CS322:Big Data

Final Class Project Report

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

SNo	Name	SRN	Class/Section
1	Ashwin Alaparthi	PES1201802062	E
2	Raghav Roy	PES1201800342	E
3	Raksha Ramesh	PES1201800345	H
4	Ritik Hariani	PES1201801558	J

Introduction

Our project YACS, or, Yet Another Centralized Scheduler is an implementation of a centralized scheduler, i.e. Jobs (Comprised of multiple tasks) are sent to a main Master (running on a dedicated machine), which distributes these tasks among multiple workers machines depending on the scheduling algorithm. The workers execute the tasks allocated to them and inform the Master when a task completes its execution. Each worker has fixed number of slots (each have independent dedicated CPU, memory, etc.), where each slot can run one task at a time. The master must check the availability of slots in the workers before allocating any task to them. Once a worker finishes executing a task, the slot held by the task is released.

Related work

Documentation of YARN in Hadoop was reviewed along with Operating System concepts like threading, semaphores and scheduling algorithms, with sockets implementation in Python for communication between Master and Workers.

Design

In our submission, we have a program for the Master server that's responsible for –

- A thread listening to new job requests from Requests Server
- Scheduling the tasks of the job, while considering any dependencies and resolving accordingly
- Keeping track of slots in workers and decrementing when scheduling a task, and incrementing when task is finished
- A thread to listen to updates from worker and accordingly log the time taken and other details for Tasks and Jobs

Similarly, the Workers are tasked with –

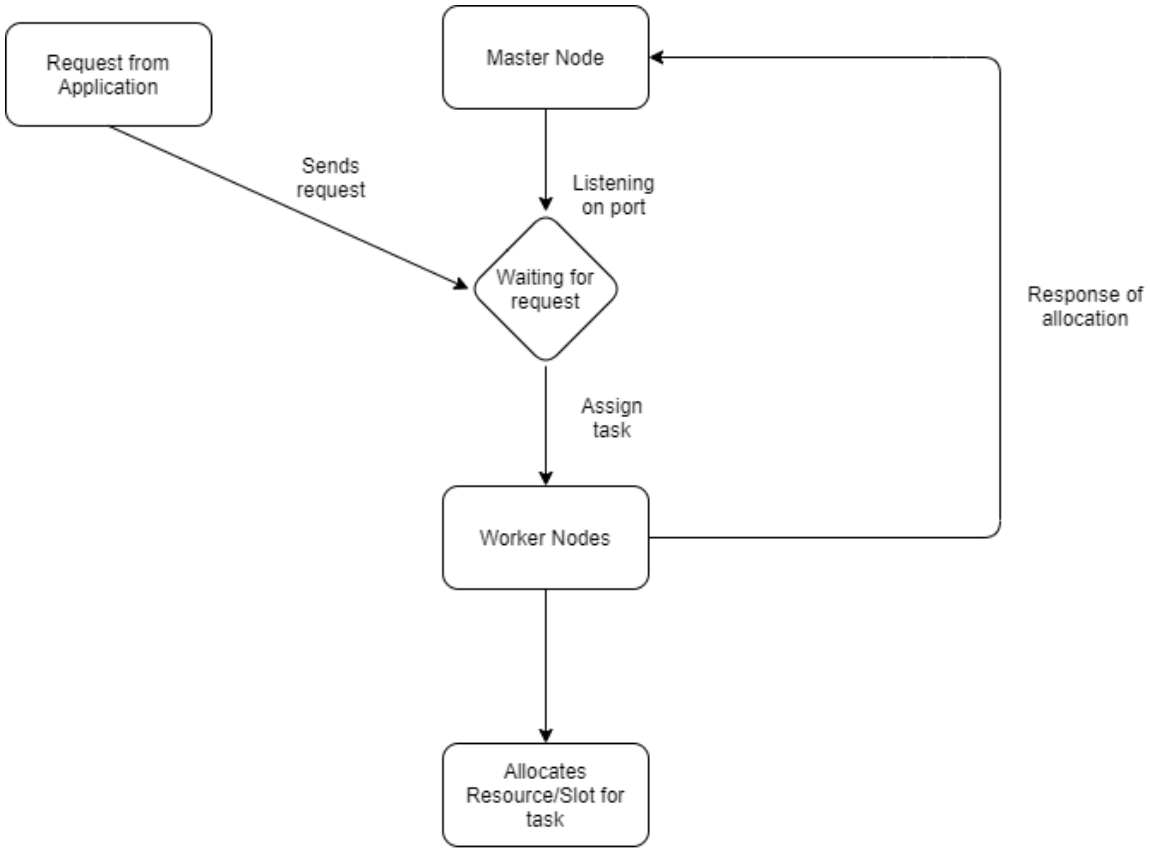
- Listening to new tasks from Master
- Performing the task for the required amount of time (simulated by making the worker thread sleep for a certain amount of time)
- Update the master back on task completion with the end time on Worker

Design of Master –

1. When the Master program is run, it creates two loggers to log the Task times and Job times. It also initiates a Task Queue to keep track of tasks to Schedule.
2. The worker config file is read and a dictionary of workers is created with the port and slots on the worker.

3. A schedule() function runs indefinitely and when the Task Queue is not empty, it schedules the task based on the required Algorithm. The 3 possible ways to schedule are:
    I. Random: In this scheduling algorithm, a random worker is chosen, if it doesn't have any free slots, a new random worker is chosen. This runs in an infinite loop till the randomly chosen worker has a free slot which can run the task.
    II. Round Robin: In this scheduling algorithm, there is a global variable maintained which indicates the worker that the most recent task was scheduled to. The incoming task is scheduled to the next chronological worker that has a free slot and the global variable is updated.
    III. Least Loaded: In this scheduling algorithm, there is a helper function that looks at the workers and returns the worker with the maximum number of slots available. Every time a new task has to be scheduled this function is called; if no worker has a free slot, then the program sleeps for 1 second and calls the function again. This is repeated until the worker is found and the task is scheduled.

