

CS322:Big Data

Final Class Project Report

**Project (FPL Analytics / YACS coding): \_\_\_\_\_YACS \_\_\_\_**  **Date: \_\_\_\_\_1/12/2020\_\_\_\_\_**

|  |  |  |  |
| --- | --- | --- | --- |
| SNo | Name | SRN | Class/Section |
| 1 | P Deepak Reddy | PES1201800344 | ‘F’ |
| 2 | Nikhil DB | PES1201801763 | ‘F’ |
| 3 | Nisha S | PES1201801777 | ‘J’ |
| 4 | Trisha C Shekar | PES1201801854 | ‘B’ |

## Introduction

YACS stands for Yet Another Centralized Scheduler. It is basically a Master-Worker Architecture for execution of jobs. The Master receives Job requests, which it then assigns the tasks related to the job to respective Workers based on the Scheduling Algorithm specified by the user. The Master and the Workers communicate with each other by the help of sockets. A job consists of multiple tasks (mapper and reducer tasks), each of these tasks is assigned to a particular slot from a chosen worker. The reducer tasks can only be executed once all the Mapper tasks have finished execution. Thus, this project is basically a Centralized Job Scheduling Master-Worker implementation.

## Related work

We had to go through the Socket programming and threading concepts. Related study about Semaphores and Locks, and how all these can be implemented in python classes and objects.

## Design

This design consists of master and workers having multiple threads to simulate a centralized scheduler.

The master maintains a the status of all the jobs received through a data structure which contains the job\_id , map\_tasks status, reduce\_tasks status for each job . The Master also maintains a task queue which contains all the tasks which satisfies all dependencies and can be scheduled to the worker. The number of free slots available in each worker is also kept track of.

The first thread of the master listens to port 5000 where it receives job requests from Requests.py. This thread also pushes all the map tasks of the received jobs to the task queue because the map tasks doesn't have any kind of dependency and can be scheduled readily. The second thread is used to schedule tasks to the workers based on scheduling algorithm specified by the user. This thread pops task from task queue and sends it to a worker based on the scheduling.

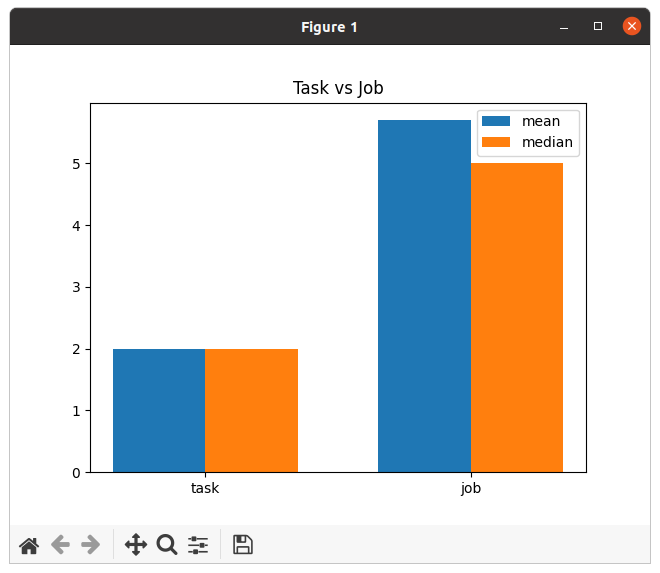
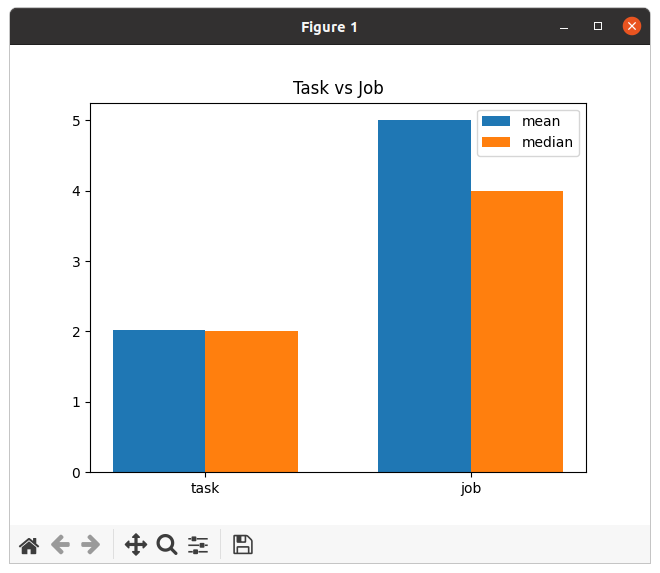
The third thread is used to listens to updates from all the workers from port 5001. This thread correspondingly updates the number of free slots of each worker, job status and also checks whether the tasks finished by worker is mapper or reducer.

