

CS322:Big Data

Final Class Project Report

**Project (FPL Analytics / YACS coding): \_\_\_ YACS coding \_\_\_**  **Date: \_\_\_\_\_\_\_\_\_\_**

|  |  |  |  |
| --- | --- | --- | --- |
| SNo | Name | SRN | Class/Section |
| 1 | Bhargavi Gummanur | PES1201800211 | 5J |
| 2 | Gaurav Pandey | PES1201801145 | 5G |
| 3 | Prithwijit Banerjee | PES1201800030 | 5H |
| 4 | Rishith Bhowmick | PES1201801912 | 5G |

## Introduction

Our final project is simulating the functioning of YACS, Yet Another Centralized Scheduler. This framework consists of one Master, which runs on a dedicated machine and manages the resources of the rest of the machines in the cluster. The other machines in the cluster have one Worker process running on each of them. The Master process makes scheduling decisions while the Worker processes execute the tasks and inform the Master when a task completes its execution. In this implementation, we have one Master and three Worker processes, all running on the same machine. The Master and the Workers maintain a log of important events.

## Related work

1. <https://realpython.com/python-sockets/>
2. <https://docs.python.org/3/library/threading.html>
3. <https://www.geeksforgeeks.org/python-how-to-lock-critical-sections/>
4. <https://www.geeksforgeeks.org/program-round-robin-scheduling-set-1/?ref=lbp>
5. <https://realpython.com/python-logging/>

## Design

In this implementation, we have one Master and three Worker processes, all running on the same machine. The Master listens for job requests and dispatches the tasks in the jobs to machines based on a scheduling algorithm. Each machine is partitioned into equal-sized resource encapsulations called slots. The number of slots in each machine is fixed. Each slot has enough resources to execute one task at a time. The number of slots is decremented when a slot is said to have been allocated to a task, and incremented when a slot is said to have been freed on a task’s completion.

The Worker process listens for task launch message from the Master. When it receives a task launch message, it adds the task to its execution pool. The Worker process simulates the running of the tasks in the execution pool by decrementing the *remaining\_duration* value of each task, until it reaches 0. Once the *remaining\_duration* of a task reaches 0, the Worker removes the task from its execution pool and reports to the Master that the task has completed its execution. A job is said to have completed execution only when all the tasks (map and reduce tasks) in the job have finished executing.

All jobs have 2 stages only. The first stage consists of map tasks and the second stage consists of reduce tasks. The reduce tasks in a job can only execute after all the map tasks in the job have finished executing. There is no ordering within reduce tasks, or within map tasks. All map tasks can run in parallel, and all reduce tasks can run in parallel.

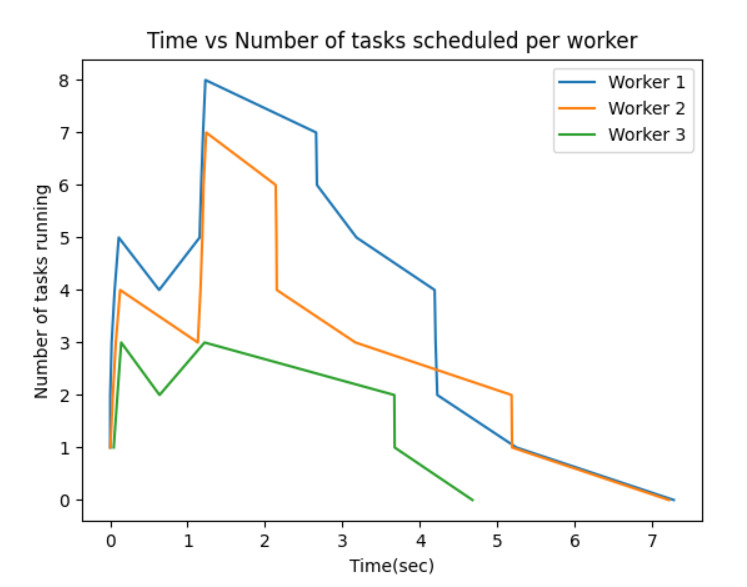
The three scheduling algorithms for a task are: *random, round-robin* and *least-loaded.*

Socket programming has been used for all communications between the Master and Workers. Each Worker’s machine will be configured with a fixed number of slots. A single slot can execute only one task at a time. When a task finishes executing, the Worker has to communicate this event to the Master. The Master keeps track of the number of slots available in each machine.

