# Job Scheduling Algorithm with Cost Savings and Resource Utilisation

Frederick Jack 45511020

# Introduction

In a distributed system, job scheduling is very important for the efficient use of resources. Scheduling is a decision-making process that typically uses an optimizing algorithm for the dispatching of jobs. Stage 2 of this project implements a client-side job scheduler working with a distributed systems server simulator. The Client is connected through a socket, so it is language independent, but this implementation is written in Java. The goal at this stage of the project is to successfully schedule and complete all available jobs. The main goal of Stage 2 is the creation of an algorithm that successfully schedules all jobs. The aim is to save on total rental cost with a turnaround time that is comparable to a worst fit algorithm without any job failures. There needs to be efficient use of costs with a superior resource utilisation metric compared to the three baseline algorithms.

# Problem Definition

When using a distributed system of servers, one of the main goals is efficient use of resources when scheduling jobs. This can be defined along many different parameters such as turnaround time, resource utilisation, and cost savings. With this project the main issue that will be addressed is whether there can be an improvement over four algorithms that are already defined: all jobs to the largest server (ATL), best fit (BF), first fit (FF), and worst fit (WF). It is already assumed that the reader will be familiar with those four algorithms. The relevance of this problem is that there can be less power consumption, increased savings for the customer and a faster processing of “jobs”. The objective is to create a “significantly” different algorithm that is superior in performance in terms of cost savings and resource utilisation. The Job Scheduling Algorithm (JSA) provides cost savings with resource utilisation being a secondary goal.

# JSA Description

* Uses TERM command to stop “idle” servers and increase resource utilization.
* GETS Avail command is used to get servers that are currently not processing other jobs to speed up execution time.
* Two lists of servers are used: A dynamic one with currently available servers and a static list of servers that is created with each xml file that is used.
* JSA searches for a server within 7 cores of the cores required for the job. It also requires that the server has the required amount of memory and disk space available for processing the job, so the Client does not crash.
* If you increase the difference of number of cores required to higher than 7 then there is a lower turnaround time.
* If you decrease the difference of number of cores required to lower than 7 then the total cost is lowered.
* When there are no “available” servers the algorithm reverts to the static server list and selects a server with enough cores, memory, and disk to handle the job.
* When a server is selected from the static server list a randomised server is picked within the limit of that server type so that jobs are evenly distributed.

# Implementation

The Client is implemented with Java version 8. The IDE used for the project was IntelliJ. The provided ds-server simulator is written with the C programming language. The Client and Server were tested using Ubuntu 20.04 with a VirtualBox virtual machine.

#### Client Class/Main

The Client class contains the main method for running the Client and contains all the methods for sending and receiving messages and the algorithm methods. It initiates the connection and contains the event loop method which is the process for scheduling jobs.

#### Server Class

The Server class contains the variables of what the GETS Avail command returns. This information is stored as an object in an array list, and it dynamically changes with each request of available servers. The array list is cleared after each job is scheduled.

#### XML Class

The XML class parses the ds-system.xml generated by the ds-sim Server. It stores the details of each server in an object array list.

#### StaticServerList Class

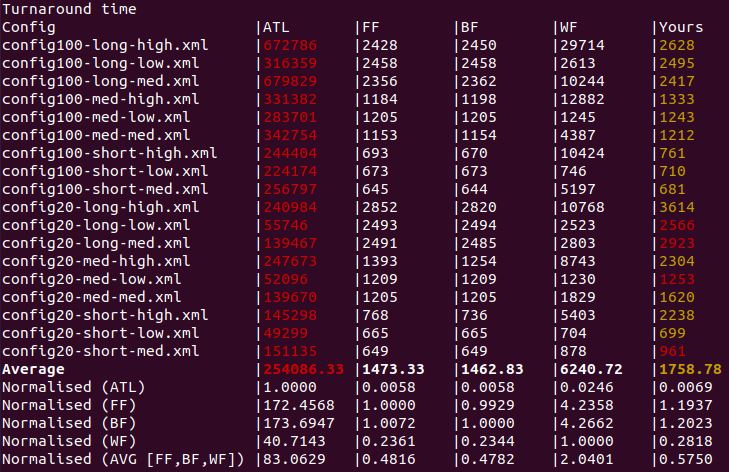
The StaticServerList class contains the variables of the static server list that is generated by the ds-system.xml file. The details of each static server are stored as an object in an array list.

