UE18CS322 : Big Data

YACS - Yet Another Centralized Scheduler

Team id : BD_1482_1903_1957_1972

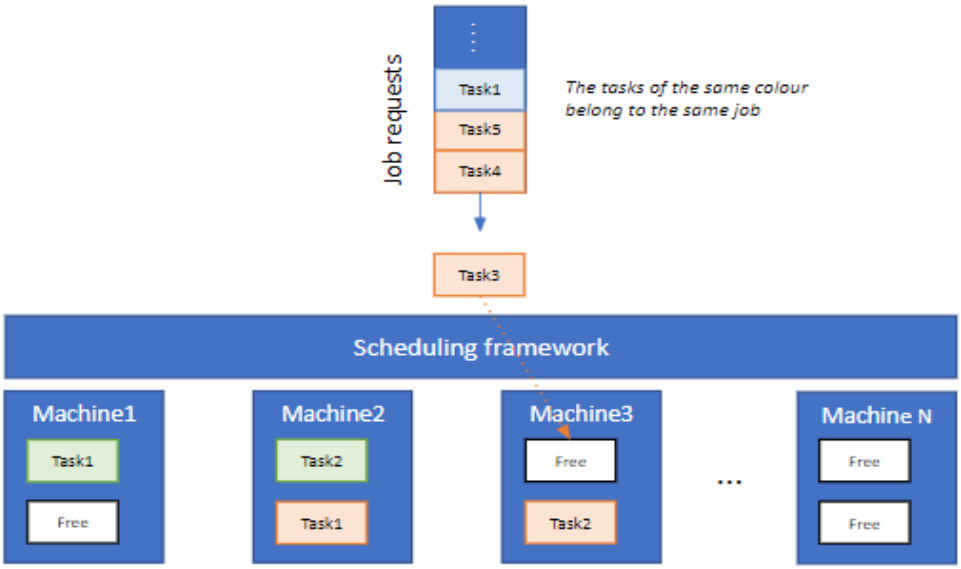
Final Class Project Report

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 - Medium.com Article
- Hadoop - Schedulers and Type of Schedulers - GeeksforGeeks
- Schedulers in YARN : from concepts to configurations - towardsdatascience.com
- Hadoop Schedulers Tutorial - Job Scheduling in Hadoop - data-flair.training

Books:-

- Hadoop: The Definitive Guide, 4th Edition - O'Reilly Publications, Chapter 4: YARN

Design

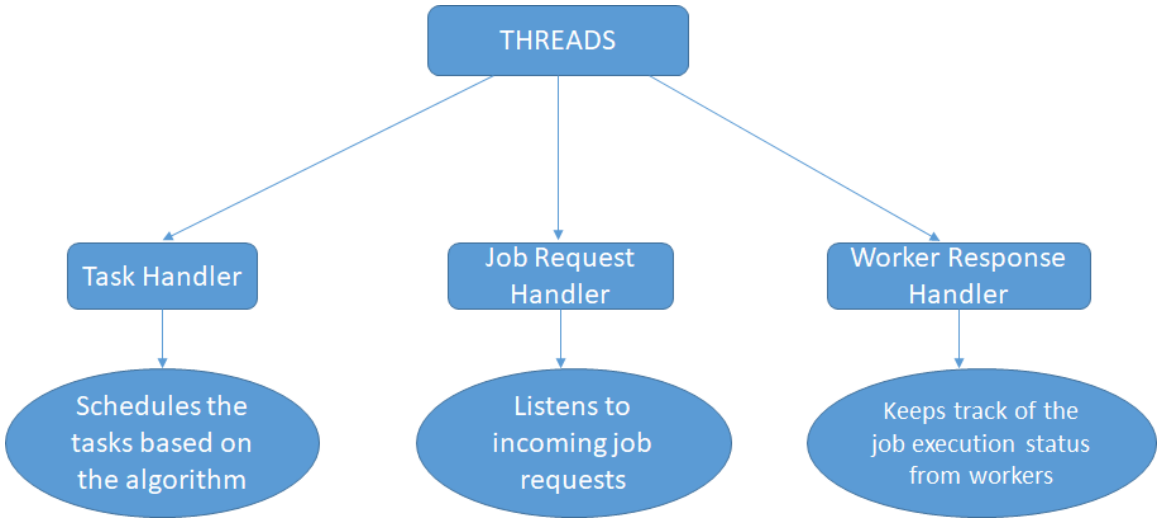
Overview of the General Design Strategy:-