The Master and the Workers maintain a log of important events. They are saved as masterlog, worker1log, worker2log and worker3log. We use the logs to analyse results for analysis.py.

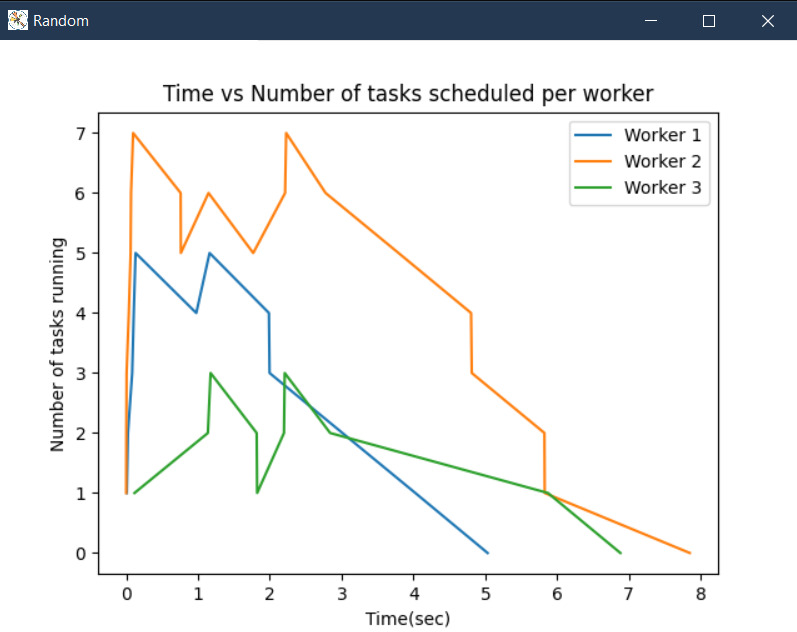
## Results

The following graphs have been made using the master and worker logs

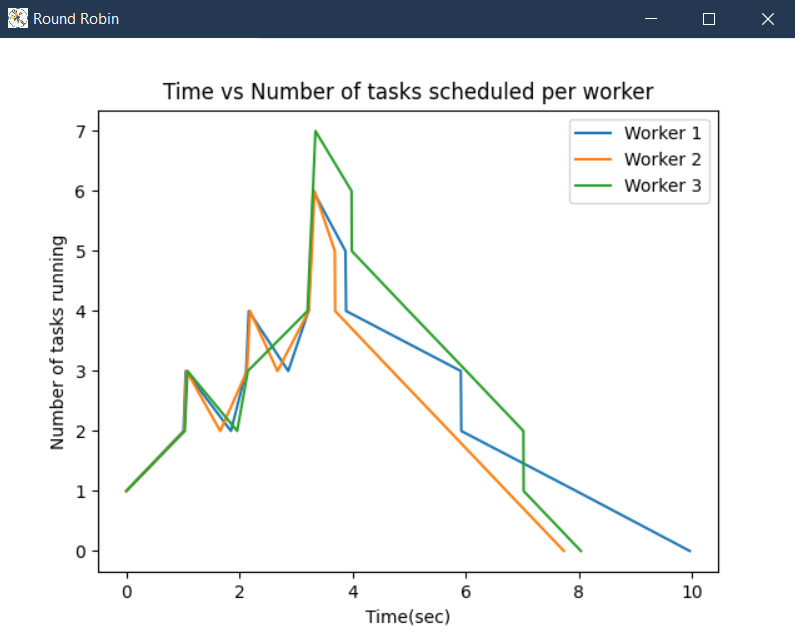


Least Loaded

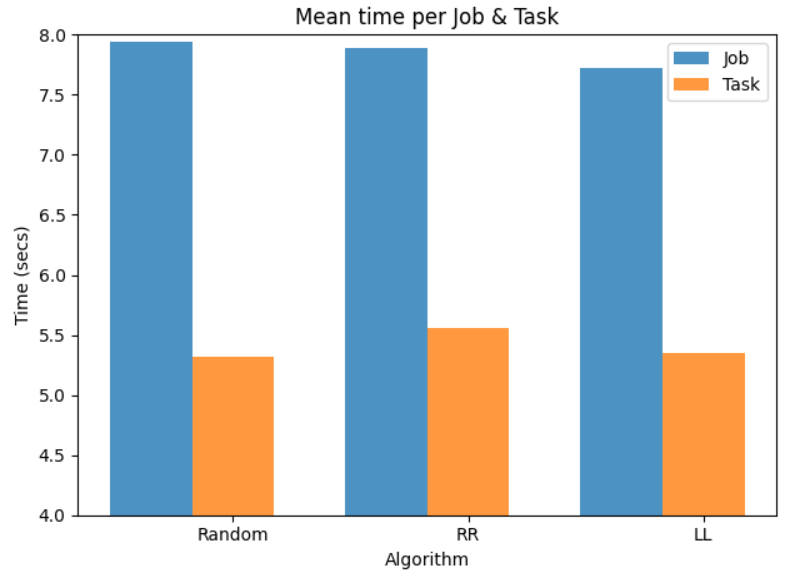
As in least loaded the worker which has the least number of slots occupied is selected, the graph is following the similar trend with the new tasks being scheduled to the workers accordingly.



In random scheduling algorithm the tasks are randomly scheduled among the workers therefore there is no particular trend seen. Same can be seen from the graph.



In Round robin the workers are given one by one in loop, therefore the number of slots being used by the workers remains the same. Similar trend can be clearly seen in the graph with each worker utilizing the same number of slots



## Problems

We had initially chosen FPL as our project, but decided to switch later on to YACS. We faced a lot of issues related to socket programming such as connection timeout error, broken pipe error, etc. While implementing locks into our algorithms also, we faced a few challenges which were mitigated by researching extensively on how to implement locking in multithreaded processes and into the three scheduling algorithms. Picking the appropriate contents and data to be output to a log file for analysis also took a considerable amount of time.

