

CS322:Big Data

Final Class Project Report

**Project (FPL Analytics / YACS coding):** YACS Coding **Date:** 01/12/2020

|  |  |  |  |
| --- | --- | --- | --- |
| SNo | Name | SRN | Class/Section |
| 1 | Snigdha S Chenjeri | PES1201800045 | 5C |
| 2 | Sakshi Shetty | PES1201800190 | 5C |
| 3 | Ananya V | PES1201800204 | 5I |
| 4 | Swanuja Maslekar | PES1201800369 | 5H |

## Introduction

The project we have chosen is YACS (Yet Another Centralised System). It’s a framework simulating a master (for handling scheduling) machine and worker processes (for execution). Our project consists of one master machine facilitating jobs—which handle multiple tasks—for three workers.

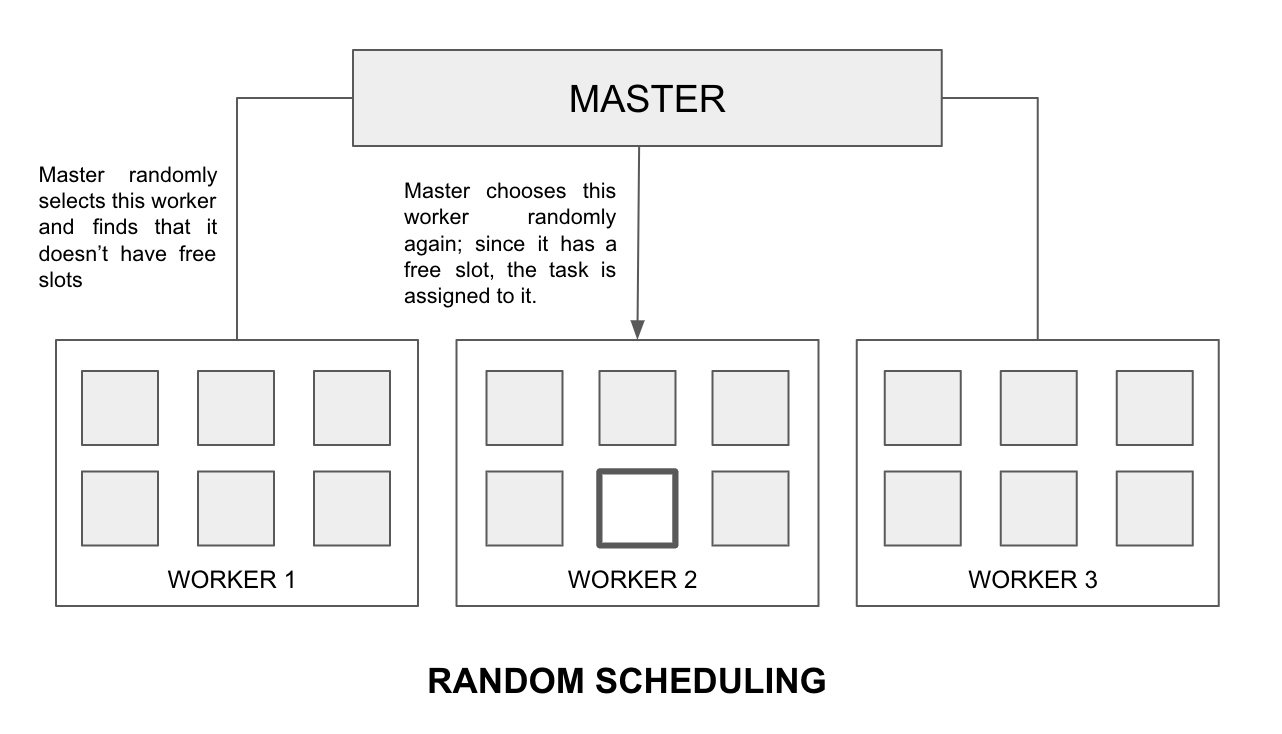
## Related work

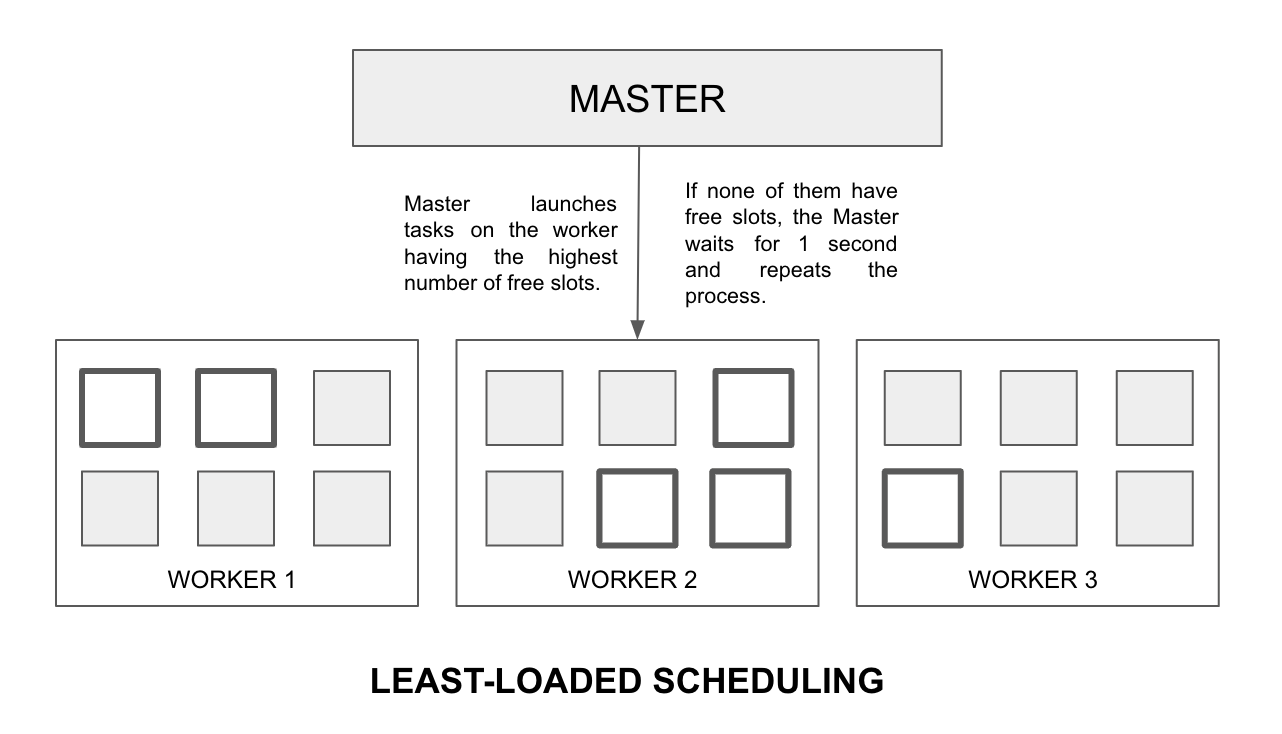
Background study material that we have read and referenced:

