CS322:Big Data

Final Class Project Report

Project (FPL Analytics / YACS coding): _YACS_ Date: 1/12/2020

SNo	Name	SRN	Class/Section
1	Sunit Koodli	PES1201800060	CSE J
2	Nikhil Kashyap	PES1201800239	CSE H
3	Subhav Vardhan	PES1201801182	CSE J
4	Naveeth P	PES1201801978	ECE A

Introduction

The topic we have chosen for the final project is YACS, where we focus on designing a centralized scheduling framework. A single machine can’t run big data workloads as they are too huge, therefore, these are run on clusters of interconnected machines. Our centralized scheduling framework manages and allocates resources of this cluster to different jobs (made of multiple tasks) in the workload. All jobs have 2 stages, the first being the map tasks and the second being reduce tasks. The reduce tasks in a job can be executed only after the execution of all map tasks in the job.

The framework consists of 1 Master process, which runs on a dedicated machine and manages the resources of the remaining machines in the cluster, and 3 Worker processes, where each of them process and execute the tasks, and then inform the Master when a task execution is completed. All communications between the Master and Workers are via sockets. Each of the Worker machines are configured with a fixed number of slots. When a task finishes executing, the information is passed to the scheduling framework and accordingly the resources are freed. These freed resources are then assigned to other tasks.

The machine that is assigned for a task is decided by the 3 scheduling algorithms, namely: random, round-robin and least-loaded.

Related work

1. "Computer Networking - A Top Down Approach" by James F. Kurose and James F. Kurose.

We used this text book as a reference to understand the communication between machines, TCP connections in servers and also to understand the socket programming and port communication which we had to implement as a part of this project.

2. "Operating Systems Concepts" by Peter Baer Galvin and Greg Gagne.

This text book was used to improve our understanding of scheduling algorithms, which have been implemented in this project and the concepts of locks and threads, which again have been extensively implemented as a part of this project.

3. "https://www.javatpoint.com/os-scheduling-algorithms"

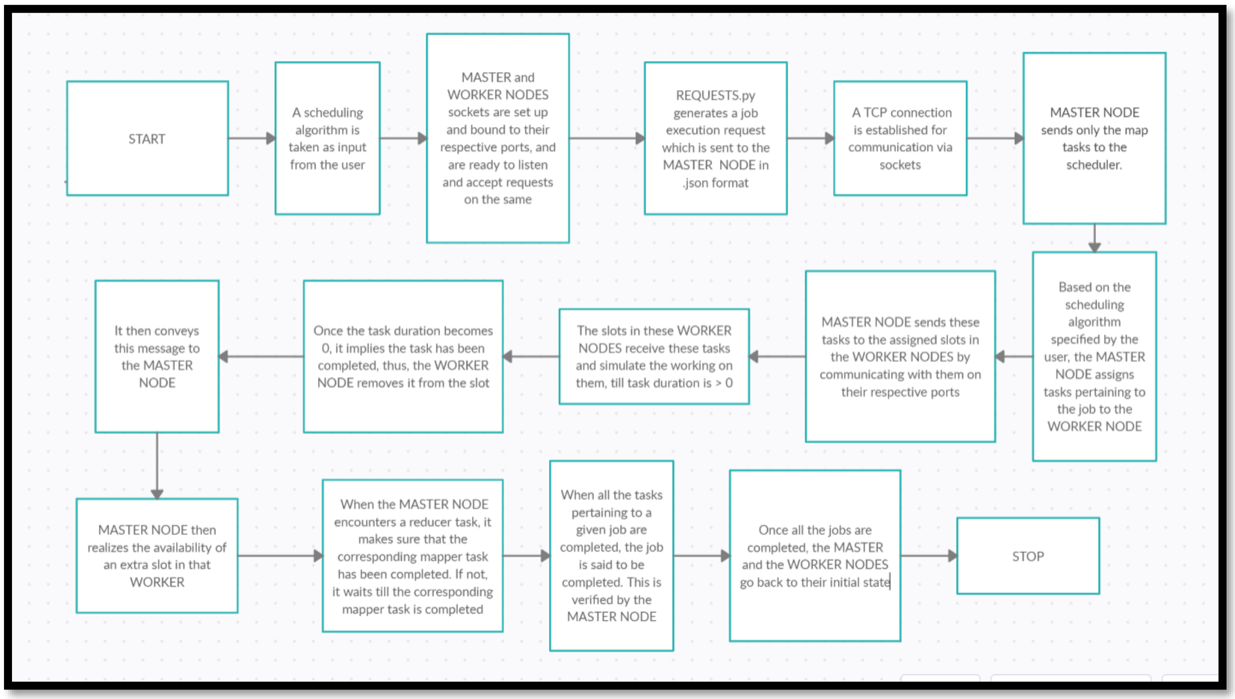
This website gave us a better understanding of scheduling algorithms, with respect to the manner in which we could emulate them using python functions, for them to be implemented as a part of this project.

