# Distributed Job Scheduler

## COMP3100 Stage 2

Michael Thygesen - 45207275

## Introduction

Stage 2 of this project is about understanding & implementing different scheduling algorithms comparable and/or better than: All to Largest (ATL), First Fit (FF), Best Fit (BF), and Worst Fit (WF). Typically these algorithms are used in operating systems for deciding how, where, and why to allocate files of different sizes into certain memory locations. Essentially, in our case, we are using one of the above baseline algorithms to try to optimise performance of job-dispatching based off of metrics such as turnaround time/waiting time, average utilisation of servers, and total cost. In contrast, stage 1, the '*Distributed Cloud Scheduler*' (DCS) had us send all available jobs to the largest server available [2].

Essentially, this project the '*Distributed Job Scheduler*' (DJS) is built upon what we have done in stage 1 in order to implement an algorithm comparable to FF, BF, and WF, whilst being entirely different to ATL.

## Problem Definition

The Scheduling Problem involves finding the optimal schedule for various objectives [5]. As stated previously there are a few objectives/measures that ds-client uses for evaluating the efficacy of a particular scheduling algorithm. Namely these are:

- Minimisation of average turnaround time (& waiting time) (TT).
- Maximisation of average resource utilisation (RU).
- Minimisation of total server rental cost (CO).

Unfortunately, none of the algorithms mentioned above (ATL, FF, BF, and WF) are capable of achieving the most optimal (or acceptable) results in all three metrics. Each are useful in certain situations. Essentially which one we decide to use is entirely dependent upon our goals. That being said, the objective of DJS is not simply the optimisation of only one of these metrics - rather we want an algorithm that can perform sufficiently well in all three metrics. However, as per the testing script given, we must make a choice at which metric to be compared to. So the primary metric we will use in measuring the success of the DJS algorithm will be CO. Optimising for cost seems like the most realistic: cheaper server fees might mean not going broke. However we do not want to sacrifice TT in the pursuit of cheaper server fees.

## Algorithm Description

Below is a brief summary/pseudocode of the DJS Custom Algorithm:

1. Receive current job and list of capable servers
2. Scan through capable server list and find any servers that DO NOT currently have any waiting jobs AND satisfy the requirements (cores, memory, disk)
    a. If above criteria is met, we add this to a new list for "ideal servers", otherwise it is not added
3. Next we check if the ideal server list is empty or not
    a. If ideal server list is empty we scan through the capable server list again and we schedule the current job to the server with the smallest number of waiting jobs
    b. If the ideal server list is NOT empty we go through this list and find the job with the highest number of running jobs. Once that has been found we schedule the current job to the server with the highest number of running jobs

The figure below is a visualisation of how the DJS Custom Algorithm schedules its jobs. The particular configuration file used here is 'ds-config01--wk9.xml' – it has 3 servers (Tiny, Small, and Medium), and there are 5 jobs to be scheduled (J0 through to J4).