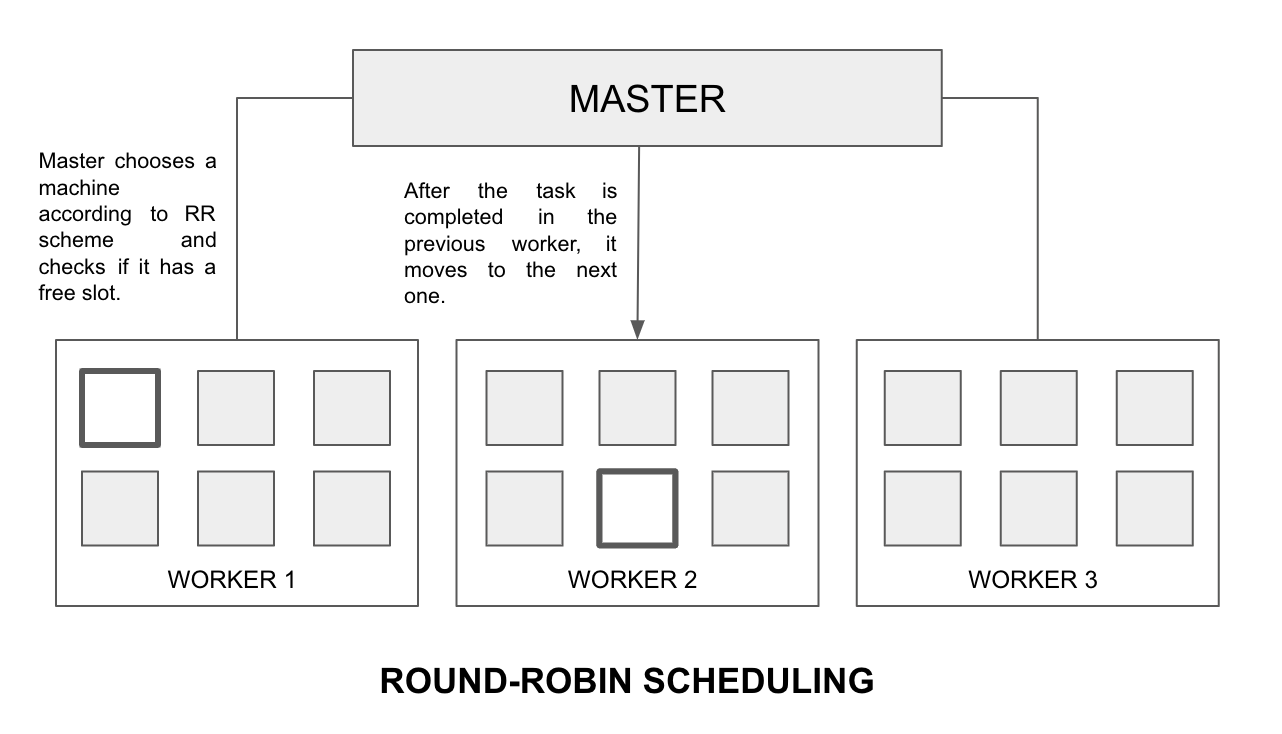
* [Theoretical references](https://elastisys.com/wp-content/uploads/2018/01/kubernetes-ha-setup.pdf?x83281)
* [Coding references](https://www.tutorialspoint.com/python/python_multithreading.htm)

## Design

With three workers running worker processes, our Master machine allocates tasks them to based on three scheduling algorithms (as we have depicted below):



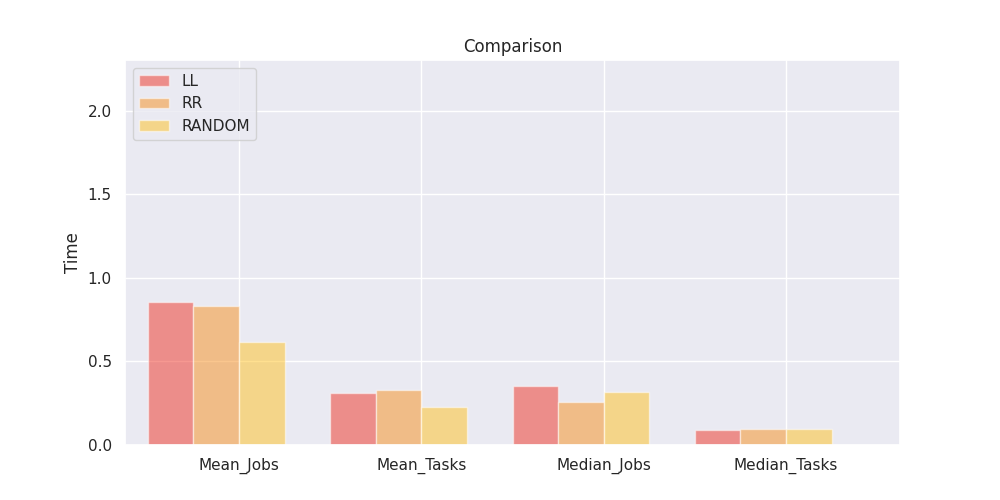




* Master.py contains three threads: one for listening to requests from Requests.py, one for allocating map tasks to workers as per requests, and one for listening to updates from workers regarding completion of map tasks and scheduling reducer tasks.
* The threads in workers listen for tasks from the master on one thread and execute the tasks on another thread.
* Tasks are allocated based on the algorithms depicted above.
* We have used classes for encapsulating the functions handling requests and allocation of workers to jobs & tasks (requests, workers, and tasks are three objects belonging to their respective classes and handle the functions corresponding to them).
* For analysis of tasks’ and jobs’ performances, we plotted graphs depicting median time, mean time, and a heatmap showing the number of tasks spread across the run time for each worker.
* Implementation of locks to take care of critical sections—reqList (list of all request objects) and execPool (list of all task objects currently being run in a worker); reqList is in Master.py whereas execPool is in Worker.py