4. "https://users.cs.duke.edu/~chase/cps196/slides/sockets.pdf"

Understanding Port Numbers and communication between the Master and Workers was key for us to implement this project in its entirety, so in order for us to get a good grasp of their functioning this website was a great source of information and really broadened our understanding of these concepts.

Design

BLOCK DIAGRAM briefing the start to end processes involved in the project


START
A scheduling algorithm is taken as input from the user
MASTER and WORKER NODES sockets are set up and bound to their respective ports, and are ready to listen and accept requests on the same
REQUESTS.py generates a job execution request which is sent to the MASTER NODE in .json format
A TCP connection is established for communication via sockets
MASTER NODE sends only the map tasks to the scheduler.
Based on the scheduling algorithm specified by the user, the MASTER NODE assigns tasks pertaining to the job to the WORKER NODE
MASTER NODE sends these tasks to the assigned slots in the WORKER NODES by communicating with them on their respective ports
The slots in these WORKER NODES receive these tasks and simulate the working on them, till task duration is > 0
Once the task duration becomes 0, it implies the task has been completed, thus, the WORKER NODE removes it from the slot
It then conveys this message to the MASTER NODE
MASTER NODE then realizes the availability of an extra slot in that WORKER
When the MASTER NODE encounters a reducer task, it makes sure that the corresponding mapper task has been completed. If not, it waits till the corresponding mapper task is completed
When all the tasks pertaining to a given job are completed, the job is said to be completed. This is verified by the MASTER NODE
Once all the jobs are completed, the MASTER and the WORKER NODES go back to their initial state
STOP

Designing the YACS System was the core part of our project. We'll begin by explaining the various components and algorithms that we implemented as a part of this project followed by a flow diagram, showing the working of the system we implemented.

1. Components and their Working:

a. The Master Node: It is the central core of our design for the system that we have implemented. It has the following functionalities:

- Running a thread Listening for Jobs from the "Requests.py" file on port 5000, to take these Job Requests.
- Allocating the Tasks pertaining to the Jobs to the respective worker nodes, based on a scheduling algorithm present in it, which is specified by the user and sending these tasks to the specified port number.
- Receiving updates from worker nodes on task completion and deallocating these resources once the tasks are completed.
- On encountering reducer tasks as part of the job, check to see if the respective map task has been executed first, and only then allows for execution of the reducer tasks.

b. The Worker Node: These are the actual compute centers of the infrastructure, which run the tasks and give the output. They have the following functionalities:

- Contain slots, which are a pool of resources set aside for task execution, capable of executing tasks one at a time. Different tasks run in other slots simultaneously.
- Running threads to listen for tasks being allocated by the Master node on their individual port numbers.
- Simulating execution of each task based on their execution time and notifying master of task completion through socket bound to their individual port numbers.

c. Config.json: This is a file that contains specifications about Worker Nodes. The specifications are as follows:

- **worker_id:** Unique ID assigned to each worker node.
- **slots:** Number of slots assigned to each worker node.
- **port:** Individual Port number for communication with the Master Node.

d. Requests.py: Creates Job Requests, and sends them to the master for execution. It creates a socket connection with the master node to carry out this process of sending requests.

2. Scheduling Algorithms Implemented:

a. Random Scheduling: It involves the Master picking a worker node at random and assigning it a task if free slots are available. If available, the task is launched, and if not, the process continues till a free slot is found.

