Intelligent Priority Scheduler for Distributed Systems

Group 8
Katrina David (45308748)

Introduction

Priority Scheduling is the process of allocating jobs to the scheduler based on their degree of importance. It is very easy to use, and it means that jobs with a high priority do not need to wait for a long time to be executed. Stage 3 of the Intelligent Priority scheduler involves working individually to design and implement a new scheduling algorithm. This new scheduling algorithm is responsible for optimising one or a number of performance metrics/objectives and constraints. It is possible that the implementation of this new scheduling algorithm may in turn have a negative effect on other optimisation metrics. In this circumstance, this would mean that one or more of these objectives will need to be sacrificed in order for the newly implemented scheduling algorithm to work efficiently and effectively.

Problem Definition/Formulation

The new scheduling algorithm that has been implemented addresses the performance objectives of average utilisation and total cost by sorting the servers based on their state and desirability. The purpose of the algorithm is to utilise the servers that are idle because they are the most desirable as they have no jobs to perform. As such by sorting the servers based on their state, more jobs can be executed by different servers at the same time.

Algorithm Description

The configuration files that have been used to test the implementation of the smartFF scheduling algorithm are the s3sampleconfig which contains seven config.xml files.

The algorithm is designed to sort the servers based on their state and desirability. Servers that are idle are the most desirable because they are on stand-by until a job has been assigned to them for them to execute it. For example, if the state of a server is 2 which means that it is idle, a high ranking of 1 would be returned because the array in the sortByState() function sorts the rankings from lowest to highest. This is illustrated by the flowchart in Figure 1.

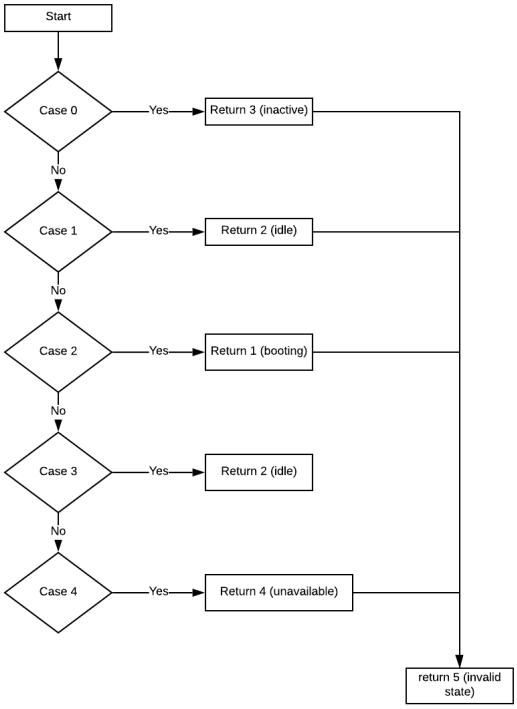

Figure 1: Flowchart of stateRanking

Pseudocode

For a given job ji,

1. Assign the states and return a high ranking based on the state (separate function)
2. Sort the servers based on their most desirable state from lowest to highest (separate function)
3. Obtain sever state information by utilising the sortByState()
4. For each server type i, s_i , from the smallest to the largest
    5. For each server j, s_i,j of server type s_i, from 0 to *limit* - 1 // j is server ID
        6. If server s_i,j has sufficient available resources to run job j_i then
            7. Return s_i,j
        8. End If
    9. End For
10. End For
11. Return the First Active Server with sufficient initial resource capacity to run job j_i

Figure 2 visually illustrates how the smartFF algorithm is implemented including the helper methods.

