CS322:Big Data

Final Class Project Report

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

SNo	Name	SRN	Class/Section
1	Tarun Gupta J	PES1201800073	5 J
2	Vignesh K Kumar	PES1201800085	5 J
3	Arun Srinivasan P	PES1201800383	5 I
4	Chandratop Chakraborty	PES1201800062	5 G

Introduction

YACS(Yet Another Centralized Schedule) the goal of this project is to construct a framework which follows the master-slave design. The framework consists of 1 master and 3 workers. The master process takes care of the scheduling decisions of the workers and monitors the communication to keep the entirety in sync .

Related work

1. YARN Structure: Apache Hadoop YARN: Yet another resource negotiator
2. Socket Programming: Socket Programming in Python (Guide) – Real Python
3. Scheduling Algorithms:
    a. Round Robin: Round Robin
    b. Least Loaded: Least Loaded

Design

We have used the following data structures to assist us in processing the jobs in the master:

- workers_state: A nested dictionary indexed by the worker_id which has the information about the worker such as the port number, and the number of occupied slots as well as the total number of slots.
- map_jobs_tbd: A list of dictionaries which contain all the map jobs to be done. It contains the details about the duration of the task and the id.
- reduce_jobs_tbd: A list of dictionaries which contain all the reduce jobs to be done. It contains the details about the duration of the task and the id. It is populated only when the corresponding map jobs are completed
- job_state: A dictionary indexed by job_id which contains the information about each job such as the map and reduce tasks as well as the completed tasks. It also contains the arrival time and completion time.

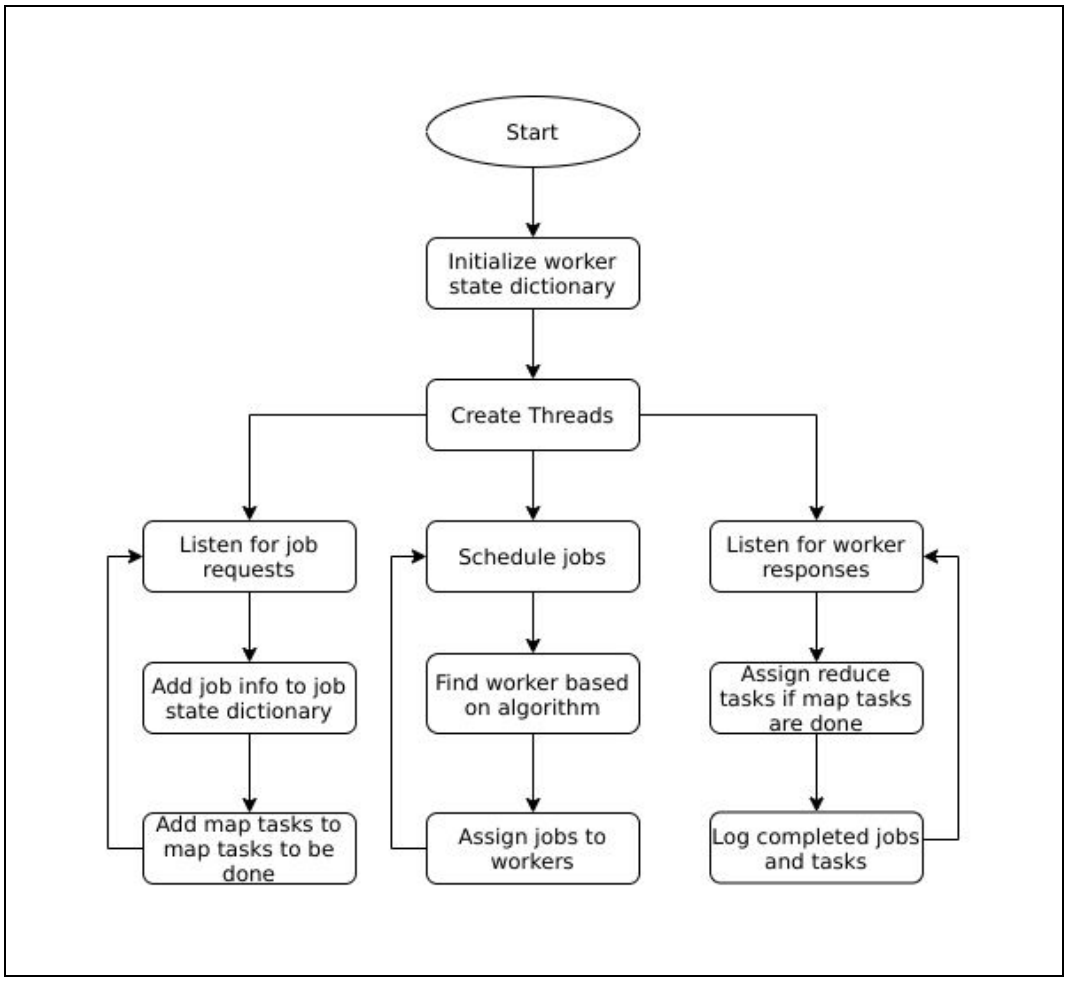
Each of these variables are shared across the master threads using a lock for each variable to ensure that there is no multiple access to maintain consistency.

