COMP3100 Algorithm Design Stage 3 Document

Group 8:

Sakura Mukhopadhyay 45435073

Project title: Intelligent priority job scheduler for distributed systems

1. Introduction

Information technologies are the driving trends for the future of computing. Information technologies require efficient and powerful platforms with high capabilities. Distributed systems have raised to be this platform that provide the option of running multiple applications parallelly at the same time while appearing coherent to the user. Distribution systems are a network collection of elements in a computing-based system that coherently work together to execute tasks in an efficient and viable method in terms of time, cost, and quality. Efficiency can be controlled in a distributed system, by implementing scheduling with complex algorithms. Scheduling in distributed systems, is the decision maker of the system by allocating resources on set conditions over a set time to ensure the system is running at maximum efficiency. Task scheduling and task allocation algorithms is essential in distributed system to ensure the optimal solution in all environments is produced. This project so far, has produced three base-line algorithms into the job scheduler which operate on different operations and logics. In stage3, the final stage of the project a new algorithm inspired by existing algorithms will be implemented and applied to the simulated distributed system, to produce a client-side simulator that executes jobs at specific set conditions to increase the efficiency and overall minimise the cost of the system compared to the existing baseline algorithms. Through this project, I have been able to build a foundation bank of knowledge about distributed systems that has allowed me to understand different possible methods and data structures in a job scheduler.

2. Problem Definition

The aim of stage 3 of my project is to implement a time scheduling algorithm to the system that prioritises and executes tasks with significantly smaller execution times compared to the baseline algorithms. I would like to minimise the wait time of tasks, and the overall execution time as I believe this is what drives the cost of a system. The amount of time taken for a set task and the server used will be a major factor to the total cost of the system. Implementing an algorithm which uses server sizes that fit the requested task and will not waste memory space is much preferable and I believe will see result in a reduce of execution and wait time. Furthermore, I hope for majority scenarios, that this will also decrease the overall cost of the system. By implementing this algorithm, I aim to minimise the overall time for the total number of tasks to be completed and minimising the cost. For the project to be successful, the created algorithm as mentioned, will have a lower execution and wait time

compared to the baseline algorithms and less overall cost. The job scheduler will run by receiving submission times of requested tasks and based on the requested data size of the job, it will schedule and prioritise it from smallest size to larger sizes executing them based on this order. The scheduler will continue to regularly have a check for any new jobs and updating the status of the task. Upon completion of the jobs, the system will produce a statistical report containing the task completion times, cost and system utilisation significantly improved than stage 2.

3. Algorithm Description

The algorithm I have created was inspired from the buddy algorithm and the first-fit algorithm. First-fit algorithms search the first block of the system and is allocated a sufficient partition that comes from the top layer of the main memory. Though it is efficient in time, it is not always efficient in cost and quality. There are possible cases that it can allocate memory spaces without sufficient space to process all tasks. For my algorithm, I wanted to use first-fit as my base inspiration to ensure my algorithm had a fast search time as well which reduces the overall time of the system to allocate and after further development, execute a task. I thought this would help me achieve the aims for my stage 3 algorithm. However, as first-fit is sometimes over costly, I wanted to adjust it to ensure it was efficient in both time and cost. The buddy system algorithm scheme inspired me to achieve this cost efficiency in my system. The buddy allocation system algorithm takes a larger memory block and splits it into two equal and smaller sized blocks known as buddies. This is done to meet the requirements of the requested task size to execute the tasks and minimise wastage of memory space. One of these smaller blocks, will be further split into two smaller block parts until the job tasks are all completed. The advantage is that the execution time is significantly less, hence reducing the overall cost. Another advantage is the way two buddy memory blocks can reform and combine as a larger block of memory depending on the memory request which ensures memory is not wasted unnecessary. Figure [1] and Figure [2] below demonstrate how these algorithms work on their own. My implemented algorithm has its own structure as in the project guidelines but was inspired by the working of both these algorithms.

