Distributed Systems Client-Server cheapestFit Algorithm Job Scheduling

Student: Joshua Devine

Student ID: 45238278

1. Introduction

This aim of this project was to further develop the client-server simulator that was initially designed to schedule jobs based on the allToLargest specifications set in stage 1, but now design and implement an algorithm which can schedule jobs that can either result in the minimisation of average turnaround time, maximisation of average resource utilisation or minimisation of total rental cost. It was also determined based on the specifications that the scheduling algorithm developed would be compared to the baseline algorithms i.e. FF, BF, WF already implemented in ds-sim based on the objective metrics and should outperform at least one of these baseline algorithms for each metric. It was made evident throughout development that the implementation of an algorithm based on specific metrics could result in major sacrifices being made to another metric i.e. resource utilisation may be greatly improved but turnaround time will be significantly slower. Hence after careful and thorough discussion with the tutors it was clear that an algorithm would need to be designed which does improve a specific criterion, but not at the large expense of another metric thus aiming for a program which is considerate of all three objectives.

2. Problem Definition

After designing the allToLargest algorithm for stage 1 it became clear that even though this algorithm successfully achieved its intention of all jobs to the largest server, there were other considerations which determined how effective this algorithm is. These concerns were the objective metrics of turnaround time, resource utilisation, and the cost of servers. After analysis of these metrics for allToLargest it was evident that turnaround time had significantly been reduced due to its design of utilising the single largest server to schedule all the jobs sent from the client. It was determined that theses metrics clearly impact one another meaning if the design approach was to improve turnaround time, then more resources would be used. It was assumed that regardless of the conflicting nature of these metrics that there had to be a more effective approach which is conscious of these issues. This assumption was reassured throughout the development of stage 2 client-server simulation as alternative baseline algorithms FF, BF, WF displayed results which indicated that a balance between the objective metrics could be reached. For stage 2 the objective metric which was the focus of the design for the algorithm was the minimisation of total rental costs. Working out the total rental cost is done by ((End Time – Start Time)/3600)*hourly rate of the server and then by adding each worked out scheduling rental cost decision. The goal of minimising total rental costs was the chosen algorithm design philosophy as it was analysed that if this client-server program were being developed for a business client which has no understanding of the technical approach to the system, the most appealing design choice that they would clearly understand is a reduction in costs. This would be a large issue for the business client as with project development one important criteria to ensuring an effective project management is the associated costs.

