.java, this algorithm will improve the utilization of resources effectively.

Algorithm description

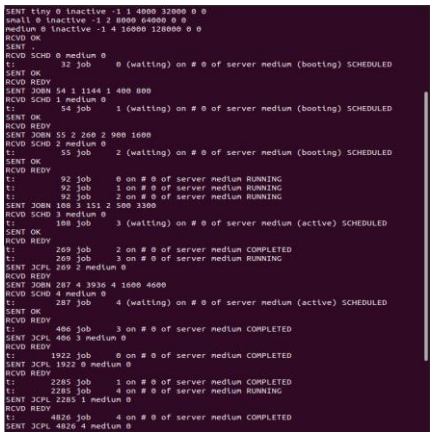
All to the largest is to allocate all jobs to the largest server. For each job, the state of the server will not be considered. If the server is occupied and there are no redundant cores, the job will wait until the idle resources are enough for the job to use. This algorithm has a obvious advantage on improving the resource allocation rate, but has no advantage in reducing the server turnover. For first fit, it allocates jobs to the first optional server in the received server list, which will achieve the purpose of fast allocation, but it may lead to higher resource allocation rate or server rent. When the first server is occupied, the server is assigned to the next server,

and so on. When all servers are occupied and there are not enough resources to allocate jobs, the job will wait in the first optional server until the resources are allowed to continue to allocate. For wrong fit, the server will assign all jobs to the last server in the descending list of servers, that is, the largest server. When the largest server is occupied, the job is assigned to the previous server, and so on. When all servers are occupied, jobs will queue on the optional largest server.

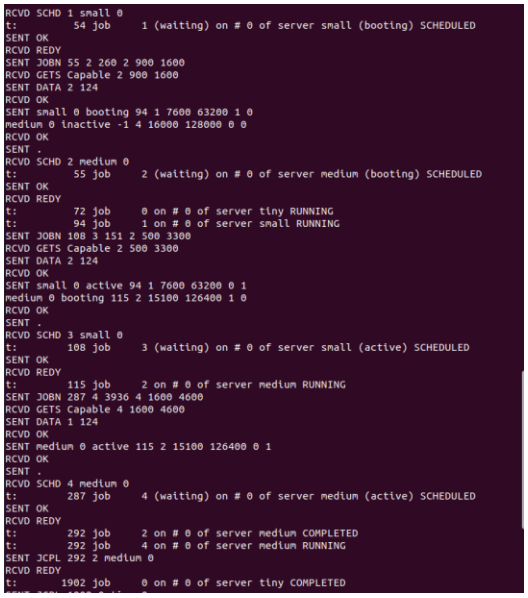
Example(wk9.xml):

```
<!-- generated by: Y. C. Lee --
<config randomSeed="65535">
<servers>
<server type="tiny" limit="1" bootupTime="40" hourlyRate="0.4" coreCount="1" memory="4000" disk="32000"/>
<server type="small" limit="1" bootupTime="40" hourlyRate="0.4" coreCount="2" memory="8000" disk="64000"/>
<server type="medium" limit="1" bootupTime="60" hourlyRate="0.8" coreCount="4" memory="16000" disk="128000"/>
</servers>
<jobs>
<job type="short" minRunTime="1" maxRunTime="60" populationRate="30"/>
<job type="medium" minRunTime="61" maxRunTime="600" populationRate="40"/>
<job type="long" minRunTime="601" maxRunTime="3600" populationRate="30"/>
</jobs>
<workload type="unknown" minLoad="20" maxLoad="60"/>
<termination>
<condition type="endtime" value="86400"/>
<condition type="jobcount" value="5"/>
</termination>
</config>
```

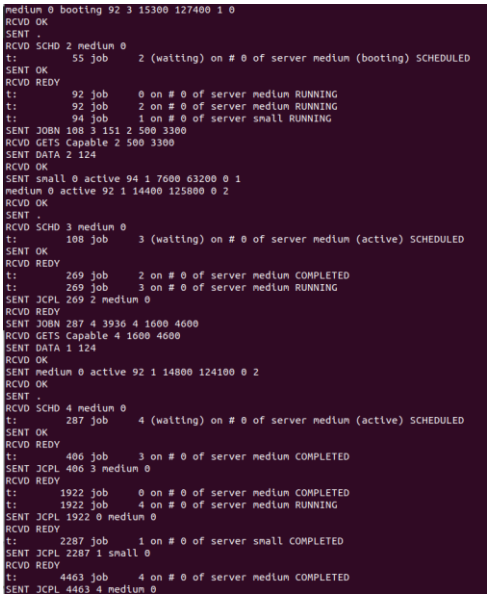
All To The Largest in Client.java



First Fit in ff.java



Worst Fit in wf2.java



Implementation details

In client.java, the required variables are created first, and then the constructor is used to read the information returned by the server. Next, the constructor creates and reads and writes the log file. Then use functions to connect and communicate with the server and client as needed. When the client receives the job information, the job information will be split by white space, and the data will be stored in the string array for use. After the client receives the list of available servers from the server, the information will be saved to the string array, and then the

server list will be saved according to the server.java class. The client allocates the received jobs according to the algorithm and repeatedly confirms whether the server has jobs to allocate. When the client receives the notification that the server has no job, the connection between the server and the client will be interrupted.