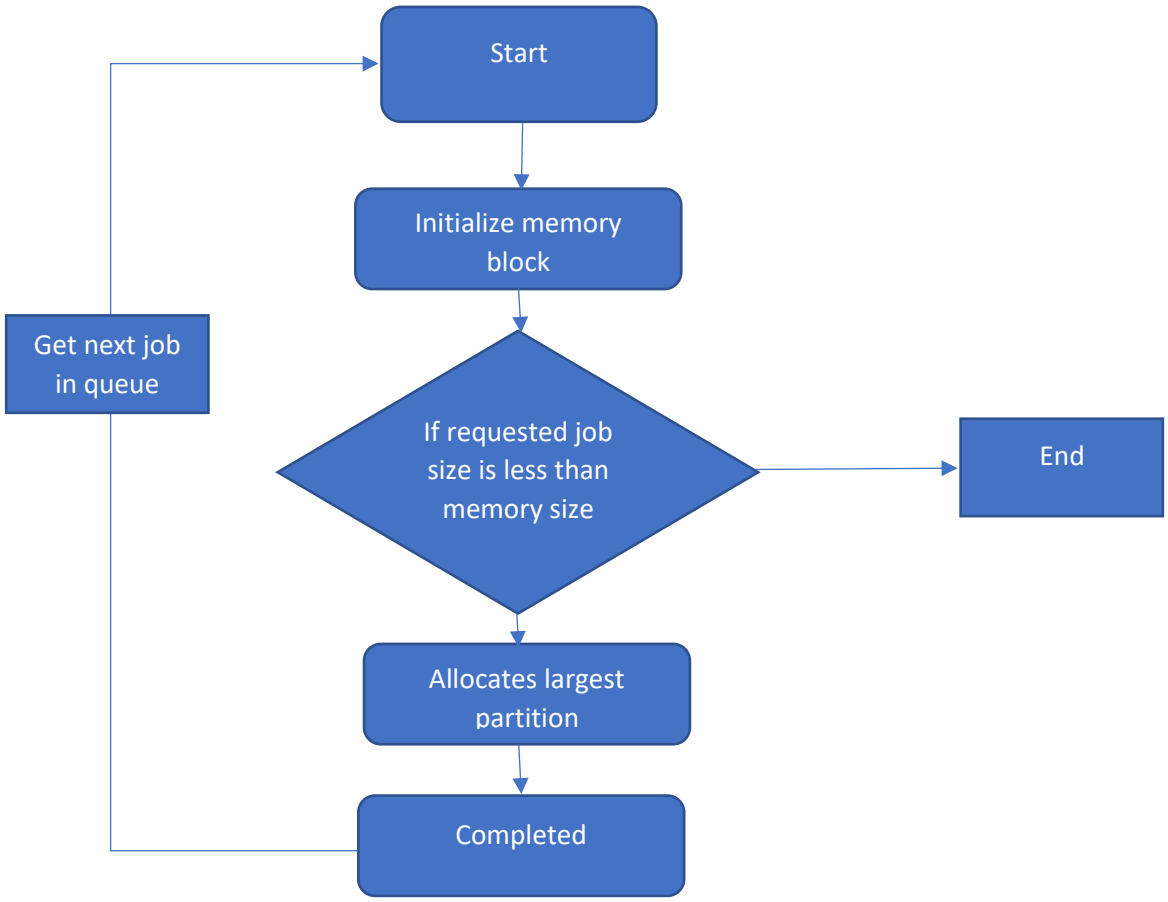

Figure [1]: First-fit algorithm in a distributed system

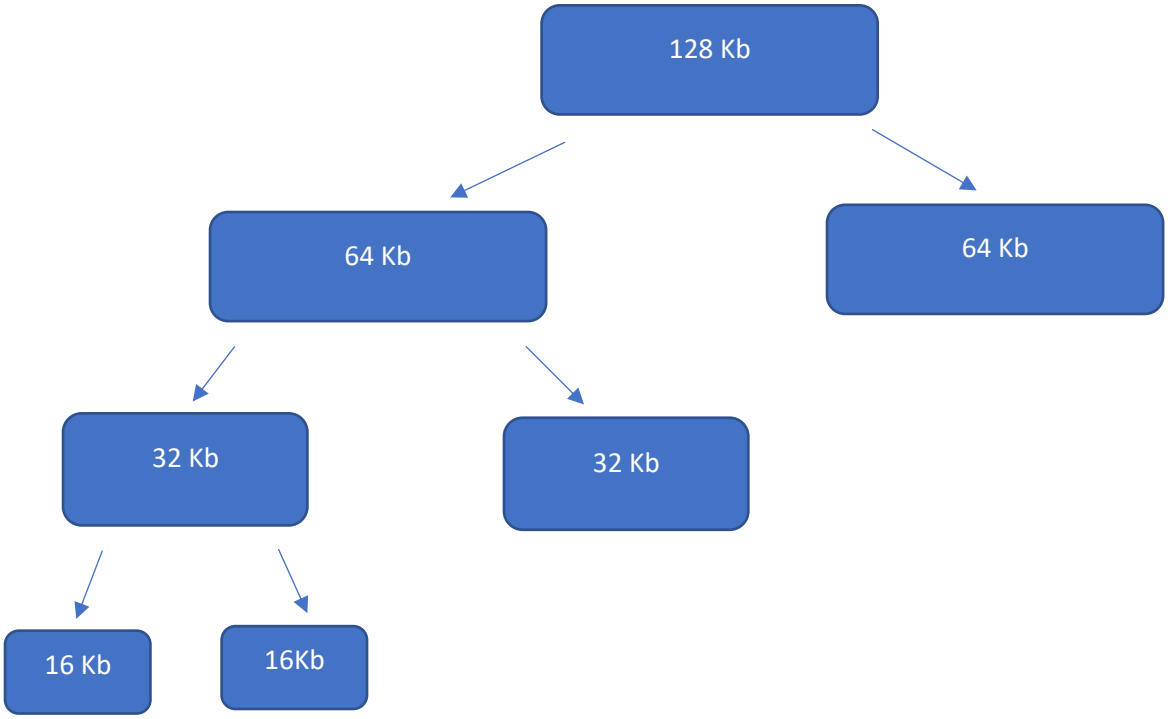

Figure [2]: Visual representation of how buddy system splits memory blocks in a distributed system

The system I have made uses the concept of both the algorithms of Figure [1] and Figure [2]. For example, if the first task size at top of memory is 128 kB then the algorithm takes this, but if the requested task only requires 64 kB then it splits it into two buddies and utilises 64 kB instead.

Below I have set up an example scheduling scenario with several configurations files and compare to the output to observe the changes. Below I have attached the analysis of the output obtained when run with my new algorithm and have compared all three baseline algorithms with the new algorithm.

Table [1] below contains the overall data values collected when running all the algorithms on the configuration files specified. Where bf is the best-fit algorithm, ff is the first-fit algorithm, f is the worst-fit algorithm and bd is the new algorithm inspired by buddy system algorithm and first fit algorithm.