The worker shares one variable across both its threads that is:

- task_tbd: A list of dictionaries which contain all the tasks that the worker has to execute.

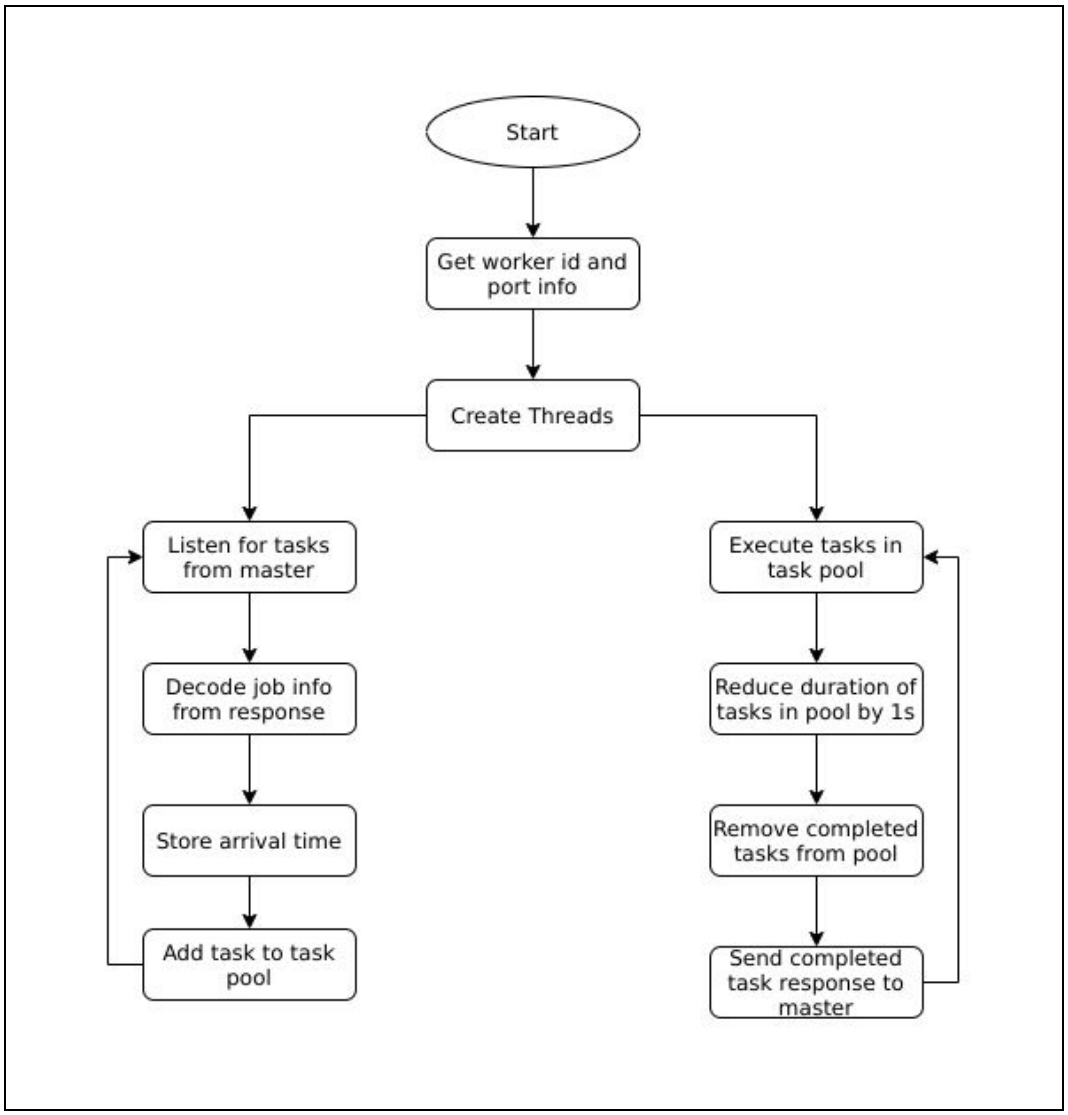
This variable is also shared across both worker threads using a lock to maintain consistency.

