Tiny Waiting Time Fit

Name: Cheuk Ying Julia Chui

SID: 45333874

Introduction

In Stage 2 of the assignment we must create at least one new scheduling algorithm that optimises one of the three metrics: turnaround time, resource utilisation, and server leasing cost.

The difference between the job's start and finish times is used to calculate turnaround time. The number of cores in operation at any given time is used to determine resource utilisation. The cost of running a server is referred to as the server rental cost. It is calculated by multiplying the hourly rate by the time it takes to complete a job in seconds/ 3600.

In the newly implemented algorithm we must improved upon one or more of the following:

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

Problem Definition

The three metrics described above are opposing performance objectives, and optimising one may require compromising optimising the others. For example, if you try to minimise average turnaround time by lowering waiting time, you may wind up utilising more resources, resulting in higher server leasing costs.

The three baseline algorithms will cost slightly to rent than ATL as FF, BF and WF use multiple servers whereas ATL only utilises one, but the turnaround time will be much longer.As a result, when developing the optimisation algorithm, we try to maximise only one metric of our choosing.

Algorithm Description

The choosen algorithm that will be implemented within Stage 2 will improve on the following:

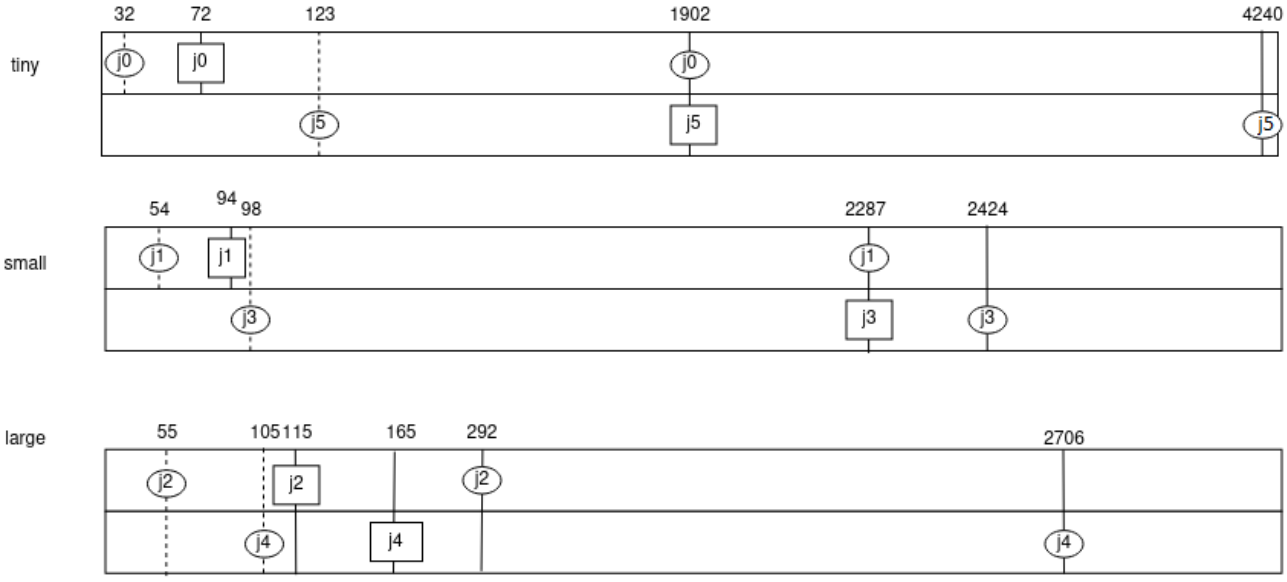
- Optimise one of the baseline algorithms
- Outperfom ALT turnaround time
- Average Turnaround time, average resoures utilisation and total rental costs outperforms one or more of the baseline algorithms, First-Fit, Best-Fit or Worst-Fit

```
# -------------------------------------------------------------------------------------
# 1 tiny servers used with a utilisation of 100.00 at the cost of $0.46
# 1 small servers used with a utilisation of 100.00 at the cost of $0.26
# 2 large servers used with a utilisation of 100.00 at the cost of $0.60
# ================================== [ Summary ] ==================================
# actual simulation end time: 4240, #jobs: 6 (failed 0 times)
# total #servers used: 4, avg util: 100.00% (ef. usage: 100.00%), total cost: $1.33
# avg waiting time: 694, avg exec time: 1536, avg turnaround time: 2230
```

Figure 1. Schedule of the Scenario

As shown in the figure above, my algorithm runs to schedule the first job within the tiny server at T:32, the second job in the small server at T:54, third job in the large server at T:55, fourth job in the small server at T:98, fifth job in the large server at T:105 and the sixth job in the tiny server at T:123. This schedule is created and shows the algorithm runs to schedule jobs to the server with the least amount of time spent waiting.