Algorithms	Total cost ($)	Waiting time (s)	Execution time (s)	Average utilisation time (%)
bf	29546.38	1760156	15555	60.27
ff	28061.26	1478146	15555	63.13
wf	42171.96	7548266	15663	100
bd	8.17	9696	6200	51.81

Table [1]: Data gathered when run with ds-config-s2-1.xml configuration file

Algorithms	Total cost ($)	Waiting time (s)	Execution time (s)	Average utilisation time (%)
bf	29546.38	1760156	15555	60.27
ff	28061.26	1478146	15555	63.13
wf	42171.96	7548266	15663	100
bd	86.04	49300	8642	63.03

Table [2]: Data gathered when run with ds-config-s2-2.xml configuration file

Algorithms	Total cost ($)	Waiting time (s)	Execution time (s)	Average utilisation time (%)
bf	29546.38	1760156	15555	60.27
ff	28061.26	1478146	15555	63.13
wf	42171.96	7548266	15663	100
bd	28400.4	1008203	20550	62.96

Table [3]: Data gathered when run with ds-config-s2-3.xml configuration file

Algorithms	Total cost ($)	Waiting time (s)	Execution time (s)	Average utilisation time (%)
bf	29546.38	1760156	15555	60.27
ff	28061.26	1478146	15555	63.13
wf	42171.96	7548266	15663	100
bd	806.36	201634	1464	53.41

