COMP3100 Assignment 2: Scheduling Algorithm

Introduction

Distributed systems come in various sizes and quantities, which can be as small as an individual PC to as large as multiple data centres working together in unison. To effectively use such computing resources, we must schedule jobs to be performed as efficiently as possible. Previously we discussed how jobs are dispatched and scheduled in distributed systems such as data centres which is simulated by ds-sim (on the server side). During this process, several performance metrics exist such as execution time, turn around time, resource utilization and costs of execution are to be optimized. The objective of this assignment is to design and implement a scheduling algorithm that allocates jobs to servers based on improving these metrics. It is difficult to optimize for all these metrics as improving one will often lead to worsening in another, thus when implementing such algorithm, the developer(s) and manager must communicate with each other to agree on the sacrificed metric(s) and which metric(s) is to be improved upon in practice.

Problem Definition

When scheduling jobs, we must decide how are they to be assigned to servers to be processed. Our approach depends on what metrics we want to optimize for:

1. Minimization of average turnaround time (TT)
2. Maximizing the average resource utilization (RU)
3. Minimization of server rental cost (CO)

The metrics are defined as:

- TT= (average wait time) / (average execution time)
- CO= (active time/3600) * hourly rate
- RU: first boot time to last job time (only for active time)
    - Active time for specific server/ (Last job on any server- job for specific server time)

However, when optimizing for one metric, another one must be sacrificed, thus the focus on maximizing resource utilization (RU) and improving server rental cost (CO) were chosen for my algorithm at the cost of turnaround time (TT). The main reason why being that I believe that this would be the most realistic approach since cost plays a major role in deciding how the overall system is setup. If we maximize resource utilization, we are getting our money’s worth, increase efficiency and improving rental cost will always be regarded as an advantage. The only scenario where this approach is not suitable is with time sensitive application that requires jobs to be completed as soon as possible. Another reason why I choose these metrics to optimize for is due to it being the easiest to implement compared to turnaround time which was my initial decision. Optimizing for that would require a modified fast/best-fit approach which was considerably harder to achieve compared to the ratio function which has been developed.

Algorithm Description

When designing the algorithm, the design focus was readability, simplicity and focusing on maximizing resource utilization and improving rental cost. To achieve this server with more resources(cores) are given more jobs over smaller servers with less resources. This led to the development of an algorithm that makes use of a ratio function to allocate the jobs.

System Overview:

1. Server handshake (HELO, AUTH, OK, etc)
2. build array list of server object(s)
3. while there are jobs to schedule:
   I. Read new job into object.
   II. [data lines] read all servers capable of doing job into array list.
   III. sort servers by ratio function
   IV. pick best server & add job.
   - rinse and repeat from step 3.

The algorithm is based on the following formula:

[Waiting jobs + 1(If booting*)] / (Number of Cores) -> (we favour servers with more cores)

*we treat booting up as a job.

Which is then used with the ratio function:

- Servers (gets filtered to)->
- Capable servers(sort)->
- Sort(A,B): (A.waiting/A.cores) VS (B.waiting/B.cores) ->
- ordered servers ->
- get list

Figure 1 visualises how this process works in a simple scheduling scenario.