## Conclusion

From learning to implement a simulation of a scheduler, we learnt how a scheduler functions internally and how scheduling algorithms actually allocate tasks. We learnt about the functions of a master node and how it utilizes the algorithms to pick the most suitable worker node to allot the task to. We also learnt how heartbeats and regular updates to the master node are crucial to maintain a working centralized scheduler. By comparing all the algorithms using analysis.py we learnt about the efficacy of scheduling algorithms

## EVALUATIONS:

|  |  |  |  |
| --- | --- | --- | --- |
| SNo | Name | SRN | Contribution (Individual) |
| 1 | Bhargavi Gummanur | PES1201800211 | Logging, Scheduling algorithms, Multithreading, Socket programming |
| 2 | Gaurav Pandey | PES1201801145 | Analysis.py, Plots, Bash scripts, Scheduling algorithms |
| 3 | Prithwijit Banerjee | PES1201800030 | Logging, Scheduling algorithms, Socket programming, Report |
| 4 | Rishith Bhowmick | PES1201801912 | Socket programming, Locks, Multithreading, Scheduling algorithms |

## (Leave this for the faculty)

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Evaluator | Comments | Score |
|  |  |  |  |

## CHECKLIST:

|  |  |  |
| --- | --- | --- |
| SNo | Item | Status |
| 1. | Source code documented |  |
| 2. | Source code uploaded to GitHub – (access link for the same, to be added in status 🡪) |  |
| 3. | Instructions for building and running the code. Your code must be usable out of the box. |  |