Implementation: Using the random.randint() function to generate random worker_id, and iteratively continuing the process using loops.

b. Round Robin Scheduling: Round Robin is a preemptive job scheduling algorithm. The worker nodes are ordered based on their worker_id, and are picked by the master in a round-robin fashion. If the worker node does not have a free slot, the Master moves on to the next worker_id in the ordering. This process continues until a free slot is found.

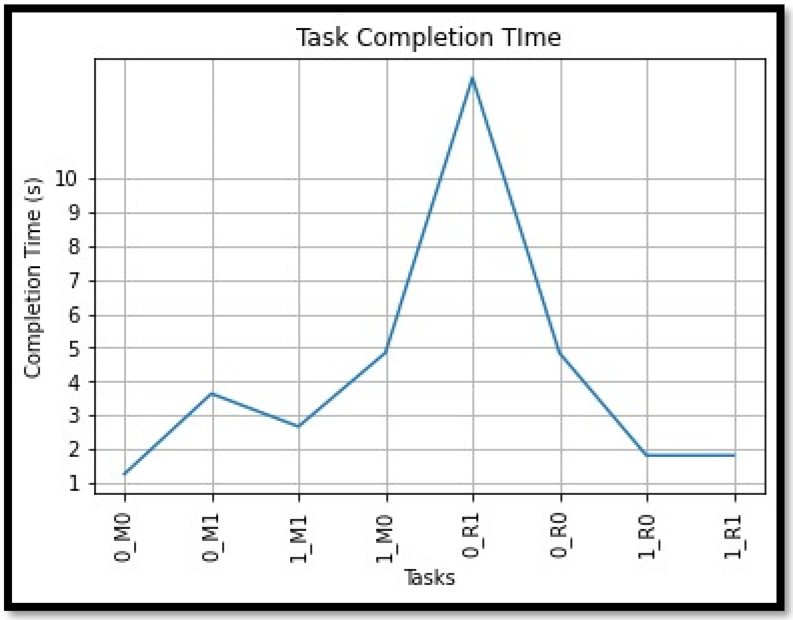
Implementation: Using sort() function to sort workers by worker_id and then iterating through searching for free slots using loops.

c. Least Loaded Scheduling: It is a type of scheduling algorithm wherein the Master node, searches for the worker node with the most available slots, and allocates the task to that worker node. In case no free slots are available it waits for a second and repeats the process till a free slot is found.

Implementation: Using nested loops, linearly iterating through the worker nodes to find the worker node with the maximum free slots available and assigning tasks based on the logic above. There is a 1 second sleep time accounted for the condition where there are no empty slots.

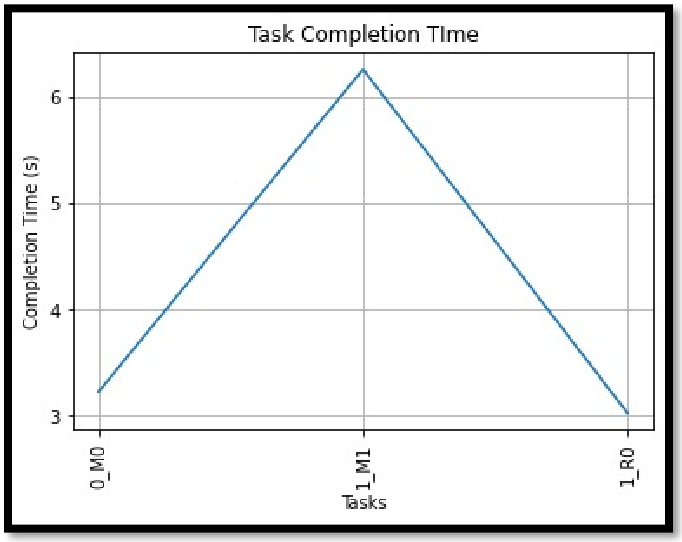
Results

1) For the Round Robin Scheduling Algorithm, we plotted a graph for the tasks completed and their respected completion times.