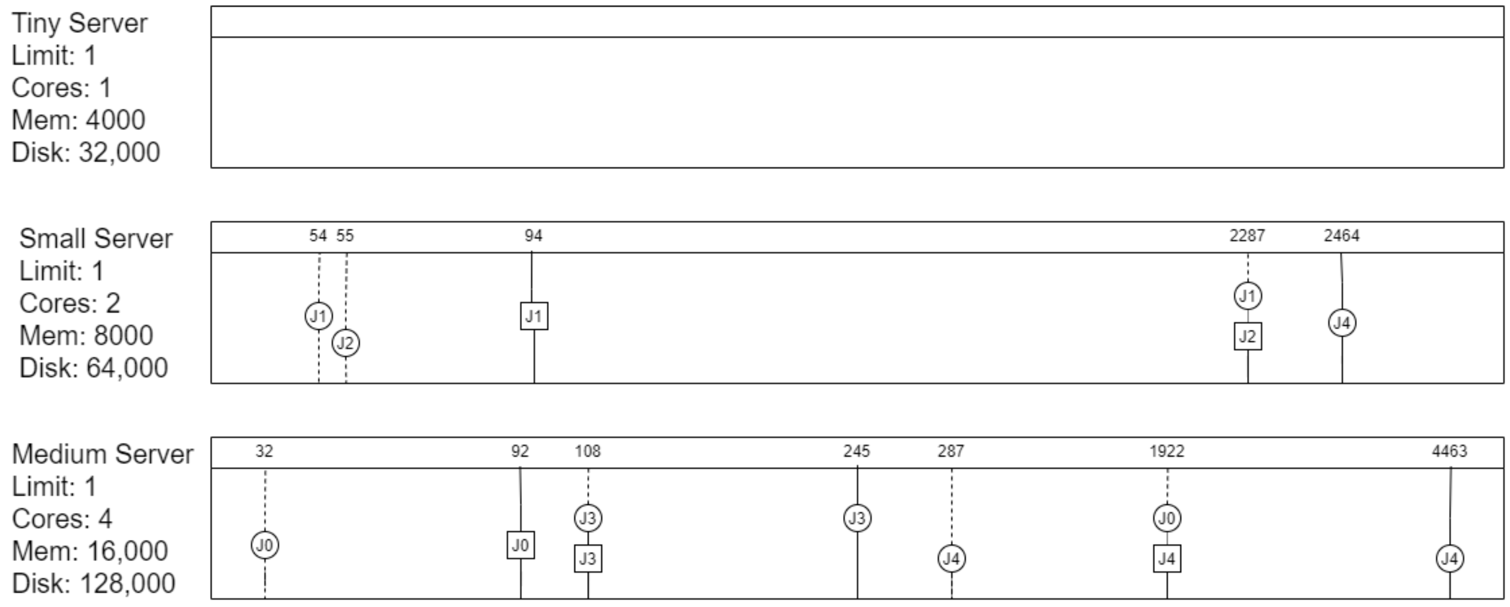

*Figure 1: DJS Custom Algorithm Visualised using the configuration file: ds-config01--wk9.xml*

In the above diagram a circle with dotted lines represents a job first being scheduled (and waiting). A square with solid line indicates that a job has just started running. And finally, a circle with a solid line indicates that a job has just finished running. The numbers at the top represents what time (in seconds) that a job has been scheduled, completed, etc.

Essentially this algorithm tends to favour larger servers over the smaller ones (in terms of cores, memory, and disk). This is not the most efficient job schedule for this particular scenario. Some of these jobs could be run on the 'Tiny' server, and some jobs can even be run concurrently on the same server. We will further discuss DJS's strengths and weakness later.

# Implementation Details

The data structures used in DJS is quite similar to the previous version (DCS). The libraries used for this project include the following [2]:

java.net.*
java.util: arrayList & concurrent.TimeUnit
java.xml.parsers: DocumentBuilder & DocumentBuilderFactory
org.w3c.dom: Document, Element & NodeList

Essentially, DJS has three files; the Client (main class), a Server class, and a Job Class. Essentially the purpose of the Server and Job class is that it allows us to store & retrieve the data coming in from ds-client. For this we construct an ArrayList of Server Objects, so that each Server and its data (e.g. core count) can be easily and readily available. As is the case with Jobs, however, once a job has been scheduled, we immediately delete that job from the Job ArrayList, so that the current Job is always at index 0. A stack or a queue would most likely be better suited for this, however this implementation works fine as it currently is.

# Evaluation

## Simulation Setup

The setup used for running the simulations were on a Linux System (specifically Ubuntu). Essentially the process involved opening two terminals at once; one would run ds-server, and the other running DJS. These programs were able to communicate with one-another via the terminals since ds-client is using the IP address '127.0.0.1' (known as the Loopback Internet Protocol or localhost). Furthermore it is using the port number '5000' – one of many ports that are known as 'Dynamic Ports' (aka private ports: range of 49152 to 65535 are not assigned) [7]. All DJS has to do is connect to the same IP address & Port number and they are able to communicate with one another via the terminals. When running ds-client we need to specify which 'Configuration File' we are using by passing it as a string in the command-line as a parameter. This leads us to the next section; testing & results of the DJS custom algorithm versus the baselines algorithms.

## Configuration Files used

For testing and evaluating this custom algorithm, a total of 31 test configuration files were used. This includes the ones used for marking Stages 1 & 2, weeks 6 & 9 sample configurations, and the initial sample configurations given. The results for all of these files were collected and arranged in an Excel Spreadsheet, in the same format that the '`test_results'` script gives its results. The table below [6] is an example of some of the configuration files used, including its name, number of jobs, and the type of workload.

| File Name | # Jobs | Workload Type | | File Name | # Jobs | Workload Type |
|---|---|---|---|---|---|---|
| config100-long-high.xml | 2000 | High | | ds-config01--wk9.xml | 5 | Unknown |
| config100-long-low.xml | 2000 | Low | | ds-sample-config01.xml | 10 | Moderate |
| config100-long-med.xml | 2000 | Med | | ds-S1-config03--demo.xml | 2000 | Towards High |

*Table 1: Examples of configuration files used. Excel file/pdf of results can be found in repository [6].*

## Results & Comparisons

Once the results of the custom algorithm, and the baselines algorithms (FF, BF, & WF) had been collected and formatted, we proceeded to take an average of the results for each metric (TT, RU, & CO) for each algorithm. The averages can be found below in tables 2, 3, & 4. This also includes whether we saw an improvement or a decline in said metrics.

| Turnaround Time (seconds) | | | | |
|---|---|---|---|---|
| Algorithm | FF | BF | WF | Custom Algorithm |
| Average | 3,653.55 | 3,659.03 | 19,371.48 | 4,144.16 |
| Improvement/worsened | -13.43% | -13.26% | 78.61% | N/A |

*Table 2: Average of baseline vs. custom algorithm for TT across 31 configuration files. NB. Lower average is better.*

| Resource Utilisation (%) | | | | |
|---|---|---|---|---|
| Algorithm | FF | BF | WF | Custom Algorithm |
| Average | 72.25 | 70.82 | 80.13 | 80.41 |
| Improvement/worsened | 11.29% | 13.54% | 0.35% | N/A |

