**Cloud Job Scheduler**

Scheduling in Distributed Systems

*COMP3100 - Distributed Systems*

Robinson Le 45948852

# Introduction: (1/2 Page)

The overall purpose of this project is to create a client to schedule jobs to various servers within a simulation, which follows a set communication protocol. The simulation tool *ds-sim* has been provided and it simulate scenarios based on set configurations files of the collections of servers and jobs to be scheduled. [2]

In stage 2 of this project, the client ‘*Cloud Job Scheduler’*, has the assignment of scheduling all jobs that will be generated in the simulation with a custom designed algorithm to the designated server. As with the requirements of the project, the *IP address* and *port number* are to be hard coded, to allow for simple testing of the client with the provided simulation program. The main achievement that is be accomplished in stage 2 is have a new scheduling algorithm implemented into the client, where it should achieve certain performance metrics. For the algorithm in this client, its main objective is to provide reduced rental costs while maintaining a high resource utilisation and still be able to deliver a low turnaround time.

Just as it was in stage 1, error handling is not necessary for this client with the current specifications and requirements.

# Problem Description: (1/2 Page)

To effectively schedule jobs in a distributed system environment it must have a focus on providing good performance metrics, which are *Turnaround Time* (TT), *Resource Utilisation* (RU) *and Rental Costs* (CO). The current three baseline algorithms *First-Fit* (FF), *Best-Fit* (BF) and *Worse-Fit* (WF) have been provided as a comparison, where they all have areas in the performance metrics that could be improved or modified. The algorithm *allToLargest* (ATL) is to be also used, which was the primary goal of stage 1 as a comparison point as well.

The custom-designed algorithm currently implemented in the Cloud Job Scheduler has the focus of minimising rental costs, ensuring a high resource utilisation while maintaining a respectable turnaround time. This was chosen as there was a need for a middle ground between ATL’s ability to have a very low cost and maintain an exceptionally high resource utilisation but its drastically poor turnaround times.

With the current design of the algorithm, it has yielded improvements in all performance metrics when compared to the baseline algorithms and ATL thus, fulfilling its goal of being the middle ground.

# Algorithm Description: (1 Page)

This new custom-designed algorithm has been referred in the Cloud Job Scheduler simply as *NEW*, to clearly signify that it is indeed the new algorithm.

Its main basis is to schedule jobs to the smallest server with the least of waiting jobs currently present. This ensures that idle and active servers are used for job scheduling where this will result in a higher resource utilisation and that there is a wide spread of servers used to reduce turnaround times. By having the smaller servers used up first, it allows for the reduction of rental costs.