Master Workflow



As shown in the flowchart the master consists of three threads that share the data structures mentioned above. It communicates with the workers by means of socket programming which sends the required information about the task to the worker using a json object. It then receives the information back from the worker in the same way and updates the job info and logs it to an external file. The master first assigns all the map tasks to a worker based on the algorithm specified as an argument. Once all the map tasks of a specific job are completed it then adds the reduce tasks of that job to the reduce job pool and assigns workers for the reduce jobs as well.

Worker Workflow

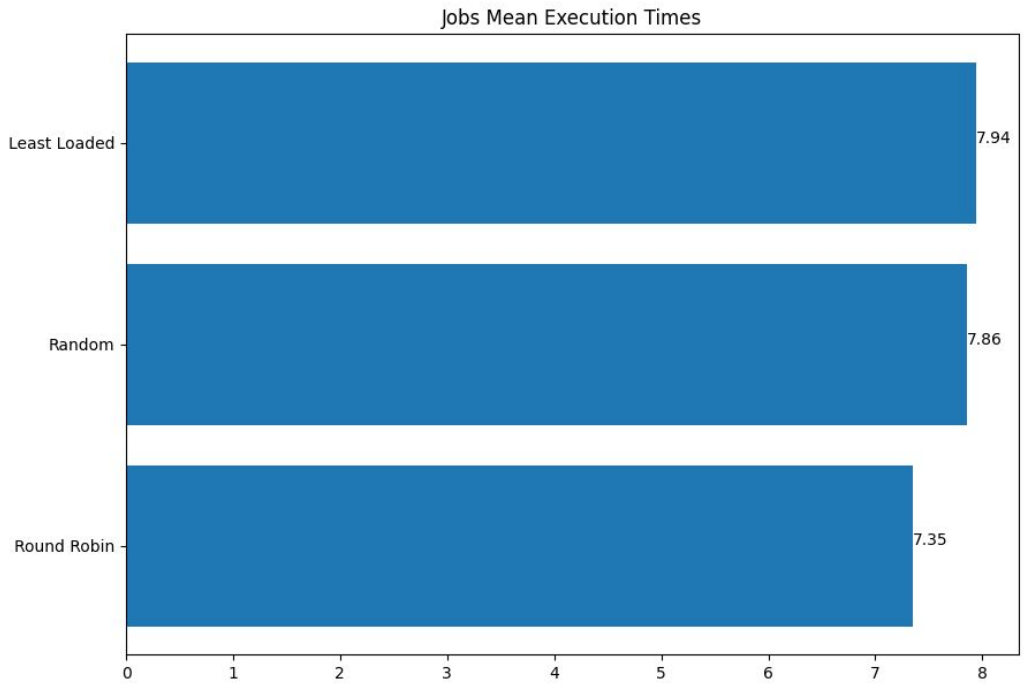


The worker consists of two threads, one which reads the incoming data from the master and adds tasks to the task pool and another which executes the tasks in the pool. When a task is completed it sends a response to the master with the necessary information such as execution time.