3. Algorithm Description

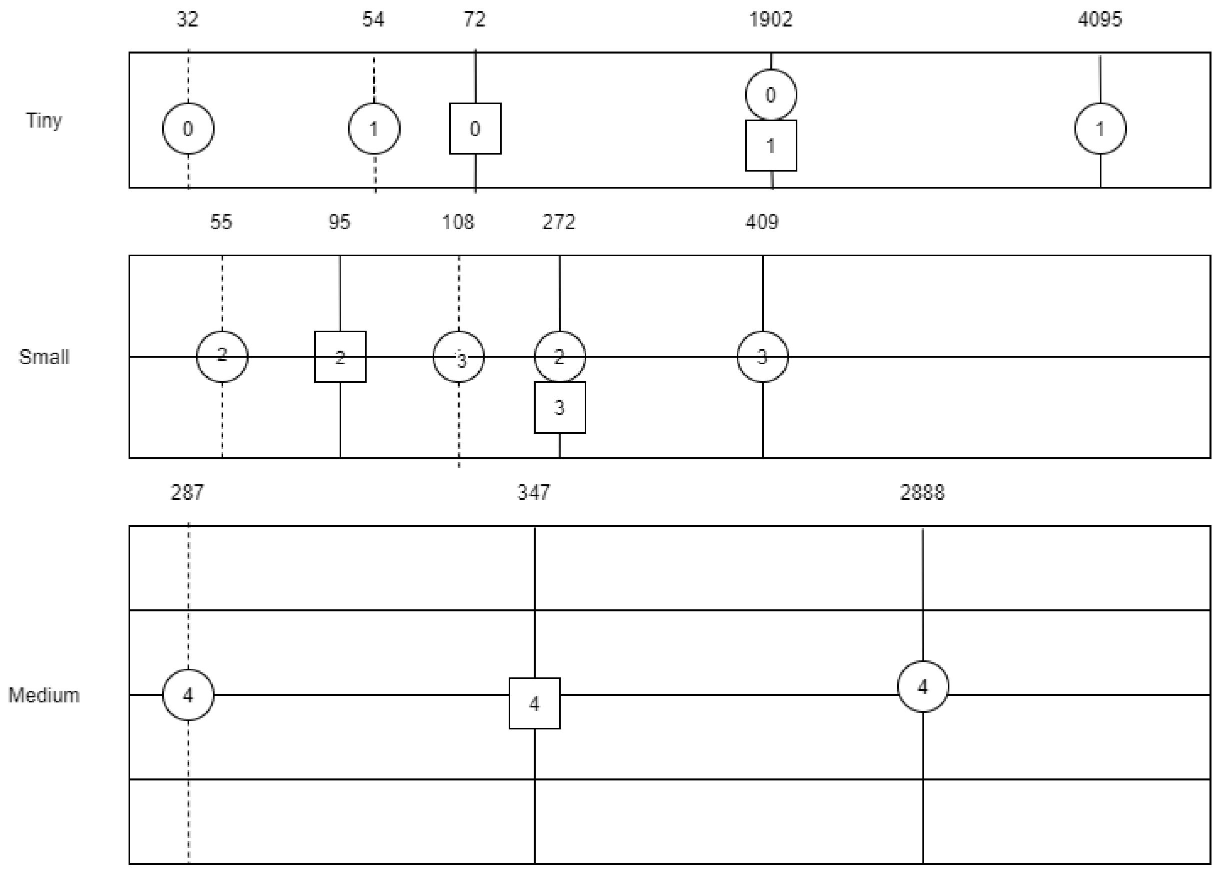


Figure 3.1 Scheduling Diagram of cheapestFit Algorithm with Config ds-config01—wk9.xml

The scheduling diagram fig 3.1 highlights the run time of the cheapestFit algorithm whilst running the config file ds-config01—wk9.xml. This is a small config which easily shows how jobs are scheduled on 3 different kinds of server. The client will initially send the GETS Capable command to the server to receive all the capable servers that can handle the current job. Initially the algorithm will check through these capable servers to see if there no waiting jobs, if there is any it will then check if there are no running jobs on the server, if there are any running jobs then it will finally proceed to the last check to see find the last server which has cores or memory or disk space that is greater than or equal to the job's cores/memory/disk space. Essentially the premise with this design of the algorithm is to find any servers which have no waiting or running jobs so that the current job can be prioritised and processed quicker. If the algorithm can't find any servers which have no waiting or running jobs then it will send the job to the last server which has greater than or equal amounts of cores or memory or disk space. With config ds-config01—wk9.xml the servers received by the client all satisfy the requirements of the algorithm of either having no waiting jobs or no running jobs hence they are scheduled to the first servers from the list. If it were a larger list of jobs then the algorithm would have to call upon the final condition as the servers eventually would have both waiting jobs and running jobs. The algorithm uses a good variety of resources which allows it to balance its costs as well ensure that turnaround time is not exponentially impacted having a better turnaround time average than WF, but worse than FF and BF.

4. Implementation

The client-side of the program implements a few techniques to allow it to function according to the stage 2 specifications. These techniques include:

A. ArrayLists

ArrayLists are the primary form of data storage for the client. ArrayLists allow for crucial data to be added, manipulated, and iterated over very easily hence the reason they were chosen to hold the important pieces of data passed into the client. Initially an ArrayList of strings is created in the getCorrectServer function which is responsible for adding and holding each server information string sent from the server when the GETS message has been sent. This ArrayList is then passed into the algorithm function cheapestFit. The creates the 2nd ArrayList of serverInfo objects serverHold which is tasked with adding and storing serverInfo objects that are created and populated with the data in the SLI ArrayList. This serverInfo list is then looped through to access each object and its required piece of data needed for the scheduling algorithm i.e. waiting jobs (serverHold.get(i).wjobs).