**Example**:

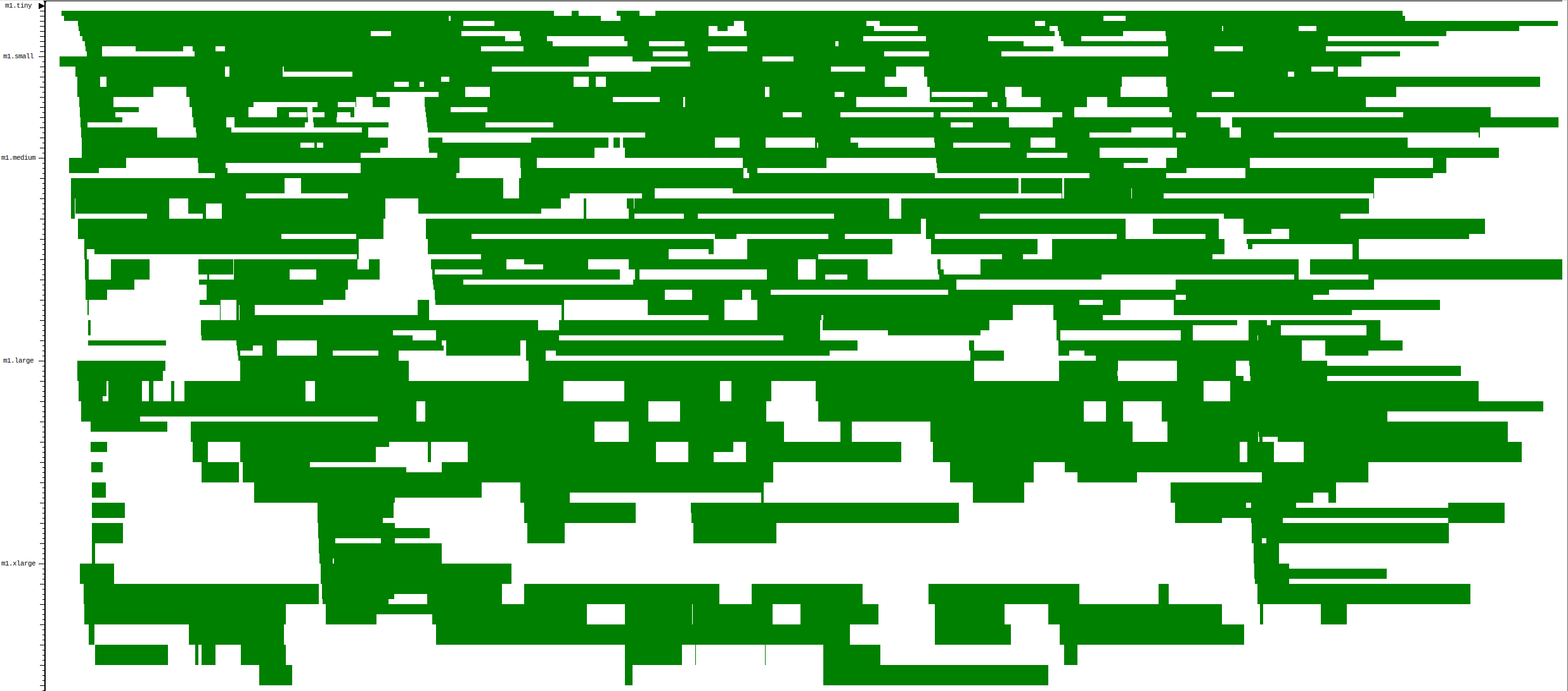


Figure - ds-viz [3]

In figure 1, the sample config ‘ds-S1-config02—demo.xml’ [4] which has been provided from the COMP3100 – Distributed System unit, was used to demonstrate the Cloud Job Scheduler in its ability to schedule jobs with its *NEW* algorithm. The vertical axis is used to represent each server present in the current simulation config and the horizontal represents the time passed during the simulation. This figure shows when a server is being used by representing a server’s utilisation with a green bar spanning horizontally reflecting how long it was used.

As shown, we can see that the Cloud Job Scheduler tends have the smaller (lower amount of CPU cores) servers more utilised as it focuses on finding on the smallest server, after checking if another server has had less waiting jobs. This is apparent when looking at the larger servers “large” and “xlarge” where large gaps of inactivity can be seen, where this causes the average resource utilisation to be lower. This negative is outweighed by the algorithm’s strength of having the smaller servers used more than the larger servers, as it ensures their utilisation is kept high and ensures that a lower rental cost is achieved. This is ideal as it costs much more to run the larger servers than it does for the smaller servers.

The figure also displays how those large gaps within certain servers could have been filled up with jobs to ensure less servers were used in general, thus further improving the resource utilisation and reducing rental costs.

# Implementation: (1/2 page)

This algorithm was implemented in the ‘Scheduler’ class, where it was provided its own scope and named “newAlgorithm()”. This class has the class attributes of an ArrayList called serverList, which stores the list of servers available to have the current job scheduled to. The algorithm is called from within the “schedule(String algorithm)” method, where it determines which algorithm is to be used based on the command line arguments provided on execution.

The class and its various methods have calls to the ‘Server’ class where it holds the server current specifications such as its: server-type, server-id, state, current-start-time, CPU-core-count, RAM, disk-storage, waiting-jobs and running-jobs. The methods within this class are “getServerTypeID()” where it has the function of returning a String with the server’s type and ID with a whitespace separating them, this is to be used when sending the scheduling message. The “getCoreCount()” and “getWaitingJobs” both have the functionality of parsing the strings stored in the class into an integer which can be used in other classes for purposes such as comparisons.

