Project report: Stage 2
COMP3100 Distributed Systems, S2, 2022
SID: 45898405, Name: Benjamin_ van_de_Vorstenbosch

1 Introduction

Provided with the job scheduling simulator ds-sim, the aim for this project was to develop a functional client that can communicate with the ds-server to allocate jobs to various servers. In Stage 1, the task was to create this client and implement the single scheduling algorithm Largest Round Robin. In Stage 2, that client has been used again to implement a custom algorithm designed to minimise turnaround time.

This report has been written to:

- Explain the purpose of this new algorithm
- Describe how it is implemented
- Compare the algorithm against 4 other baseline algorithms - first capable, first fit, best fit, and worst fit
- Make an informed decision about which algorithm is best for decreasing turnaround time

The new algorithm has been labelled 'Custom Turnaround' or 'CT' for short. When you see these terms throughout the report, they are referring to the newly designed algorithm for minimising turnaround time.

2 Problem Definition

The purpose of this new algorithm is to minimise the average turnaround time of jobs. Turnaround time is the amount of time taken for a job to be completed, counted from the time first submitted. Therefore to improve turnaround time the algorithm needs to minimise the amount of time spent waiting for a job to begin after submission.

Unfortunately, focusing entirely on improving one aspect of job scheduling leads to other aspects decreasing in efficiency. In the case of ds-sim, completely minimising turnaround time will lead to an increase in rental costs and a decrease in resource utilisation (both of which is bad). The Custom Turnaround algorithm has been designed to not only improve turnaround time but to also mitigate the negative impacts on rental cost and resource utilisation. This makes it a viable option for a real-world solution where these are important factors that need to be taken into account as well.

With the balancing of efficiency in mind, the final goal of this algorithm is to have a better average turnaround time than the First Capable, First Fit, Best Fit, and Worst Fit algorithms, whilst still remaining in the top 3 algorithms for the average rental cost and resource utilisation efficiency.

3 Algorithm Description

The Custom Turnaround algorithm is, in theory, quite a simple one. It will calculate an estimate time that a job will have to wait before it can be executed on each server, and will pick the server with the lowest wait time. This means that the jobs will have a smaller length of time before they can begin running, and therefore minimal turnaround time. If there are two servers with the same estimated wait time, it will choose the server with the least number of cores total rather than unnecessarily filling up cores for larger servers should a bigger job (with more cores required) come along.

The complexity of this algorithm comes from the calculation of the estimated runtime. Each job that ds-server sends to the client is given an estimate runtime - a naive solution for this algorithm would simply be to add all of the job estimated runtimes on a given server to determine how long the job will be waiting. However, that solution doesn't take into account the fact that each server has multiple cores, and therefore multiple jobs could be running parallel to each other. The estimate runtime would therefore be up to n times larger than it actually is, where n is the total number of cores the server has.

Parallel Wait Time Calculations

The figures below indicate a single 3 core server at two different points in time. Because this algorithm is more mathematically based rather than shifting jobs around or allocating based on status, it is easier to describe based on these snapshots rather than an entire configuration. They don't describe exactly how the ds-server has allocated jobs, but instead theoretically display how the jobs could be optimally stacked within a server in order to calculate optimal scheduling. Yellow indicates running jobs, red indicates waiting jobs, and green indicates the job that will be added. They have each been allocated a job ID and an estimate runtime.

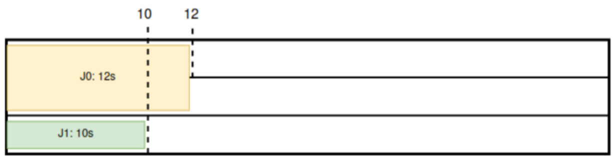

Figure 1: No wait time

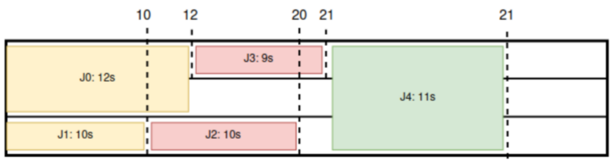

Figure 2: Calculated wait time

1. If the server has more available cores than the job requires, it can be immediately run (as displayed in Figure 1. Therefore, this simply returns a parallel wait time of 0. Otherwise, some more maths is involved to determine the estimate parallel wait time - the following steps will be using the snapshot in Figure 2.