B. Arrays

Arrays were used extensively throughout stage 2 of the client, primarily for the purpose of having a temporary method of holding certain strings sent from the server. Arrays were not the primary storage method of data as they are limited in terms of manipulating the data or modifying the size of the array when compared to ArrayLists. However, they were useful when accessing parts of data in the strings sent from the server, as the string could be split into pieces wherever there was any whitespaces and then the data could be accessed depending on its corresponding index and parsed into a variable responsible for storing it i.e. si.id = Integer.parseInt(SLIHold[1]);.

C. Gets Capable

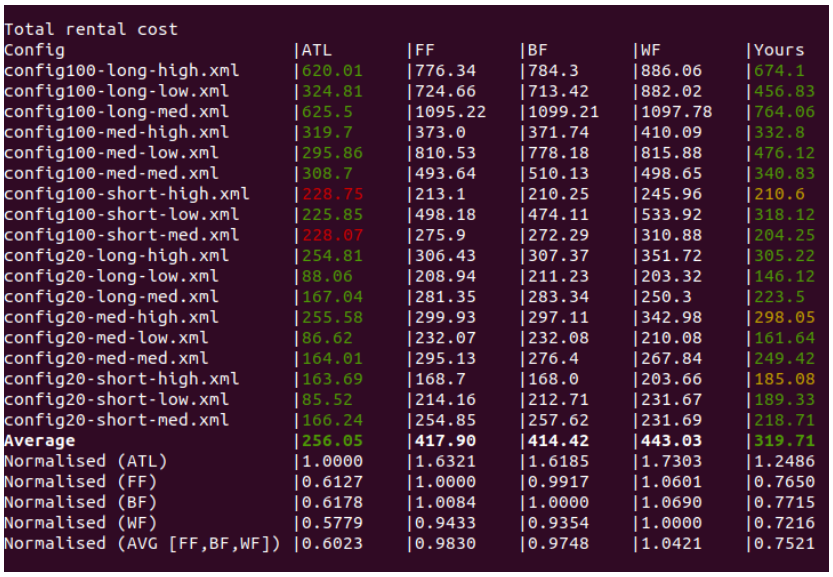
In stage 1 of this project the GETS All message was used to receive all the servers so that the allToLargest algorithm could check each server and find the largest corresponding server to send the jobs to. This was changed for the stage 2 of the project to the GETS Capable message as the message returns all the capable server information for the current job. This means that rather than sorting through all the servers to find servers that meet the needs of the current job the GETS Capable provides the already sorted list of servers which meet the requirements of the job. This change resulted in an easier implementation of the algorithm which meets the specifications of stage 2.

D. Class Objects

An additional class ServerInfo was made with a constructor so that additional objects of that class could be made. The purpose of this class was to hold the server information data which was being stored in the SLI list. Each server was split into the SLIHold array so that each piece of the server string could be accessed and assigned to the corresponding ServerInfo object variable. Each new ServerInfo object with its appropriate data was then added into a ArrayList of ServerInfo serverHold so that it could be looped through and have each bit of required data accessed so that scheduling algorithm could be completed.