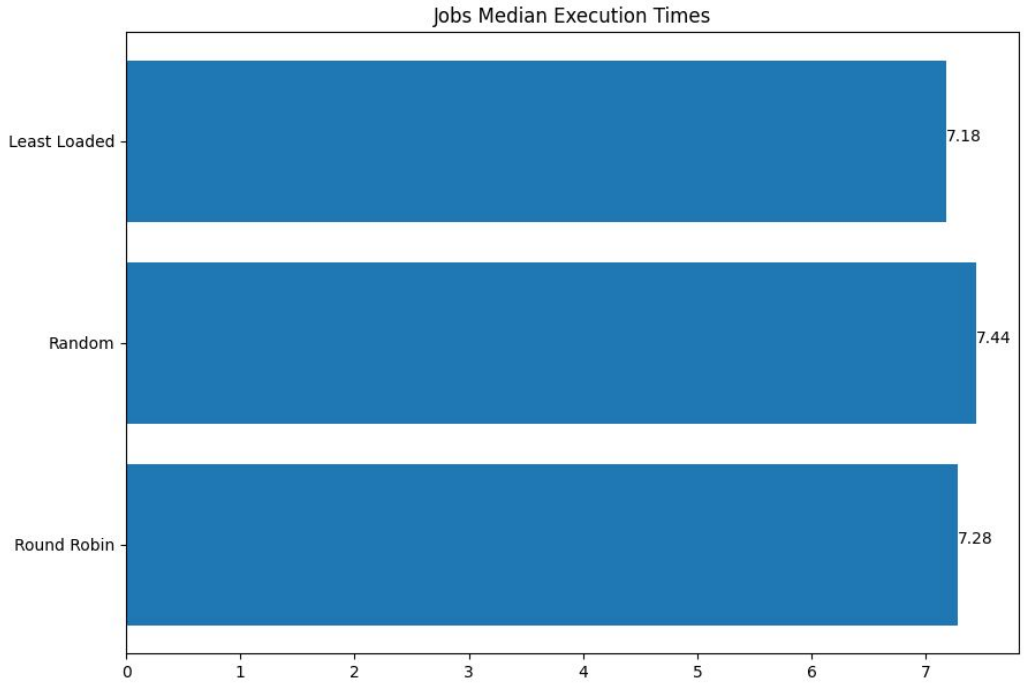
Results

We have plotted three visualizations which are given below:

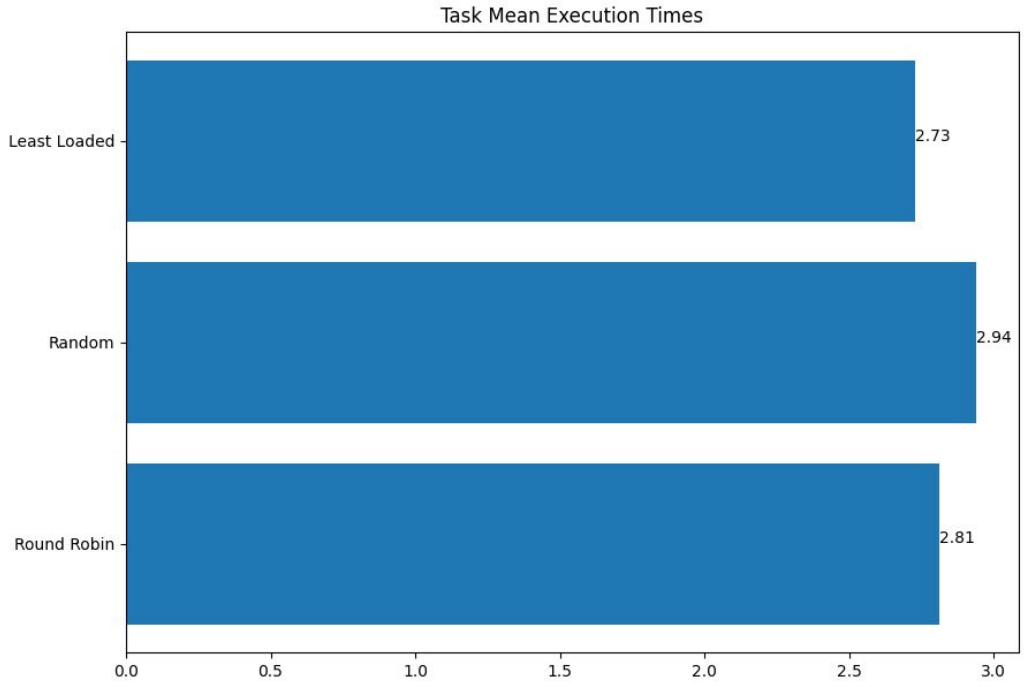
1) Mean and Median of the execution time with respect to the tasks
2) Mean and Median of the execution time with respect to the jobs
3) Heatmap which represents the distribution of tasks at each worker at a given period of time.