## Results

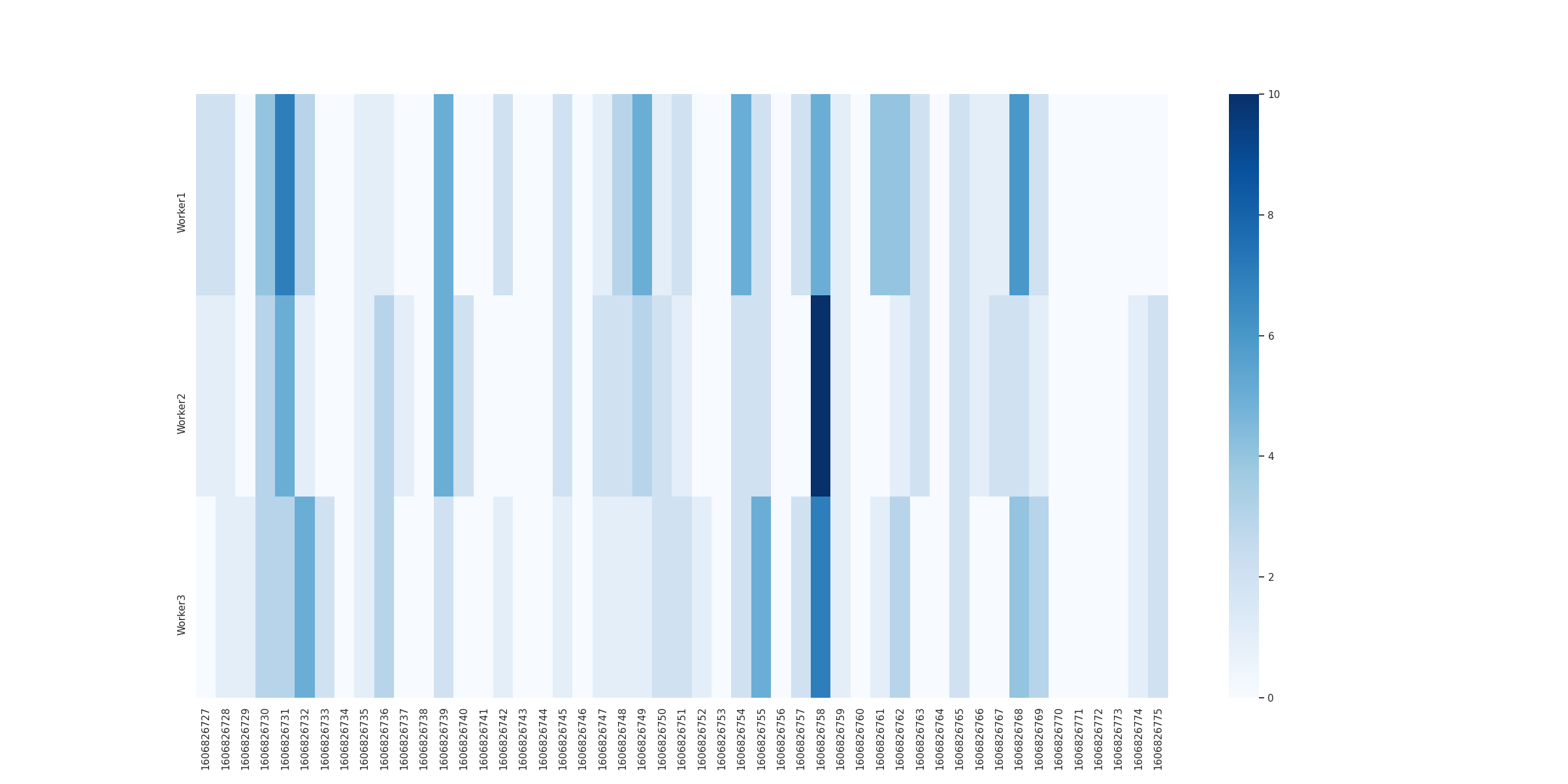
The results of programs gave us the following insights:



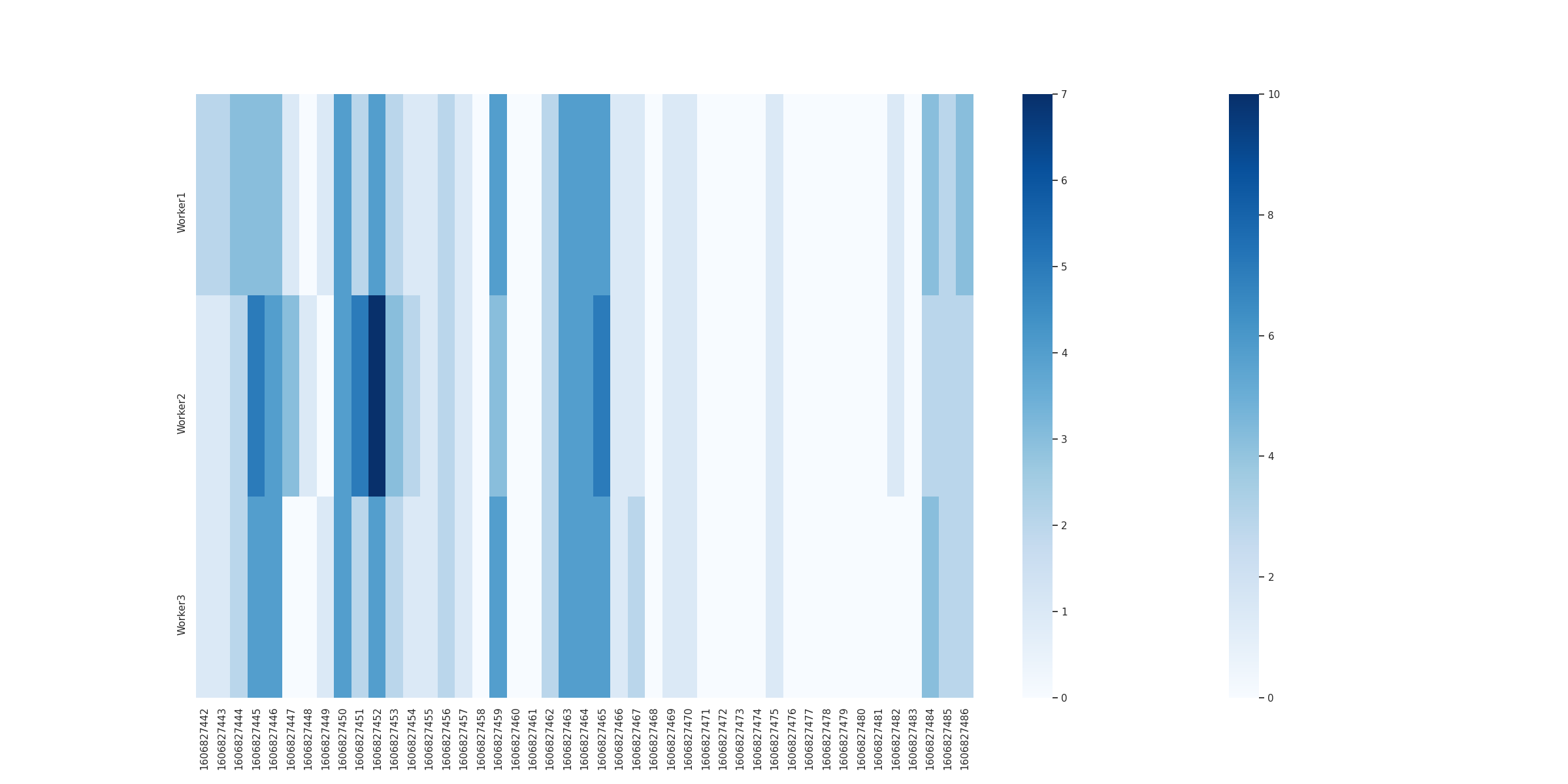
The efficiency was highest consistently for round-robin scheduling throughout the run of the program for various numbers of jobs and tasks.