In the figure below it shows a scheduling scenario



*Note: Time 1902 where two jobs are done simultaneously, j0 is completed and j5 is still running. Similarly at Time 2287 with j1 being completed and j3 still running.

* In the figure below it shows the schedule of the scheduling scenario in ds-config01--wk9 which has been configured to simulated 6 jobs

Figure 2. Scheduling Scenario

Implementation

The algorithm used to schedule jobs to the server with the least amount of time spent waiting is depicted below.

```
public class TinyWaitingTimeFit implements ServerProvider {

    @Override
    public String getServer(String serverType, Server[] serverList) {
        Server min = serverList[0];

        for (int i = 0; i < serverList.length; i++) {
            if (min.getWaittime() > serverList[i].getWaittime()) {
                min = serverList[i];
            }
        }
        return min.getType()+" "+min.getId();
    }
}
```

Figure 3. Newly implemented algorithm

The goal of the algorithm depicted in the above figure is to find the server object within the array with the least amount of time spent waiting. After calculating the least waiting time, the algorithm will return both the server's id and the server type, and then proceed to schedule the next job inside the returned server type.

Evaluation

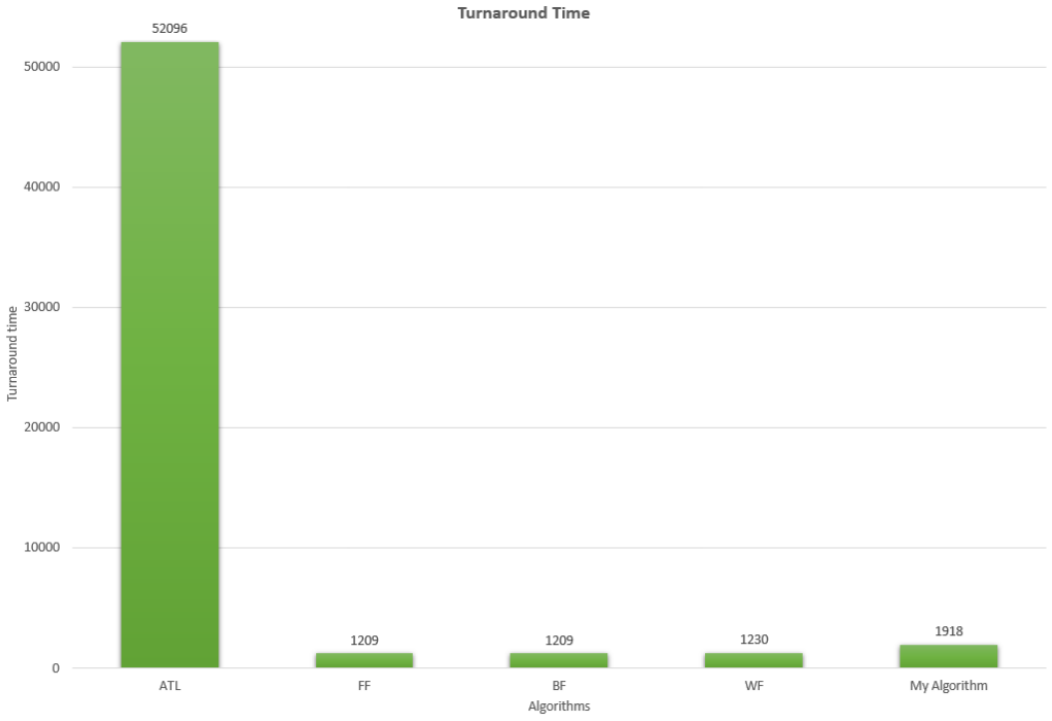


Figure 4. Turnaround Time Comparison

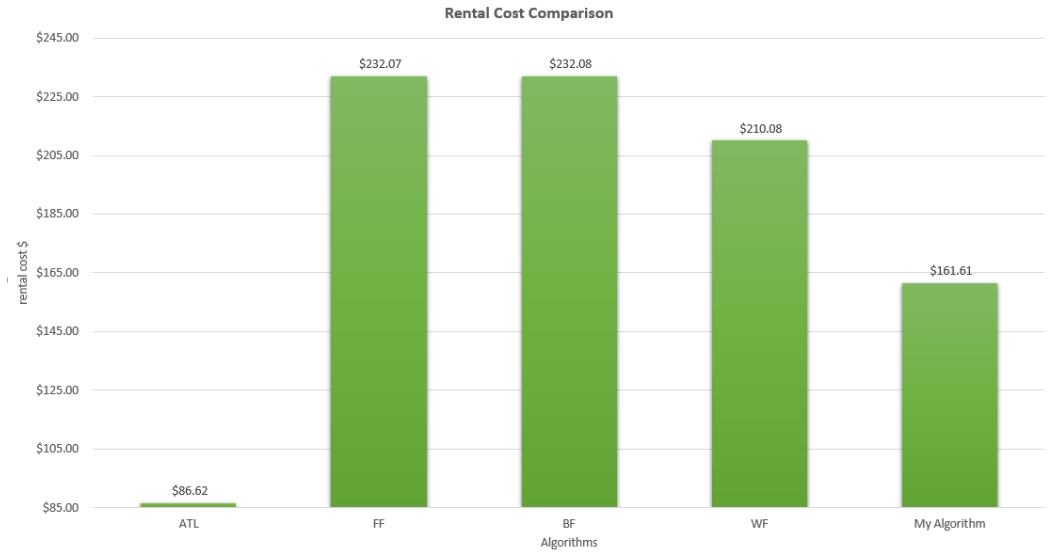


Figure 5. Rental Cost Comparison

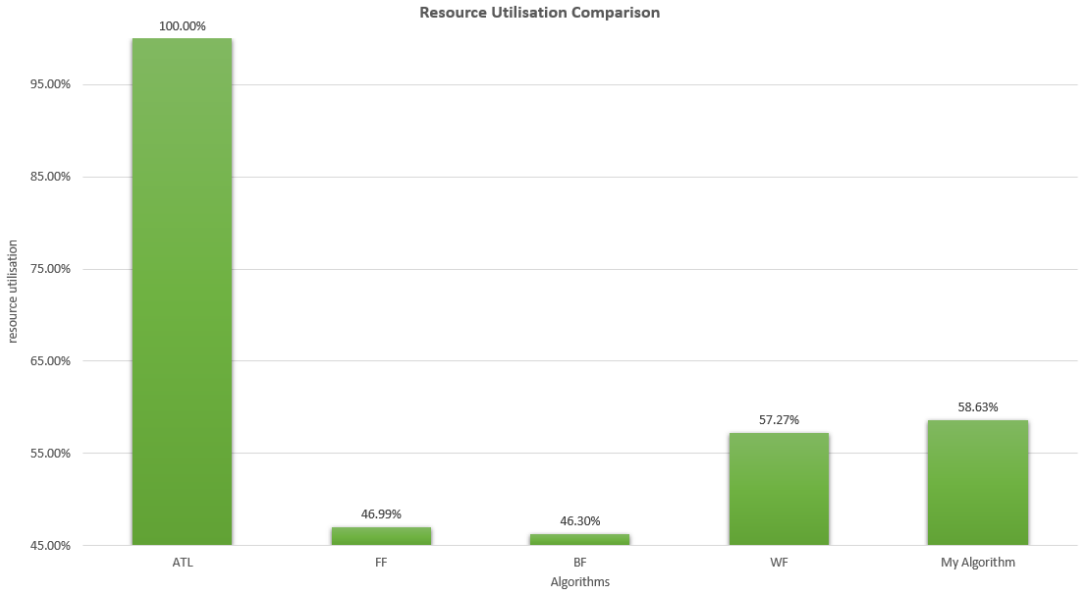


Figure 6. Resource Utilisation Comparison