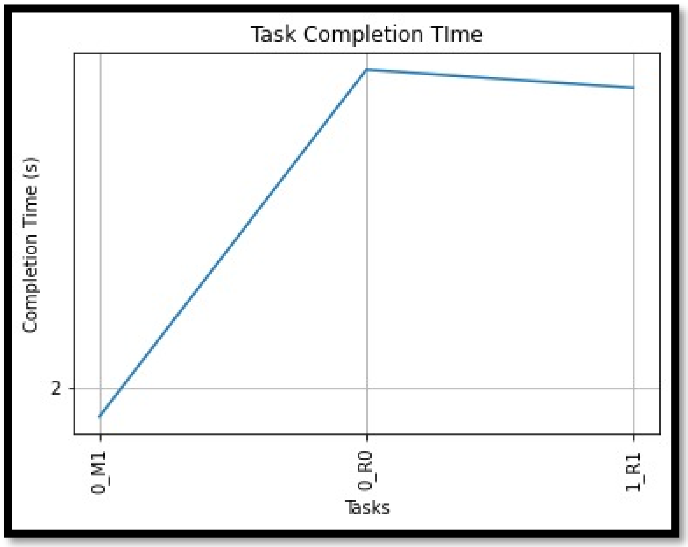


For each worker, we plotted a graph of the tasks assigned to the worker against the time of completion for each task.

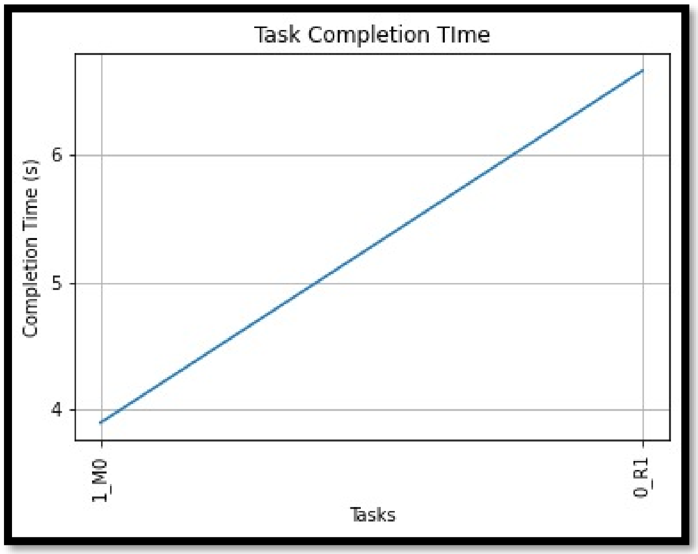
WORKER 1:



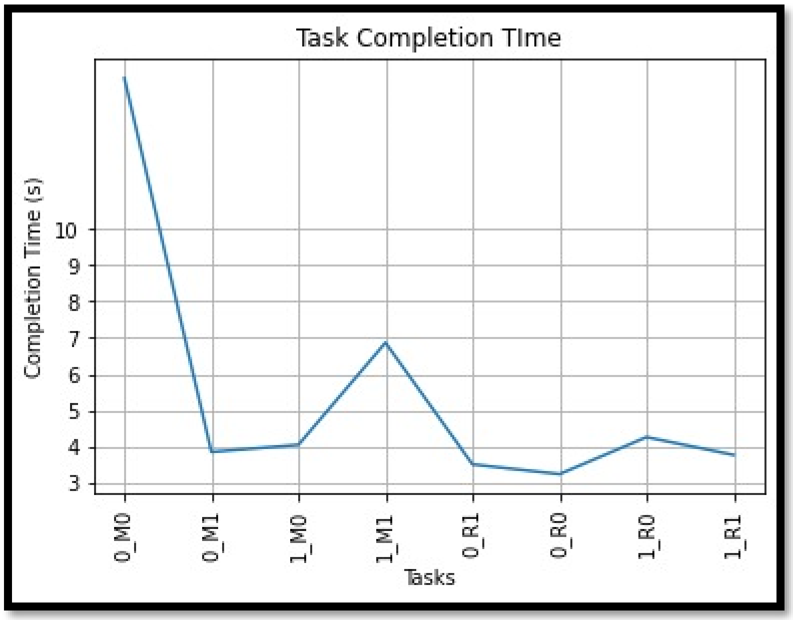
WORKER 2:



WORKER 3:

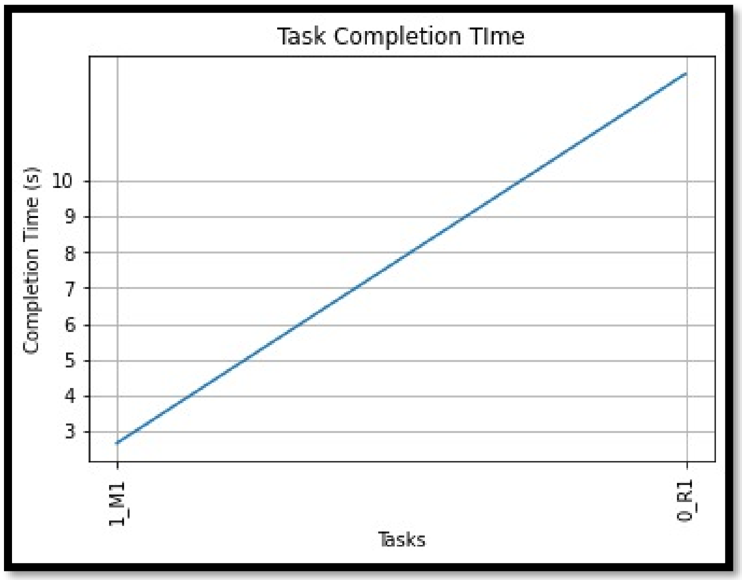


2) For the Least Loaded Scheduling Algorithm, we plotted a graph for the tasks completed and their respected completion times.

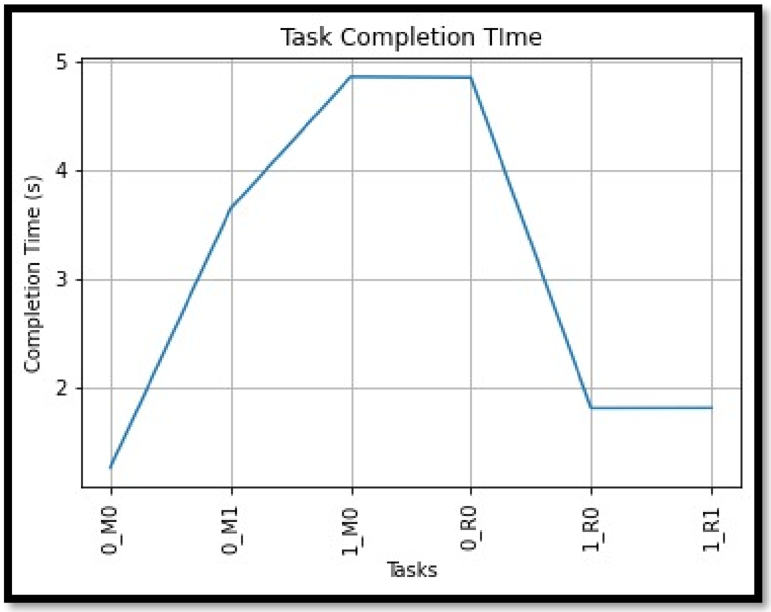


For each worker, we plotted a graph of the tasks assigned to the worker against the time of completion for each task.

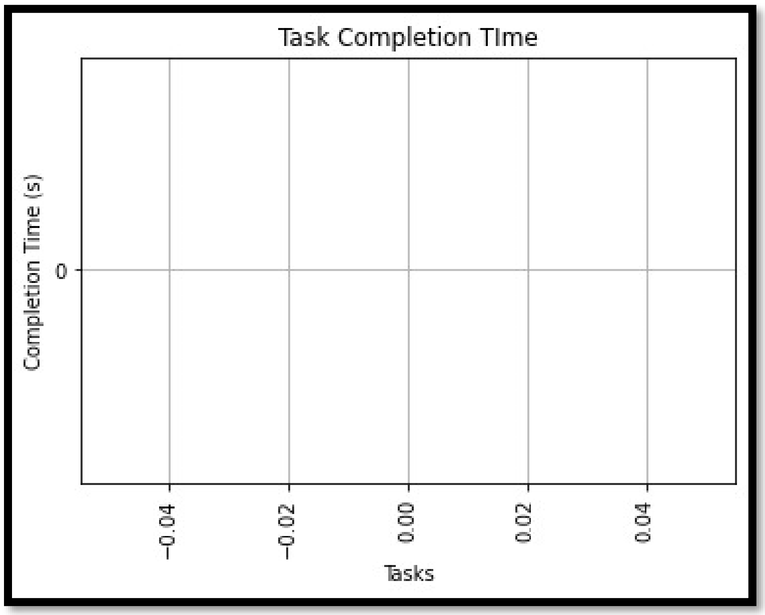
WORKER 1:



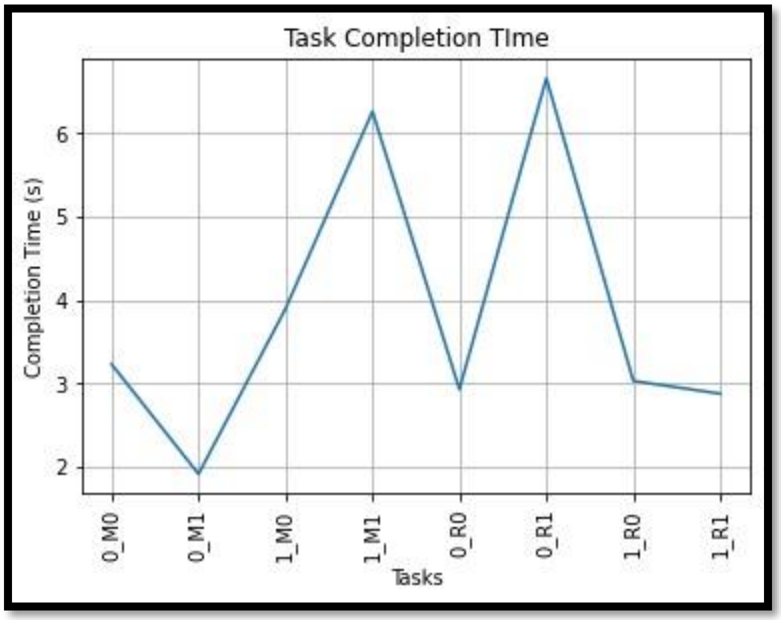
WORKER 2:



WORKER 3:

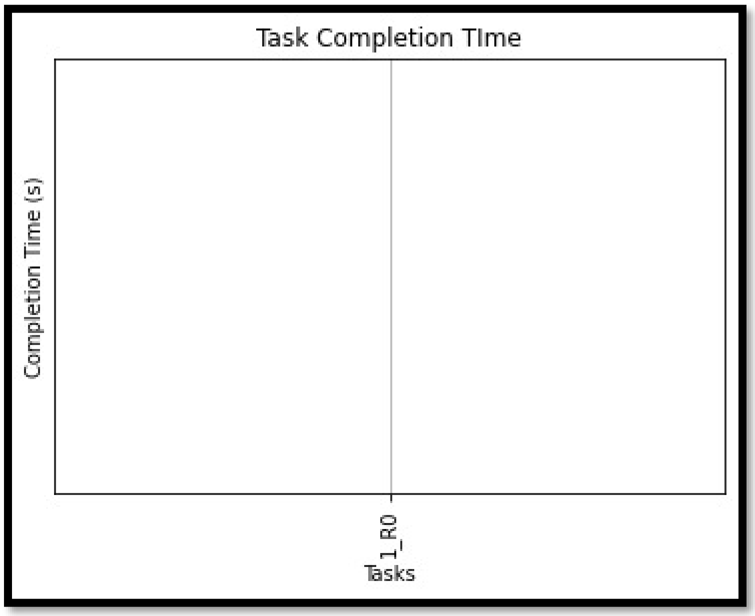


3) For the Random Scheduling Algorithm, we plotted a graph for the tasks completed and their respected completion times.

