CS322:Big Data

Final Class Project Report

Project (FPL Analytics / YACS coding): __YACS coding__ Date: __1/12/2020__

SNo	Name	SRN	Class/Section
1	Tejas Srinivasan	PES1201800110	5-'A'
2	Arjun Chengappa	PES1201800119	5-'A'
3	Lavitra Kshitij Madan	PES1201800137	5-'A'
4	Priyanka Gopi	PES1201801797	5-'B'

Introduction

Yeet Another Centralized Scheduler (christened with the same acronym provided - 'YACS') is a master-slave architecture that is implemented on one machine but can be scaled to include many workers on many different machines. The 'master' is hosted on a dedicated machine and it schedules requests from and allocates resources to 'workers' present in a cluster of machines (although it is all in the same machine in the current implementation). The Master 'schedules' requests with the help of 'scheduling algorithms' (round robin,least loaded etc.) and the worker processes execute tasks assigned to them by listening for task launch messages from the master. Once this 'message' or signal to launch is received, the Worker adds it to the execution pool (of tasks to be run) on the machine it is on. The value of the number of available slots is decremented when it is occupied and incremented once the slot is freed up (tasks have finished completion).

The centralized framework consists of 1 master and 3 workers which are scheduled on different threads. The scheduling algorithms are provided via command line interface and have been described in detail in the master.py file.

The intricacies of both the worker and the tasks is described in detail in the master file in their respective class functions.

Related work

Reference books -

Core Python Applications Programming, 3rd edition by Wesley Chun

Operating System Concepts by Abraham Silberschatz, Peter B Galvin, Gerg Gagna

Operating Systems: Three Easy Pieces by Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau (University of Wisconsin-Madison)

Computer Networks: A Top Down Approach by Jim. F. Kurose for socket programming concepts

Reference websites (articles/papers etc) -

RealPython and w3schools for Pythonic syntax and implementation of data structures and other concepts required.

GeeksforGeeks for concepts for implementation of sockets,locks,threads etc.

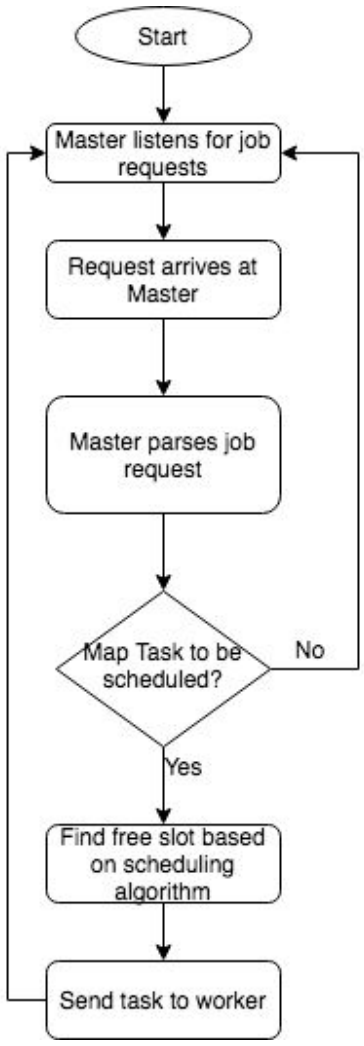
And of course, a lot of Stackoverflow :P

Design

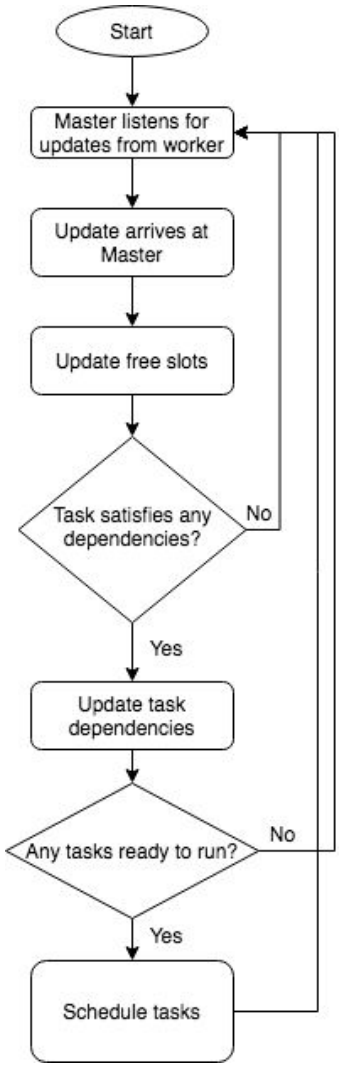
The Master listens to requests on port 5000 and updates requests from Workers on port 5001. The flowcharts from the slides provided us with great clarity on what functionality should be included in the master and worker files and hence the same have been used for our understanding and included in this report.