- 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:-

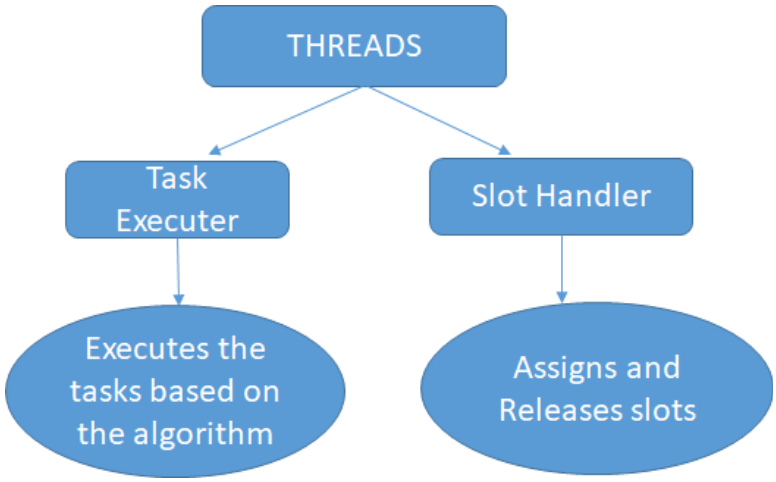
1. task_handler :
    a. Connects to the worker based on specified port no.
    b. Determines the type of algorithm used for scheduling
    c. Schedules the jobs based on the Algorithm
    d. It logs the start of each task in a job
    e. Also keeps track of no of free slots on the workers

2. job_request_handler:

a. Accepts the incoming job requests from requests.py
b. Reads and stores the JSON object consisting of Worker configurations
c. Appends the jobs to a job pool (queue)
d. Also logs the Start of every job

3. worker_response_handler:
    a. Receives the task completion status from the workers
    b. Logs the completion of each task
    c. As and when the jobs are completed, it pops them from the job pool
    d. 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

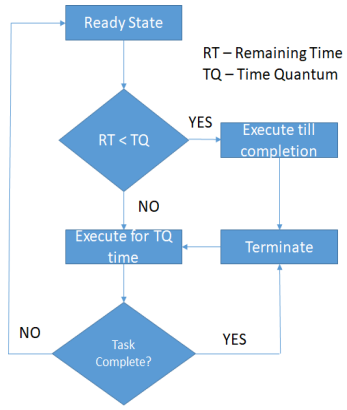
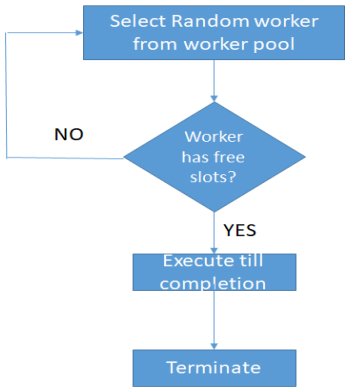
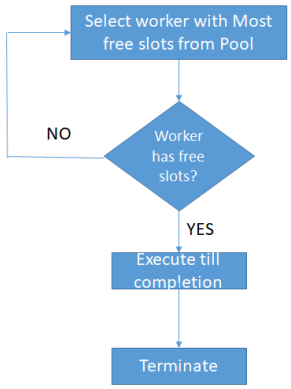
The threads are utilized by 2 classes namely:-

1. task_executer :
    a. Connects to master on port 5001
    b. Receives Map and Reduce tasks scheduled by master
    c. Checks for availability of slots
    d. Decreases the duration till the task is executed

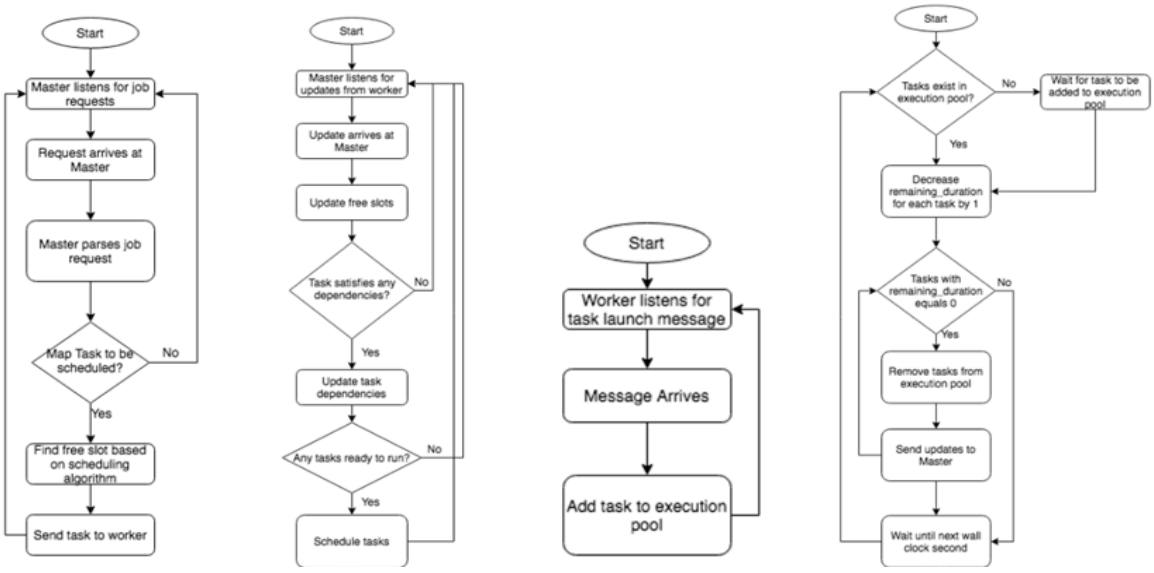
e. Acquires and releases lock while decreasing duration

