

UE18CS322 : Big Data

YACS - Yet Another Centralized Scheduler

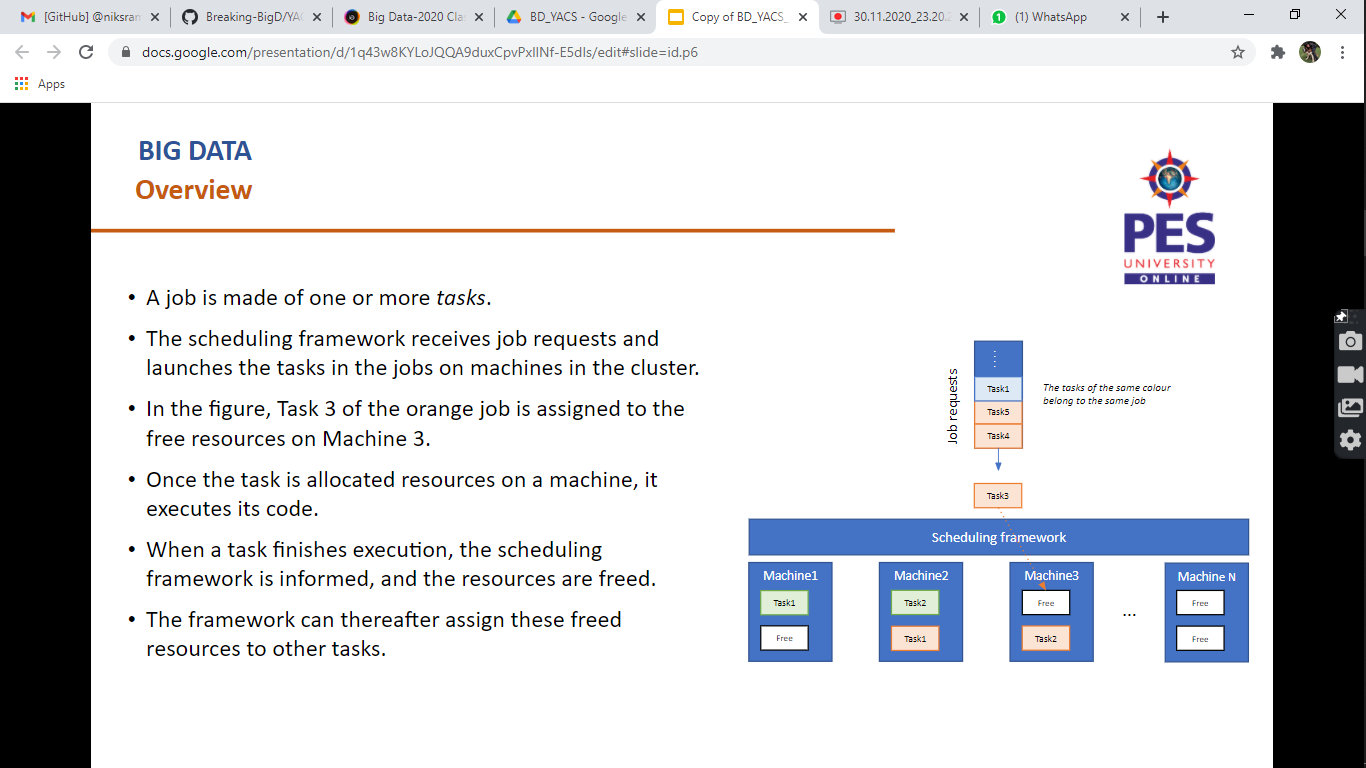
Team id : BD\_1482\_1903\_1957\_1972

Final Class Project Report

**Project: YACS Coding**  **Date:01/12/2020**

|  |  |  |  |
| --- | --- | --- | --- |
| SNo | Name | SRN | Class/Section |
| 1 | S Nikhil Ram | PES1201801972 | 5 -B |
| 2 | Rakshith C | PES1201801903 | 5 - D |
| 3 | K Vikas Gowda | PES1201801957 | 5 - F |
| 4 | Yash Gawankar | PES1201801482 | 5 - J |

## Introduction



* Big Data workloads consist of multiple jobs from different applications, which need to be run on a cluster of interconnected machines known as workers due to its large resource requirements.
* In the Real World, due to the limitation of resources, a scheduling framework is necessary in order to allocate resources to the cluster to various incoming jobs arriving from a wide range of applications.
* An incoming job could contain multiple tasks (like Map and Reduce), and hence resources need to be allocated in compliance with their dependencies (ex. all Reduce tasks depend on the previous Map tasks).
* Thus, in order to coordinate the scheduling of tasks among numerous interconnected clusters, a Centralized Scheduling Framework is needed.
* 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.

## Related work

**Research Papers:-**

* Dazhao Cheng, Xiaobo Zhou, Palden Lama, Jun Wu, Changjun Jiang, Cross-platform resource scheduling for spark and MapReduce on YARN, IEEE Trans. Comput., PP (99) (2017)
* Jyoti V. Gautam, Harshad kumar B. Prajapati, Vipul K. Dabhi, Sanjay Chaudhary, A survey on job scheduling algorithms in big data processing, IEEE Conf. Pap. (March 2015)
* Thomas C. Bressoud, Qiuyi Tang, “Results of a model for hadoop YARN MapReduce tasks, IEEE Int. Conf. Clust. Comput. (2016)
* Mohd Usama, Mengchen Liu, Min Chen, Job Schedulers for Big Data Processing in Hadoop Environment: Testing real-life schedulers using benchmark programs
* Islam, Muhammed Tawfiqul & Buyya, Rajkumar, Resource Management and Scheduling for Big Data Applications in Cloud Computing Environments (2018).