5. Evaluation

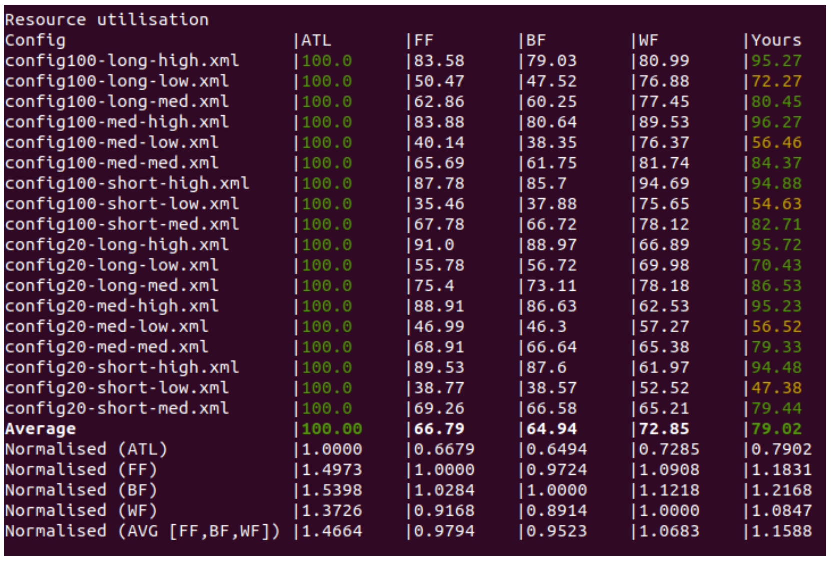
A. Total Rental Cost



```
Total rental cost
Config                      |ATL       |FF        |BF        |WF        |Yours
config100-long-high.xml     |620.01    |776.34    |784.3     |886.06    |674.1
config100-long-low.xml      |324.81    |724.66    |713.42    |882.02    |456.83
config100-long-med.xml      |625.5     |1095.22   |1099.21   |1097.78   |764.06
config100-med-high.xml      |319.7     |373.0     |371.74    |410.09    |332.8
config100-med-low.xml       |295.86    |810.53    |778.18    |815.88    |476.12
config100-med-med.xml       |308.7     |493.64    |510.13    |498.65    |340.83
config100-short-high.xml    |228.75    |213.1     |210.25    |245.96    |210.6
config100-short-low.xml     |225.85    |498.18    |474.11    |533.92    |318.12
config100-short-med.xml     |228.07    |275.9     |272.29    |310.88    |204.25
config20-long-high.xml      |254.81    |306.43    |307.37    |351.72    |305.22
config20-long-low.xml       |88.06     |208.94    |211.23    |203.32    |146.12
config20-long-med.xml       |167.04    |281.35    |283.34    |250.3     |223.5
config20-med-high.xml       |255.58    |299.93    |297.11    |342.98    |298.05
config20-med-low.xml        |86.62     |232.07    |232.08    |210.08    |161.64
config20-med-med.xml        |164.01    |295.13    |276.4     |267.84    |249.42
config20-short-high.xml     |163.69    |168.7     |168.0     |203.66    |185.08
config20-short-low.xml      |85.52     |214.16    |212.71    |231.67    |189.33
config20-short-med.xml      |166.24    |254.85    |257.62    |231.69    |218.71
Average                     |256.05    |417.90    |414.42    |443.03    |319.71
Normalised (ATL)            |1.0000    |1.6321    |1.6185    |1.7303    |1.2486
Normalised (FF)             |0.6127    |1.0000    |0.9917    |1.0601    |0.7650
Normalised (BF)             |0.6178    |1.0084    |1.0000    |1.0690    |0.7715
Normalised (WF)             |0.5779    |0.9433    |0.9354    |1.0000    |0.7216
Normalised (AVG [FF,BF,WF]) |0.6023    |0.9830    |0.9748    |1.0421    |0.7521
```