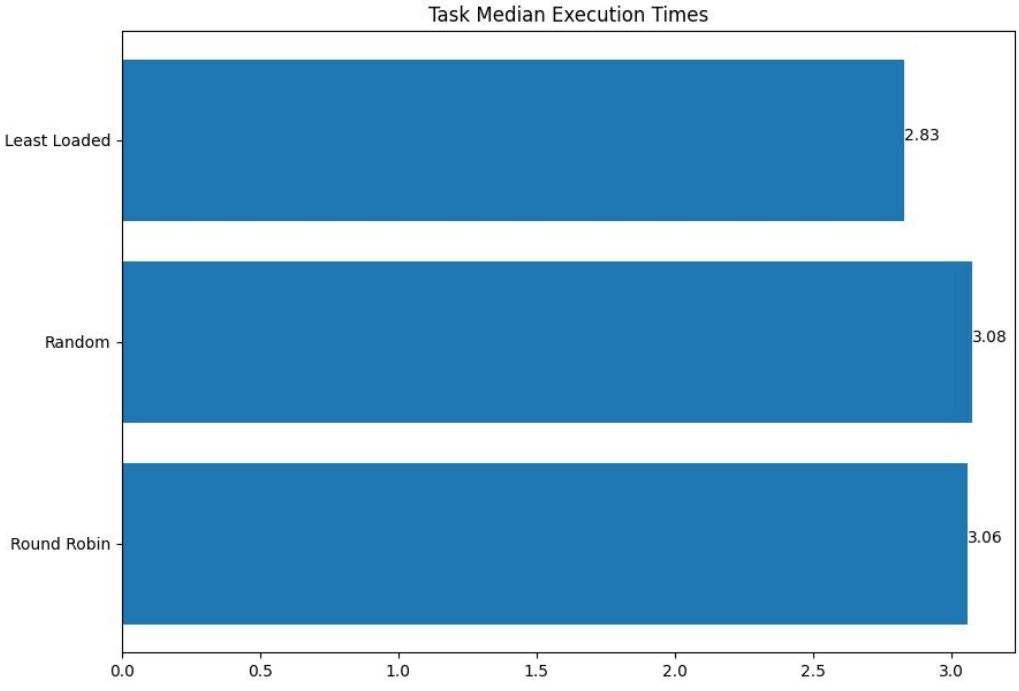
The above visualization gives us the distribution of mean time of execution of the jobs for the 3 algorithms. The above is for a particular execution and doesn't portray any relationship.



The above visualization gives us the distribution of median time of execution of the jobs for the 3 algorithms. Similarly the above image is for a particular execution and doesn't portray any relationship.

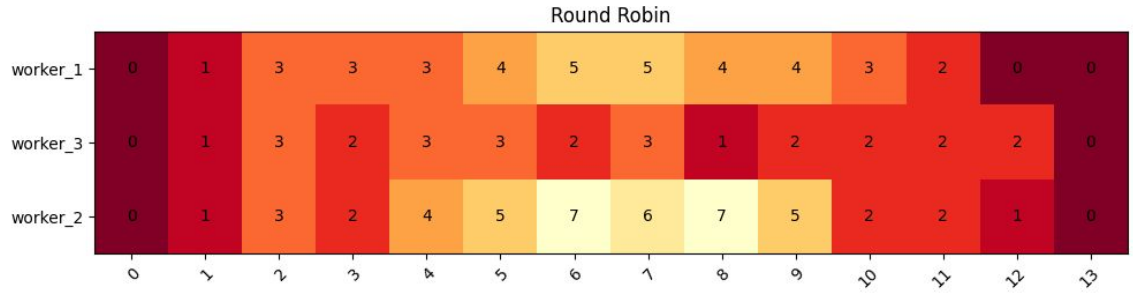


The above visualization gives us the distribution of mean time of execution of the tasks for the 3 algorithms. We can see that since a job consists of multiple tasks the execution time of a job is considerably higher than a single task execution time. Similarly the above image is for a particular execution and doesn't portray any relationship.

