Efficient-Cheap-Fit-Scheduling-Algorithm

StudentID: 45984026

StudentName: Richmond Toh

Introduction

The purpose of this project was the design a new scheduling algorithm which optimizes on one or more of the following objectives.

- Minimisation of average turnaround time
- Maximisation of average resource utilisation
- Minimisation of total server rental cost

Turnaround time – The amount of time taken to complete a job. In other words, a job's turnaround time is calculated based on the difference between the start and end time of the job.

Resource utilization – is the logical server utilization in comparison to how many cores are being utilized by the jobs scheduled to that server. In other words, the average amount of cores being utilized at any given time.

Rental cost – is the total cost associated with utilizing a sever to schedule a job. The rental cost is calculated by the time take to complete a job in seconds/3600 multiplied by the hourly rate.

However, these three-performance metrics are in direct opposition to one another and the optimization of one metric may lead to decrease in performance of the other metrics. So, in this case it is important to not over optimize one metric due to diminishing returns. For instance, if you were to reduce the server rental cost by 40% at the expense of a 200% increase of turnaround time it is not possible to justify the performance of the algorithm itself.

Problem definition

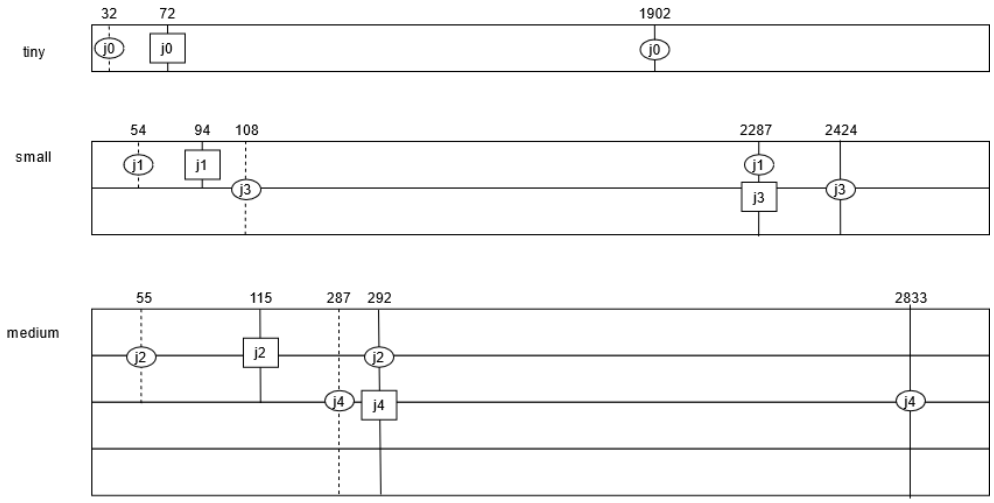
Given the three baseline algorithms First-Fit, Best-Fit, Worst-fit and the AllToLargest implementation created in stage 1 of the assignment. These algorithms have their own short comings and are useful only in certain scenarios. For instance, the AllToLargest implementation will almost always have the lowest rental cost but at the expense of an extremely high turnaround time. Whereas the other three baseline algorithms will have a slightly higher rental cost than AllToLargest but they will have a considerably shorter turnaround time.

The algorithm that I am designing aims to beat all three baseline algorithms in rental cost but at the same time be marginally similar in terms of turnaround time to all three baseline algorithms while being vastly superior to AllToLargest's turnaround time. This algorithm aims to be the most cost-effective scheduling algorithm that can complete jobs at a very cheap rate without having an extreme turnaround time like AllToLargest.

The objective of the new algorithm can be easily achieved by not over optimising the rental costs which brings on diminishing returns since decreasing the cost after you reach a certain point will inversely mean that the other metrics will suffer, which is why AllToLargest's turnaround time is inversely proportional to its rental cost.

