**COMP 3100 Stage 2: Half Available Algorithm**

Jonathan Skirtun (45982333)

**Introduction**

Stage 2 of COMP 3100 project is to implement a scheduling algorithm that would be able to optimise one or more aspects of the scheduling process these being the minimisation of average turnaround time, the maximisation of average resource utilisation and the minimisation of total server rental cost. These objectives would be graded with respect to the First Fit (FF), Best Fit (BF) and Worst Fit (WF) and All to Largest algorithms.

**Problem Definition**

The issue presented by All to Largest developed in the first stage is the excessive time taken to schedule jobs due to poor usage of servers as such the goal of this algorithm was to improve upon the average turnaround time while keeping resource utilisation and resource cost metrics reasonable. Turnaround time was chosen as the focus of this algorithm due to the abysmal turn around time of the All to Largest algorithm which is unlikely to handle many jobs efficiently in a workplace environment. Other metrics such as cost were improved compared to the All to Largest algorithm but were not the focus of the scheduling algorithm.

**Algorithm Description**

The algorithm developed initially scheduled jobs based on the minimum required cores, memory, and disk space available only scheduling additional jobs if all servers were being utilised however, it was discovered that scheduling the jobs in this manner was inefficient as servers would of schedule at most one job per server. This was highly inefficient as cores were left unused, a solution to this problem was developed which would schedule jobs on half of the servers available and only scheduling on other servers once those servers were fully utilised where it would schedule the job to the largest servers. This method decreased the turnaround time substantially and left the resource utilisation and cost comparable and, in some cases, better than the FF, BF and WF algorithms.

When the job scheduler receives a job it constructs a SCHD message with the job ID and passes the job information to the getHalfAvail() scheduling function. This component sends a GETS Avail message to the server using the information of the job parsed in the scheduling loop and receives a list of servers that are available to run the job, if all servers are currently unavailable it will send a GETS All message to the server which it will use to schedule the job instead, in both cases the resulting server list is copied into an array which is then searched for the most suitable server to run the job. The suitability of a server is determined by five factors, core count, memory, disk space, ID and jobs waiting. First the scheduler checks the core count of the sever against the cores required by the job being scheduled and ensures it schedules on every other server to maximise the utilisation of the servers running and minimise the cost of running jobs on multiple servers. Next the server memory and disk space are checked against the job’s requirements to ensure that the server can handle the job. Finally, the server is checked for the number of waiting jobs, this is to ensure that jobs are scheduled evenly across server types and do not overload individual servers with jobs which would impact the turn around time of the algorithm. In the case of using a server list created by GETS ALL the algorithm will schedule the job to any available server it can which are frequently the even numbered servers which have been unused to save on costs and optimise turnaround terms.

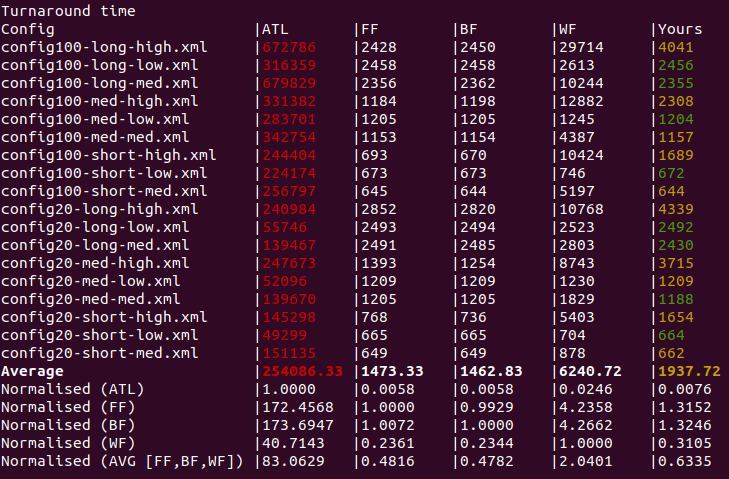
**Implementation Details**

The scheduling algorithm uses a Server class which contains the server type, ID cores memory, disk, waiting jobs and running jobs. This class is used to hold server information for comparison and scheduling. For converting job messages from the server into data that can be stored in an array a parsing function is used in conjunction with a job array which holds the information. To send and receive messages from the server sending and receiving functions were developed and used to communicate with the server.

