**---------------------------------------------------------------------------**

**Cost Efficient Job Scheduling Algorithm for Distributed Systems**

**Prithivi Sharma, 45705704**

**---------------------------------------------------------------------------**

# Introduction:

This project particularly focuses on creating a job scheduling algorithm that allows for job scheduling within servers to be done much more efficiently and optimises key objectives within servers like the turnaround time, the average utilisation of resources for job scheduling and the total rental cost of the server. This is achieved through the use of a function called “processed\_alg” created on a new java file, new\_alg, which is used in the client side of the system to implement a variation of the first fit algorithm to improve the aforementioned areas of job scheduling.

With that in consideration, the main goal of this stage is to ensure that job scheduling can be performed and applied on distributed systems, much more efficiently, allowing the client-side simulator and server-side simulator to be to schedule jobs and have them completed much more quickly.

# Problem Definition:

The scheduling of jobs within a distributed system involves the use of three baseline algorithms, The Best Fit, The Worst Fit and The First Fit algorithms. The problem presented by the First Fit algorithm however, is that if it does not find a server for the job to be scheduled based on the resources that are available on the first iteration and having no waiting, it will loop through the entire server to find the first server that is active or booting up and runs the server there, wasting a lot of memory in the process. As a result, this wastes a lot of memory by just allocating the memory even if the allocation of the size of the pattern for the job is bigger compared to the job or not. As a result, external fragmentation can occur from the implementation of the first fit algorithm.

The processed\_alg attempts to rectify this by running the allocated job along with a job that has previously been assigned to the server that already has a scheduled job if the submission time of the new job is less than the starting time of the job in the server, allowing both of the jobs to run in parallel.

# Algorithm Description:

This algorithm involves the use of a variation of the first fit algorithm which is used to find the first available servers. This algorithm still follows the initial condition of finding the first available servers to run jobs in which is determined through the memory disk space and the amount of CPU cores in comparison to the job’s requirements. This algorithm firstly starts off by iterating through the servers which are sorted through selection sort on this algorithm where it compares each of the jobs with the server to determine if that server has the capability to run the job. If that server has the capability to run the job, then that server assigns the job. In the event that it cannot do so, the start time of the waiting job within the server is found and those jobs are made to work in parallel.

# Implementation Details:

The implementation of this algorithm involves the use of two classes, given\_job and new\_alg. Given\_job contains the details of the job broken down into 6 different fields, submission time, the job ID, the estimated running time of the algorithm, the core count of the job, the resource memory and the resource disk. This second class, new\_alg have 2 different algorithms, server\_sort, which involves the use of the selection sort to sort the server into ascending order to be used when scheduiling the jobs. The processed\_alg is used with a given\_job object. The processed\_alg algorithm uses the sorted server array and iterates through it along with the array list which is iterated through as well to find a server which has the resource capabilities of scheduling the job.

In the event of no server being found, the start time of the waiting job within a server which has the resource capabilities but contains a job already in the server, the server then runs both of the jobs together as well. A scheduling example where this algorithm can be used is when a job needs to be scheduled and all servers already have a waiting job. This algorithm can allow the job to be scheduled much more quickly, increasing the turnaround time and the resource utilisation and the cost for keeping a server open as well for first fit.

The data structures that were used for this algorithm consisted of array lists to store the servers and arrays storing the servers and the sorted servers.

# Evaluation:

Upon use, this method allows jobs to be scheduled much more quickly and is much more efficient upon use. Due to this, this algorithm has a quicker turnaround time in comparison to the first fit algorithm and can reduce the resource utilisation, taking up less memory upon use when compared to the first fit algorithm which would require looping through the entire list of servers to find a suitable one to store the job, taking up more memory. Furthermore, the rental cost of the servers would also reduce primarily due to both jobs being executed in parallel to one another and completing much more quickly.

This algorithm also performs more efficiently than the Worst Fit Algorithm upon execution. This is mainly because this algorithm contains less conditions to schedule a job into a server and allows the jobs to be completed together rather than waiting for one job to complete after the other.

In comparison to the worst-fit algorithm, however, processed\_alg has shortcomings on some areas as it would hav much less resource utilisation than processed\_alg and therefore would be more efficient in use.

# Conclusion:

In conclusion, it is clear that processed\_alg is much more efficient than first and worst fit but would require more modifications to be better in quality than the best-fit algorithm.

# Reference:

# 