Efficient Cloud based job scheduling algorithm

Name: Bruce Liu
SID: 45958114

Introduction:

The primary goal of project two is essentially to generate an algorithm from project one that provides a better scheduling decision from the provided algorithms of best fit, worst fit and first fit in terms of resource utilization, resource usage or turnaround over. As there is no arbitrary judgement value for the results of the self made algorithm to be compared against, the ultimate goal is to provide an algorithm that is comparatively better in terms of job scheduling for all three algorithms. This can be a marginal improvement all around to any form of drastic improvement in one area but the main point is to reduce the unnecessary or drastic sacrifice of one area of improvement for a minor improvement in another. This report shall be the summarisation of the efforts placed into the project and acts as a documentation for the provision of an explanation for all the decisions behind any changes including reasoning in relation to both the project and real world applications.

Problem definition:

Cloud computing is a heavily utilised computing paradigm by modern society due to several functions pertaining to a pay-as-you-go scheme and scalability. But it comes with certain issues in terms of overall cost and power efficiency where once employed incorrectly, can massively increase the overall cost and delay in the running of different applications and therefore heavily affect both profit margins and upkeep costs. This can stem from many factors but one of the primary ones is currently job scheduling and how cloud servers should be rented in order to handle jobs of different complexity that require different levels of computational resources and therefore must be rented at different costs.

The penultimate goal of this assignment is to improve one or all of the following;

1. **Resource utilization**
2. **Resource cost**
3. **Turnaround time**

Using project 1 as a basis, my current goal is to balance out the turnaround time gained from utilising the all to largest algorithm so that it is no longer so monumentally detrimental in comparison to the baseline algorithms. It must be able to majorly decrement the turnaround time till a point where it is at minimum marginally better or of equal standard in comparison to one of the scheduling algorithms whether it would be the Best Fit, Worst Fit or First fit algorithm. Whilst that is one goal, another concurrent goal for the algorithm is to form an improvement based around the resource utilisation and resource usages. As the assignment is arbitrary in nature, there is no defined level of improvement hence my goal is to seek and match or minor improvement to turnaround time and a greater improvement in resource utilization and resource cost. This is so that in this trade off, the major focus for many profit organisations is as listed, for profit. Resource utilization and usage both primarily combine to improve overall profits for

many companies as the less resources used or the better they are used, then the more decrease in costs are to be expected. Therefore the saved liquid assets can be transferred directly to a different investment project instead of simply being wasted on a project for no reason, even a single cent saved in resource costs eventually add up to the sum of a greater amount as hour rental costs will decrement and further resource utilization will free up servers for other jobs or even not require them at all, therefore directly impacting cost and time.

Algorithm description:

Configuration file ds-config01--wk9, servers are listed as follows;

type	limit	boot up time	hourly rate	Core count	memory	disk
tiny	1	40	0.4	1	4000	32000
small	1	40	0.4	2	8000	64000
medium	1	60	0.8	4	16000	128000

The jobs to be scheduled are as follows;

Submit time	ID	Est runtime	Core	Memory	Disk
50	0	221	1	2000	30000
121	1	73	1	2000	20000
167	2	557	2	50000	40000
231	3	1021	4	2000	60000

Time:
50 - Job 0 → medium
110 - Job 0 running (server finished booting)
121 - Job 1 → small
161 - Job 1 running (server finished booting)
167 - Job 2 → medium (waiting)
231 - Job 3 → medium (waiting)
234 - Job 1 finished
331 - Job 0 finished, Job 2 running
888 - Job 2 finished, Job 3 running
1909 - Job 3 finished

The jobs will begin scheduling as follows, once the server sends back a job, depending on the attached message, the client will either ask the server to resend the job or take it and send it to a separate function. The function will hold the job as a single string and parse it into multiple

separate strings based on white spacing. Then the function will take the job core, memory and disk and using that information, send it to the GETS Capable line in order to list all possible servers that can schedule that job. The servers will individually be stored into an arrayList and sent to a separate function that will shift through all the available servers in order to find an optimal one. The idea is for it to schedule jobs based on the current jobs running or waiting on that server, starting from the largest server, and returning the first one that has no jobs running or waiting. Failing to do that, return the largest server that currently has the fewest jobs running or waiting and once that is done, then simply schedule that job on the specified server, this will continue until the NONE message has been received.