**Articles:-**

* [Job Scheduling in Hadoop](https://medium.com/@rinu.gour123/job-scheduling-in-hadoop-941ac254fb84) - Medium.com Article
* [Hadoop - Schedulers and Type of Schedulers](https://www.geeksforgeeks.org/hadoop-schedulers-and-types-of-schedulers/) - GeeksforGeeks
* [Schedulers in YARN : from concepts to configurations](https://towardsdatascience.com/schedulers-in-yarn-concepts-to-configurations-5dd7ced6c214#:~:text=There%20are%20three%20types%20of,placing%20them%20in%20a%20queue.) - towardsdatascience.com
* [Hadoop Schedulers Tutorial - Job Scheduling in Hadoop](https://data-flair.training/blogs/hadoop-schedulers/) - data-flair.training

**Books:-**

* [Hadoop: The Definitive Guide, 4th Edition](https://www.oreilly.com/library/view/hadoop-the-definitive/9781491901687/ch04.html) - O’Reilly Publications, Chapter 4: YARN

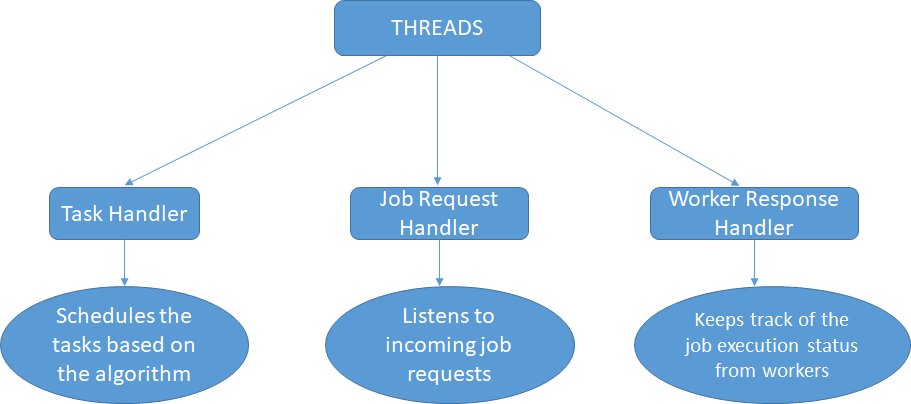
## Design

**Overview of the General Design Strategy:-**

* The Master listens for job requests and dispatches the tasks in the jobs to the workers based on a *scheduling algorithm*.
* The Worker processes listen for Task Launch messages from the Master.
* On receiving a Launch message, the Worker adds the task to the execution pool of the machine it runs on.
* The execution pool consists of all currently running tasks in the machine.
* When a task completes execution, the Worker process on the machine informs the Master.
* The Master then updates its information about the number of free slots available on the machine.

**Implementation:-**

**Master :-**



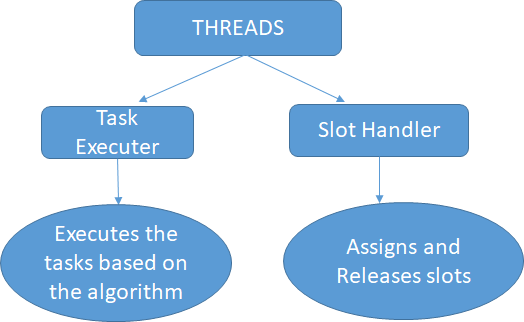
It is capable of performing 3 tasks simultaneously using threads:

1. Listening to incoming Map and Reduce requests from the requests.py
2. Scheduling the tasks based on specified algorithms
3. Keeping track of execution status of the tasks and availability of resources on the workers

The threads are utilized by 3 classes namely:-

1. task\_handler :
   1. Connects to the worker based on specified port no.
   2. Determines the type of algorithm used for scheduling
   3. Schedules the jobs based on the Algorithm
   4. It logs the start of each task in a job
   5. Also keeps track of no of free slots on the workers
2. job\_request\_handler:
   1. Accepts the incoming job requests from requests.py
   2. Reads and stores the JSON object consisting of Worker configurations
   3. Appends the jobs to a job pool (queue)
   4. Also logs the Start of every job
3. worker\_response\_handler:
   1. Receives the task completion status from the workers
   2. Logs the completion of each task
   3. As and when the jobs are completed, it pops them from the job pool
   4. Ensures that all Reduce tasks are implemented only after the Map tasks and thus maintaining the dependencies

**Worker:-**



It is capable of performing 2 tasks simultaneously using threads:

1. Accepting incoming Map and Reduce tasks from the master
2. Assigning available slots to each tasks

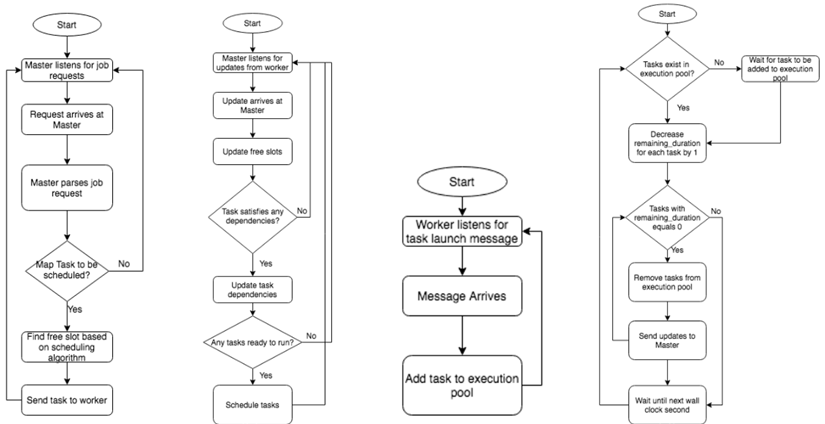
The threads are utilized by 2 classes namely:-