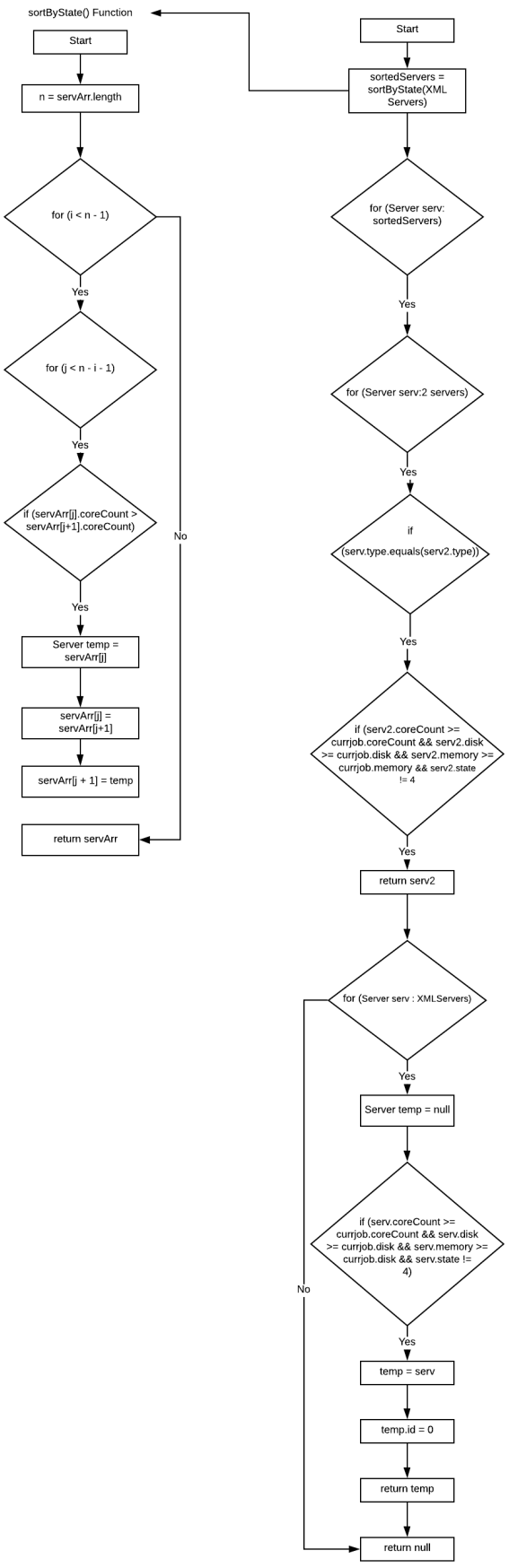

Figure 2: Flowchart of how smartFF algorithm works

Implementation Details

The 'smartFF' algorithm is applied in Client.java and has been implemented based on the First Fit Algorithm. Helper methods such as stateRanking and sortByState have also been created to aid in the execution of the algorithm. The stateRanking function takes in the states of the servers which range from 0-4. The first state '0' means the server is inactive, the second state '1' means that the server is booting, the third state '2' means that the server is idle, the fourth state '3' means that the server is active and the fifth and final state '4' means that the server is unavailable to perform any jobs. The sortByState function uses the stateRanking function to sort the servers based on their most desirable state from lowest to highest. The stateRanking function then utilises the sortByState function to go through the servers to see which servers are using too much resources thus wasting power that can be used to execute bigger jobs. Hence, the algorithm prioritises the servers that are idle and are ready to go.

Evaluation

After conducting a number of tests and experimentations, the smartFF scheduling algorithm did in fact optimise the performance objectives of average utilisation and total cost. Prior to optimising these objectives, other objectives were trialled and tested to see if they would improve or deteriorate based on the implementation. This is seen in Figure 3 to Figure 6 where the smartFF algorithm had a negative effect on other optimisation metrics. Figure 3 shows an increase in Actual Simulation Time when compared to FF which is what the smartFF algorithm is based upon. Figure 4 illustrates that the smartFF algorithm provided an inconsistent calculation of the Waiting Time with the results showing some of the config files increasing, decreasing and maintaining their original value when comparing all baseline algorithms side by side. Figure 5 maintained and slightly improved the Execution Time for ds-config-s3-5 and ds-config-s3-5 and Figure 6 where the smartFF algorithm similarly provided an inconsistent calculation of the Turnaround Time.

Simulation End Time (sec)				
Config File	FF	BF	WF	smartFF
ds-config-s3-1	49136	49136	71656	49136
ds-config-s3-2	6893491	6893491	10198778	10198778
ds-config-s3-3	28243327	28474874	37797027	28243327
ds-config-s3-4	222731	238864	311144	311144
ds-config-s3-5	279818	279818	417606	279818
ds-config-s3-6	445008	449168	500740	500740
ds-config-s3-7	478802	518966	1789345	540253

Figure 3: Simulation End Time Comparison Table for s3sampleconfig files

Waiting Time				
Config File	FF	BF	WF	smartFF
ds-config-s3-1	9696	9696	18356	9696
ds-config-s3-2	2249670	2249670	4817933	4817933
ds-config-s3-3	9491026	9615375	18668067	9276463
ds-config-s3-4	51421	54636	11049	11049
ds-config-s3-5	49300	49300	176095	49300
ds-config-s3-6	128969	130456	206168	206168
ds-config-s3-7	55710	60224	735532	122765

Figure 4: Waiting Time Comparison Table for s3sampleconfig files

Execution Time				
Config File	FF	BF	WF	smartFF
ds-config-s3-1	6200	6200	6764	6200
ds-config-s3-2	10419	10419	10429	10429
ds-config-s3-3	46180	46180	46319	46226
ds-config-s3-4	15	15	15	15
ds-config-s3-5	8642	8642	9150	8642
ds-config-s3-6	775	775	776	776
ds-config-s3-7	1325	1325	1334	1331

