Assignment 1
Project report: Stage 2 of 2
COMP3100 Distributed Systems, S1, 2023
SID: 46131434, Name: Ravi Inder Singh

Contents

1 Introduction

This is stage 2 and final stage of the project. The project's aim was to develop understanding of optimal job scheduling in a distributed system. This stage is developed on top of stage 1. In stage 1, I implemented Largest Round Robin (LRR) job scheduling algorithm. However, in this stage, we were given to devise a new job scheduling algorithm that aim to optimise the average turnaround time. As the three components, avg.turnaround time, resource utilisation and rental cost are inter-dependent, change in one element comes with the cost in others. So after careful consideration of all the factors, I came to this new job scheduling algorithm, which I named as "first available then capable" (FATC).

This report is organised in 6 sections. Section 2 aims of defining the problem faced and need for new scheduling algorithm. Section 3 talks about newly devised algorithm. Section 4 is about the implementation of the algorithm. Section 5 contains the comparison between my algorithm and five base line algorithms (FF,BF,FFQ,BFQ and WFQ). Finally we have report conclusion and link to the project's **GitHub** repository.

2 Problem Definition

While job scheduling we look for performance of 5 Baseline algorithms in three different parameters: average turnaround time, rental cost and resource utilisation. The problem that I encountered was that these algorithms perform better in few metrics than each other and none of these was good enough that incorporated the three metrics to the optimal level. For example: BF and FF has lower turnaround time as compared to the WFQ. However, WFQ was using less resources than the rest. These conflicting objectives pave the way for new algorithm i.e., FATC. My algorithm reduce the average turnaround time while using optimal resources.

3 Algorithm Description

This section gives the overview algorithm and its working. Algorithm design is simple and easy to understand. After setting up successful connection, client ask for a job to schedule. It uses the job requirements i.e., cores, memory and disk, to search for the readily available servers and then assign this job to the first server from the list. In case, there are no readily available servers, it looks for the servers that are capable of handling this particular job and then assign this job to the first server in this list. This process is repeated until all the jobs are scheduled.

Following is the diagram of Week09 sample config file when it is scheduled using FATC algorithm. We

got three servers Tiny (1 core), Small (2 cores) and Medium (4 cores). Jobs are indicated by j0, j1 ... j4. Red lines indicate the job submission time, yellow indicates the starting time and green represents the completion time. It is evident from the figure and submission times - 32,54,55 that the first 3 jobs out of 5 were scheduled at readily available servers. When the job j0 was scheduled at 'tiny', this server cease to be readily available and then next job had to be scheduled on either of the two remaining ones. This process continues until no server is readily available and then we look for the capable server and assign the job to it. Average turnaround time was 1839 and 100% resource utilisation.