Algorithm Description

Scheduling scenario



The schedule

```
# -------------------------------------------------------------------------------
# 1 tiny servers used with a utilisation of 100.00 at the cost of $0.20
# 1 small servers used with a utilisation of 100.00 at the cost of $0.26
# 1 medium servers used with a utilisation of 100.00 at the cost of $0.60
# ===================================== [ Summary ] ====================================
# actual simulation end time: 2833, #jobs: 5 (failed 0 times)
# total #servers used: 3, avg util: 100.00% (ef. usage: 100.00%), total cost: $1.07
# avg waiting time: 464, avg exec time: 1375, avg turnaround time: 1839
```

Using the “ds-config01--wk9” Efficient-Cheap-Fit-Scheduling-Algorithm [1]

Description of the algorithm

Essentially the algorithm which I designed takes in an ArrayList of server objects, the algorithm assumes the ArrayList only contains servers which are capable of running the job. The objective of the algorithm is to return the server type and server id of the server which has the least estimated waiting time, in other words it returns the optimal server with the lowest runtime to schedule the next job to.

```
public String getServer(ArrayList<Server> serverList) {

    //Iterates through the serverList and finds the server with the lowest estimated waiting time.
      Server optimalServer =  Collections.min(serverList, new Comparator<Server>() {

            @Override
            public int compare(Server S1, Server S2) {
                //Compares Server1 with Server2 in the list and check which has the lowest estimated waiting time.
                 return Integer.compare(S1.getEstimatedWaittime(), S2.getEstimatedWaittime());
            }
       });

      //returns something like "xlarge 1"
      return optimalServer.getType()+" "+optimalServer.getId();

}
```

Discussion of scenario

Within the given Scheduling scenario, my algorithm first schedules job 1 to the server with the lowest least estimated wait time which happens to be the tiny server at “32”, then job 2 is scheduled on the small server at “54”, then the job 3 is scheduled on the small server at “108” and finally job 4 is scheduled on the medium server at “287”.

However, there is a flaw within the algorithm where job 3 should be instead scheduled the medium server and not the small server; but nevertheless, the algorithm does improve on rental cost while not averaging a high turnaround time.

Implementation

Implementing this new algorithm required me to update the Server class and allow the estimatedwait time and estimatedruntime to be stored.

Data Type	Before (Stage 1)	After (Stage 2)
String	type	type
Int	id	id
String	State	State
Int	bootupTime	bootupTime
Int	coreCount	coreCount
Int	Memory	Memory
int	disk	disk
Int	**N/A**	estimatedWaittime
int	**N/A**	estimatedRuntime

When the client sends the "Gets Capable" message + the job requirements (core, memory, disk) to the server; the server will send back a message containing all the servers which are capable of running the job including all the information listed in the table above (Stage 2) which then needs to be added to an ArrayList of type Servers. This ArrayList is then passed into the newAlgorithm class where it searches through the server list and finds the server with the lowest estimated wait time and schedules the job to that server. Before every job is scheduled the serverList will update itself and only show the server which are capable of running the job and therefore we do not have to worry about scheduling to a server which is unavailable or does not meet the job requirements.