**Evaluation**

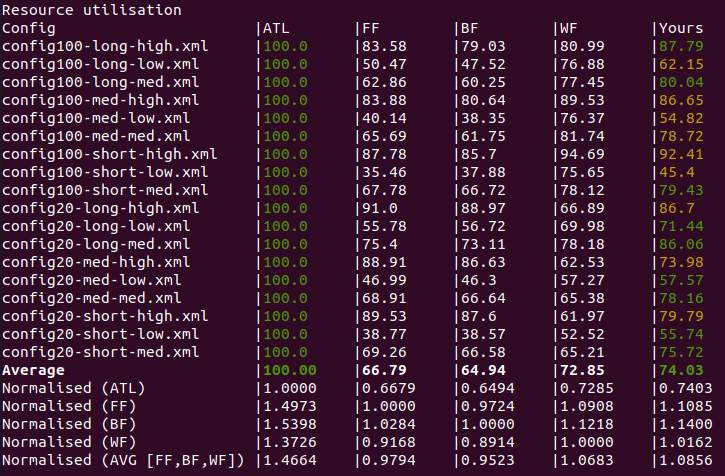
**Turnaround time**

The Half Available scheduling algorithm improves upon turnaround time greatly in comparison to All to Largest as it was intended to do when run against the configurations and test case provided using the “./test\_results “java Stage2Client -a fff” -o tt -n -c ../../configs/other” command in the terminal window. Compared to the WF algorithm, Half Available reduces turnaround time in each config file by a substantial amount and is a better choice overall. Half Available however when compared to FF and BF is marginally slower in about half of the configuration files, this however is not a major issue in the context of being utilised in the field as it is only slightly slower it still manages to beat or match the FF and BF algorithms in half of the configuration files.



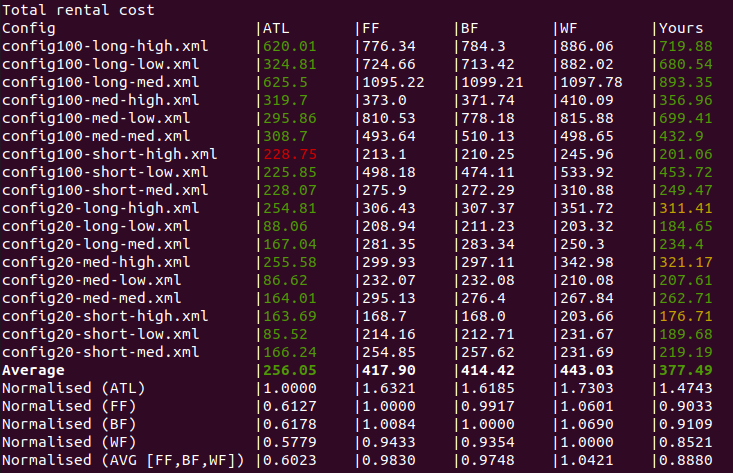
**Resource Utilisation**

Resource utilisation is the only metric which Half Available is unable to beat the All to Largest algorithm due to the algorithm’s nature of assigning jobs to one server resulting in a 100% resource utilisation. When compared to the three baseline algorithms however, Half Available can use improve resource utilisation when compared half of the baseline algorithms and manages to achieve the best average resource utilisation with the exception of All to Largest.



**Cost of execution**

Additionally, it manages to achieve a lower cost than all three of the baseline algorithms. The Half available algorithm is unable to beat the All to Largest algorithm in this respect however in all but one of the cases.



**Conclusion**

The Half Available algorithm developed for Stage 2 can improve the three metrics of turnaround time, resource utilisation and cost of execution when compared to the three baseline algorithms and the All to Largest algorithms. It does this by scheduling jobs on a select number of servers only using other servers when there are no longer any available to schedule on. This allows it to improve in all metrics and keep up with the baseline algorithms. The Half Available algorithm is suggested to be used over the baseline algorithms due to its overall efficiency and low cost of use when compared to the baseline algorithms, it works best in environments that do not always require 100% usage of all servers due to its requirement of keeping unused servers to compute jobs when there are no available servers remaining.

**References**

<https://github.com/JSGreg/COMP3100.git>