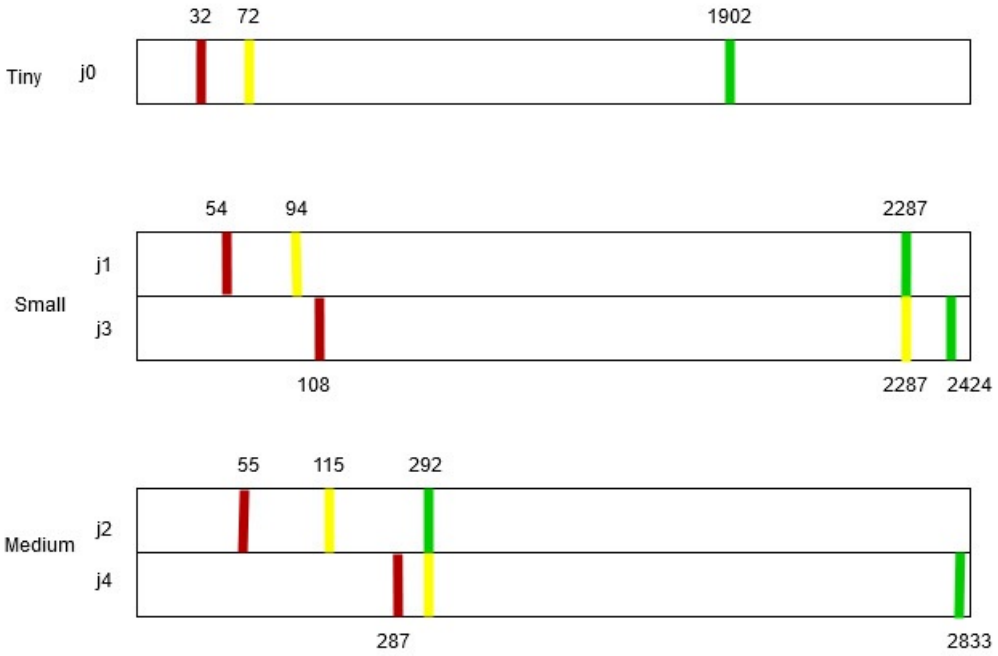

Figure 1: Scheduling of sample config file using FATC

4 Implementation

This algorithm is developed using JAVA and its socket and I/O libraries. Firstly, the connection is set up using three way handshake - **HELO, AUTH username, REDY**. After successful setup, server and java client communicates using data in and out buffer streams. My FATC java client make use of variety of data structures such as - string, int, string array etc. The 'while loop' is driving force of the scheduling it allows client to make further enquiries with the server until all the jobs are scheduled. String arrays are very useful in storing server's response and using by split function on these, we can extract meaningful information.
Following is the pseudo-code of my FATC implementation:
Server's response is SR and Client msg is CR.

1. connection setup
2. HELO - OK, Auth xx - Ok
3. While SR is not none
4. CR - REDY
5. receive either JOBN, JCPL or NONE
6. if received JOBN a b c d e f
7. Send 'GETS Avail d e f'
8. if DATA x not 0
9. schedule job to first listed server
10. else

11. send 'GETS Capable d e f'
12. schedule job on first listed server
13. end if
14. end while loop
15. CR - QUIT
16. SR - QUIT
17. Close the socket

5 Evaluation

Simulation is setup by installing python3 on the Linux machine. Then installing required modules of python3 such as pip. We were provided with the tests for stage2 from the ilearn. After downloading and extracting the tar file in the working folder which has the FATC java client file, we can run it using following command:
python3 ./s2_demo.py "java Stage2Client -a superfast" -n -r results/ref_results.json
where Stage2Client is executable java file.

5.1 Comparison

Following is comparison of FATC algorithm with five baseline algorithms.

Turnaround time

Config	FF	BF	FFQ	BFQ	WFQ	Yours
config12-long-med.xml	2400	2397	2802	2630	6880	2407
config12-med-alt.xml	388	373	813	698	3804	367
config12-med-med.xml	654	653	893	831	4576	653
config12-short-med.xml	61	60	106	100	677	60
config16-long-high.xml	3123	4674	6536	6666	23972	2671
config16-long-med.xml	2548	2562	3997	3778	19538	2556
config16-med-high.xml	3749	5328	6123	5494	23740	1408
config16-short-high.xml	3215	8260	5517	4369	24814	991
config16-short-med.xml	699	851	1952	1972	17139	667
config40-long-high.xml	4506	4164	3642	3568	8461	3369
config40-long-med.xml	3023	3022	3330	3346	11724	3026
config40-med-high.xml	1505	896	940	933	3097	906
config40-med-med.xml	963	972	1237	1232	9123	964
config40-short-high.xml	485	1365	373	301	2817	228
config40-short-med.xml	180	182	198	193	1945	180
Average	**1833.27**	**2383.93**	**2563.93**	**2407.40**	**10820.47**	**1363.53**
Normalised (FF)	1.0000	1.3004	1.3986	1.3132	5.9023	0.7438
Normalised (BF)	0.7690	1.0000	1.0755	1.0098	4.5389	0.5720
Normalised (FFQ)	0.7150	0.9298	1.0000	0.9389	4.2203	0.5318
Normalised (BFQ)	0.7615	0.9903	1.0650	1.0000	4.4947	0.5664
Normalised (WFQ)	0.1694	0.2203	0.2370	0.2225	1.0000	0.1260
Normalised (Average)	0.4581	0.5957	0.6407	0.6016	2.7039	0.3407