Table [4]: Data gathered when run with ds-config-s2-4.xml configuration file

Algorithms	Total cost ($)	Waiting time (s)	Execution time (s)	Average utilisation time (%)
bf	29546.38	1760156	15555	60.27
ff	4133.96	142103	1476	67.35
wf	42171.96	7548266	15663	100
bd	4280.76	203986	1479	85.01

Table [5]: Data gathered when run with ds-config-s2-5.xml configuration file

For majority of the test cases we observe the overall waiting and execution time by the new algorithm is much less compared to the other algorithms reducing the overall cost of the system. However in config files shown in table [3] and table [5], it is observed that though the total cost of the new algorithm to execute the tasks is slightly higher compared to first-fit , it meets the aim of having a lower waiting time, less utilisation percentage as well as number of servers. This shows that though in some scenarios, the new algorithm showed to be the most efficient in terms of cost, it can be more expensive due to the run time on that server. This is something to keep into consideration and can be investigated further to develop more. A suggestion could be to implement an algorithm that limits the number of large or medium servers, those that are more costly at a time.

4. Implementation details

The created algorithm requires the implementation of the ‘RESC’ action like the existing baseline algorithms. The RESC command sends the information of server availability to a resource to the client upon the client’s request. It obtains the information of the server’s core, disk, memory, and current state of the system. In our algorithm, we first sort the servers in ascending order by its server ID through the function sortByID(). When the servers are sorted, we check the availability of each server. A loop is used which checks the servers one by one depending on the coreCount, disk and memory size. This is where the buddy system algorithm is implemented. In the sorting algorithm we calculate the free server available starting from the smallest block that has sufficient memory space

to execute the request. This is done through the integer variable x, which is defined as int x = (int)Math.ceil(Math.log(s) / Math.log(2));. This variable splits the memory space into two as seen in Figure [1] like a buddy system. This ensures no server is selected that will use up extra memory space and instead have a server that has a memory space closest to the requested job. From this, the jobs are dedicated servers based on the requested memory size and execute from smallest memory size to largest memory size. This also completes the goal of a 'priority job scheduler' as it prioritises the tasks in terms of job task memory

5. Evaluation

The algorithms have been tested with six configuration files with the comparisons attached below. In some configuration files the new algorithm has seen to be less in cost and time, however in others it appears to be slightly costly with less time due to the server size being used. The results showed me the importance of testing it with different amount and size jobs as even though it may be ideal for one scenario it may not be suitable for another.

We further evaluate the three baseline algorithms and the new algorithm known as 'bd' with the configuration file of 'ds-config-s2-1.xml'. This configuration file should have a minimum cost however the three baseline algorithms up to this point were proving to be overcostly for this set of tasks. The four algorithms are compared below.

Execution time of a task in the scheduling system is the amount of time the job takes to execute from running on the server till complete. Generally, the higher the execution time, the higher the cost. As we can see in Figure [3], the new algorithm notated as 'bd', has the smallest execution time compared to the baseline algorithms. This means the total time taken for all tasks to be completed is less and will make the overall speed of the system much faster. The waiting time of a task is defined as the amount of time that a job task has to sit ideal before the task gets setup to be processed. Large waiting times can clog up and delay a system making it inefficient and adding to the cost. As seen in Figure [3], worst-fit has the largest waiting time however the new algorithm has a very small waiting time compared to all three of the baseline algorithms. We can say that this is highly preferable which increases the efficiency of the scheduler and reducing the overall cost.