2. slot_handler :
   a. Assigns the slots to each task
   b. Keeps track of no of available slots and reports to the master
   c. Acquires and releases lock while assigning the slots

Scheduling Algorithms implemented:-

Round Robin	Random Scheduler	Least Loaded
		

Model and WorkFlow:-



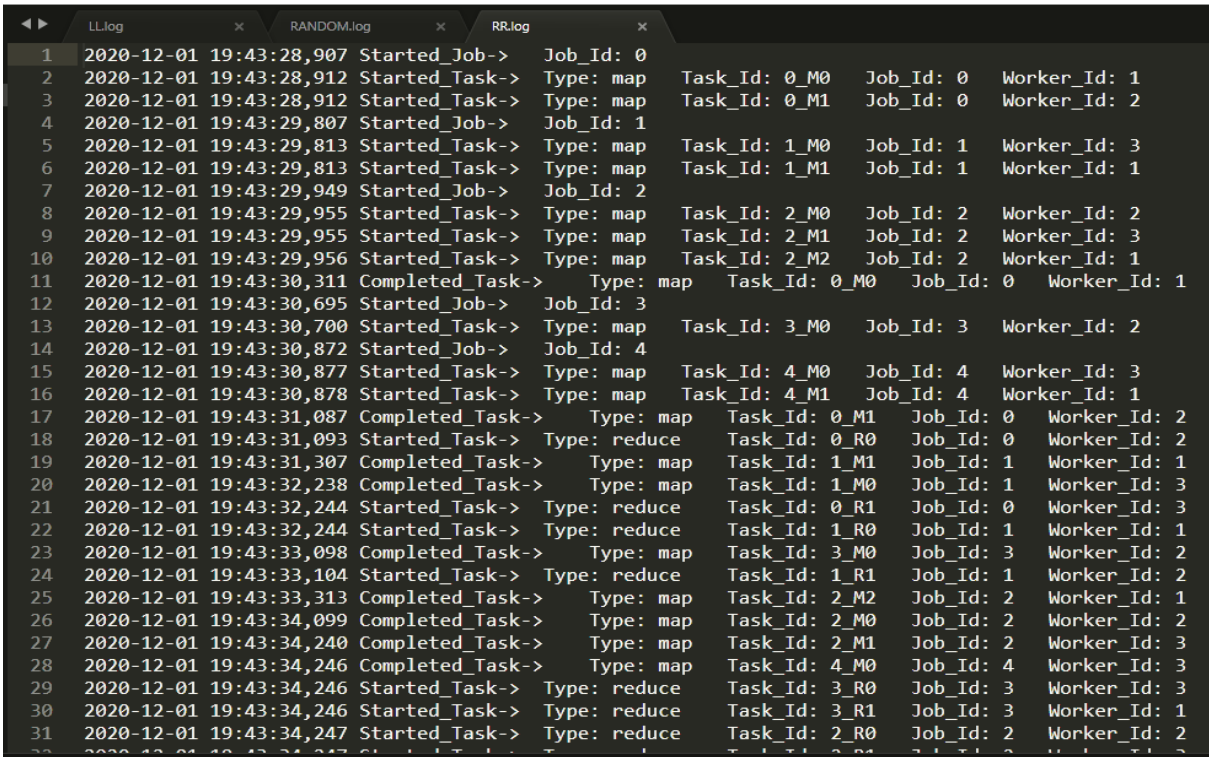
- The scheduler in the master has been introduced as the third thread in the implementation. This thread encomposses the scheduling algorithms

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.