Flowchart of working of master -

Thread 1 - (getJobRequests)



Thread 2 - (getWorkerMessage)



Master Thread 1 - (getJobRequests)

The Master listens for job requests from worker. When the request arrives at the Master, the Master parses through this request and then if there is a Map Task scheduled, the master finds a free slot based on the scheduling algorithm of choice and then sends the task to the receiver. Else, the Master just continues to listen for updates.

Master Thread 2 - (getWorkerMessage)

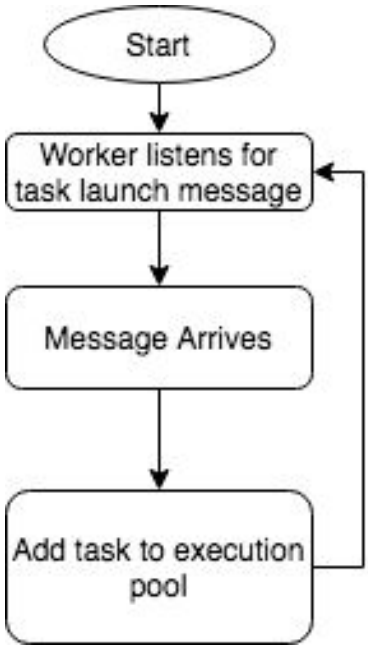
The Master listens for a message (update) from the worker. If there is a message received then the number of free slots is updated. Then if all the dependencies are satisfied, the tasks are scheduled, else the Master just continues to listen for any updates.

Master Thread 3 - (yeetacs)

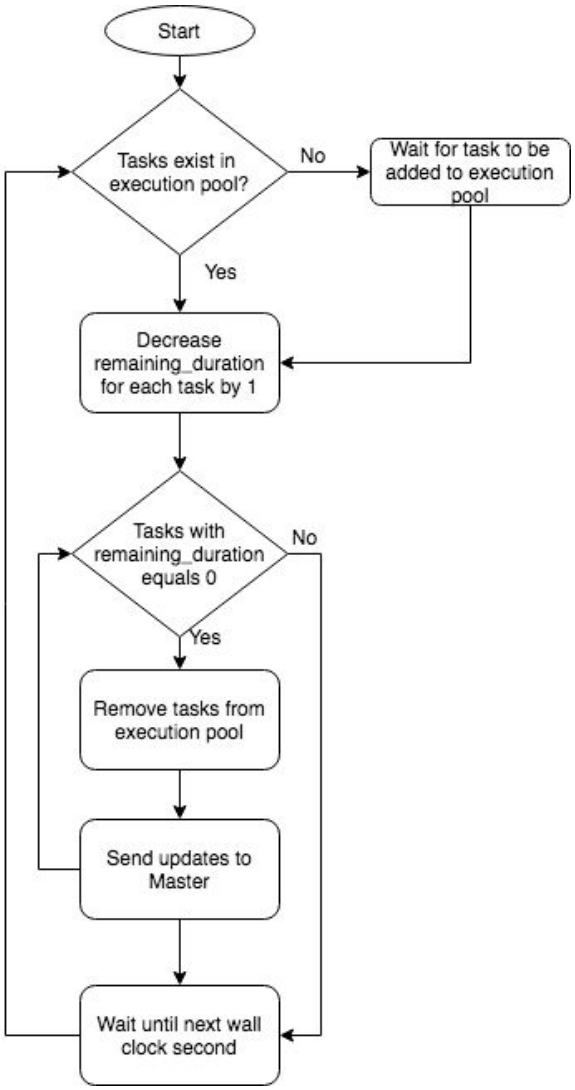
This thread runs the actual scheduler (been appropriately named) to assign requests using any of the scheduling algorithms implemented (which can be mentioned via command line arguments). The worker id is found using the scheduling algorithm and then we get the first task from the pool (or task queue) and send the worker with corresponding worker id.Every time a task is received we reduce the number of available slots by one and every time one is dispatched, a slot is freed up.

Flowchart of working of worker -

Thread 1 -



Thread 2 -



Worker Thread 1-

Worker listens for a task execution message from master and on its arrival, adds it to its execution pool.