If it is map task, it checks whether all the map tasks of for this completed. If all map tasks are completed, it appends the corresponding reduce tasks to the queue. If the task finished by worker is a reduce task, it checks whether all the reduce tasks are completed, if all reduce tasks are completed, it means that job is finished

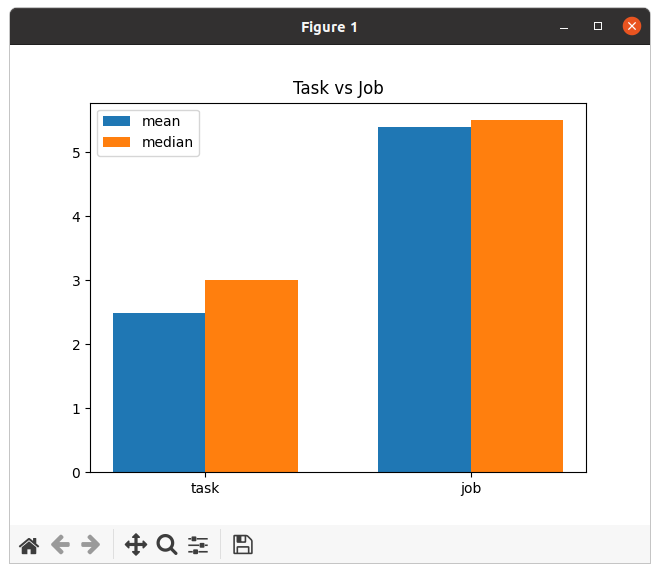
Each worker has 2 threads running parallel. The first thread is listens to the master for the incoming tasks and appends the tasks to a tasks\_list if there are any. The second thread will simulate the execution of tasks by decreasing the total duration of all tasks in the tasks\_list by 1 every second. This thread also checks the tasks\_list if any of the tasks are finished, in such a case, it sends a message back to the master.

This design assumes that data required for a job is replicated/shared across all the workers and output of map tasks can be shared to different workers where reduce tasks can be executed

## Results



Least loaded Random



Round Robin

It was observed that the mean job execution time when random scheduling algorithm was used was the highest. It was followed by round robin. When least loaded algorithm was used we got the least mean job execution time.

Mean job execution was the highest when random and round robin were used as scheduling algorithm because proper load balancing is not done here unlike least loaded where a particular worker is chosen only if it is less loaded.

In random we are choosing a worker irrespective of whether it is heavily loaded or not. A task is sent to any worker at random, this randomly chosen worker might have just one free slot while the other workers might be less loaded. This would stress the heavily loaded worker and the time taken to execute a task would definitely increase and hence high mean task and job completion time.

In round robin we are choosing workers on a cyclic basis. Some tasks might take longer time to complete so the number of free slots in a particular worker might be less than the others, but the tasks gets allocated to this free slot even though other workers have more free slots. Workers are chosen on a cyclic basis to distribute load on all the workers but in the case as mentioned before, it might increase the time to complete a job.

Load distribution happens properly in case of least loaded because a worker is chosen only if the number of free slots are higher and this would mean lesser mean time for completion when compared to the other scheduling algorithms.

Tasks were allotted to slots in a particular worker based on the scheduling algorithm specified. Variations were observed when we plotted number of tasks scheduled vs. time. This variation is because of the load balancing in different scheduling algorithms as mention earlier.

## Problems

A common shared variable which can be accessed by both the threads was required, hence we made use of class and objects.

Race Condition when the shared variable was accessed by multiple threads at the same time, in order to solve this, we made use of semaphores and locks.

The data transfer between the sockets happens via String of bytes, python module ast was used to convert it into dictionary.

## Conclusion

We learnt how:

-> The Master-Worker Architecture works.

-> Hadoop schedules jobs over various clusters, across nodes.

-> The Map-Reduce framework in Hadoop handles dependencies of job requests.

-> YARN performs scheduling and resource allocation for the Hadoop System.

## EVALUATIONS:

|  |  |  |  |
| --- | --- | --- | --- |
| SNo | Name | SRN | Contribution (Individual) |
| 1 | P Deepak Reddy | PES1201800344 | Master.py, Semaphores, shared data structure, checking dependencies for reducer tasks, threading |
| 2 | Nikhil DB | PES1201801763 | Worker.py, Socket Programming, Threading, listening for incoming requests, Logging data to .csv file |
| 3 | Nisha S | PES1201801777 | Analysis.py, Scheduling Algorithms, Bar Graph, Line Plot and other illustrations |
| 4 | Trisha C Shekar | PES1201801854 | Scheduling Algorithms |

## (Leave this for the faculty)

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Evaluator | Comments | Score |
|  |  |  |  |

## CHECKLIST:

|  |  |  |
| --- | --- | --- |
| SNo | Item | Status |
| 1. | Source code documented | Done |
| 2. | Source code uploaded to GitHub – (access link for the same, to be added in status 🡪) | Link : https://github.com/nisha-s/BD\_0344\_1763\_1777\_1854 |
| 3. | Instructions for building and running the code. Your code must be usable out of the box. | Instructions in readme.txt (file uploaded on github) |