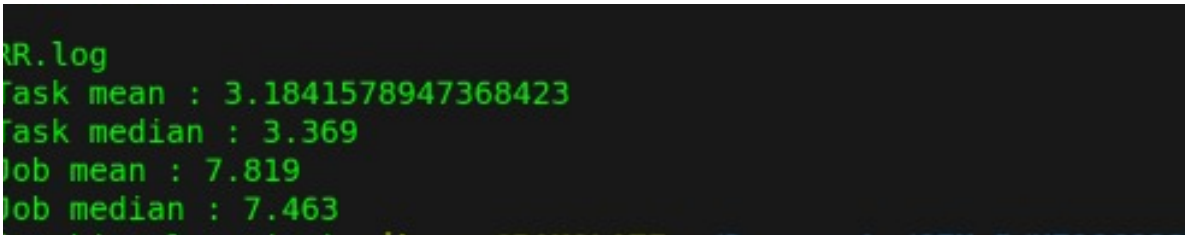
Results

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



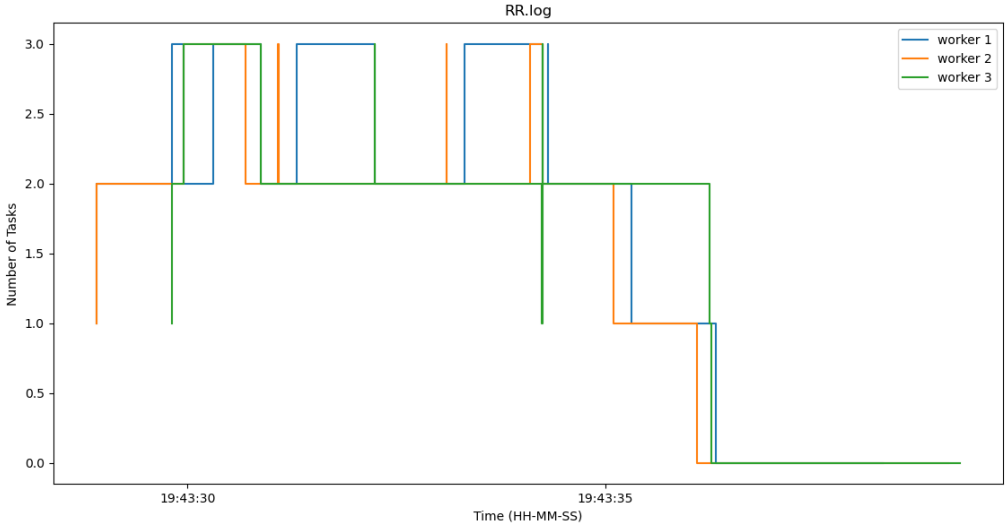
```
LL.log   RANDOM.log   RR.log
2020-12-01 19:43:28,907 Started_Job->   Job_Id: 0
2020-12-01 19:43:28,912 Started_Task->  Type: map   Task_Id: 0_M0   Job_Id: 0   Worker_Id: 1
2020-12-01 19:43:28,912 Started_Task->  Type: map   Task_Id: 0_M1   Job_Id: 0   Worker_Id: 2
2020-12-01 19:43:29,807 Started_Job->   Job_Id: 1
2020-12-01 19:43:29,813 Started_Task->  Type: map   Task_Id: 1_M0   Job_Id: 1   Worker_Id: 3
2020-12-01 19:43:29,813 Started_Task->  Type: map   Task_Id: 1_M1   Job_Id: 1   Worker_Id: 1
2020-12-01 19:43:29,949 Started_Job->   Job_Id: 2
2020-12-01 19:43:29,955 Started_Task->  Type: map   Task_Id: 2_M0   Job_Id: 2   Worker_Id: 2
2020-12-01 19:43:29,955 Started_Task->  Type: map   Task_Id: 2_M1   Job_Id: 2   Worker_Id: 3
2020-12-01 19:43:29,956 Started_Task->  Type: map   Task_Id: 2_M2   Job_Id: 2   Worker_Id: 1
2020-12-01 19:43:30,311 Completed_Task->    Type: map   Task_Id: 0_M0   Job_Id: 0   Worker_Id: 1
2020-12-01 19:43:30,695 Started_Job->   Job_Id: 3
2020-12-01 19:43:30,700 Started_Task->  Type: map   Task_Id: 3_M0   Job_Id: 3   Worker_Id: 2
2020-12-01 19:43:30,872 Started_Job->   Job_Id: 4
2020-12-01 19:43:30,877 Started_Task->  Type: map   Task_Id: 4_M0   Job_Id: 4   Worker_Id: 3
2020-12-01 19:43:30,878 Started_Task->  Type: map   Task_Id: 4_M1   Job_Id: 4   Worker_Id: 1
2020-12-01 19:43:31,087 Completed_Task->    Type: map   Task_Id: 0_M1   Job_Id: 0   Worker_Id: 2
2020-12-01 19:43:31,093 Started_Task->  Type: reduce    Task_Id: 0_R0   Job_Id: 0   Worker_Id: 2
2020-12-01 19:43:31,307 Completed_Task->    Type: map   Task_Id: 1_M1   Job_Id: 1   Worker_Id: 1
2020-12-01 19:43:32,238 Completed_Task->    Type: map   Task_Id: 1_M0   Job_Id: 1   Worker_Id: 3
2020-12-01 19:43:32,244 Started_Task->  Type: reduce    Task_Id: 0_R1   Job_Id: 0   Worker_Id: 3
2020-12-01 19:43:32,244 Started_Task->  Type: reduce    Task_Id: 1_R0   Job_Id: 1   Worker_Id: 1
2020-12-01 19:43:33,098 Completed_Task->    Type: map   Task_Id: 3_M0   Job_Id: 3   Worker_Id: 2
2020-12-01 19:43:33,104 Started_Task->  Type: reduce    Task_Id: 1_R1   Job_Id: 1   Worker_Id: 2
2020-12-01 19:43:33,313 Completed_Task->    Type: map   Task_Id: 2_M2   Job_Id: 2   Worker_Id: 1
2020-12-01 19:43:34,099 Completed_Task->    Type: map   Task_Id: 2_M0   Job_Id: 2   Worker_Id: 2
2020-12-01 19:43:34,240 Completed_Task->    Type: map   Task_Id: 2_M1   Job_Id: 2   Worker_Id: 3
2020-12-01 19:43:34,246 Completed_Task->    Type: map   Task_Id: 4_M0   Job_Id: 4   Worker_Id: 3
2020-12-01 19:43:34,246 Started_Task->  Type: reduce    Task_Id: 3_R0   Job_Id: 3   Worker_Id: 3
2020-12-01 19:43:34,246 Started_Task->  Type: reduce    Task_Id: 3_R1   Job_Id: 3   Worker_Id: 1
2020-12-01 19:43:34,247 Started_Task->  Type: reduce    Task_Id: 2_R0   Job_Id: 2   Worker_Id: 2
```