Worker Thread 2-

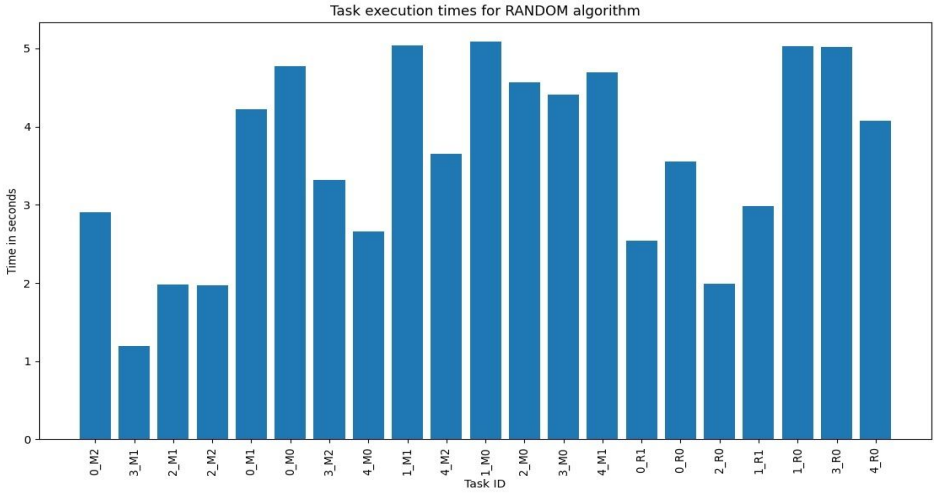
For all the map and reduce tasks present in the execution pool, the remaining duration is decremented by 1 every time and once the count reaches 0 for any task, it is removed from the execution pool. This is an iterative task and is done once every wall clock second.

Results

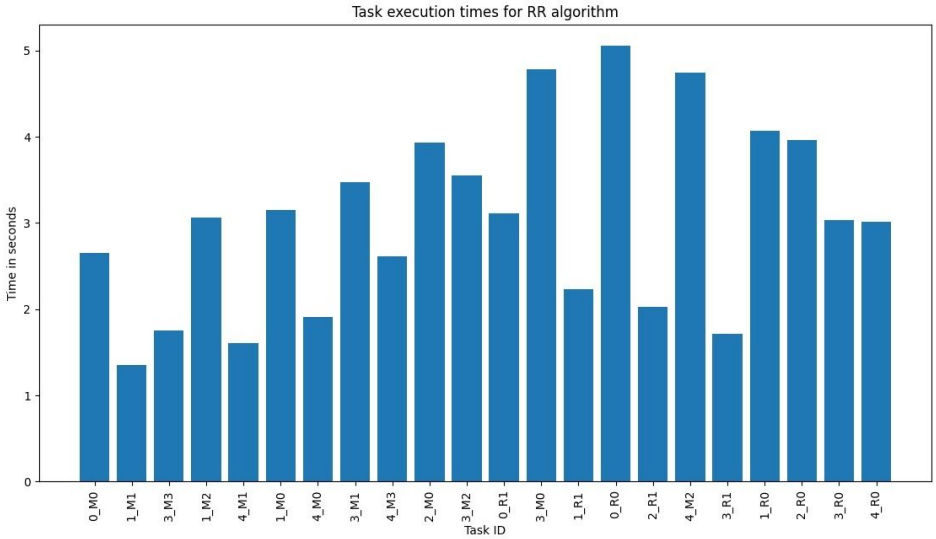
The Results obtained are basically the results obtained by plotting the graphs in task 2 of the project -

Time taken to complete tasks vs Task IDs for each of the Scheduling Algorithms - (Can specify for a certain number of jobs in the requests.py/requests_eval.py file)

