**Job Scheduler with Turnaround Time Factor for Distributed Systems**

Student Name : Saidul Islam kabboy

Student ID : 45669147

When it comes to properly balancing a variety of job requirements and available resources, the purpose of a distributed systems job scheduler is critical. The ability to maintain resource request spikes and lows effectively is the measure of a good scheduling algorithm. Designers may optimize their algorithms for three critical metrics: turnaround time, overall utilization, and cost. In most circumstances, it is only viable to significantly optimize for one or two metrics while sacrificing the others. The scheduling issues stem from resource inconsistencies and irregularly timed work requests. Creating a scheduler that delivers the best solution for the specified measure is difficult, and it must be consistent. For this assignment, I emphasized on the turnaround time. Despite the fact that my primary goal was resource utilization.

I chose to keep information within the scheduler's classes rather of the classes I had initially preferred. The algorithm begins by requesting work from a set of servers that it has access to. Each work's internal work tracker is kept up to date with real-time information, ensuring that each task schedule is as efficient as possible. It then examines the list just for servers that can currently or will soon be able to process the task request. This phase is performed without sending any server queries in order to reduce communication overhead to a minimum.While iterating over the options, each server is assigned a current workload value; the greater the number, the less desirable it is to choose that server for the new task. If there are no jobs on the server, it is an immediate zero, and the new task is allocated. Similarly, if there is enough space on a server to run this job concurrently and quickly, its workload value is 0 and it is picked. If the first two procedures fail to provide a discovered pairing, the third and most often used technique determines a non-zero workload number.

This is the most often utilized section because as more task requests arrive and overall server utilisation grows, the likelihood of obtaining a vacant execution slot decreases. This code snippet iterates over all running and waiting tasks for a specific server and computes a result that is a combination of all the tasks' projected running time multiplied by the percent of cores necessary for the task multiplied by the total number of cores on the server. The servers are compared, and the server with the lowest number is picked. All three duties just stated are carried out in the sequence just mentioned, which distributes the workload evenly in terms of task completion assessed in terms of time.

### **References:**

Github Repo:https://github.com/Saidul98/Comp3100-stage2/tree/main