1. task\_executer :
   1. Connects to master on port 5001
   2. Receives Map and Reduce tasks scheduled by master
   3. Checks for availability of slots
   4. Decreases the duration till the task is executed
   5. Acquires and releases lock while decreasing duration
2. slot\_handler :
   1. Assigns the slots to each task
   2. Keeps track of no of available slots and reports to the master
   3. Acquires and releases lock while assigning the slots

**Scheduling Algorithms implemented:-**

|  |  |  |
| --- | --- | --- |
| Round Robin | Random Scheduler | Least Loaded |
|  |  |  |

**Model and WorkFlow:-**



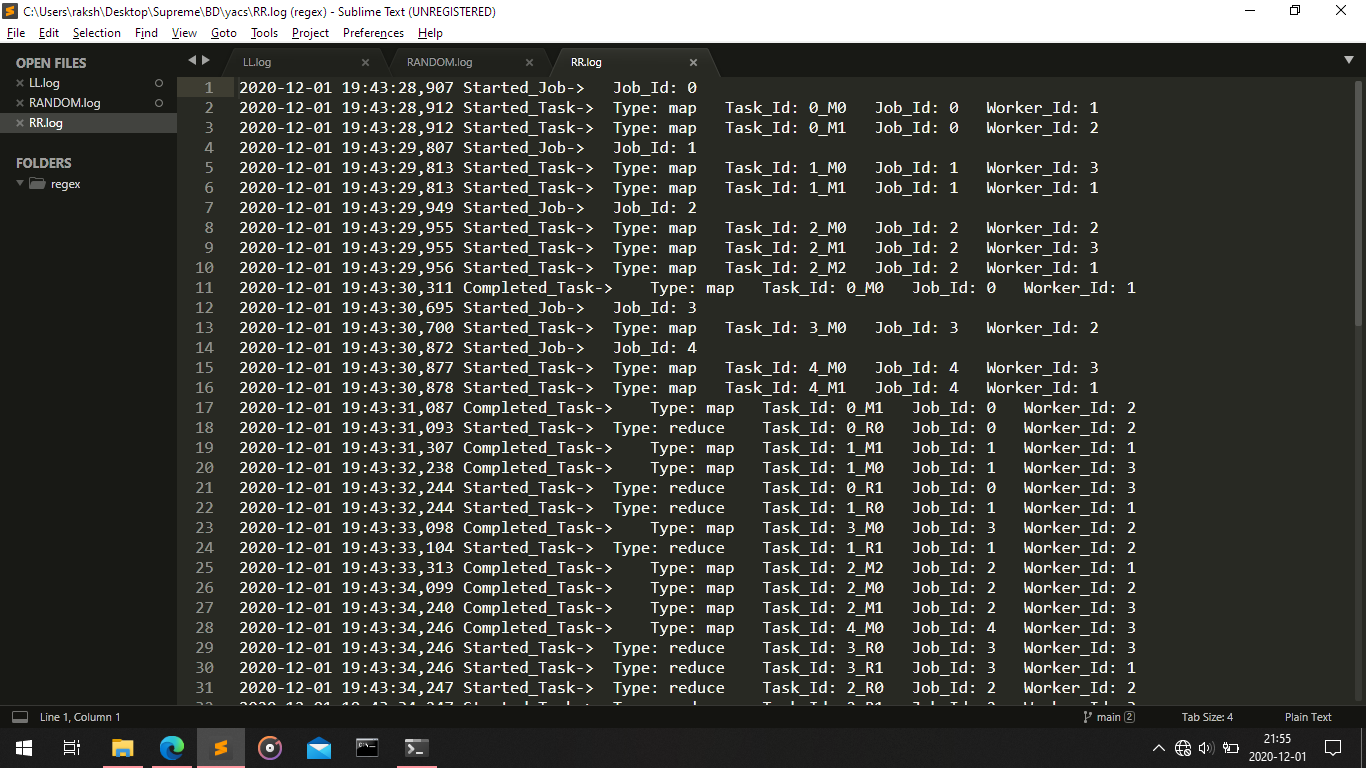
* The scheduler in the master has been introduced as the third thread in the implementation. This thread encomposses the scheduling algorithms