Theoretically it would be an optimal decision for resources to schedule jobs to large servers in order to decrease running time and rental costs. Additionally in order to improve turnaround times, multiple servers will start up concurrently as instead of having many small servers that run many jobs, scheduling jobs to large servers at once is more efficient in that it allows for less job run time with less cost and increased utilization, also the more concurrently the server startup times are run, the less turnaround time will occur. Therefore to balance out the two factors, the servers utilised are as many largest possible ones that run individual jobs. When adding resource utilization into the considerations, the algorithm intends to constantly comb through the server list dynamically and therefore explore when a job has finished running on a server and instantly assign another job to that server therefore maximizing utilization, alternatively it also checks for waiting jobs and always will leave a job waiting on that server should there be a job running already but having no waiting jobs. This is done in order to ensure that the provided servers are always in use and always available and therefore increment the utilization level as close to 100% as possible.

Implementation details:

In terms of implementation the current algorithm follows an exact copy of the foundational technologies and libraries in comparison to project 1. The initial structure also is similar to the first project in which there is the handshake protocol of the user's username and the HELO and REDY token. But from then on, it diverts as it no longer utilises the GETS All command but instead uses the more improved GETS Capable command instead. The new major functions of the code is as follows;

1. **ServerState -** Requires an arraylist containing all the servers, returns a specific server based on factors of the current jobs running on the server and whether there are any jobs waiting to be performed, otherwise schedule the job to the largest server.
2. **ScheduleJob -** Requires the printwriter and bufferedreader along with a string containing the current job, will utilise the GETS Capable function in order to list all servers capable of running the job store in the string and schedule it to a specific server in that list based on the ServerState function.

The flow of data is similar to project one with the exception of the server scheduling decisions, in order to balance out the turnaround time, the jobs are scheduled to a mix of different servers based on the factor of whether they have a job running or waiting on them, otherwise they can be sent to a predefined largest server.

Evaluation:

Configuration files utilized includes configs from the others file;
Ds-sim-master → configs → other
Results are as follows;

Turnaround time

Config	ATL	FF	BF	WF	Yours
config100-long-high.xml	672786	2428	2450	29714	6526
config100-long-low.xml	316359	2458	2458	2613	3289
config100-long-med.xml	679829	2356	2362	10244	4043
config100-med-high.xml	331382	1184	1198	12882	3564
config100-med-low.xml	283701	1205	1205	1245	1580
config100-med-med.xml	342754	1153	1154	4387	2031
config100-short-high.xml	244404	693	670	10424	3925
config100-short-low.xml	224174	673	673	746	1026
config100-short-med.xml	256797	645	644	5197	1850
config20-long-high.xml	240984	2852	2820	10768	3725
config20-long-low.xml	55746	2493	2494	2523	2766
config20-long-med.xml	139467	2491	2485	2803	2982
config20-med-high.xml	247673	1393	1254	8743	2005
config20-med-low.xml	52096	1209	1209	1230	1374
config20-med-med.xml	139670	1205	1205	1829	1448
config20-short-high.xml	145298	768	736	5403	1626
config20-short-low.xml	49299	665	665	704	773
config20-short-med.xml	151135	649	649	878	844
Average	**254086.33**	**1473.33**	**1462.83**	**6240.72**	**2520.94**
Normalised (ATL)	1.0000	0.0058	0.0058	0.0246	0.0099
Normalised (FF)	172.4568	1.0000	0.9929	4.2358	1.7110
Normalised (BF)	173.6947	1.0072	1.0000	4.2662	1.7233
Normalised (WF)	40.7143	0.2361	0.2344	1.0000	0.4040
Normalised (AVG [FF,BF,WF])	83.0629	0.4816	0.4782	2.0401	0.8241

Resource utilisation

Config	ATL	FF	BF	WF	Yours
config100-long-high.xml	100.0	83.58	79.03	80.99	97.83
config100-long-low.xml	100.0	50.47	47.52	76.88	85.66
config100-long-med.xml	100.0	62.86	60.25	77.45	87.14
config100-med-high.xml	100.0	83.88	80.64	89.53	98.09
config100-med-low.xml	100.0	40.14	38.35	76.37	73.08
config100-med-med.xml	100.0	65.69	61.75	81.74	91.31
config100-short-high.xml	100.0	87.78	85.7	94.69	96.47
config100-short-low.xml	100.0	35.46	37.88	75.65	71.54
config100-short-med.xml	100.0	67.78	66.72	78.12	94.54
config20-long-high.xml	100.0	91.0	88.97	66.89	84.18
config20-long-low.xml	100.0	55.78	56.72	69.98	74.17
config20-long-med.xml	100.0	75.4	73.11	78.18	82.94
config20-med-high.xml	100.0	88.91	86.63	62.53	70.75
config20-med-low.xml	100.0	46.99	46.3	57.27	72.85
config20-med-med.xml	100.0	68.91	66.64	65.38	71.71
config20-short-high.xml	100.0	89.53	87.6	61.97	76.47
config20-short-low.xml	100.0	38.77	38.57	52.52	77.26
config20-short-med.xml	100.0	69.26	66.58	65.21	69.96
Average	**100.00**	**66.79**	**64.94**	**72.85**	**82.00**
Normalised (ATL)	1.0000	0.6679	0.6494	0.7285	0.8200
Normalised (FF)	1.4973	1.0000	0.9724	1.0908	1.2277
Normalised (BF)	1.5398	1.0284	1.0000	1.1218	1.2626
Normalised (WF)	1.3726	0.9168	0.8914	1.0000	1.1255
Normalised (AVG [FF,BF,WF])	1.4664	0.9794	0.9523	1.0683	1.2024