Evaluation

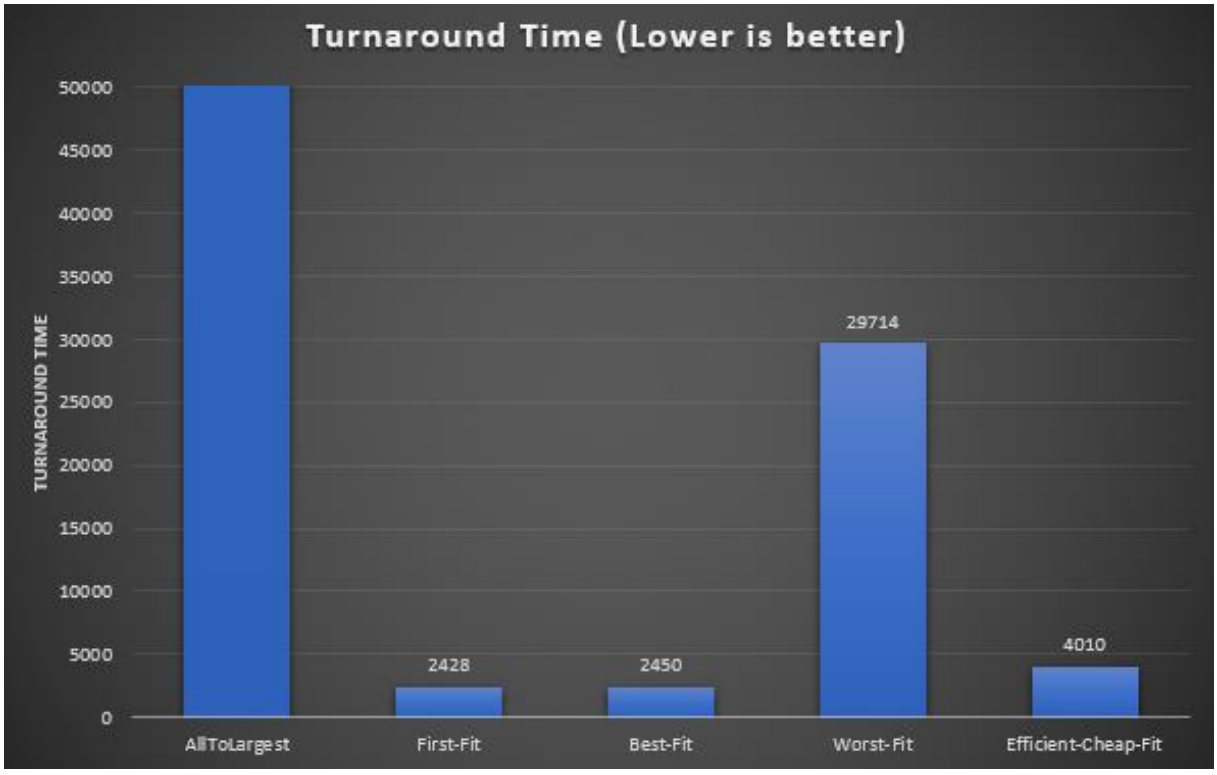


Figure 1 – Turnaround time (Lower is better)

Note: AllToLargest value is "672786", the diagram has been scaled this way so that the other algorithms metrics can be clearly seen.

As clearly demonstrated above in Figure 1, the AllToLargest is a ($\frac{672786-4010}{4010}$ x 100 = **16677.7%**) increase on turnaround time compared to the efficient-cheap-fit algorithm; and Worst-Fit is a ($\frac{29714-4010}{4010}$ x 100 = **640.9%**) increase in turnaround time.

However efficient-cheap-fit marginally loses to both First-Fit and Best-Fit by ($\frac{4010-2428}{2428}$ x 100 = **65.1%**) and ($\frac{4010-2450}{2450}$ x 100 = **63.6%**) increase in turnaround time, respectively.