Figure 5: Execution Time Comparison Table for s3sampleconfig files

Turnaround Time				
Config File	FF	BF	WF	smartFF
ds-config-s3-1	15896	15896	25120	15896
ds-config-s3-2	2260089	2260089	4828362	4828362
ds-config-s3-3	9537206	9661555	18714386	93226
ds-config-s3-4	51436	54651	110364	110364
ds-config-s3-5	57942	57942	185245	57942
ds-config-s3-6	129745	131232	206944	206944
ds-config-s3-7	57035	61549	736866	124097

Figure 6: Turnaround Time Comparison Table for s3sampleconfig files

The smartFF algorithm did however successfully optimise the performance objectives of Average Utilisation and Total costs as indicated by the tables in Figure 7 and Figure 9 and the graphs illustrated in Figure 8 and Figure 10. Figures 7 and 9 clearly show the improvement of average utilisation in all configuration files except for ds-config-s3-3 whereas Figure 8 and 10 illustrate the significant improvement of the Total Cost for four out of seven configuration files when compared to all other scheduling algorithms.

Average Utilisation				
Config File	FF	BF	WF	smartFF
ds-config-s3-1	100.0%	100.0%	100.0%	100.0%
ds-config-s3-2	75.7%	75.7%	100.0%	100.0%
ds-config-s3-3	81.2%	80.7%	100.0%	77.4%
ds-config-s3-4	86.7%	77.2%	100.0%	100.0%
ds-config-s3-5	63.0%	63.0%	100.0%	63.0%
ds-config-s3-6	76.3%	73.7%	100.0%	100.0%
ds-config-s3-7	48.0%	46.0%	100.0%	83.3%

Figure 7: Average Utilisation Table for s3sampleconfig files

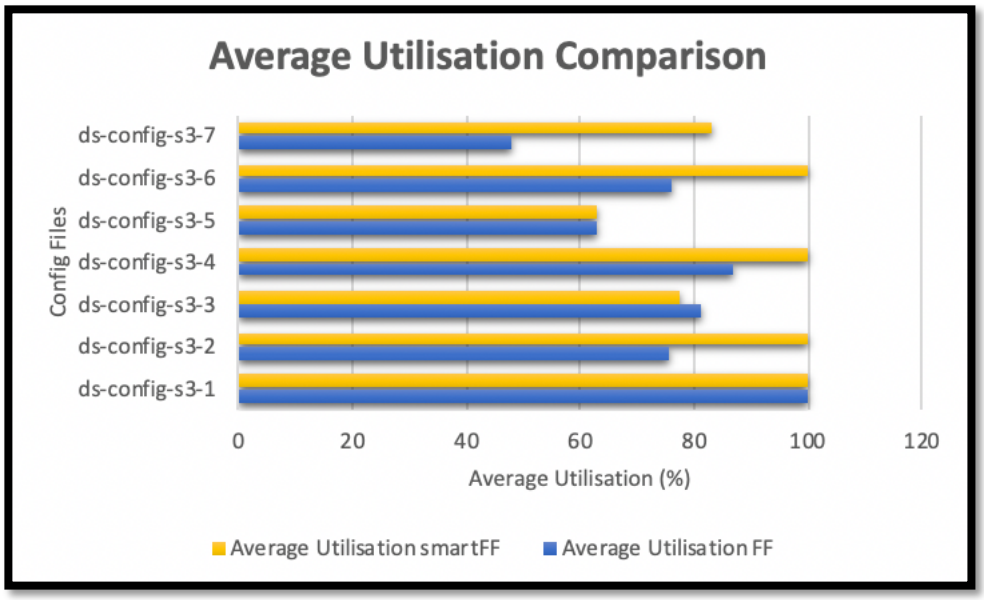

Figure 8: Average Utilisation Comparison Chart

Total Cost				
Config File	**FF**	**BF**	**WF**	**smartFF**
ds-config-s3-1	$ 6.86	$ 6.86	$ 7.95	$ 6.86
ds-config-s3-2	$ 1,148.89	$ 1,148.89	$ 1,133.19	$ 1,133.19
ds-config-s3-3	$ 16,327.66	$ 16,430.57	$ 16,798.64	$ 15,785.06
ds-config-s3-4	$ 148.37	$ 159.13	$ 138.24	$ 138.24
ds-config-s3-5	$ 86.04	$ 86.04	$ 92.78	$ 86.04
ds-config-s3-6	$ 450.92	$ 455.13	$ 444.90	$ 444.90
ds-config-s3-7	$ 6,379.16	$ 6,604.79	$ 1,759.11	$ 5,853.90