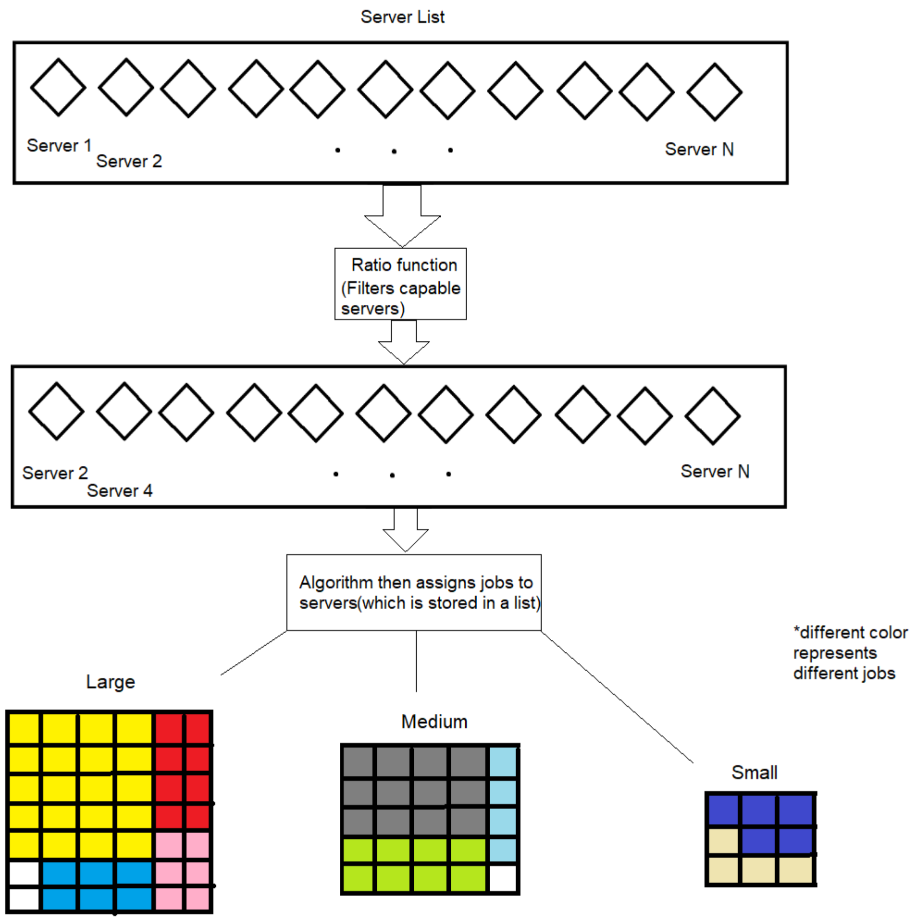

Figure 1

Implementation

The solution was built on top of the previous assignment and was iterated upon to achieve our goal of maximizing RU and improving CO.

The use of various software and technologies include:

- **Ubuntu 20.04**- used as the operating system to run Linux based programs such as ds-sim , used in conjunction with VirtualBox to ensure development can occur on windows based computers.
- **VirtualBox 6.1.18**- Virtualization software that creates a separate environment for the development and execution of the algorithm without interfering the main windows environment in case of fatal errors.
- **Visual Studio Code 1.55.2**- used for developing code for the algorithm. It was chosen because of its user friendliness and helps with readability of the code. Also allows isolated, quick testing within the IDE using bash terminals.
- **Java 16**- the language that the algorithm was written in, and allows the use of java specific libraries such as:
    - **Java.net** – allows a connection between the client and server to share information
    - **Java.io** – needed to develop the read and write functions for the program to communicate between server and client sides
    - **Java.util** – used for implementing arraylists to allow dynamic allocation of job and server scheduling information.

The types of data structures used are:

- **Arraylist(s)**- to store and sort servers and jobs in a list dynamically.
- **Classes** – used for jobs and storage and their associated attributes.

Evaluation

Turnaround time

Config	ATL	FF	BF	WF	Yours
config100-long-high.xml	672786	2428	2450	29714	3261
config100-long-low.xml	316359	2458	2458	2613	2503
config100-long-med.xml	679829	2356	2362	10244	2651
config100-med-high.xml	331382	1184	1198	12882	1842
config100-med-low.xml	283701	1205	1205	1245	1221
config100-med-med.xml	342754	1153	1154	4387	1335
config100-short-high.xml	244404	693	670	10424	1501
config100-short-low.xml	224174	673	673	746	709
config100-short-med.xml	256797	645	644	5197	893
config20-long-high.xml	240984	2852	2820	10768	3058
config20-long-low.xml	55746	2493	2494	2523	2518
config20-long-med.xml	139467	2491	2485	2803	2608
config20-med-high.xml	247673	1393	1254	8743	2099
config20-med-low.xml	52096	1209	1209	1230	1224
config20-med-med.xml	139670	1205	1205	1829	1250
config20-short-high.xml	145298	768	736	5403	2056
config20-short-low.xml	49299	665	665	704	709
config20-short-med.xml	151135	649	649	878	856
Average	**254086.33**	**1473.33**	**1462.83**	**6240.72**	**1794.11**
Normalised (ATL)	1.0000	0.0058	0.0058	0.0246	0.0071
Normalised (FF)	172.4568	1.0000	0.9929	4.2358	1.2177
Normalised (BF)	173.6947	1.0072	1.0000	4.2662	1.2265
Normalised (WF)	40.7143	0.2361	0.2344	1.0000	0.2875
Normalised (AVG [FF,BF,WF])	83.0629	0.4816	0.4782	2.0401	0.5865