# Evaluation

A testing script with 18 configuration files was provided to test the JSA. In the three figures the turnaround time, resource utilisation, and the total rental cost were tested.

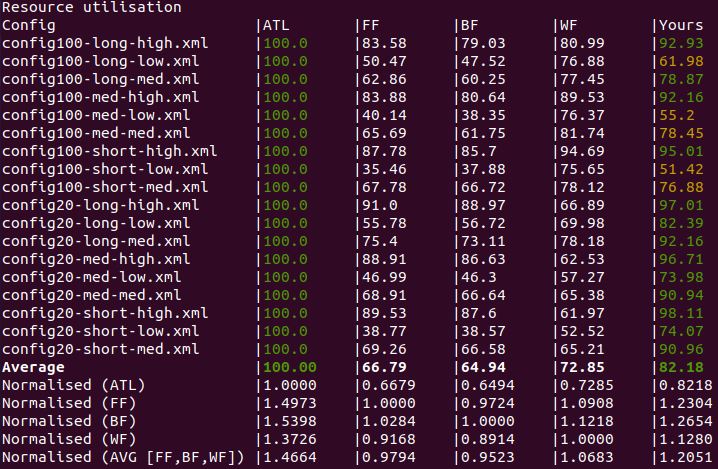
#### Turnaround Time

* In terms of turnaround time averages the JSA was within 20% of FF and BF. While having around a 30% cost reduction
* The JSA was much better than ATL (1758 vs 254086) and WF (1758 vs 6421) in turnaround time.
* The JSA performed closest to FF in terms of turnaround time when there was a “high” number of jobs.
* The JSA performed much better than WF except for four cases but could beat those times if the algorithm modifier number was increased



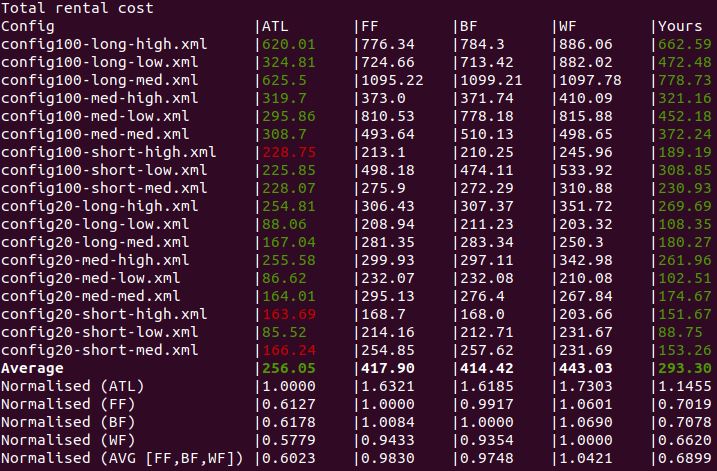
#### Resource Utilisation

* In terms of resource utilisation, the JSA performed the worst when there were a “low” number of jobs. In those cases, it performed worse than the WF algorithm. In most other cases it had the best resource utilisation of BF, FF and WF.
* JSA performed on average about 25% better than FF and BF in resource utilisation



#### Total Rental Cost

* In terms of each configuration file ATL outperforms the JSA except for three cases. But if you factor in turnaround time the JSA is far superior in terms of value. ATL is using large servers to process many small jobs and the JSA is using servers more conservatively.



# Conclusion

The JSA almost offers an average cost improvement over at ATL option which is the best performing of the four algorithms in terms of cost. The combination of total rental cost, resource utilisation, and turnaround time makes it the most well-rounded option out of the other four algorithms. One of the limitations of the JSA is that the TERM command makes an “idle” server inactive and there is an associated bootup time once it is needed again which increases the turnaround time slightly. The JSA also has the option of being modified to adjust the user’s current needs.

# References

GitHub repository: <https://github.com/fjack2114/COMP3100-Stage-2>