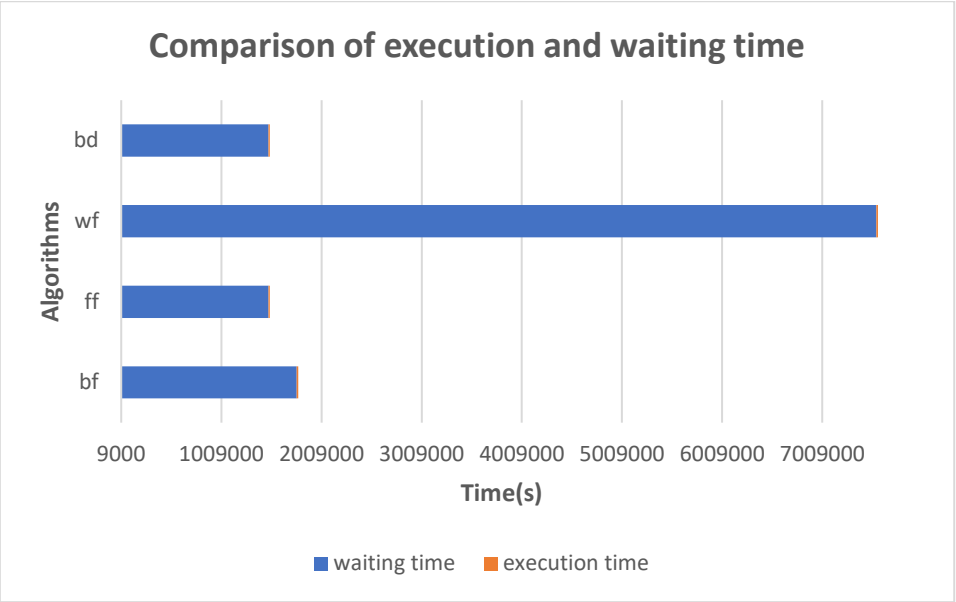

Figure [3]: Comparison of execution and waiting times between algorithms for ds-config-s2-1.xml file

The average utilisation time is a percentage rate that is calculated by taking the average of the computational work load a system performs over a time period. The smaller the average utilisation time means the system can perform more tasks over a shorter period of time. As seen in Figure [4] we can see that the average utilisation time between the new algorithm (bd), best-fit and first-fit are very close. This means that all three utilise the total time well to perform the overall number of tasks. However, the new algorithm does have a smaller time overall.

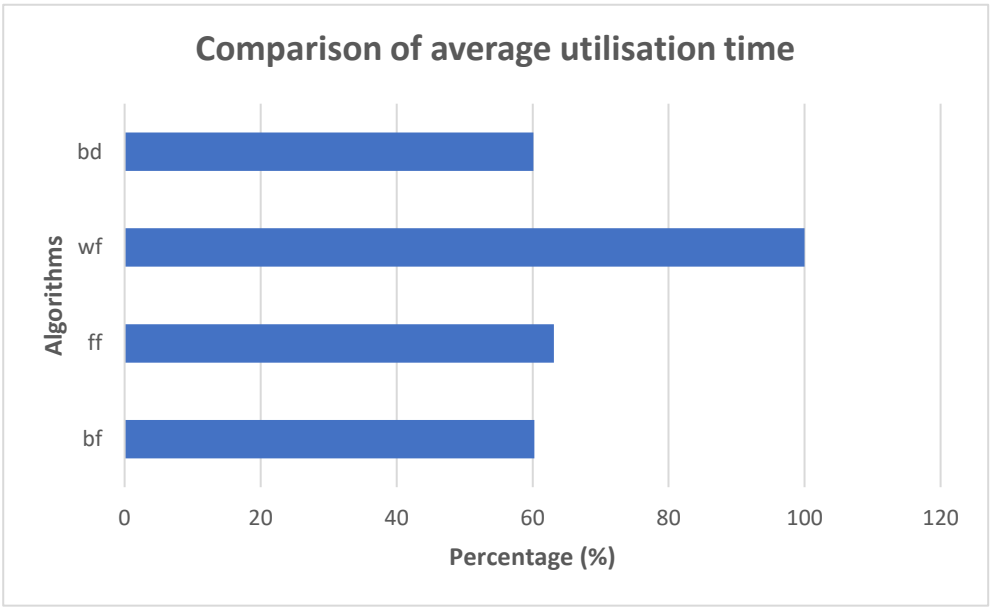

Figure [4]: Comparison of average utilisation times between algorithms for ds-config-s2-1.xml file

The total cost of each algorithm implemented in the system is given below in figure [5]. Total cost is based on many factors such as the waiting time, execution time and overall efficiency of the output.

We can see that for the configuration file of ds-config-s2-1.xml that the new algorithm has slightly lower cost. However, the standard cost for this configuration file will differ with the new algorithm not having an extremely cheap cost. Hence, we cannot say the new algorithm will always give the lowest cost but we can say that it will be more efficient overall in terms of time and less wastage of memory space.