## Execution

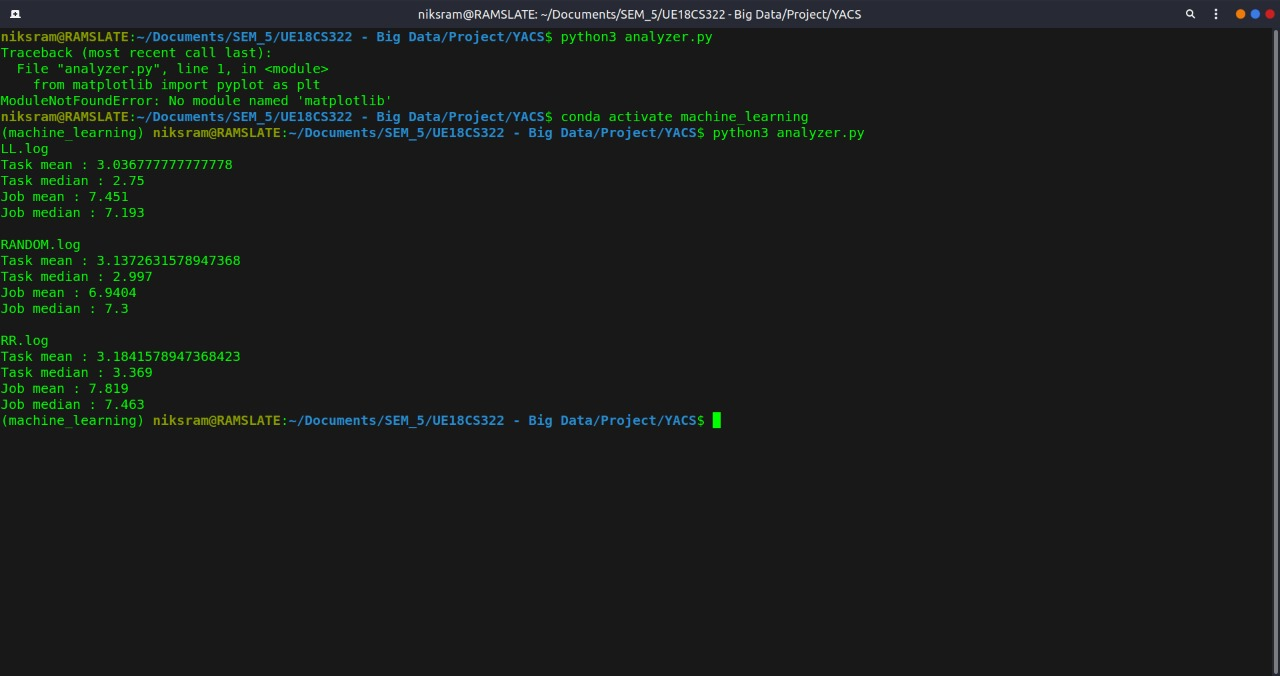
* The **workers** must initially be spawned by running **worker.py** with the appropriate port numbers and the slots.
* The **master.py** must be executed with the appropriate **config.json** whose ports and slots tally with the corresponding workers. The number of workers is not fixed to 3 and can be dynamic.
* Then **request.py** must be run with any amount of requests.
* The **Jobs** shall be executed and the corresponding log file will get created/overwritten based on the **algorithm.**
* Once all log files are created, the **analyzer.py** must be executed which reads all the log files and generates the results.

## Results

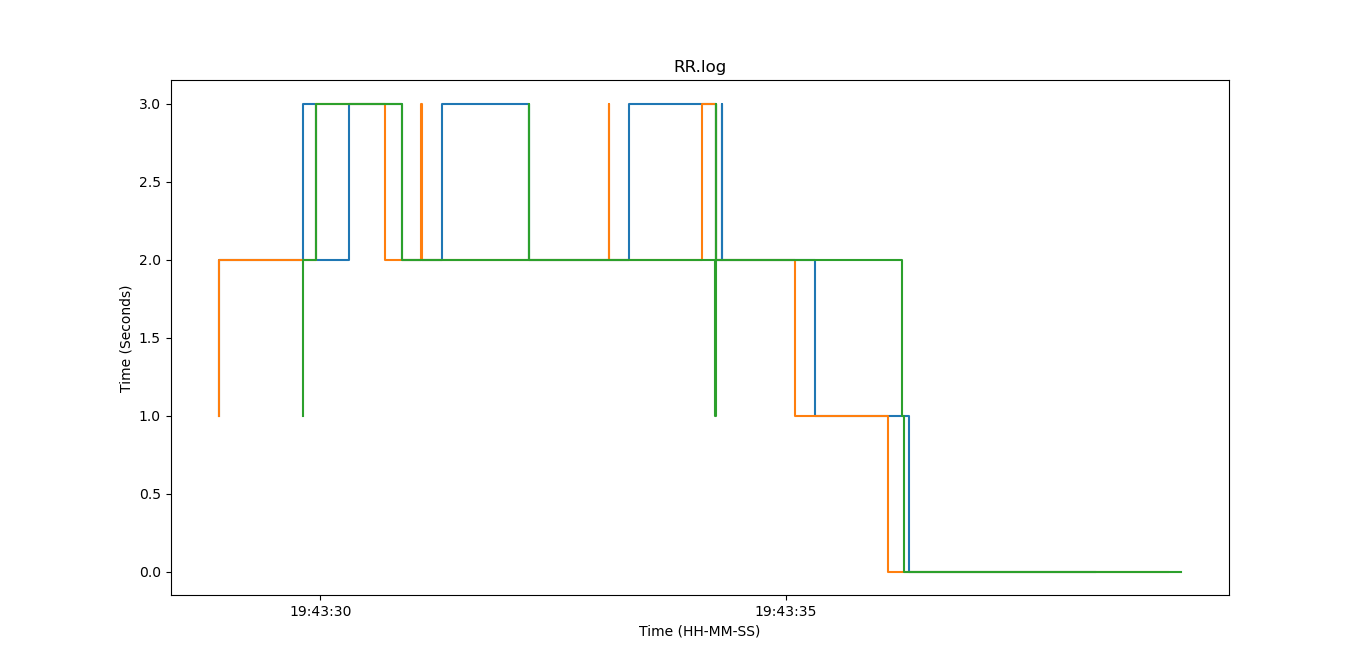
1. **Implementation of Round Robin Scheduler**
   1. Log File