2. Estimate the number of jobs that will be running non-parallel to each other. This can be done using the following calculation:

$$\text{roundup}\left(\frac{\text{incompleteCores}}{\text{serverCores}}\right) = \frac{2+1+1+1}{3} = 2 \tag{1}$$

3. Get the sum and multiple of the n largest jobs estimated runtime, where n is the number of non-parallel jobs from the previous step.

$$\text{estRunTime}_1 + .. + \text{estRunTime}_n = 12 + 10 = 22 \tag{2}$$

$$\text{estRunTime}_1 * .. * \text{estRunTime}_n = 12 * 10 = 120 \tag{3}$$

4. If the return result is simply the sum of these maximum runtimes, the estimate increasingly grew above the actual runtime as more waiting jobs were added. To limit this overflow, the final wait time estimate is instead:

$$sumMaxRuntimes - \frac{\text{sumMaxRuntimes}}{multMaxRuntimes} = 22 - \frac{12+10}{12*10} = 21 \tag{4}$$

By looking at Figure 2, it is clear that 21 is an accurate estimate for the runtime of the optimally scheduled waiting and running jobs for this server (and therefore the waiting time for the new job).

4 Implementation

The implementation of this client has not changed from Stage 1. It still prioritises scale-ability and maintenance by making use of the Factory Pattern[1]. As a result, there are 6 primary classes and 3 algorithm classes in this program. The algorithm classes implement the Algorithm interface, and are selected by the AlgorithmPattern based on arguments provided when running the client.

Class	**Description**
Client	The main class of the program that controls the overall flow of the system.
Connection	An abstract class for communication with the ds-sim server from anywhere in the program
Algorithm	An interface for all algorithms
AlgorithmFactory	Where algorithms are requested and created from
Server	Stores the base and current details of servers, based on the data provided by the ds-server
Job	Stores the base and current details of jobs, based on the data provided by the ds-server
LRRAlgorithm	The Largest Round Robin algorithm schedules jobs to the server with the largest number of total cores. If multiple are found, it chooses the first one.
FCAlgorithm	The First Capable algorithm simply allocates a job to the first server returned in the GETS Capable request to ds-server.
CTAlgorithm	The Custom Turnaround algorithmw as written for Stage 2 and is designed to decrease job turnaround time. More information about how this algorithm is implemented was included in section 3

5 Evaluation

Setup

The custom turnaround algorithm was tested against the First Capable, Best Fit, Worst Fit, and First Fit algorithms using the Stage 2 test cases provided. To access and build the client,

1. Clone the client from GitHub[3] using the command
   git clone git@github.com:BVengo/comp3100Project.git
2. Create the build folder in the main directory of the project (if missing)
3. Run the 'compile.sh' script to compile the project into the build directory. If the compile script doesn't have execute permissions, run 'chmod +x ./compile.sh'

After compiling the client, there are a few different options that can be chosen.

- Run the ds-sim server from the main folder, using the command
  ./ds-sim/ds-server -v brief -c ./configs/<config> -n
  where <config> is the chosen config file from the configs folder
- Run all tests in the config file using the command
  ./tests.sh Client.Class -n localhost 50000 <algorithm>
  where <algorithm> can be lrr, fc, or ct
- Run the original Stage 2 tests for turnaround time using the command
  cd ./configs && ./stage2-test-x86 "java -cp ../build Client localhost 50000 ct" -o tt -n & cd ..

If choosing the first option (running ds-server), the client can be connected by running the command
java -cp ./build Client localhost 50000 <algorithm>
where <algorithm> can be lrr, fc, or ct. This command must be run *after* the server has been started.