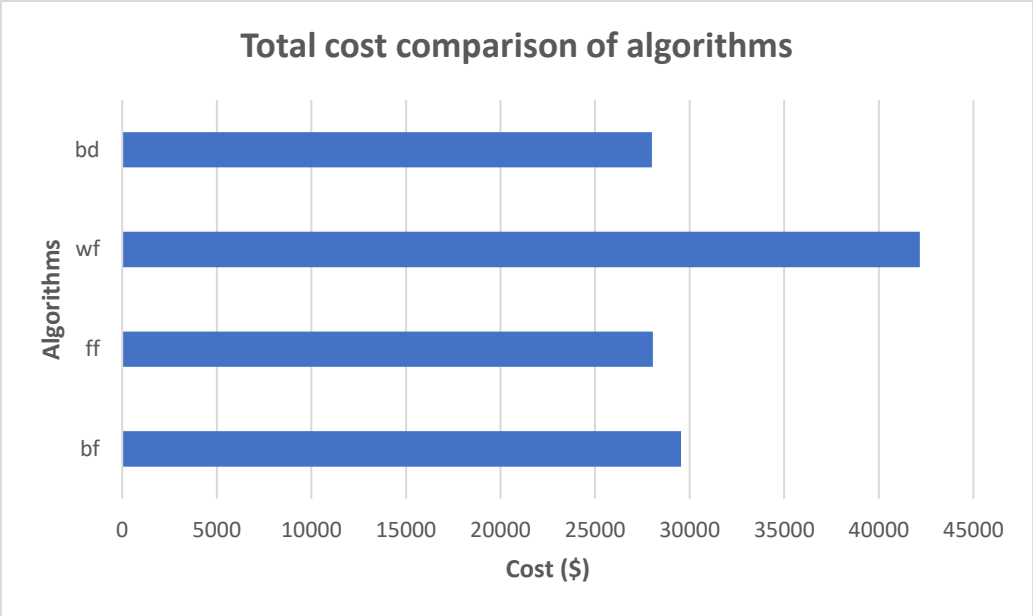

Figure [4]: Total cost of each algorithm for ds-config-s2-1.xml file

6. Conclusion

Overall, I have been able to create a priority intelligent job scheduler in a distributed system that prioritises jobs in terms of job size and is efficient in terms of time, cost, and quality. I believe the algorithms I have implemented are sufficient but may not be reliable for a large number of tasks in different scenarios, due to the size of servers which are more costly and also my lack of experience in java. However, I am happy with the understanding and learning I have gained throughout this project. In the future, the system could be more efficient if an algorithm that timed the execution time of each job is included which can determine the order of shortest to longest execution speed. By sorting it in this order, the tasks can be completed possibly quicker with a shorter waiting time. I also think it can be very useful to add time complexity more into the system, thought the buddy system does do that. Scheduling in distributed systems is very important and essential for a successful and efficient system in terms of cost, time and output quality and is reiterated through this project.

7. References

[1] https://github.com/Katrina1999/COMP3100/tree/Buddy-Fit/Stage3

[2] Ieeexplore.ieee.org. 2020. *A New Approach For Task Scheduling In Distributed Systems Using Learning Automata - IEEE Conference Publication*. [online]
Available at: https://ieeexplore.ieee.org/document/5262978
Accessed 14 March 2020

[3] GeeksforGeeks. 2020. *Buddy Memory Allocation Program | Set 1 (Allocation) - Geeksforgeeks*. [online] Available at: <https://www.geeksforgeeks.org/buddy-memory-allocation-program-set-1-allocation/> [Accessed 24 May 2020].

[4] GeeksforGeeks. 2020. *Program For First Fit Algorithm In Memory Management - Geeksforgeeks*. [online] Available at: <https://www.geeksforgeeks.org/program-first-fit-algorithm-memory-management/> [Accessed 24 May 2020].

[5] Gupta, P. and G. Vishwakarma, R., 2012. Comparison of Various Election Algorithms in Distributed System. *International Journal of Computer Applications*, 53(12), pp.1-4.