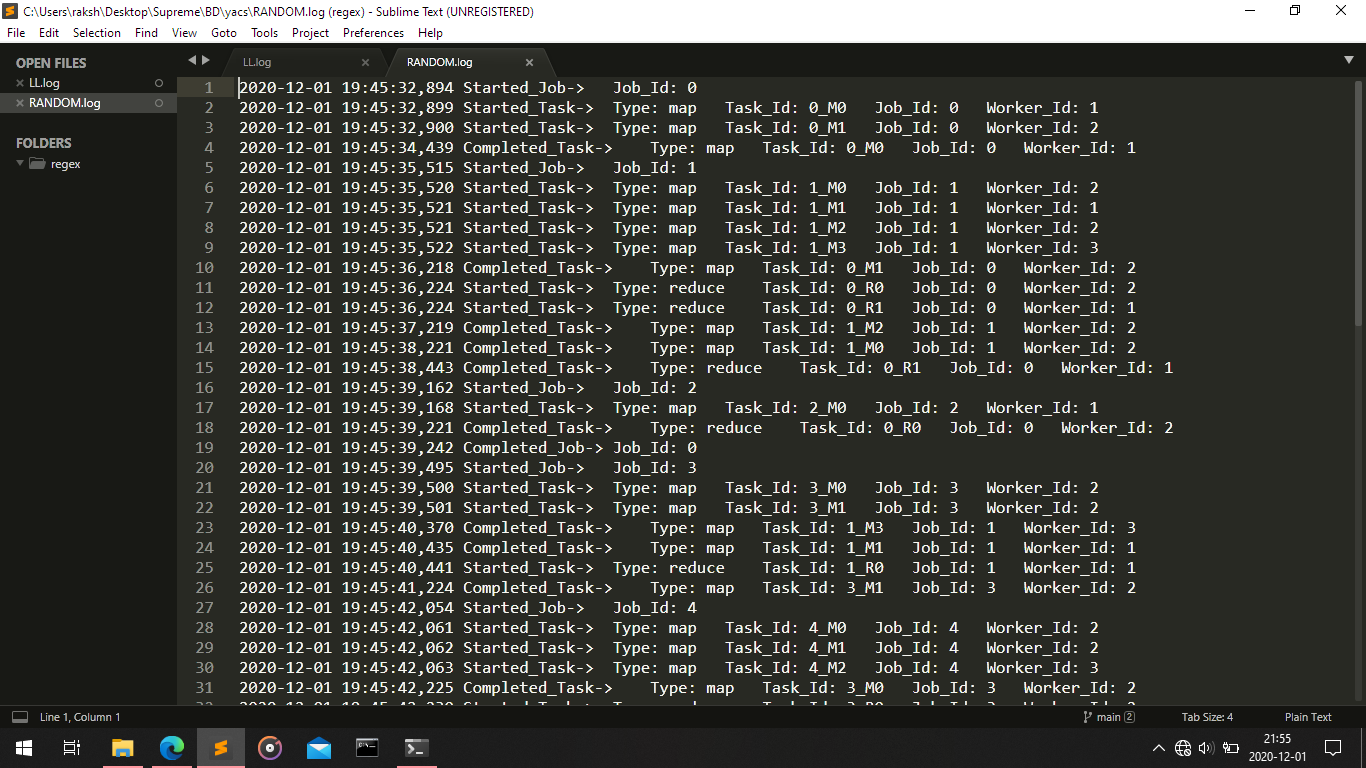
* 1. Mean and Median results :-



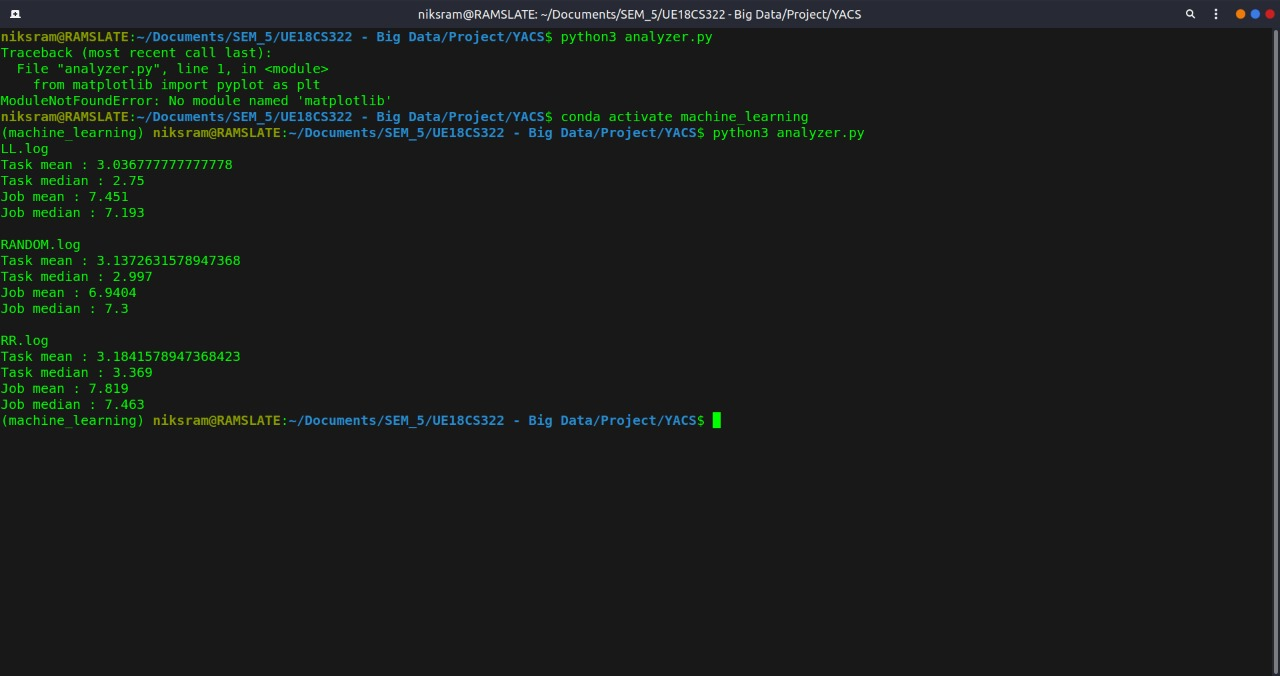
* 1. Plot of Scheduled tasks on Workers :