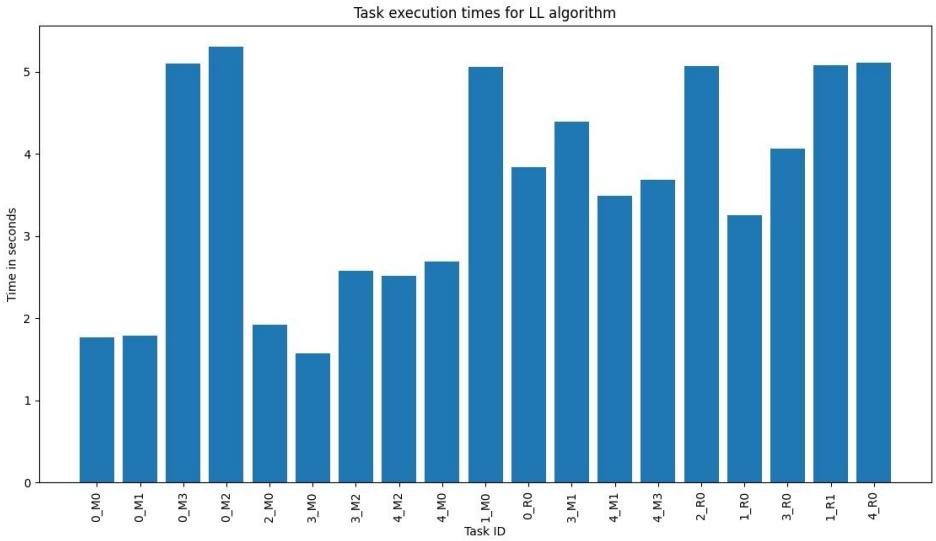
Random Selection -



Round Robin -



Least Loaded -

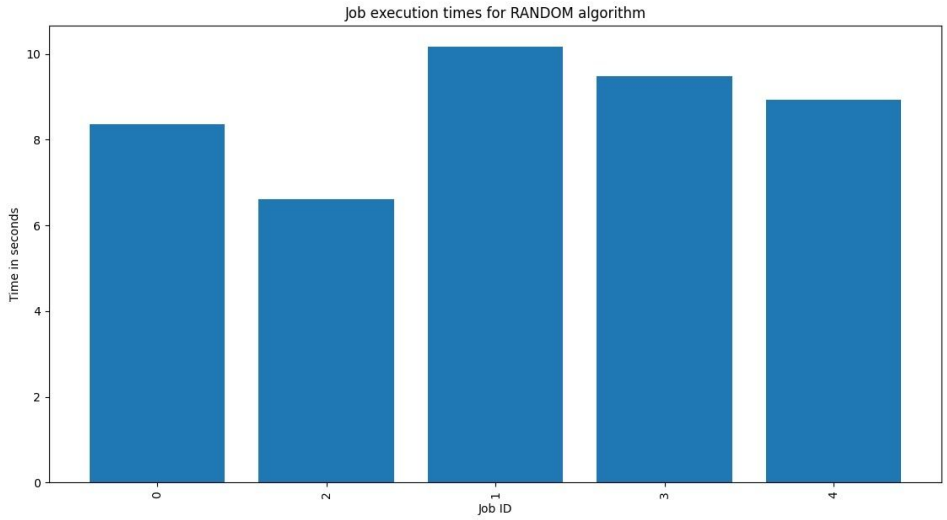


Mean and Median execution times recorded -

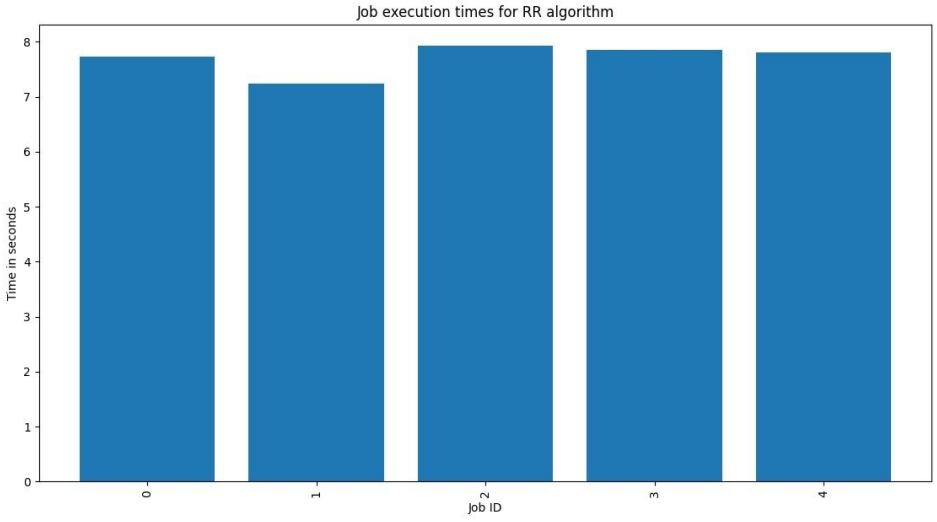
```
Metrics for task execution time using RANDOM algorithm
        Mean: 3.602530523809524 s
        Median: 3.653528 s
Metrics for task execution time using RR algorithm
        Mean: 3.035933909090909 s
        Median: 3.0490845 s
Metrics for task execution time using LL algorithm
        Mean: 3.593647 s
        Median: 3.680556 s

Metrics for job execution time using RANDOM algorithm
        Mean: 8.711372 s
```