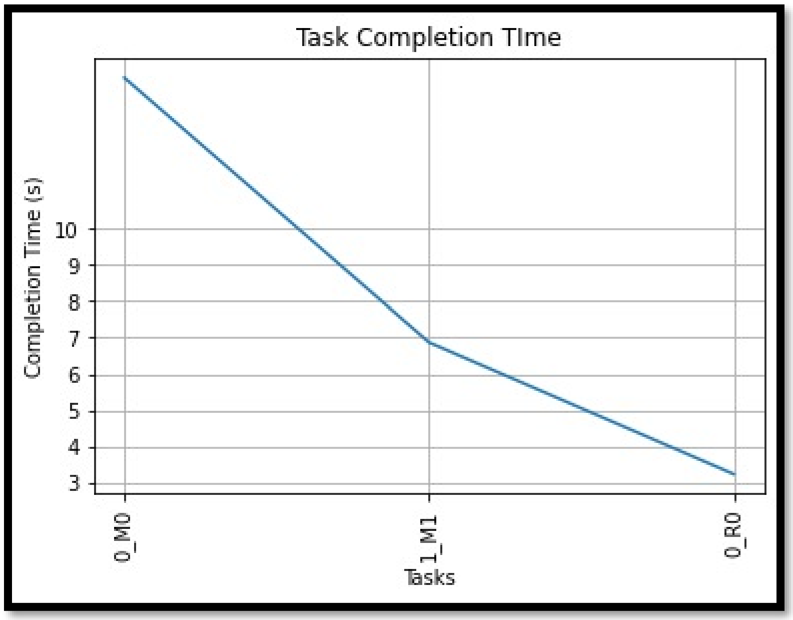


For each worker, we plotted a graph of the tasks assigned to the worker against the time of completion for each task.

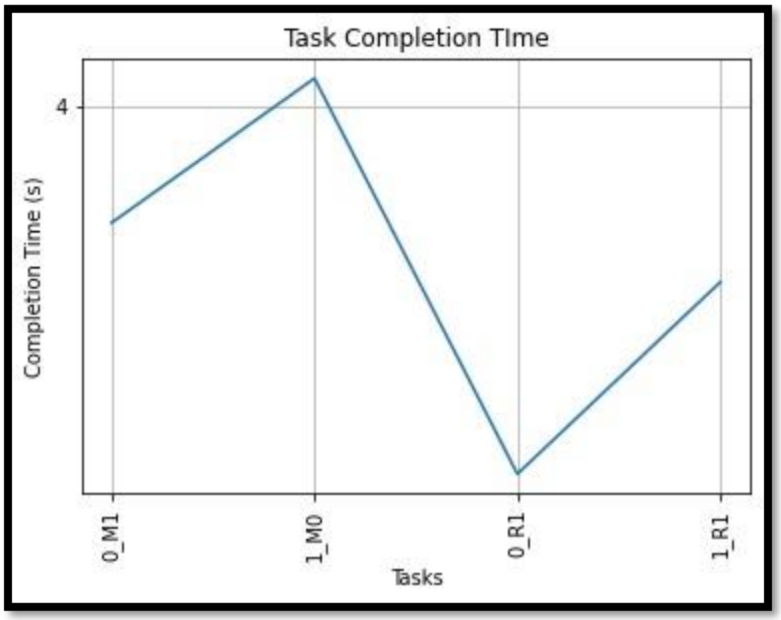
WORKER 1:



WORKER 2:



WORKER 3:



Problems

The following are the problems we encountered during the execution of our program:

Deadlock Issues: It is a state in which each member of a group is waiting for another member, including itself, to take action, such as sending a message or more commonly releasing a lock.

This was solved by using Locks and the acquire() and release() function, by which the resources assigned to the thread were not allowed to be accessed by any other threads.

ConnectionRefusedError: While accepting and sending requests from one node to the other, multiple workers used the same port, and also there was a mismatch of ports which led to this error.

This was resolved by rectifying and ensuring all port numbers were correctly matched.

Conclusion

Implementing the centralized scheduler as a part of this Big Data Final Project has been a great learning experience for us. We learnt in-depth about the various scheduling algorithms, namely: round-robin, random and least loaded and their timing efficiencies. We also learnt about inter-communication between nodes, via the mechanisms of sockets, ports, threads etc. It was a great experience designing the framework of a centralized scheduling algorithm which helps us understand how big data workflows are handled on a large scale.

EVALUATIONS:

SNo	Name	SRN	Contribution (Individual)
1	Sunit Koodli	PES1201800060	Round robin scheduling, assisted with master.py, report writing, block diagram
2	Nikhil Kashyap	PES1201800239	Least loaded scheduling, assisted with master.py, log analysis
3	Subhav Vardhan	PES1201801182	Random scheduling, assisted with master.py, report writing, block diagram
4	Naveeth P	PES1201801978	Worker.py

(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.	