*Table 3: Average of baseline vs. custom algorithm for RU across 31 configuration files. NB. Higher is better.*

| Total Rental Cost ($) | | | | |
|---|---|---|---|---|
| Algorithm | FF | BF | WF | Custom Algorithm |
| Average | $823.29 | $785.66 | $879.36 | $732.09 |
| Improvement/worsened | 11.08% | 6.82% | 16.75% | N/A |

*Table 4: Average of baseline vs. custom algorithm for CO across 31 configuration files. NB. Lower average is better.*

The colour schemes are indicative of whether we saw an improvement (Green) or a worsening (Red) for the given metrics. For TT the Custom Algorithm performed marginally worse than FF & BF, but performed significantly better than WF, hence the colour Yellow. From this we can determine that the Custom Algorithm performs better on average for CO, whilst performing on average slightly worse for TT.

It is worth noting that there are two configuration files that have been trimmed from the calculations of the averages due to the degree in which they skew the data overall. However the results of said configuration files can be seen below in table 5.

| Algorithm | Configuration File | Turnaround Time | Resource Util. (%) | Cost ($) |
|---|---|---|---|---|
| FF | **ds-S1-config04--demo.xml** | **1,055,036** | **70.46%** | **$69,333.12** |
| | ds-sample-config04.xml | 11,409,075 | 92.88% | $680,058.12 |
| BF | **ds-S1-config04--demo.xml** | **1,060,352** | **70.20%** | **$69,289.91** |
| | ds-sample-config04.xml | 11,409,075 | 91.95% | $680,509.38 |
| Mine | **ds-S1-config04--demo.xml** | **170,983** | **91.25%** | **$76,925.05** |
| | ds-sample-config04.xml | 1,456,116 | 92.26% | $694,444.38 |

*Table 5: Removed Configuration files. N.B. These were removed due to being such massive outliers in size.*

## Discussion

As stated above, the Custom Algorithm is able to provide a relatively fast TT, for cheaper than the Baseline Algorithms. ATL hasn't been discussed nor compared to, since it optimizes for cost at the detriment of TT, to an absolutely unacceptable degree. WF has similar issues. Furthermore, when we look at the results of Table 5 (FF & BF have been coloured differently to assist in legibility), the Custom Algorithm absolutely outperforms FF & BF for TT. However, it is able to do so at the detriment of cost (but not to an unacceptable degree). Upon further inspection of both these configuration files, both have a job count of 3000, and a workload type of "overloaded". It would appear that this Custom Algorithm is better at TT optimisation for the larger job counts/workloads, whilst for smaller job counts/workloads it works better at reducing total cost. Furthermore, it tends to favour the larger servers more. Therefore, it seems as if this Custom Algorithm would work better in fulfilling a niche role, rather than being an algorithm for general use.

For further inspection of the data given earlier, please refer to the files: 'Stage 2 Test Results.pdf', or 'Stage 2 Results.xlsx'

## Conclusion

In conclusion, the findings of this report indicates that the DJS Custom Algorithm is a decent job scheduling algorithm when compared with the baseline algorithms (FF, BF, & WF). As with each, they have their respective strengths and weakness. When and why to use a particular way of scheduling is entirely dependent on the context and goals in mind. For example, ATL achieves significantly cheaper prices than all other mentioned algorithms, but in doing so greatly sacrifices its TT. The DJS Custom Algorithm on the other hand seems to perform better in terms of cost across all algorithms (except ATL), especially for increased work-loads. However, it does not sacrifice TT in doing so – in fact there is only a very marginal difference in TT for the DJS Custom Algorithm when compared to FF & BF. So all in all, this algorithm would perhaps work best during times of higher workloads, or as a cost-saving measure for smaller workloads.

## References

[1] https://github.com/MrThygesen16/COMP3100Stage2 (Stage 2 repo)

[2] https://github.com/CazDev/Distributed-Cloud-Scheduler (Stage 1 group repo)

[3] https://github.com/distsys-MQ/ds-sim/blob/master/docs/ (ds-sim user guide)

[4] https://github.com/distsys-MQ/ds-sim/blob/master/src/pre-compiled/ds-client (ds-client)

[5]https://riot.ieor.berkeley.edu/Applications/Scheduling/index.html#:~:text=Scheduling%20problems%20involve%20solving%20for,and%20characteristics%20of%20the%20jobs.&text=The%20user%20can%20select%20any,any%20number%20of%20parallel%20machines.

[6] Further Results (PDF) / Stage 2 Results (Excel File) – can be found in 'docs' folder in [1].

[7] "Windows TCP/IP Ephemeral, Reserved, and Blocked Port Behaviour", *Docs.microsoft.com*, 2010. [Online]. Available: https://docs.microsoft.com/en-us/previous-versions//bb878133(v=technet.10)?redirectedfrom=MSDN.