Results

As can be seen in Figure 3, the Custom Turnaround algorithm does indeed provide a smaller turnaround time than the other algorithms, averaging at 1515.56. This meant that it met its project aim to deliver the minimum average turnaround out of all the baseline algorithms. The First Capable algorithm is the most notable other algorithm here, sitting with the largest 246,382.78.

CT	FF	BF	WF	FC
1516	1,804	1,867	19,610	246,383

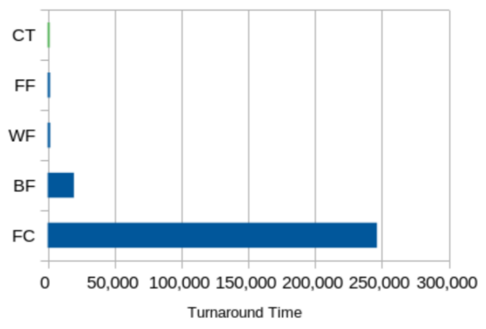

Figure 3: Turnaround time against other algorithms

Figure 4 displays that the Custom Turnaround algorithm still averages second in the overall rental costs despite focusing on turnaround time. Not only does this meet the requirements of efficiency in rental costs (as mentioned in section 2, but it also indicates that this is the best option in terms of rental costs when aiming for efficient turnaround. First Capable is the only algorithm to do (quite substantially) better, however it has the worst turnaround time by many factors more.

CT	FF	BF	WF	FC
634.61	638.18	636.89	665.09	379.29

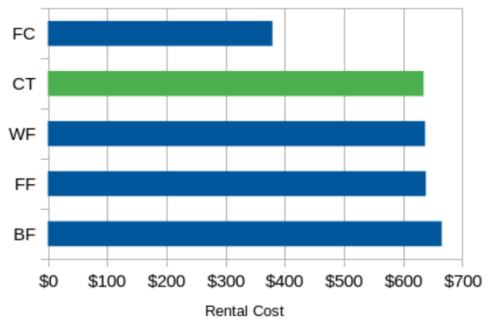

Figure 4: Rental cost against other algorithms

Finally Figure 5 demonstrates once again that, despite focusing on turnaround, the Custom Turnaround algorithm does not discard the importance of resource utilisation. Instead it comes in 3rd in the overall averages. Although this isn't perfect, it meets the requirements laid down in section 2 providing an acceptable level of performance in comparison to the other algorithms. This would not be the algorithm to choose if resource utilisation was top priority, however that is beyond the scope of this project.

CT	FF	BF	WF	FC
63.66	63.30	60.72	68.79	94.37

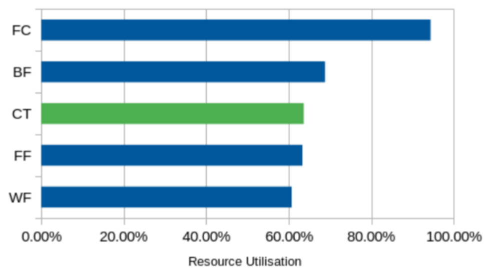

Figure 5: Resource utilisation against other algorithms

6 Conclusion

For minimising turnaround time without losing too much efficiency in other areas, the Custom Turnaround algorithm is far more optimised than the First Fit, Worst Fit, Best Fit, and First Capable algorithms. However, if resource utilisation and rental cost isn't an important factor during the decision of which algorithm to use, it is instead recommended than a different algorithm be designed that will utilise ds-servers MIGJ command to shift waiting jobs around instead of the Custom Turnaround algorithm which only schedules a job once and then moves on.

References

[1] DoFactory. "C# factory method." (), [Online]. Available: https://www.dofactory.com/net/factory-method-design-pattern.

[2] Y. C. Lee. "Ds-sim github repo." (Mar. 9, 2022), [Online]. Available: https://github.com/distsys-MQ/ds-sim.

[3] B. van de Vorstenbosch. "Project github repo." (), [Online]. Available: https://github.com/BVengo/comp3100Project.