Figure 5.1 Test Output for RC

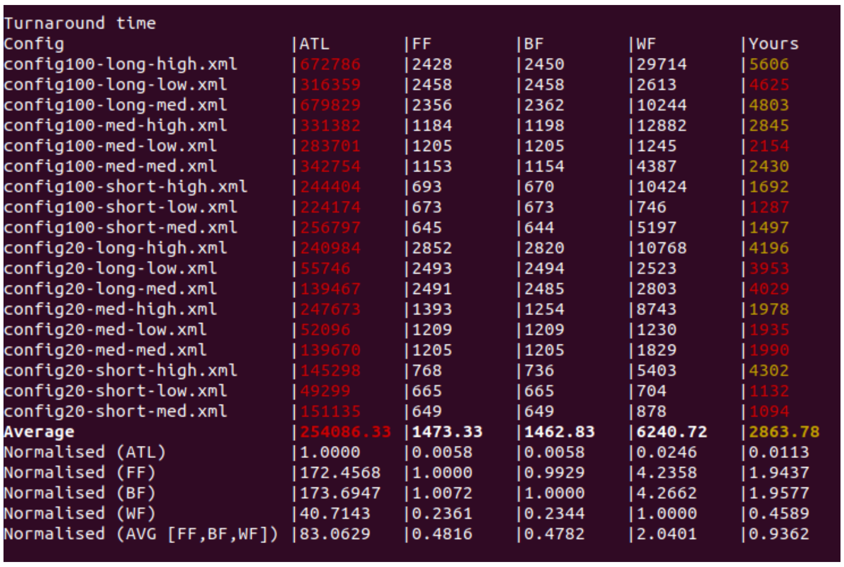
B. Resource Utilisation



```
Resource utilisation
Config                      |ATL       |FF        |BF        |WF        |Yours
config100-long-high.xml     |100.0     |83.58     |79.03     |80.99     |95.27
config100-long-low.xml      |100.0     |50.47     |47.52     |76.88     |72.27
config100-long-med.xml      |100.0     |62.86     |60.25     |77.45     |80.45
config100-med-high.xml      |100.0     |83.88     |80.64     |89.53     |96.27
config100-med-low.xml       |100.0     |40.14     |38.35     |76.37     |56.46
config100-med-med.xml       |100.0     |65.69     |61.75     |81.74     |84.37
config100-short-high.xml    |100.0     |87.78     |85.7      |94.69     |94.88
config100-short-low.xml     |100.0     |35.46     |37.88     |75.65     |54.63
config100-short-med.xml     |100.0     |67.78     |66.72     |78.12     |82.71
config20-long-high.xml      |100.0     |91.0      |88.97     |66.89     |95.72
config20-long-low.xml       |100.0     |55.78     |56.72     |69.98     |70.43
config20-long-med.xml       |100.0     |75.4      |73.11     |78.18     |86.53
config20-med-high.xml       |100.0     |88.91     |86.63     |62.53     |95.23
config20-med-low.xml        |100.0     |46.99     |46.3      |57.27     |56.52
config20-med-med.xml        |100.0     |68.91     |66.64     |65.38     |79.33
config20-short-high.xml     |100.0     |89.53     |87.6      |61.97     |94.48
config20-short-low.xml      |100.0     |38.77     |38.57     |52.52     |47.38
config20-short-med.xml      |100.0     |69.26     |66.58     |65.21     |79.44
Average                     |100.00    |66.79     |64.94     |72.85     |79.02
Normalised (ATL)            |1.0000    |0.6679    |0.6494    |0.7285    |0.7902
Normalised (FF)             |1.4973    |1.0000    |0.9724    |1.0908    |1.1831
Normalised (BF)             |1.5398    |1.0284    |1.0000    |1.1218    |1.2168
Normalised (WF)             |1.3726    |0.9168    |0.8914    |1.0000    |1.0847
Normalised (AVG [FF,BF,WF]) |1.4664    |0.9794    |0.9523    |1.0683    |1.1588
```