b. Mean and Median results :-



```
RR.log
Task mean : 3.1841578947368423
Task median : 3.369
Job mean : 7.819
Job median : 7.463
```

c. Plot of Scheduled tasks on Workers :



2. Implementation of Random Scheduler

a. Log File

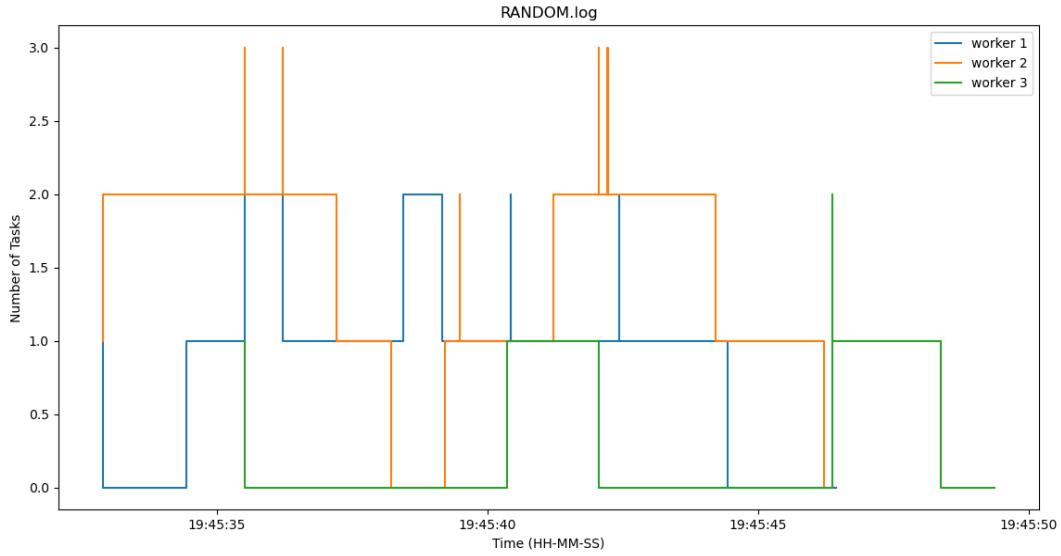
LL.log | RANDOM.log

```
2020-12-01 19:45:32,894 Started_Job->   Job_Id: 0
2020-12-01 19:45:32,899 Started_Task->  Type: map   Task_Id: 0_M0   Job_Id: 0   Worker_Id: 1
2020-12-01 19:45:32,900 Started_Task->  Type: map   Task_Id: 0_M1   Job_Id: 0   Worker_Id: 2
2020-12-01 19:45:34,439 Completed_Task->    Type: map   Task_Id: 0_M0   Job_Id: 0   Worker_Id: 1
2020-12-01 19:45:35,515 Started_Job->   Job_Id: 1
2020-12-01 19:45:35,520 Started_Task->  Type: map   Task_Id: 1_M0   Job_Id: 1   Worker_Id: 2
2020-12-01 19:45:35,521 Started_Task->  Type: map   Task_Id: 1_M1   Job_Id: 1   Worker_Id: 1
2020-12-01 19:45:35,521 Started_Task->  Type: map   Task_Id: 1_M2   Job_Id: 1   Worker_Id: 2
2020-12-01 19:45:35,522 Started_Task->  Type: map   Task_Id: 1_M3   Job_Id: 1   Worker_Id: 3
2020-12-01 19:45:36,218 Completed_Task->    Type: map   Task_Id: 0_M1   Job_Id: 0   Worker_Id: 2
2020-12-01 19:45:36,224 Started_Task->  Type: reduce    Task_Id: 0_R0   Job_Id: 0   Worker_Id: 2
2020-12-01 19:45:36,224 Started_Task->  Type: reduce    Task_Id: 0_R1   Job_Id: 0   Worker_Id: 1
2020-12-01 19:45:37,219 Completed_Task->    Type: map   Task_Id: 1_M2   Job_Id: 1   Worker_Id: 2
2020-12-01 19:45:38,221 Completed_Task->    Type: map   Task_Id: 1_M0   Job_Id: 1   Worker_Id: 2
2020-12-01 19:45:38,443 Completed_Task->    Type: reduce    Task_Id: 0_R1   Job_Id: 0   Worker_Id: 1
2020-12-01 19:45:39,162 Started_Job->   Job_Id: 2
2020-12-01 19:45:39,168 Started_Task->  Type: map   Task_Id: 2_M0   Job_Id: 2   Worker_Id: 1
2020-12-01 19:45:39,221 Completed_Task->    Type: reduce    Task_Id: 0_R0   Job_Id: 0   Worker_Id: 2
2020-12-01 19:45:39,242 Completed_Job-> Job_Id: 0
2020-12-01 19:45:39,495 Started_Job->   Job_Id: 3
2020-12-01 19:45:39,500 Started_Task->  Type: map   Task_Id: 3_M0   Job_Id: 3   Worker_Id: 2
2020-12-01 19:45:39,501 Started_Task->  Type: map   Task_Id: 3_M1   Job_Id: 3   Worker_Id: 2
2020-12-01 19:45:40,370 Completed_Task->    Type: map   Task_Id: 1_M3   Job_Id: 1   Worker_Id: 3
2020-12-01 19:45:40,435 Completed_Task->    Type: map   Task_Id: 1_M1   Job_Id: 1   Worker_Id: 1
2020-12-01 19:45:40,441 Started_Task->  Type: reduce    Task_Id: 1_R0   Job_Id: 1   Worker_Id: 1
2020-12-01 19:45:41,224 Completed_Task->    Type: map   Task_Id: 3_M1   Job_Id: 3   Worker_Id: 2
2020-12-01 19:45:42,054 Started_Job->   Job_Id: 4
2020-12-01 19:45:42,061 Started_Task->  Type: map   Task_Id: 4_M0   Job_Id: 4   Worker_Id: 2
2020-12-01 19:45:42,062 Started_Task->  Type: map   Task_Id: 4_M1   Job_Id: 4   Worker_Id: 2
2020-12-01 19:45:42,063 Started_Task->  Type: map   Task_Id: 4_M2   Job_Id: 4   Worker_Id: 3
2020-12-01 19:45:42,225 Completed_Task->    Type: map   Task_Id: 3_M0   Job_Id: 3   Worker_Id: 2
```