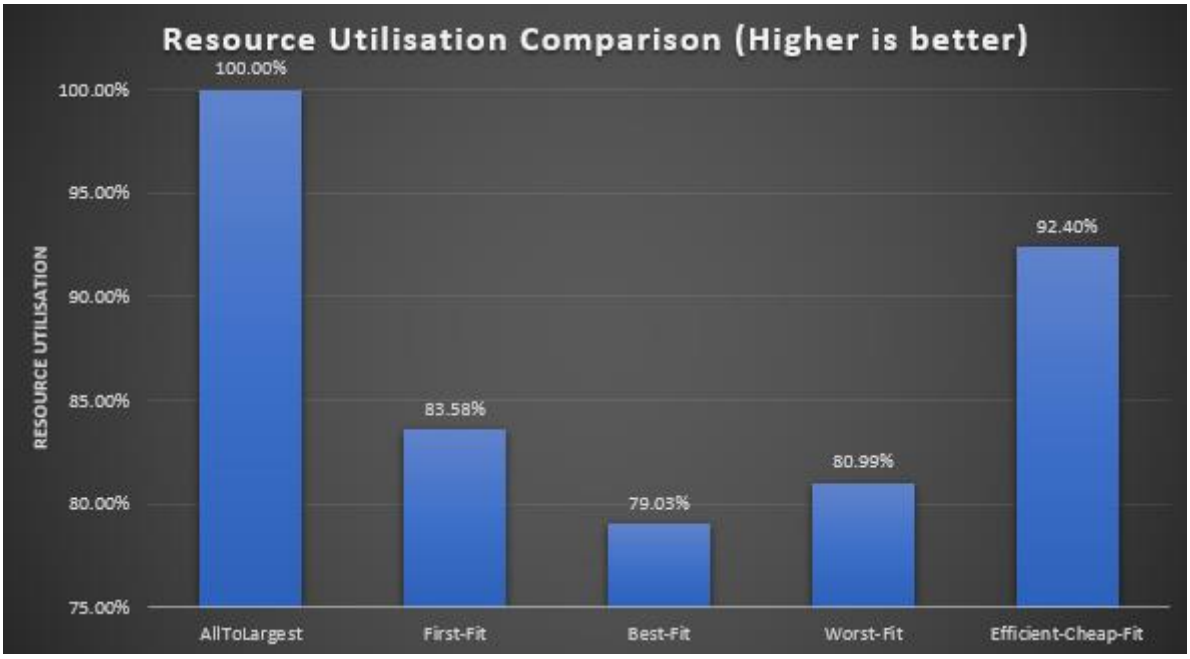


Figure 2 – Resource Utilisation (Higher is better)

In terms of resource utilization, you can clearly see that Efficient-Cheap-Fit has better resource utilization than Fit-fit, Best-fit and Worst-fit by 8.82%, 13.37% and 11.41% respectively, while only being 7.6% worse than AllToLargest.