Figure 9: Total Cost for s3sampleconfig files

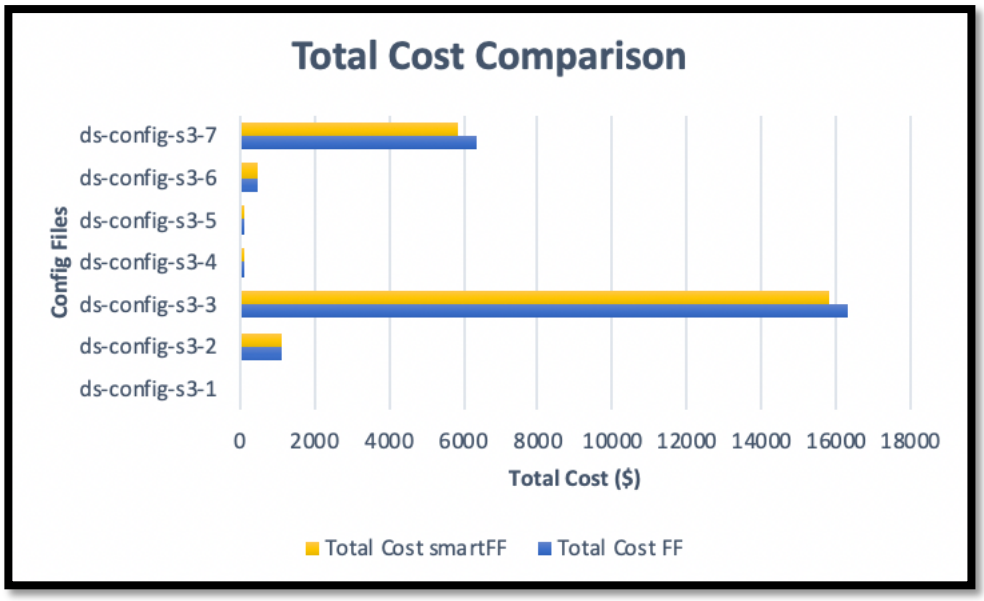

Figure 10: Total Cost Comparison Chart

Out of the two performance objectives that were optimised, based on the results the total cost was significantly optimised as a lot amount of money was saved for four out of seven configuration files when compared to all other scheduling algorithms (Figure 11). On the whole, when comparing the First Fit algorithm and the smartFF algorithm, approximately $1099.71 was saved across all the configuration files. Prior to the implementation of smartFF, all configuration files had a very large price attached to it which is why it became appropriate to optimise the performance objective.

Total Cost Saving					
Config File	FF		smartFF		Saving
ds-config-s3-1	$	6.86	$	6.86	$ -
ds-config-s3-2	$	1,148.89	$	1,133.19	$ 15.70
ds-config-s3-3	$	16,327.66	$	15,785.06	$ 542.60
ds-config-s3-4	$	148.37	$	138.24	$ 10.13
ds-config-s3-5	$	86.04	$	86.04	$ -
ds-config-s3-6	$	450.92	$	444.90	$ 6.02
ds-config-s3-7	$	6,379.16	$	5,853.90	$ 525.26

Figure 11: Total Saving Comparison Table

Furthermore, the smartFF algorithm performs better than the baseline algorithms when optimising two performance objectives particularly average utilisation and total cost. This new implementation performs better than the baseline algorithms as it searches and sorts the servers based on their state and assigns jobs to the servers particularly if they are waiting for something to do. This means that resources will used efficiently and effectively in order to complete the jobs assigned to them.

Conclusion

Overall, the new scheduling algorithm 'smartFF' considerably improved the Average Utilisation which successfully measures how much as a percentage of a resource is used. This was clearly evident in all configuration files except for ds-config-s3-3. The smartFF algorithm also significantly improved the Total Cost for four out of seven configuration files when compared to all other scheduling algorithms. It is suggested that the smartFF algorithm be refactored in order to improve the average utilisation of ds-config-s3-3 as well as the total cost of ds-config-s3-1 and ds-config-s3-5 as no saving is evident.

References

- Anuar, A., 2020. *Assignment 2 OS*. [online] Academia.edu. Available at: https://www.academia.edu/10122881/Assignment_2_OS [Accessed 29 May 2020].

- David, K., Mukhopadhyay, S. and Lal Das, T., 2020. *COMP3100*. [GitHub Repository] https://github.com/Katrina1999/COMP3100.git.

- www.javatpoint.com. 2020. *OS Scheduling Algorithms - Javatpoint*. [online] Available at: https://www.javatpoint.com/os-scheduling-algorithms [Accessed 27 May 2020].

- Tutorialspoint.com. 2020. *Operating System Scheduling Algorithms - Tutorialspoint*. [online] Available at: https://www.tutorialspoint.com/operating_system/os_process_scheduling_algorithms.htm [Accessed 28 May 2020].