b. Mean and Median results :

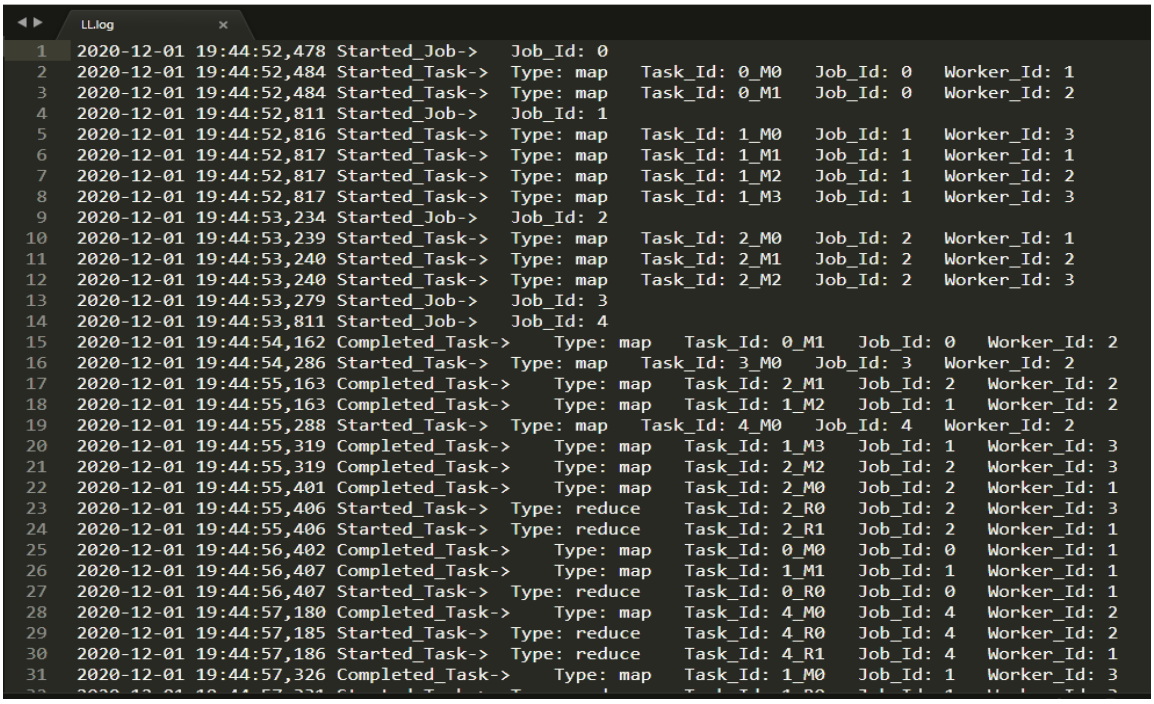
```
RANDOM.log
Task mean : 3.1372631578947368
Task median : 2.997
Job mean : 6.9404
Job median : 7.3
```