Time taken to complete jobs vs Job IDs for each of the Scheduling Algorithms - (Can specify for a certain number of jobs in the requests.py file)

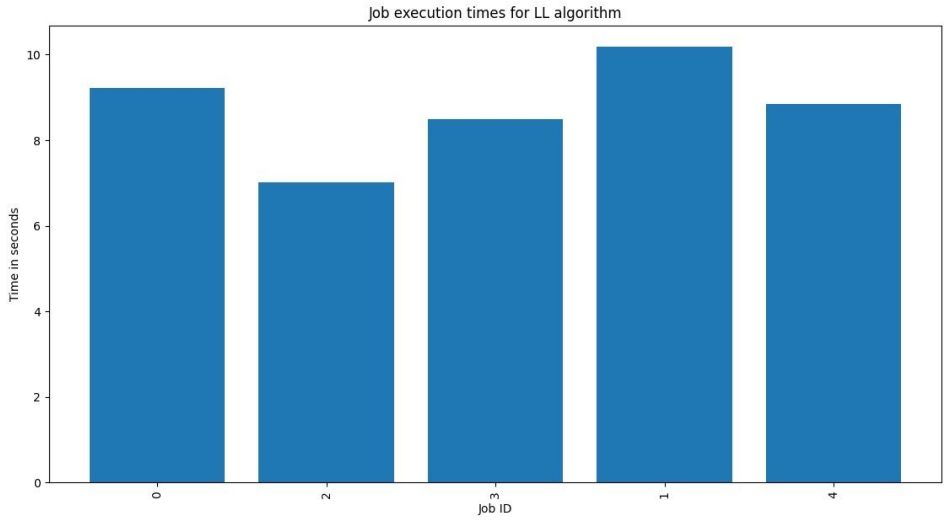
Random Selection -



Round Robin -



Least Loaded -

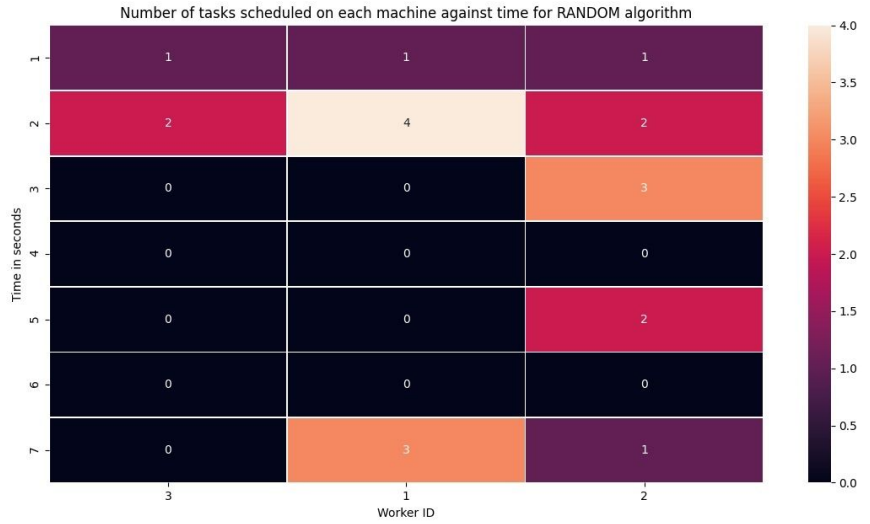


Mean and Median execution times recorded -

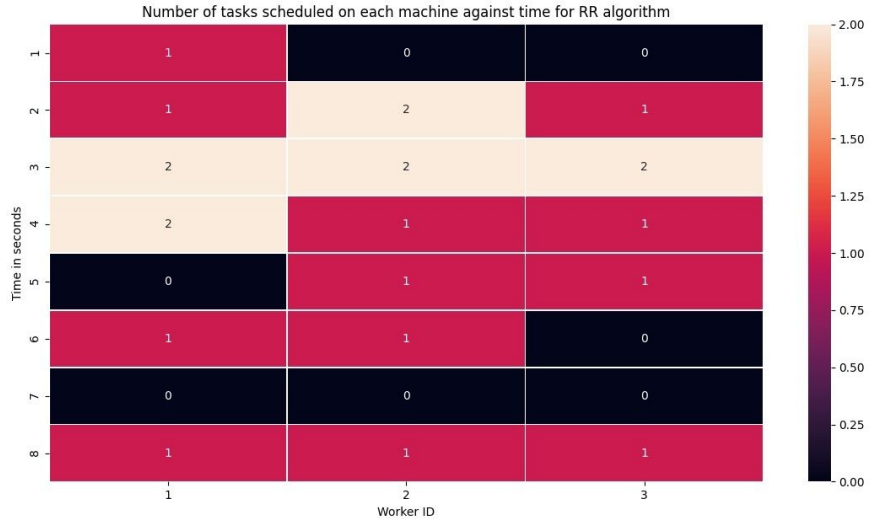
```
Metrics for job execution time using RANDOM algorithm
        Mean: 8.711372 s
        Median: 8.93615 s
Metrics for job execution time using RR algorithm
        Mean: 7.7148434 s
        Median: 7.814036 s
Metrics for job execution time using LL algorithm
        Mean: 8.7477778 s
        Median: 8.844909 s
```

Heatmap of number of tasks scheduled against tasks for each scheduling algorithm -

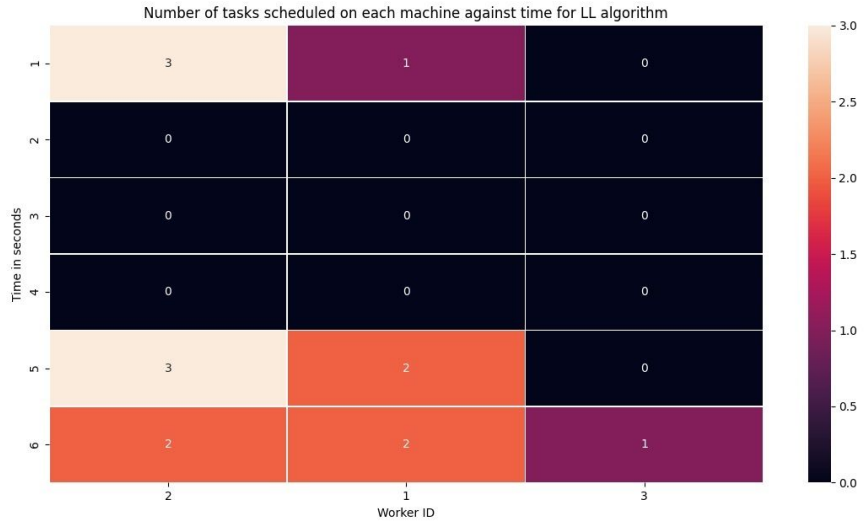
Random Selection -



Round Robin



Least Loaded



Problems

Since we had initially started off with the FPL project we would like to mention that the streaming and the usage of RDDs in PySpark and their conversion were a problem we spent a lot of time trying to solve but could not and hence moved onto the YACS scheduler project.

While we started working on the YACS, we found that the scheduler stops assigning tasks to the worker once all the slots were being utilized without checking for which ones were free.

We found a solution to this problem by having a separate thread on which the yeetacs centralized scheduler was run to read all the tasks from the queue and schedule them in order.

There was also initially the problem of correctly interpreting the log files after the master-worker exchange which proved to be an issue while plotting the graphs for Task-2 but this was quickly dealt with by the usage of 'dictionaries' instead of lists to store the tasks and job execution times.

PS - We had also faced various compile-time issues as in natural with a project and also run time issues with dying threads and unresponsive workers.

The time we found to work on the project satisfactorily was also insufficient.

Conclusion

Through this project we learnt a lot about the workings of a master-worker architecture, how it scales and how to make the implementation possible for workers located on a cluster of machines. Simulation of these tasks using concepts of socket programming,threads etc. gave us a lot of insight on the core subjects in computer science.

In terms of soft skills, we also learnt a lot about teamwork and time management.

EVALUATIONS:

SNo	Name	SRN	Contribution (Individual)
1	Tejas Srinivasan	PES1201800110	
2	Arjun Chengappa	PES1201800119	
3	Lavitra Kshitij Madan	PES1201800137	
4	Priyanka Gopi	PES1201801797	

(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.	