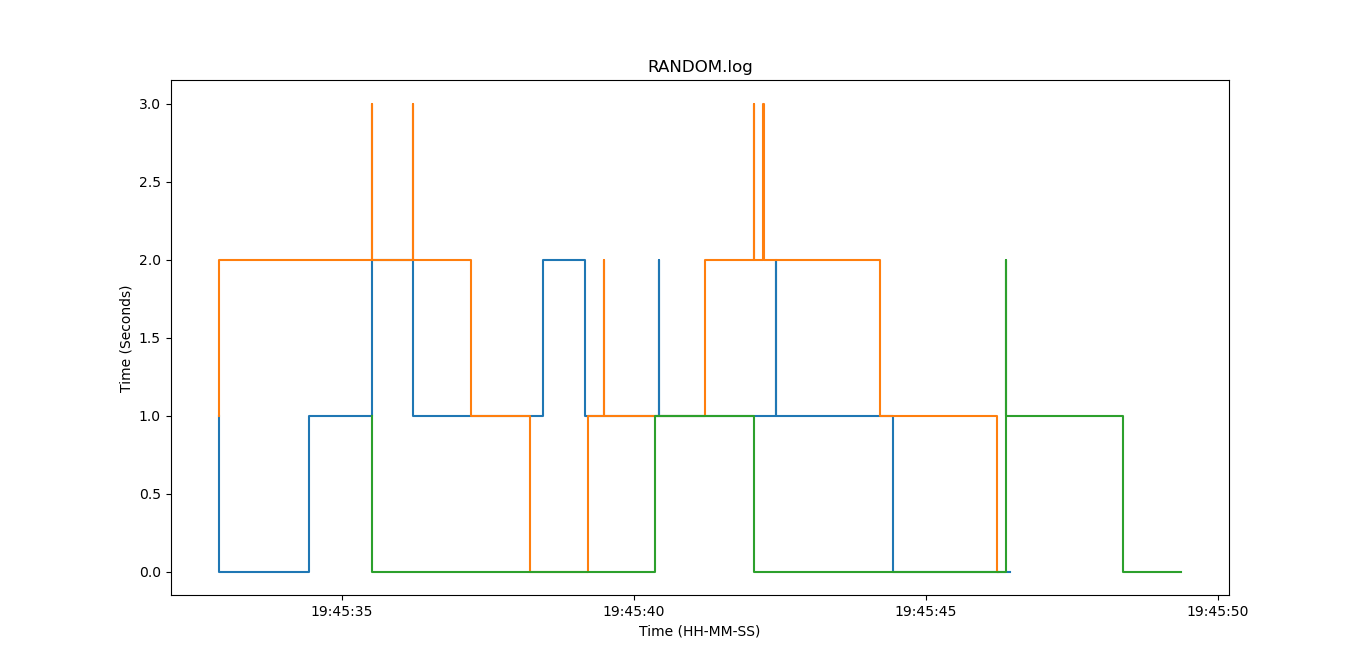
1. **Implementation of Random Scheduler**
   1. Log File



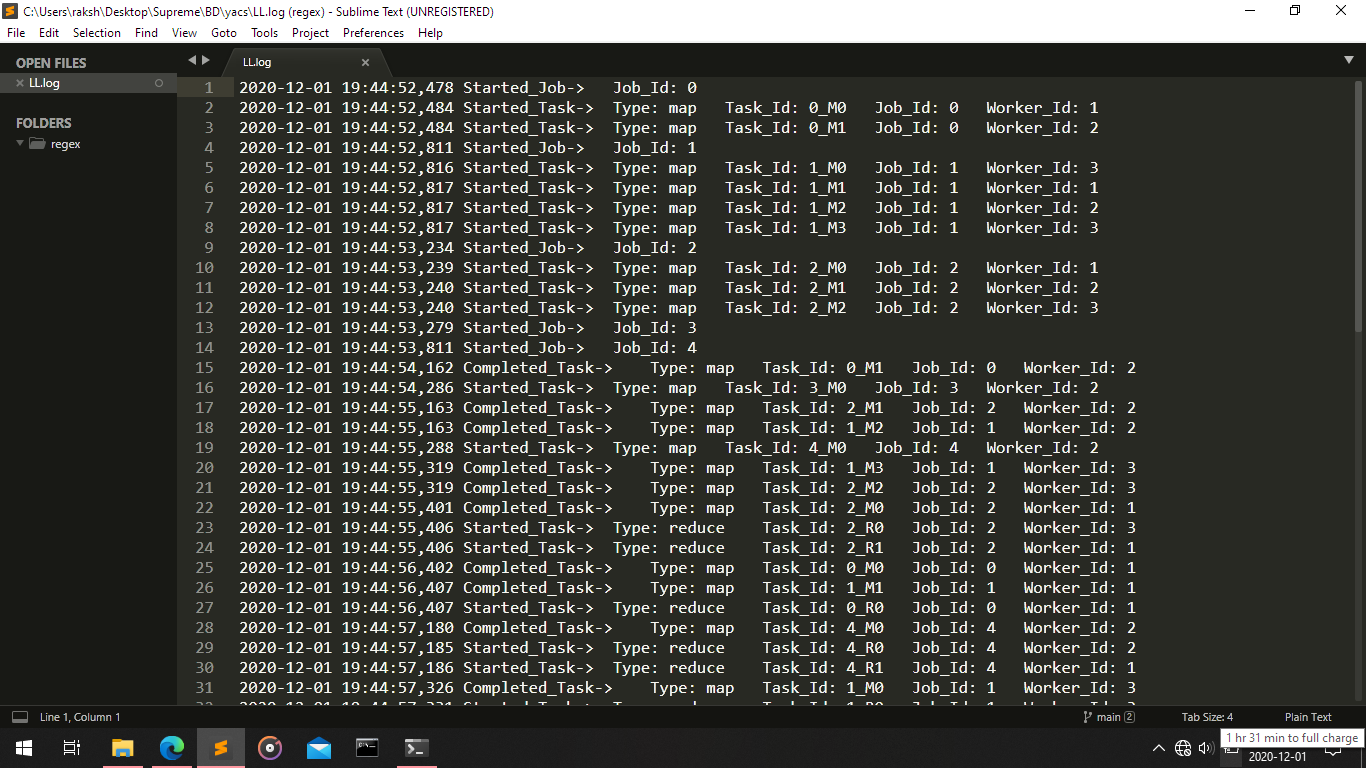
* 1. Mean and Median results :