c. Plot of Scheduled tasks on Workers



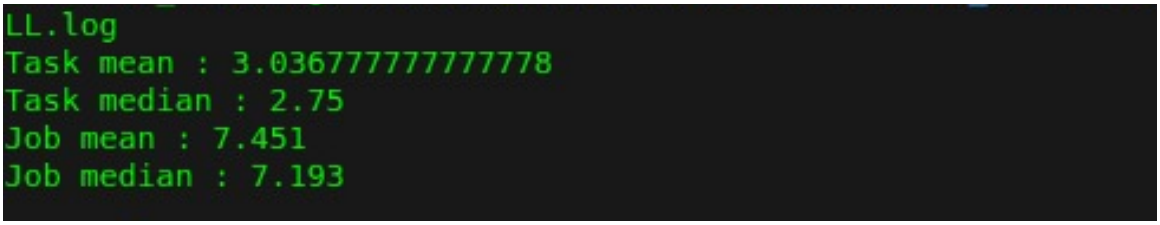
3. Implementation of Least Loaded Scheduler

a. Log File



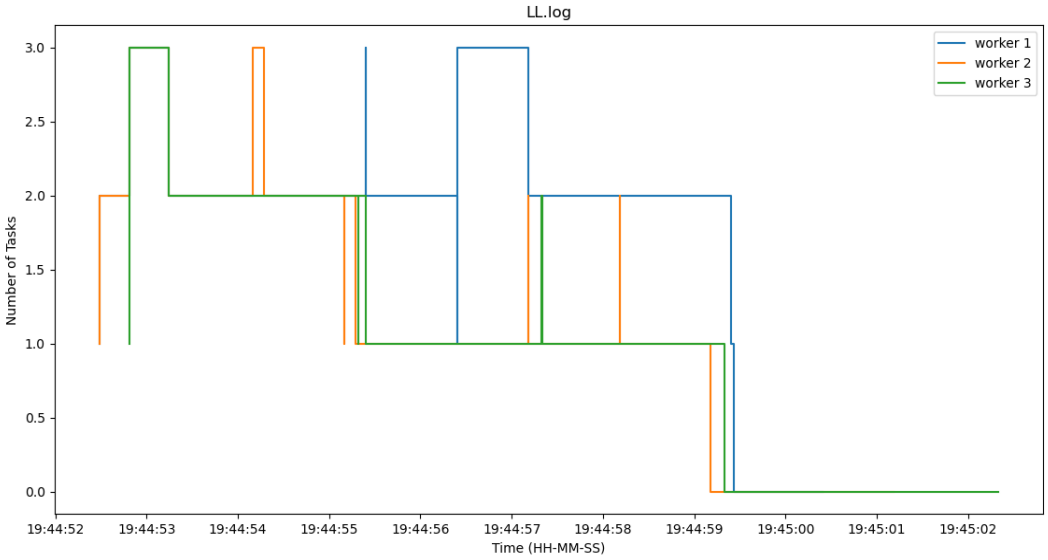
```
LL.log
1  2020-12-01 19:44:52,478 Started_Job->   Job_Id: 0
2  2020-12-01 19:44:52,484 Started_Task->  Type: map   Task_Id: 0_M0   Job_Id: 0   Worker_Id: 1
3  2020-12-01 19:44:52,484 Started_Task->  Type: map   Task_Id: 0_M1   Job_Id: 0   Worker_Id: 2
4  2020-12-01 19:44:52,811 Started_Job->   Job_Id: 1
5  2020-12-01 19:44:52,816 Started_Task->  Type: map   Task_Id: 1_M0   Job_Id: 1   Worker_Id: 3
6  2020-12-01 19:44:52,817 Started_Task->  Type: map   Task_Id: 1_M1   Job_Id: 1   Worker_Id: 1
7  2020-12-01 19:44:52,817 Started_Task->  Type: map   Task_Id: 1_M2   Job_Id: 1   Worker_Id: 2
8  2020-12-01 19:44:52,817 Started_Task->  Type: map   Task_Id: 1_M3   Job_Id: 1   Worker_Id: 3
9  2020-12-01 19:44:53,234 Started_Job->   Job_Id: 2
10 2020-12-01 19:44:53,239 Started_Task->  Type: map   Task_Id: 2_M0   Job_Id: 2   Worker_Id: 1
11 2020-12-01 19:44:53,240 Started_Task->  Type: map   Task_Id: 2_M1   Job_Id: 2   Worker_Id: 2
12 2020-12-01 19:44:53,240 Started_Task->  Type: map   Task_Id: 2_M2   Job_Id: 2   Worker_Id: 3
13 2020-12-01 19:44:53,279 Started_Job->   Job_Id: 3
14 2020-12-01 19:44:53,811 Started_Job->   Job_Id: 4
15 2020-12-01 19:44:54,162 Completed_Task->    Type: map   Task_Id: 0_M1   Job_Id: 0   Worker_Id: 2
16 2020-12-01 19:44:54,286 Started_Task->  Type: map   Task_Id: 3_M0   Job_Id: 3   Worker_Id: 2
17 2020-12-01 19:44:55,163 Completed_Task->    Type: map   Task_Id: 2_M1   Job_Id: 2   Worker_Id: 2
18 2020-12-01 19:44:55,163 Completed_Task->    Type: map   Task_Id: 1_M2   Job_Id: 1   Worker_Id: 2
19 2020-12-01 19:44:55,288 Started_Task->  Type: map   Task_Id: 4_M0   Job_Id: 4   Worker_Id: 2
20 2020-12-01 19:44:55,319 Completed_Task->    Type: map   Task_Id: 1_M3   Job_Id: 1   Worker_Id: 3
21 2020-12-01 19:44:55,319 Completed_Task->    Type: map   Task_Id: 2_M2   Job_Id: 2   Worker_Id: 3
22 2020-12-01 19:44:55,401 Completed_Task->    Type: map   Task_Id: 2_M0   Job_Id: 2   Worker_Id: 1
23 2020-12-01 19:44:55,406 Started_Task->  Type: reduce    Task_Id: 2_R0   Job_Id: 2   Worker_Id: 3
24 2020-12-01 19:44:55,406 Started_Task->  Type: reduce    Task_Id: 2_R1   Job_Id: 2   Worker_Id: 1
25 2020-12-01 19:44:56,402 Completed_Task->    Type: map   Task_Id: 0_M0   Job_Id: 0   Worker_Id: 1
26 2020-12-01 19:44:56,407 Completed_Task->    Type: map   Task_Id: 1_M1   Job_Id: 1   Worker_Id: 1
27 2020-12-01 19:44:56,407 Started_Task->  Type: reduce    Task_Id: 0_R0   Job_Id: 0   Worker_Id: 1
28 2020-12-01 19:44:57,180 Completed_Task->    Type: map   Task_Id: 4_M0   Job_Id: 4   Worker_Id: 2
29 2020-12-01 19:44:57,185 Started_Task->  Type: reduce    Task_Id: 4_R0   Job_Id: 4   Worker_Id: 2
30 2020-12-01 19:44:57,186 Started_Task->  Type: reduce    Task_Id: 4_R1   Job_Id: 4   Worker_Id: 1
31 2020-12-01 19:44:57,326 Completed_Task->    Type: map   Task_Id: 1_M0   Job_Id: 1   Worker_Id: 3
```