Figure 5.2 Test Output for RU

C. Turnaround Time



```
Turnaround time
Config                      |ATL       |FF        |BF        |WF        |Yours
config100-long-high.xml     |672786    |2428      |2450      |29714     |5606
config100-long-low.xml      |316359    |2458      |2458      |2613      |4625
config100-long-med.xml      |679829    |2356      |2362      |10244     |4803
config100-med-high.xml      |331382    |1184      |1198      |12882     |2845
config100-med-low.xml       |283701    |1205      |1205      |1245      |2154
config100-med-med.xml       |342754    |1153      |1154      |4387      |2430
config100-short-high.xml    |244404    |693       |670       |10424     |1692
config100-short-low.xml     |224174    |673       |673       |746       |1287
config100-short-med.xml     |256797    |645       |644       |5197      |1497
config20-long-high.xml      |240984    |2852      |2820      |10768     |4196
config20-long-low.xml       |55746     |2493      |2494      |2523      |3953
config20-long-med.xml       |139467    |2491      |2485      |2803      |4029
config20-med-high.xml       |247673    |1393      |1254      |8743      |1978
config20-med-low.xml        |52096     |1209      |1209      |1230      |1935
config20-med-med.xml        |139670    |1205      |1205      |1829      |1990
config20-short-high.xml     |145298    |768       |736       |5403      |4302
config20-short-low.xml      |49299     |665       |665       |704       |1132
config20-short-med.xml      |151135    |649       |649       |878       |1094
Average                     |254086.33 |1473.33   |1462.83   |6240.72   |2863.78
Normalised (ATL)            |1.0000    |0.0058    |0.0058    |0.0246    |0.0113
Normalised (FF)             |172.4568  |1.0000    |0.9929    |4.2358    |1.9437
Normalised (BF)             |173.6947  |1.0072    |1.0000    |4.2662    |1.9577
Normalised (WF)             |40.7143   |0.2361    |0.2344    |1.0000    |0.4589
Normalised (AVG [FF,BF,WF]) |83.0629   |0.4816    |0.4782    |2.0401    |0.9362
```

Figure 5.3 Test Output for TT

As seen with outputs for RC, RU and TT, the objective of reducing rental costs was achieved with an improvement of nearly 25% made when compared to FF, BF and WF baseline algorithms. Consideration towards the other metrics was also achieved with this algorithm as a 15% improvement was made towards RU and a 6% improvement made towards TT. It is evident that turnaround time was the least successful metric for the cheapestFit algorithm as it was only able to outperform the WF baseline algorithm for it, however this was expected as there is no way to have a balanced algorithm which outperforms the baseline algorithms for all objectives. This means that the time between a job being sent to a server and finished being processed is slower when compared to FF and BF. This is due to when there are no more servers than satisfy the waiting jobs or running jobs check they are then scheduled to the last server which is greater than or equal to the job's cores. During the development stages of the algorithm an early version provided results which highlighted 46% improvement too costs. This result did satisfy the initial objective of improving the rental cost metric, though extensively impacted the turnaround time, thus bringing the performance of that metric below all three baseline algorithms. This further highlights the issue of balancing the algorithm to have a spread performance will come at the cost of maximising the performance of one objective. Ultimately the pros and cons of this algorithm are:

Pros:

- Minimises rental costs
- Utilises a good spread of resources
- Has a better turnaround time compared to allToLargest and WF
- Is a balanced algorithm which considers the performance of all metrics

Cons:

- Slower turnaround time when compared to FF, BF
- Does not maximise the objective metric to its full potential
- allToLargest has lower average costs

6. Conclusion

In conclusion, the intent of designing an algorithm which schedules jobs to servers for a lower total rental cost has been successfully achieved. It may have not been able to outperform allToLargest for lowest costs however, has achieved at providing an alternative solution which considers for providing spread performance of each objective metric whilst also having lower total rental compared to the three baseline algorithms FF, BF, WF. During development it was noted and observed the effect of specialising in one metric and how it greatly effects the performance of another. Based on this observation it would be suggested that a balanced approach is the ideal design philosophy as it ensures no great sacrifices must be made, thus avoiding any potential user problems.

7. References

[1]J. Devine, "JDC0DE/COMP3100-Stage2-Group_52", *GitHub*, 2021. [Online]. Available: https://github.com/JDC0DE/COMP3100-Stage2-Group_52. [Accessed: 01- May- 2021].

[2]Y. Lee, Y. Kim and J. King, ds-sim: A Distributed Systems Simulator User Guide. Sydney: Macquarie University, Department of Science and Engineering, 2021, pp. 1-40.