In Figure 4. It displays a computed turnaround time comparison between Stage 1's ATL algorithm, the three basline algorithms and my algorithm. Lower turnaround time is better as it means the time it takes for the job to be completed is minimal and as you can see that my algorithm has outpeformed against ATL.

In Figure 5. It displays a computed rental cost comparison between Stage 1's ATL algorithm, the three basline algorithms and my algorithm. Lower server rental cost is better as it means the cost it takes for the job to be completed is minimal. As explained before server rental cost is calculated by multiplying the time it takes to finish a project in seconds/ 3600 by the hourly rate. As you can see that my algorithm has outpeformed against the three baseline algorithm.

In Figure 6. It displays a computed turnaround time comparison between Stage 1's ATL algorithm, the three basline algorithms and my algorithm. Higher utilisation is better as it means that the number of cores being used at any one time is higher and as you can see that my algorithm has outpeformed against the three baseline algorithm.

Conclusion

To summarise, compared to the three basic algorithms, my tiny waiting time fit job scheduler has a cheaper rental cost and a faster turnaround time than the ATL algorithm. In comparison to ATL, my newly developed technique is meant to schedule jobs with the least amount of run time feasible.

Although Tiny Waiting job scheduler could be enhanced further in terms of the three metrics of rental cost, resource utilisation, and turnaround time, it has improved upon the required metrics and objectives, as indicated in this report.

References

GitHub repository URL: https://github.com/JY2D/Distributed-Systems-Assig-2