Total rental cost

Config	ATL	FF	BF	WF	Yours
config100-long-high.xml	620.01	776.34	784.3	886.06	669.4
config100-long-low.xml	324.81	724.66	713.42	882.02	461.18
config100-long-med.xml	625.5	1095.22	1099.21	1097.78	768.45
config100-med-high.xml	319.7	373.0	371.74	410.09	335.07
config100-med-low.xml	295.86	810.53	778.18	815.88	500.51
config100-med-med.xml	308.7	493.64	510.13	498.65	367.9
config100-short-high.xml	228.75	213.1	210.25	245.96	**220.79**
config100-short-low.xml	225.85	498.18	474.11	533.92	366.16
config100-short-med.xml	228.07	275.9	272.29	310.88	237.26
config20-long-high.xml	254.81	306.43	307.37	351.72	290.9
config20-long-low.xml	88.06	208.94	211.23	203.32	158.58
config20-long-med.xml	167.04	281.35	283.34	250.3	220.27
config20-med-high.xml	255.58	299.93	297.11	342.98	290.06
config20-med-low.xml	86.62	232.07	232.08	210.08	151.26
config20-med-med.xml	164.01	295.13	276.4	267.84	224.33
config20-short-high.xml	163.69	168.7	168.0	203.66	**171.46**
config20-short-low.xml	85.52	214.16	212.71	231.67	159.05
config20-short-med.xml	166.24	254.85	257.62	231.69	217.02
Average	**256.05**	**417.90**	**414.42**	**443.03**	**322.76**
Normalised (ATL)	1.0000	1.6321	1.6185	1.7303	1.2606
Normalised (FF)	0.6127	1.0000	0.9917	1.0601	0.7723
Normalised (BF)	0.6178	1.0084	1.0000	1.0690	0.7788
Normalised (WF)	0.5779	0.9433	0.9354	1.0000	0.7285
Normalised (AVG [FF,BF,WF])	0.6023	0.9830	0.9748	1.0421	0.7592

Comparatively speaking, the algorithm I implemented sacrifices rental costs and utilization to decrease the turnaround time in regards to the all to largest algorithm. My algorithm has the lowest average rental cost in comparison to all algorithms, this is purely based on a corporate perspective as the lower the cost, the more profitable the organization becomes. In comparison, the resource utilization is mostly better than all three algorithms but there is an exception for certain configurations, mainly in those holding a medium number of jobs. This is primarily due to scheduling decisions in which for the algorithm to become at an optimal state, all servers must be opened and operating hence when there are too many servers open and not enough jobs to be shared around, this causes some form of rental cost issue. Finally the turnaround time is much better than all to largest and worst fit, this creates a balance in order to achieve growth in the other factors but not entirely ignore turnaround time. Overall my algorithm improves rental cost and resource utilization whilst also maintaining a certain standard of turnaround time, this is beneficial as it acts as an all around algorithm therefore it is applicable in many general situations. But the drawback is that it requires a certain number of jobs with either high or low resources in order to reach an optimal point, otherwise certain servers would not be utilized for periods of time.

Conclusion:

In summary, my algorithm chooses to ensure at minimum some form of improvement for all factors of resource utilization, rental cost and turnaround time. I have chosen to drastically reduce turnaround time whilst simultaneously improving cost and utilization to a minor extent in order to balance out all three attributes. I have discovered that whilst the idea of having all servers booted is good, a necessity for job number and resource requirements must be fulfilled in order to increase efficiency and this therefore is a consideration to be solved. The suggestion is for this algorithm to be primarily deployed in situations involving many jobs with either high or low resource costs. For future improvements, the scheduling algorithm should be more resource focused as well, scheduling jobs based both on the jobs on the server and what the resource requirements of each job are.

References:

1. **Github -** https://github.com/SneakyDaOne/COMP3100Ass2