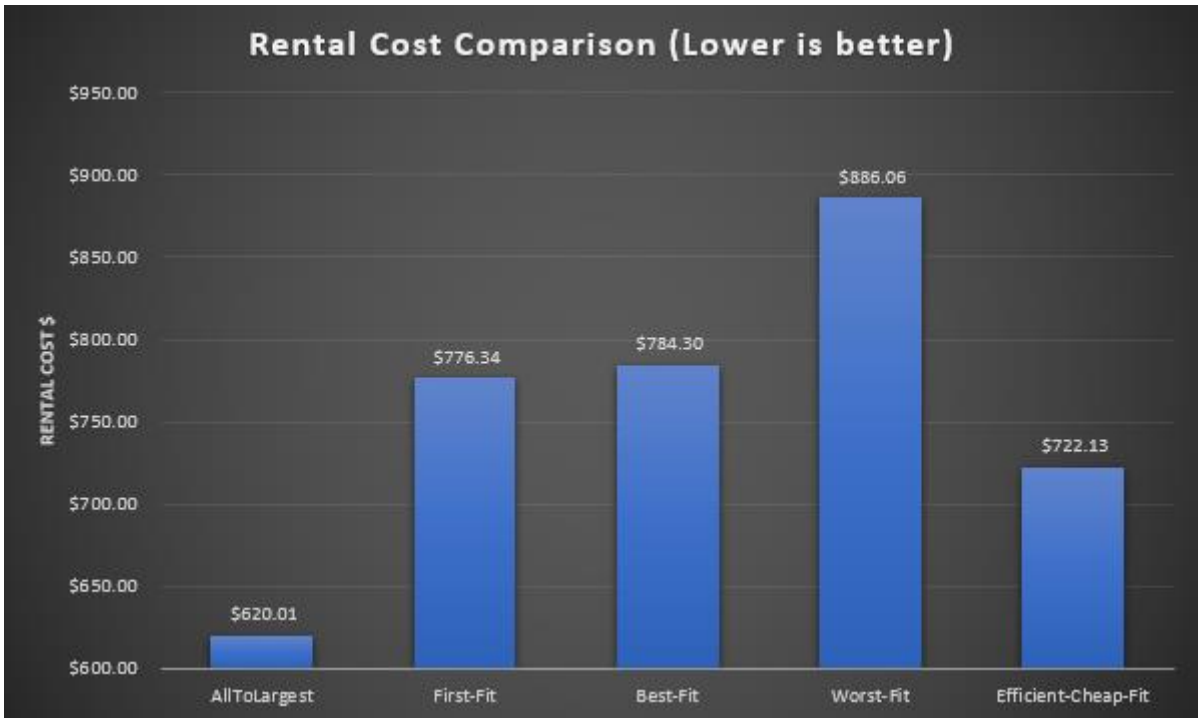


Figure 3 – Rental cost (Lower is better)

As you can see in Figure 3 it is clearly evident that Efficient-cheap-fit is better in regard to cost compared to all three baseline algorithms Fit-fit has a ($\frac{776.34-722.13}{722.13}$ x 100 = **7.5%**) increase in cost and Best-Fit has a ($\frac{784.30-722.13}{722.13}$ x 100 = **8.6%**) increase in cost and Worst-Fit is a ($\frac{886.06-722.13}{722.13}$ x 100 = **22.7%**) increase cost. However, it is not better than the AllToLargest algorithm which has a ($\frac{722.13-886.06}{722.13}$ x 100 = **14.1%**) decrease in rental cost.

Table Summary of all baseline algorithms and Stage 1 AllToLargest Implementation

	AllToLargest	***First-Fit***	***Best-Fit***	***Worst-Fit***	***Efficient-Cheap-Fit***
Simulation End Time	1395077	50898	50898	512518	54082
Rental Cost	**$620.01**	**$776.34**	**$784.30**	**$886.06**	**722.13**
Total Servers Used	1	100	100	60	100
Resource Utilisation	100%	83.83%	79.14%	83.58%	92.95%
Wait Time	670371	13	35	27299	1595
Turnaround Time	**672786**	**2428**	**2450**	**29714**	**4010**

As expected, the Efficient-Cheap-Fit has a lower rental cost than all three base line algorithms but only costs slightly more than AllToLargest, however the in the turnaround time the Efficient-Cheap-Fit algorithm has a considerably better turnaround time than AllToLargest making this algorithm more suitable for complete a large amount of jobs in a short amount of time at a low cost. This algorithm sits in between Best-fit and AllToLargest, where Best-fits objective is to have the shortest turnaround time but at the expense of a higher rental cost, and inversely AllToLargest has one of the worst turnaround time but has the lowest rental cost of any algorithm.

Conclusion

Ultimately, the Efficient-Cheap-Fit-Scheduling-Algorithm which I designed successfully reduces the total rental cost compared to all three baseline algorithms while only being marginally slower in turnaround time but massively improves upon AllToLargest's turnaround time due to its overoptimization on one metric leading to diminishing returns. All in all, the algorithm I designed helps find a middle ground between the baseline algorithms and AllToLargest as it is important to understand that over optimizing one metric will negatively effectively affect the other metrics. However, in the scenario where rental cost and resource utilization are not a factor my algorithm falls slightly short in terms of turnaround time compared to First fit and best fit although in certain circumstances my algorithm does beat worst fit. The Efficient-Cheap-Fit-Scheduling-Algorithm can be improved which can be seen in the scheduling scenario on page 2 where the third job should have been scheduled on the medium server and not the small server. Nevertheless, while the algorithm could be improved even further it was not necessary as the Efficient-Cheap-Fit-Scheduling-Algorithm has already met the criteria of beating AllToLargest in turnaround time while on average it beats Worst-fit, Best-fit and first-fit in regards to the rental cost most of the time.

References

[1] https://ilearn.mq.edu.au/pluginfile.php/7168960/mod_resource/content/1/ds-config01--wk9.xml

[2] https://github.com/RichmondToh/Comp3100DistributedSystems