Resource utilisation

Config	ATL	FF	BF	WF	Yours
config100-long-high.xml	100.0	83.58	79.03	80.99	96.85
config100-long-low.xml	100.0	50.47	47.52	76.88	76.16
config100-long-med.xml	100.0	62.86	60.25	77.45	92.7
config100-med-high.xml	100.0	83.88	80.64	89.53	95.8
config100-med-low.xml	100.0	40.14	38.35	76.37	68.92
config100-med-med.xml	100.0	65.69	61.75	81.74	95.69
config100-short-high.xml	100.0	87.78	85.7	94.69	93.6
config100-short-low.xml	100.0	35.46	37.88	75.65	57.41
config100-short-med.xml	100.0	67.78	66.72	78.12	94.83
config20-long-high.xml	100.0	91.0	88.97	66.89	78.72
config20-long-low.xml	100.0	55.78	56.72	69.98	74.61
config20-long-med.xml	100.0	75.4	73.11	78.18	73.08
config20-med-high.xml	100.0	88.91	86.63	62.53	84.54
config20-med-low.xml	100.0	46.99	46.3	57.27	64.4
config20-med-med.xml	100.0	68.91	66.64	65.38	67.02
config20-short-high.xml	100.0	89.53	87.6	61.97	90.1
config20-short-low.xml	100.0	38.77	38.57	52.52	61.72
config20-short-med.xml	100.0	69.26	66.58	65.21	80.07
Average	**100.00**	**66.79**	**64.94**	**72.85**	**80.35**
Normalised (ATL)	1.0000	0.6679	0.6494	0.7285	0.8035
Normalised (FF)	1.4973	1.0000	0.9724	1.0908	1.2030
Normalised (BF)	1.5398	1.0284	1.0000	1.1218	1.2372
Normalised (WF)	1.3726	0.9168	0.8914	1.0000	1.1028
Normalised (AVG [FF,BF,WF])	1.4664	0.9794	0.9523	1.0683	1.1782

Total rental cost

Config	ATL	FF	BF	WF	Yours
config100-long-high.xml	620.01	776.34	784.3	886.06	731.97
config100-long-low.xml	324.81	724.66	713.42	882.02	852.78
config100-long-med.xml	625.5	1095.22	1099.21	1097.78	885.99
config100-med-high.xml	319.7	373.0	371.74	410.09	368.25
config100-med-low.xml	295.86	810.53	778.18	815.88	868.32
config100-med-med.xml	308.7	493.64	510.13	498.65	403.75
config100-short-high.xml	228.75	213.1	210.25	245.96	236.12
config100-short-low.xml	225.85	498.18	474.11	533.92	576.68
config100-short-med.xml	228.07	275.9	272.29	310.88	272.53
config20-long-high.xml	254.81	306.43	307.37	351.72	302.93
config20-long-low.xml	88.06	208.94	211.23	203.32	194.82
config20-long-med.xml	167.04	281.35	283.34	250.3	270.37
config20-med-high.xml	255.58	299.93	297.11	342.98	297.59
config20-med-low.xml	86.62	232.07	232.08	210.08	199.18
config20-med-med.xml	164.01	295.13	276.4	267.84	271.54
config20-short-high.xml	163.69	168.7	168.0	203.66	176.2
config20-short-low.xml	85.52	214.16	212.71	231.67	216.68
config20-short-med.xml	166.24	254.85	257.62	231.69	236.26
Average	**256.05**	**417.90**	**414.42**	**443.03**	**409.00**
Normalised (ATL)	1.0000	1.6321	1.6185	1.7303	1.5974
Normalised (FF)	0.6127	1.0000	0.9917	1.0601	0.9787
Normalised (BF)	0.6178	1.0084	1.0000	1.0690	0.9869
Normalised (WF)	0.5779	0.9433	0.9354	1.0000	0.9232
Normalised (AVG [FF,BF,WF])	0.6023	0.9830	0.9748	1.0421	0.9621

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

As we can observe from the results, we can see that all test configurations are properly handled and the algorithm performs well in RU in most cases and improves CO (usually a middle ground between the best of middle result) when running the supplied test that uses the 18 sample configurations. When running the test in RU and CO modes the performance improvement is satisfactory (according to the test file supplied) however with respect to TT it does not achieve the improvements required.

In TT tests, we can see that the algorithm performs slightly worse compared to FF and BF but is better than WF and ATL for most configurations and is reflected in the average result of 1794.11 seconds. In RU tests, we can see that the algorithm performs better than the other algorithms (FF and BF) in most configurations, but sometimes gets beaten by WF, overall, we see an improvement from the baseline algorithms with an average of 80.35%. In the CO test we can see that the algorithm has mixed results, sometimes being beaten by BF and less in WF but the overall average lower at $409.

From this we can see that the overall pros

- Performs better than baseline algorithms in terms of resource utilization in most configurations.
- Can improve the rental cost in some cases compared to the baseline algorithms.

The con being:

- Overall, longer turnaround times compared to BF and FF algorithms.

Conclusion

Based on the results we observed just now, we can see that the objective of improving resource utilisation has been achieved and rental cost in some cases. This comes at the expense of turn around time which was worse compared to BF and FF baseline algorithms. In the real world I believe my approach/algorithm would be chosen in most cases where there are no time sensitive jobs required as costs play a major role in how the system is laid out, be more appealing to management and can be a major impact on other key decisions. In time sensitive application I would have tuned for turn around time and gone with a different algorithm probably based on a modified fast/best fit algorithm.

References

Github:

https://github.com/danknewen/COMP3100-assignment-2