Evaluation

Figure1

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 (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

Improvement: 46.64%

Fugure2

Turnaround time

Config	ATL	FF	BF	WF	Yours
config100-long-high.xml	672786	2428	2450	29714	672786
config100-long-low.xml	316359	2458	2458	2613	316359
config100-long-med.xml	679829	2356	2362	10244	679829
config100-med-high.xml	331382	1184	1198	12882	331382
config100-med-low.xml	283701	1205	1205	1245	283701
config100-med-med.xml	342754	1153	1154	4387	342754
config100-short-high.xml	244404	693	670	10424	244404
config100-short-low.xml	224174	673	673	746	224174
config100-short-med.xml	256797	645	644	5197	1434
config20-long-high.xml	240984	2852	2820	10768	240984
config20-long-low.xml	55746	2493	2494	2523	55746
config20-long-med.xml	139467	2491	2485	2803	139467
config20-med-high.xml	247673	1393	1254	8743	247673
config20-med-low.xml	52096	1209	1209	1230	52096
config20-med-med.xml	139670	1205	1205	1829	139670
config20-short-high.xml	145298	768	736	5403	145298
config20-short-low.xml	49299	665	665	704	49299
config20-short-med.xml	151135	649	649	878	151135
Average	254086.33	**1473.33**	**1462.83**	**6240.72**	239899.50
Normalised (FF)	172.4568	1.0000	0.9929	4.2358	162.8277
Normalised (BF)	173.6947	1.0072	1.0000	4.2662	163.9965
Normalised (WF)	40.7143	0.2361	0.2344	1.0000	38.4410

Improvement: -7742.51%

Total rental cost

Config	ATL	FF	BF	WF	Yours
config100-long-high.xml	620.01	776.34	784.3	886.06	620.01
config100-long-low.xml	324.81	724.66	713.42	882.02	324.81
config100-long-med.xml	625.5	1095.22	1099.21	1097.78	625.5
config100-med-high.xml	319.7	373.0	371.74	410.09	319.7
config100-med-low.xml	295.86	810.53	778.18	815.88	295.86
config100-med-med.xml	308.7	493.64	510.13	498.65	308.7
config100-short-high.xml	228.75	213.1	210.25	245.96	228.75
config100-short-low.xml	225.85	498.18	474.11	533.92	225.85
config100-short-med.xml	228.07	275.9	272.29	310.88	2.33
config20-long-high.xml	254.81	306.43	307.37	351.72	254.81
config20-long-low.xml	88.06	208.94	211.23	203.32	88.06
config20-long-med.xml	167.04	281.35	283.34	250.3	167.04
config20-med-high.xml	255.58	299.93	297.11	342.98	255.58
config20-med-low.xml	86.62	232.07	232.08	210.08	86.62
config20-med-med.xml	164.01	295.13	276.4	267.84	164.01
config20-short-high.xml	163.69	168.7	168.0	203.66	163.69
config20-short-low.xml	85.52	214.16	212.71	231.67	85.52
config20-short-med.xml	166.24	254.85	257.62	231.69	166.24
Average	256.05	**417.90**	**414.42**	**443.03**	243.50
Normalised (FF)	0.6127	1.0000	0.9917	1.0601	0.5827
Normalised (BF)	0.6178	1.0084	1.0000	1.0690	0.5876
Normalised (WF)	0.5779	0.9433	0.9354	1.0000	0.5496

Improvement: 42.72%

Figure3

```
Final results:
2.1: 1/1
2.2: 1/1
2.3: 0/1
2.4: 0/6
```

Figure4

The test cases for Client.java are 18 test cases provided in DS-SIM folder, which are in the other folder of the config folder. The test used is the binary file “test_results” provided by ilearn. According to figure 1, we can easily find that the algorithm in client. Java can make the resource utilization rate reach an amazing 100%, which makes it far superior to the other three algorithms in improving the resource utilization rate. However, the reason for high resource utilization is that all jobs are assigned to one server, which means super long queue and turnaround time. According to figure 2, we find that its turnaround time is significantly longer than the other three algorithms. As the above analysis shows, all jobs are allocated to the same server, and a lot of time is spent on queuing, while other algorithms allocate jobs to multiple servers, so the algorithm in the Client.java does not have any advantage in turnround time. According to figure 3, this algorithm can achieve great advantages in reducing the server rental price. This is because this algorithm greatly increases the resource utilization, which makes the server always in full load, so in the same time, the efficiency of a single server in this algorithm is far higher than other algorithms. Due to the high efficiency, if the rent of each server is

proportional to the load capacity, this algorithm will greatly reduce the rent consumption of the server.

Conclusion

In a word, all to the largest can greatly improve resource utilization and significantly reduce server rental costs, but it can also significantly extend the turnaround time. First fit can significantly reduce turnaround time, and worst fit can relatively reduce server rental costs. It is impossible to achieve perfect control of server rental cost, turnaround time and resource utilization, so the algorithm should be selected according to the demand.

References

GitHub repository: https://github.com/WenmingZH/comp3100stage2