The “newAlgorithm()” begins by assign a Server object with the name of best and calling the “shortestQueue(ArrayList<Server>)” method to find the smallest server that also has the least amount of waiting jobs, where it aims to find the smallest server with the method “getSmallestServer(ArrayList<Server>)” and checking with “Server.getWaitingJobs()” to find the server that best fits the criteria. The “newAlgorithm()” will then proceed with returning the resulting server from shortestQueue() and calling the “getServerTypeID()” from the ‘Server’ class.

# Evaluation: (2 pages)

The setup used to effectively evaluate the improvements of the custom-designed algorithm in comparison to the baseline algorithms and ATL includes:

* Ubuntu
* ds-sim
* test\_results

The *test\_results* binary script was generously provided by the COMP3100 – Distributed System unit in order to allow the preparation of students for the live demo as well as giving an early insight in the marks that could be received. [5]

By using the provided binary script, it uses the eighteen sample configs supplied from the *ds-sim* GitHub repository. [6] It then runs the Cloud Job Scheduler with the eighteen configurations and compares the results generated from them with values derived from the baseline algorithms and the ATL algorithm, where an average is determined and is then normalised to easily show the differences.

The eighteen sample configurations have a wide variety of jobs and servers to show the algorithm’s performance in as many scenarios as possible. These variations include the job’s lengths, requirements and when they are to be scheduled, for the servers it also includes how many are present in the simulation, their hardware specifications, and their rental costs.

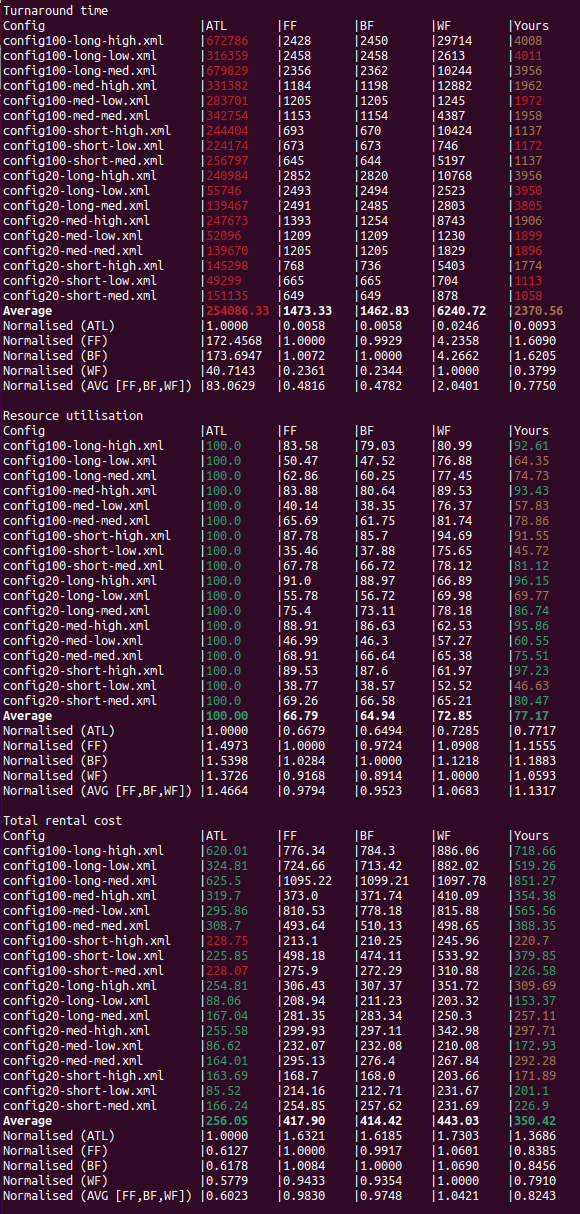
From the *test\_results* script it displays the results typically generated from the summary after a *ds-sim* instance is completed in a simple format to easily compare between the algorithms. This is seen in figure 2, where the colours of the numbers in the ‘ATL’ and ‘Yours’ shows the algorithm’s performance against each other and the baseline algorithms. Green in the ‘ATL’ column meant it was able to beat the Cloud Job Scheduler in that performance metric, and a red meant it was worse than the current algorithm implemented.