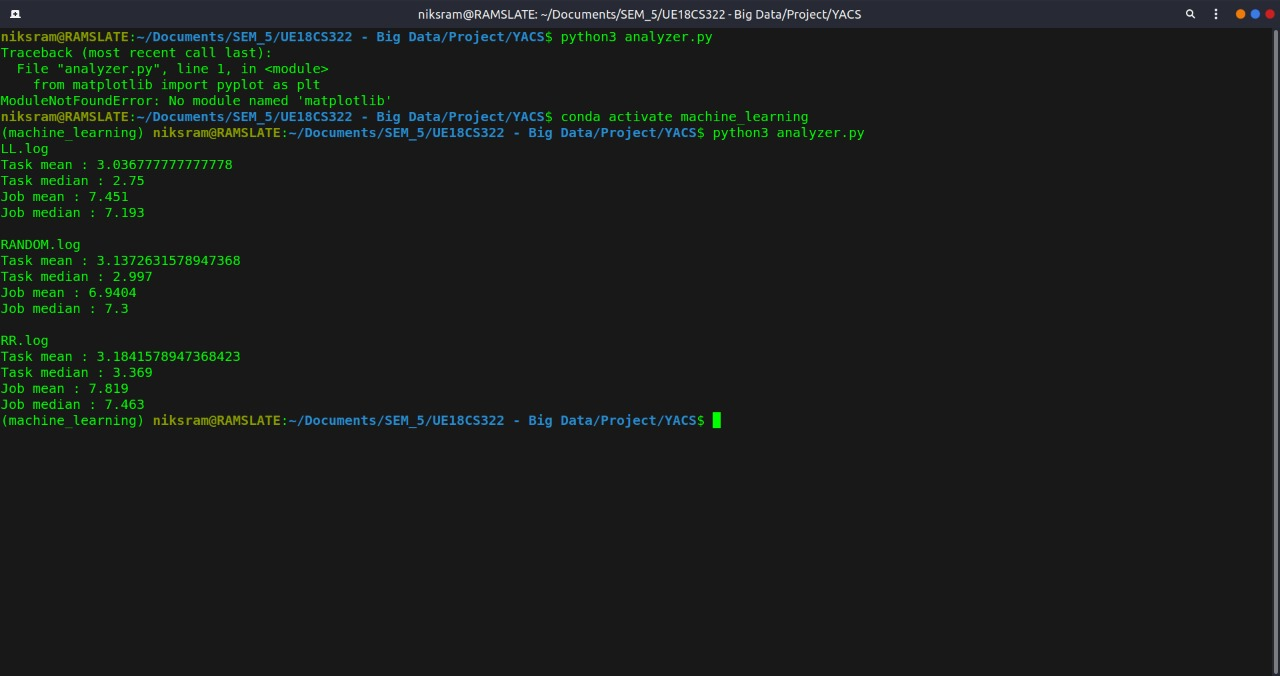
* 1. Plot of Scheduled tasks on Workers



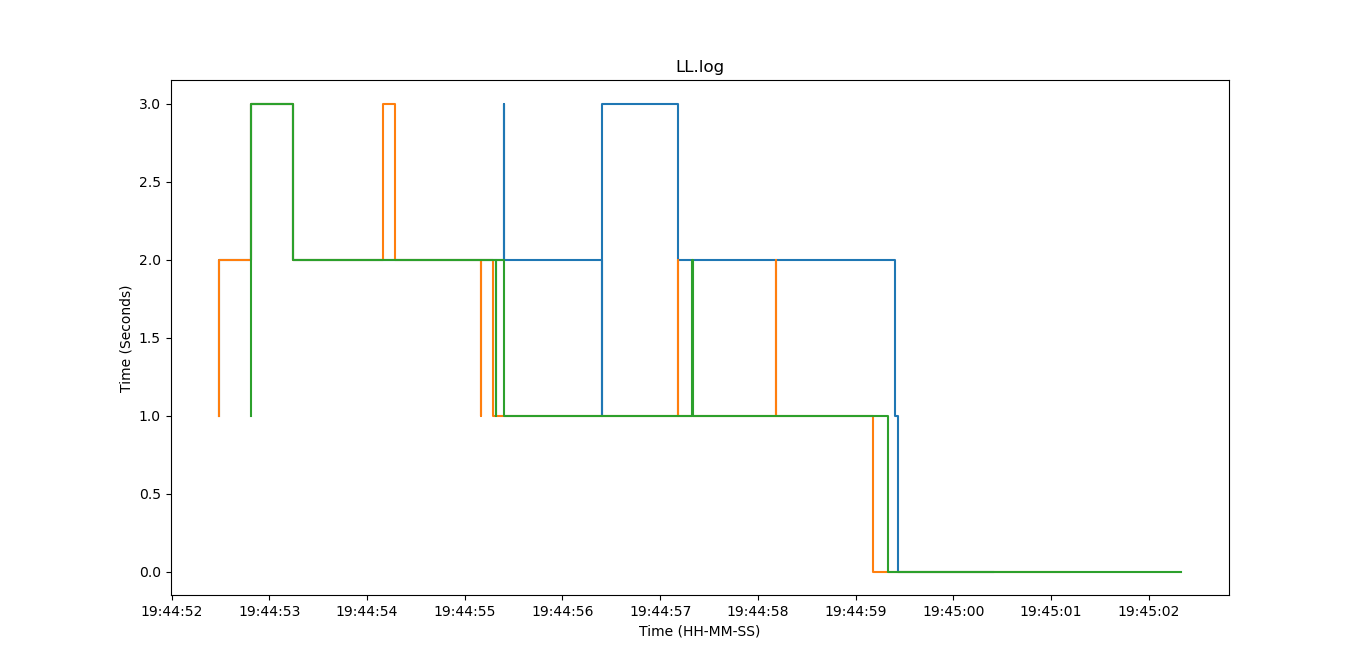
1. **Implementation of Least Loaded Scheduler**
   1. Log File