Figure 2: Comparing Turnaround time

Resource utilisation

Config	FF	BF	FFQ	BFQ	WFQ	Yours
config12-long-med.xml	69.39	66.87	68.24	66.45	79.22	69.76
config12-med-alt.xml	66.93	63.88	66.46	63.39	49.55	66.90
config12-med-med.xml	66.22	63.06	65.92	62.68	71.52	66.29
config12-short-med.xml	61.56	58.46	61.32	58.18	61.48	61.62
config16-long-high.xml	79.97	74.70	77.70	74.06	66.17	79.98
config16-long-med.xml	68.16	64.71	67.59	63.86	65.06	68.35
config16-med-high.xml	77.45	73.06	76.10	73.14	48.18	78.72
config16-short-high.xml	76.88	71.21	74.44	70.87	50.94	78.62
config16-short-med.xml	66.22	62.42	65.54	62.08	61.40	66.18
config40-long-high.xml	85.81	80.91	86.34	81.81	79.46	87.62
config40-long-med.xml	75.19	69.43	74.92	69.00	65.26	75.21
config40-med-high.xml	83.01	79.51	83.87	79.32	67.73	84.45
config40-med-med.xml	58.92	54.46	58.83	54.53	45.66	58.96
config40-short-high.xml	86.54	79.64	86.68	80.98	78.09	87.22
config40-short-med.xml	77.23	71.66	77.18	71.68	87.92	77.27
Average	**73.30**	**68.93**	**72.74**	**68.80**	**65.18**	**73.81**
Normalised (FF)	1.0000	0.9404	0.9924	0.9387	0.8892	1.0070
Normalised (BF)	1.0633	1.0000	1.0553	0.9981	0.9455	1.0708
Normalised (FFQ)	1.0077	0.9476	1.0000	0.9458	0.8960	1.0147
Normalised (BFQ)	1.0654	1.0019	1.0573	1.0000	0.9473	1.0728
Normalised (WFQ)	1.1246	1.0576	1.1161	1.0556	1.0000	1.1325
Normalised (Average)	1.0503	0.9877	1.0423	0.9858	0.9339	1.0576

Figure 3: Comparing Resource Utilisation

Total rental cost

Config	FF	BF	FFQ	BFQ	WFQ	Yours
config12-long-med.xml	131.37	131.75	136.01	133.20	132.04	132.04
config12-med-alt.xml	113.25	113.19	115.75	115.21	118.77	113.25
config12-med-med.xml	132.82	133.01	134.87	134.60	129.86	132.93
config12-short-med.xml	21.50	21.49	21.68	21.68	21.00	21.50
config16-long-high.xml	589.16	596.10	632.23	632.18	653.30	583.50
config16-long-med.xml	581.93	581.49	600.44	599.28	580.79	582.39
config16-med-high.xml	598.42	598.83	632.22	617.53	696.20	578.73
config16-short-high.xml	594.10	612.40	635.48	626.12	674.29	580.47
config16-short-med.xml	579.43	578.62	592.54	590.29	587.31	579.52
config40-long-high.xml	1244.09	1236.91	1250.35	1239.58	1297.00	1232.80
config40-long-med.xml	1552.56	1567.68	1583.64	1584.85	1552.89	1561.71
config40-med-high.xml	614.35	602.11	605.53	605.93	657.50	600.52
config40-med-med.xml	1281.93	1281.04	1294.66	1293.52	1183.21	1282.44
config40-short-high.xml	600.37	612.79	601.77	597.96	655.30	592.86
config40-short-med.xml	586.94	587.55	592.51	588.60	566.19	587.31
Average	**614.81**	**617.00**	**628.65**	**625.37**	**633.71**	**610.80**
Normalised (FF)	1.0000	1.0036	1.0225	1.0172	1.0307	0.9935
Normalised (BF)	0.9965	1.0000	1.0189	1.0136	1.0271	0.9900
Normalised (FFQ)	0.9780	0.9815	1.0000	0.9948	1.0081	0.9716
Normalised (BFQ)	0.9831	0.9866	1.0052	1.0000	1.0133	0.9767
Normalised (WFQ)	0.9702	0.9736	0.9920	0.9868	1.0000	0.9638
Normalised (Average)	0.9854	0.9889	1.0076	1.0023	1.0157	0.9790

Final results:
2.1: 1/1
2.2: 1/1
2.3: 7/7

Figure 4: Comparing Rental Cost

My algorithm FATC performed a lot better in terms of average turnaround time, it was least among the all baseline algorithms. Optimal use of resources and rental is being observed in the use of FATC.

5.2 Pro and cons

Pro: FATC is fastest among all other given algorithms, with least average turnaround time. It also has the least rental cost associated.
Cons: Resource utilisation is more than 'Worse Fit Queue(WFQ)'.

6 Conclusion

It can be concluded that my own algorithm - FATC use some of the basics of **FF or BF** however, it perform relatively better than these five baseline algorithms. So the required results of reducing average turnaround time and rental cost can be achieved using FATC algorithm.

7 References

Link to the **GitHub** project is https://github.com/Ravi-mq/COMP3100DSAssignment.git