b. Mean and Median results :

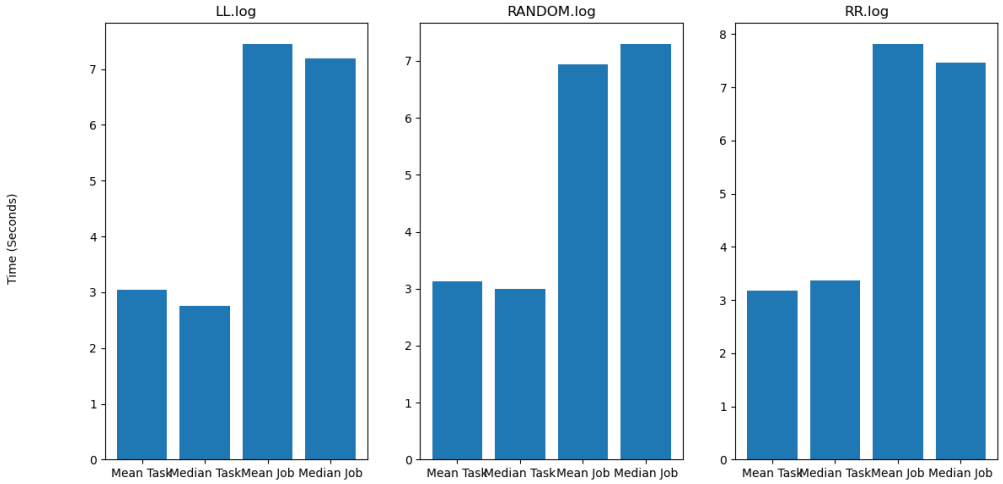


```
LL.log
Task mean : 3.036777777777778
Task median : 2.75
Job mean : 7.451
Job median : 7.193
```

c. 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
2. Parallelizing the implementation of multiple jobs	Creation and utilization of a 3rd thread on master to send tasks to workers
3. 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

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.	