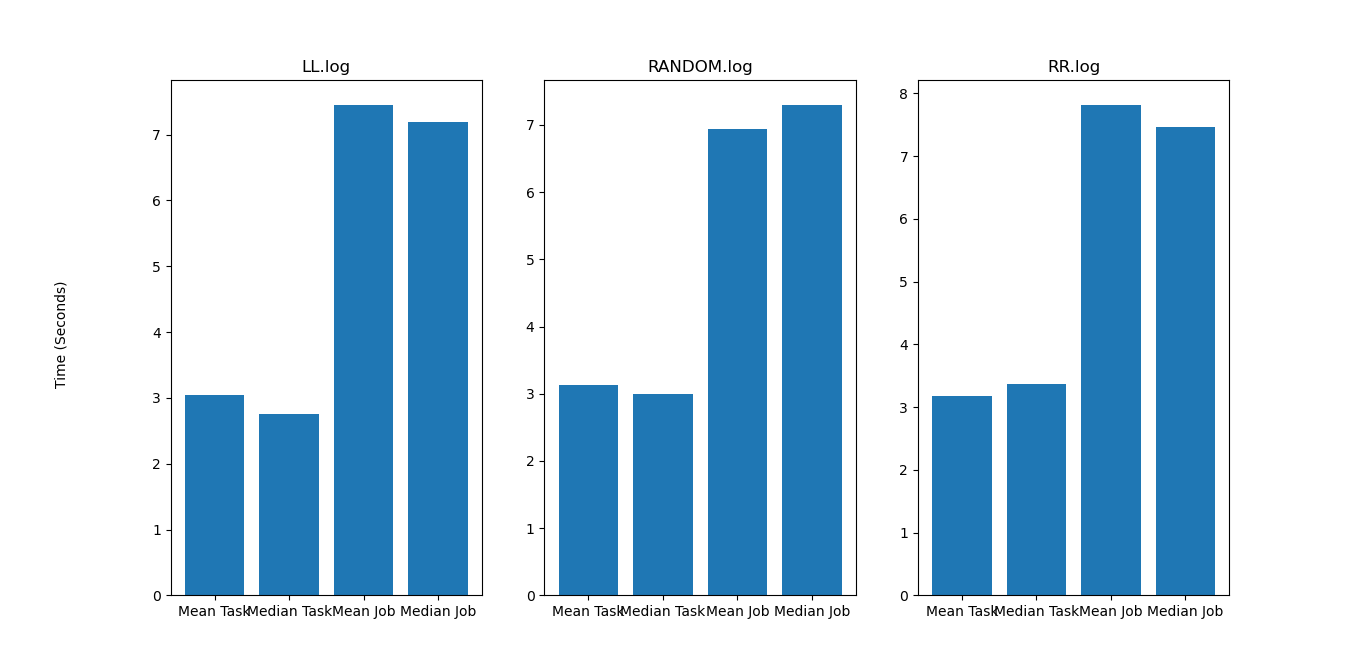


* 1. Mean and Median results :



* 1. Plot of Scheduled tasks on Workers



**Comparison of Performances of each Scheduling Algorithms**

## Problems

|  |  |
| --- | --- |
| Problem | Solution |
| 1. Standardized format of messages used to communicate between ports. | Json format was used for simplified encoding and decoding at sender and receiver |
| 1. Parallelizing the implementation of multiple jobs | Creation and utilization of a 3rd thread on master to send tasks to workers |
| 1. Consistency in format of logging messages | Using in-built logging module in python |

## Conclusion

**Main Learnings (Observations) from the Project:-**

* Resolving concurrency issues with the help of locking mechanism
* When the no. of slots in each worker is 1, the mean and median of the performance of each Scheduling algorithm is similar
* In some cases Random Scheduling algorithm proved to be more effective compared to Round Robin Scheduling
* In Round Robin, there was a fair distribution of load among all workers
* Least Loaded algorithm was found to be the most effective
* Random Scheduling algorithm caused an increase in total execution time in a few cases, due to queuing, increasing the waiting time unnecessarily in spite of workers being free.

## EVALUATIONS:

|  |  |  |  |
| --- | --- | --- | --- |
| SNo | Name | SRN | Contribution (Individual) |
| 1 | S Nikhil Ram | PES1201801972 | Implementation of Master |
| 2 | Rakshith C | PES1201801903 | Logging, Analyser and Scheduling algorithms |
| 3 | Vikas Gowda | PES1201801957 | Implementation of Worker |
| 4 | Yash Gawankar | PES1201801482 | Socket Programming and Resolving Concurrency using Locking |

## (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. |  |