Here are some heatmaps showing the number of tasks running on each worker from start to end time:

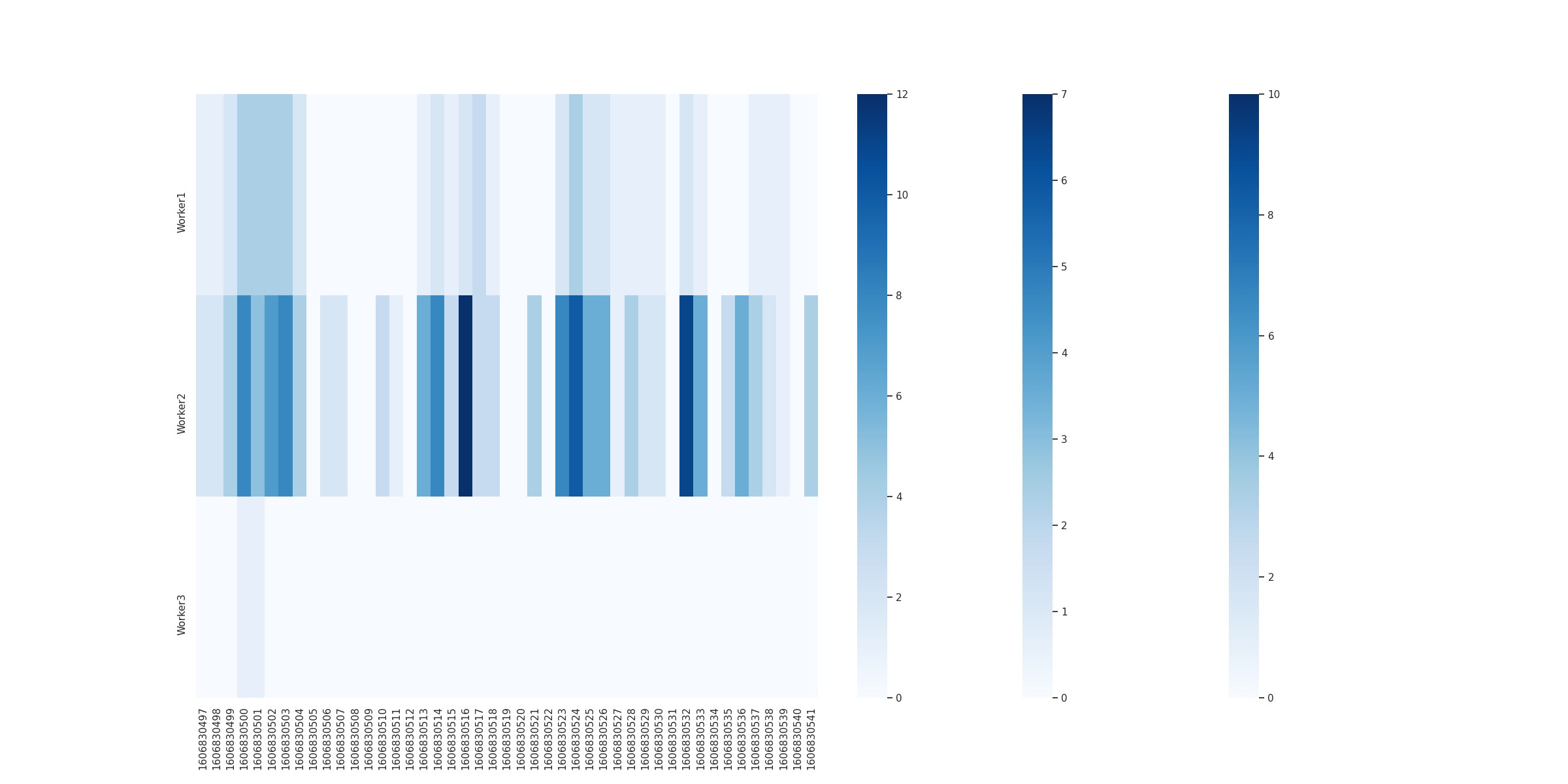
Random:



Round robin:



Least-loaded:



## Problems

* Distributing the work appropriately among threads
* Ensuring parallel distribution of work across mappers and the reducer
* Time intervals and delays (and keeping a log of these times).

## Conclusion

From this project, we learned how a centralised scheduling framework actually translates from paper to code. Although the problem statement itself is based on Big Data’s concepts, the actual coding required core OS concepts and computer networking, which were added bonuses for our learning component in this project. Scheduling as a whole is a lot clearer and more structured for us to understand now.

## EVALUATIONS:

|  |  |  |  |
| --- | --- | --- | --- |
| SNo | Name | SRN | Contribution (Individual) |
| 1 | Snigdha S Chenjeri | PES1201800045 |  |
| 2 | Sakshi Shetty | PES1201800190 |  |
| 3 | Ananya V | PES1201800204 |  |
| 4 | Swanuja Maslekar | PES1201800369 |  |

## (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. |  |