Figure 2 - Results from test\_results.bin with Cloud Job Scheduler

The colours in the ‘Yours’ column represent whether the algorithm was able to beat the baselines, with green meaning it was superior to all three, brown meaning it beat two, brown being superior to one and red for being worse than all three baselines.

The configs are categorised based on the amount of servers present, represented by the number after ‘config’. The number of jobs that will be present is shown with “-long, -med, -short” and the job characteristics are presented with “-high, -med, -low” as workloads.

The results as seen in figure 2 was generated from the currently implemented algorithm in the Cloud Job Scheduler easily reveal how it provides on average an improvement in both resource utilisation and rental cost when compared to the baseline algorithms. This is also accomplished with how it provides a turnaround time that is massively lower than ATL and lower than *WF*. It will also edge out ATL with rental costs in certain specific cases, but it tends to be within its range of rental costs and resource utilisation.

From the figure, we can determine that they could have been a larger improvement in its ability to turnaround jobs and still be able to provide the improvements in the other metrics.

The disadvantages in the current implementation of the algorithm present in the Cloud Job Scheduler is its inherent weakness with scheduling the jobs effectively for rental costs, where there are scenarios with a presence of a low number of servers and high workloads that are not many in numbers. It is also apparent when it mostly consists of medium workloads with both large amount and a medium of jobs. The extreme scenario of when there are lots of servers, with a low number of jobs to be scheduled and the jobs are considered a high workload. In these cases, we can see the Cloud Job Schedule struggle to beat all the baselines, always losing out to two of them at one moment.

In resource utilisation the areas that the algorithm is weaker at include scenarios with a low amount of server present with low workloads for both low and high amount jobs as well as high work loads with a small job amount. There is also another situation with low and medium workloads that come with lots and a medium amount of jobs present with many servers.

There are cases of weakness with the algorithm in its ability to beat at least one baseline algorithm in turnaround time, with situations where in both low and high number of servers with all job numbers again that low workloads will result in being worse than all the baselines. There is another case when there are not many servers present and in any number of jobs that low and medium workloads will cause the algorithm to be worse than all three baselines.

# Conclusion:

Conclusively, we can determine that the new custom-designed algorithm has fulfill its purpose of being the middle-ground between ATL and the three baseline algorithms in terms of the performance metrics discussed earlier. The lack of an ability to view all the jobs that are to be scheduled and requiring the need to reassign jobs to different servers would have made a more efficient job scheduler. But this would have increased the complexity and execution times of the Cloud Job Scheduler by increasing the need for further communication between the client and server.

During the design of this algorithm, they could have been the possibility of having the algorithm having stateful information of a job’s estimated run time. This would have allowed it to schedule them more appropriately to larger idle servers in order to reduce turnaround times even further.

# References: (IEEE)

1. R. Le (2021), “COMP3100 – Project2”, May 2021 [Online]. Available: <https://github.com/RobbieLe/COMP3100-Project2>
2. Y.G. Lee, Y.K. Kim, J. King, COMP3100. Simulation Program, “ds-sim” Faculty of Engineering and Science, Macquarie University, Sydney, NSW, Feb. 18th, 2021. [Online]. Available: <https://github.com/distsys-MQ/ds-sim>
3. King J (2019), “ds-viz”, May 2019 [Online]. Available: <https://bitbucket.org/JaydenKing32/ds-viz/master>
4. Y.G. Lee, Y.K. Kim, J. King, COMP3100. Sample Configuration, “ds-config02—demo.xml” Faculty of Engineering and Science, Macquarie University, Sydney, NSW, Feb. 18th, 2021.
5. Y.G. Lee, COMP3100. Testing Script, “test\_results.bin” Faculty of Engineering and Science, Macquarie University, Sydney, NSW, Feb. 18th, 2021.
6. Y.G. Lee, Y.K. Kim, J. King, COMP3100. Sample Configurations, “config.xml” Faculty of Engineering and Science, Macquarie University, Sydney, NSW, Feb. 18th, 2021. [Online]. Available: <https://github.com/distsys-MQ/ds-sim/tree/master/configs/other>