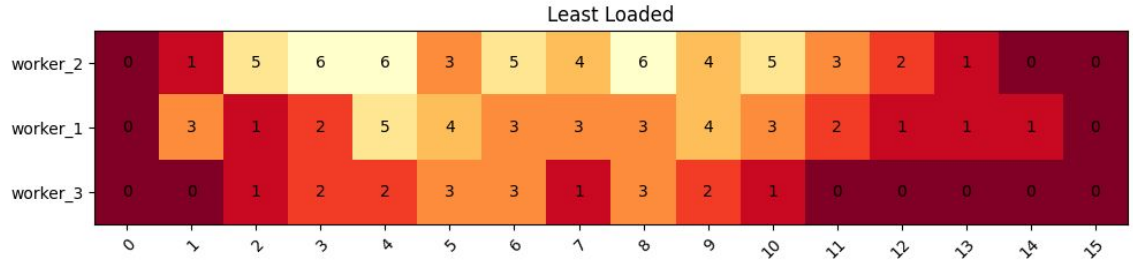


The above visualization gives us the distribution of median time of execution of the tasks for the 3 algorithms. We can see that since a job consists of multiple tasks the execution time of a job is considerably higher than a single task execution time. Similarly the above image is for a particular execution and doesn't portray any relationship.

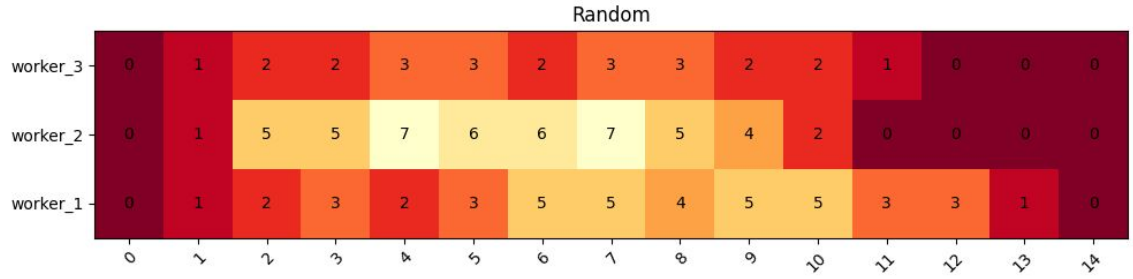
Heatmaps



The above heatmap gives us a visual representation of the distribution of tasks across the given workers for the round robin algorithm. We can infer from the heatmap that the number of tasks seem to remain consistent across all 3 workers, considering the variance in number of slots across workers and the difference in execution time for each task.



The above heatmap gives us a visual representation of the distribution of tasks across the given workers for the least loaded algorithm. We can infer from the heatmap that the worker with maximum slots seems to serve the maximum number of tasks. (In this case the worker 2 has a maximum number of slots).



The above heatmap gives us a visual representation of the distribution of tasks across the given workers for the random algorithm.

Problems

Problems Faced during the building of the project

1. Deadlock occurrence due to the structure of acquiring and releasing in the while loop.
   Solved by adding sleep() to give time for the locks to be released and acquired by another thread. Otherwise the same thread would acquire the lock without releasing it. Without providing any buffer time for another thread to acquire it.
2. Keeping track of multiple structures to store information about the map and reduce tasks for later analysis and logging.
3. Understanding the time intervals for visually representing the state of each worker through heatmaps.

Conclusion

Main Learnings:

1. Extensive understanding of threading
2. Accessing resources appropriately utilizing locks through methods such as acquire and release.
3. Use of a centralized distributor modeled in a master and slave structure.
4. Understanding the importance Socket Programming for Communication

EVALUATIONS:

The entire project was on video-conferencing platforms with screen share facility with all the members contributing equally to the work because of which no part of the project can be attributed to an individual member of the team.

SNo	Name	SRN	Contribution (Individual)
1	Tarun Gupta J	PES1201800073	Equal
2	Vignesh K Kumar	PES1201800085	Equal
3	Arun Srinivasan P	PES1201800383	Equal
4	Chandratop Chakratop	PES1201800062	Equal

(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.	