4. A thread of the master listens to new Job requests on Port 5000. Upon receiving a new job, it notes down the Job ID and job entry time. It then stores the job in a dictionary, with the map tasks and reduce tasks. The map tasks of the job are added to the Tasks Queue.
5. Another thread of the master listens for updates from the worker. The update contains worker ID, Completed Task, and time completed. It then logs the task with start and end time, and checks if it's a Map or Reduce task. If all Map tasks for that job have been completed, we're scheduling all the reduce tasks by adding them to Task Queue. If all Reduce tasks are completed, then Log the job with start and end time.



Design workflow

Results

The results were as expected. The different values obtained are given below with respect to their scheduling algorithm for 25 requests (in seconds):

1. **Random:**
   Mean time taken by jobs: 6.001155529022217
   Median time taken by jobs: 5.6103980541229 25
   Mean time taken by tasks: 2.564184374809265
   Median time taken by tasks: 3.005800008773 8037
2. **Round Robin:**
   Mean time taken by jobs: 6.139268074035645
   Median time taken by jobs: 6.3885228633880615
   Mean time taken by tasks: 2.3966108654059615
   Median time taken by tasks: 2.01705539226532

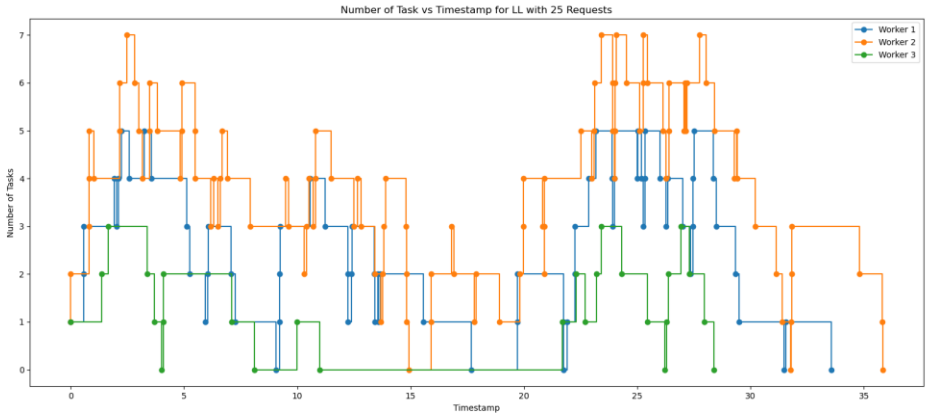
3. **Least Loaded**

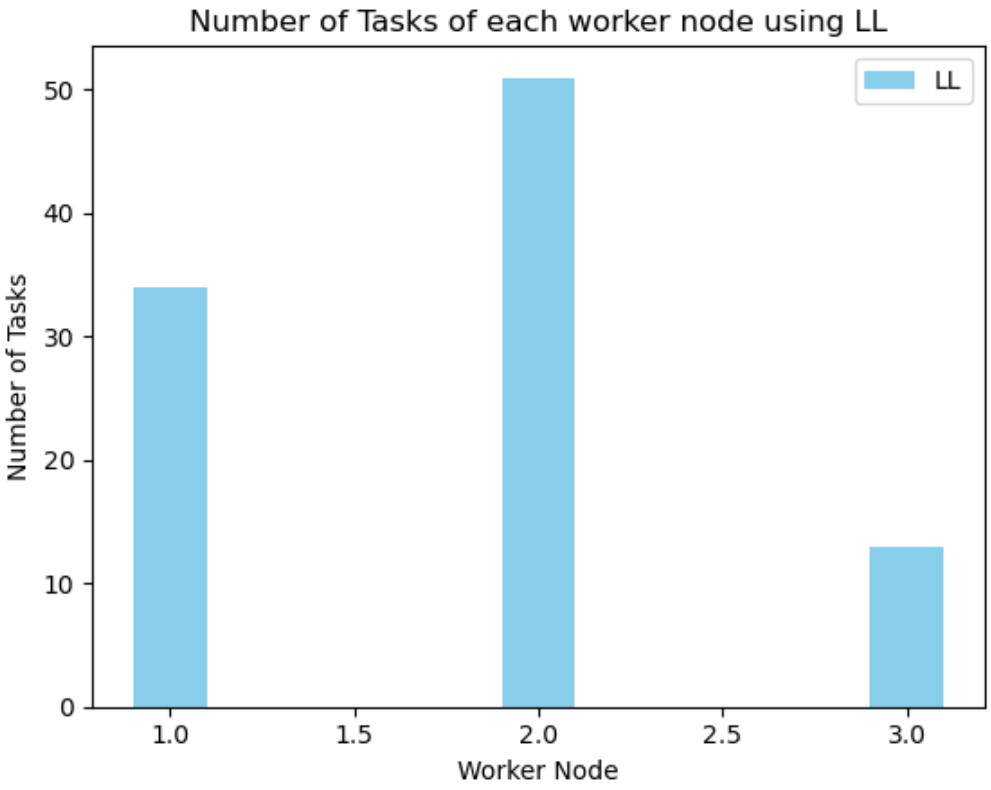
Mean time taken by jobs: 6.1543086814880 37
Median time taken by jobs: 6.3290925025939 94
Mean time taken by tasks: 2.5047072391120757
Median time taken by tasks: 2.0263891220092773

Graphs:

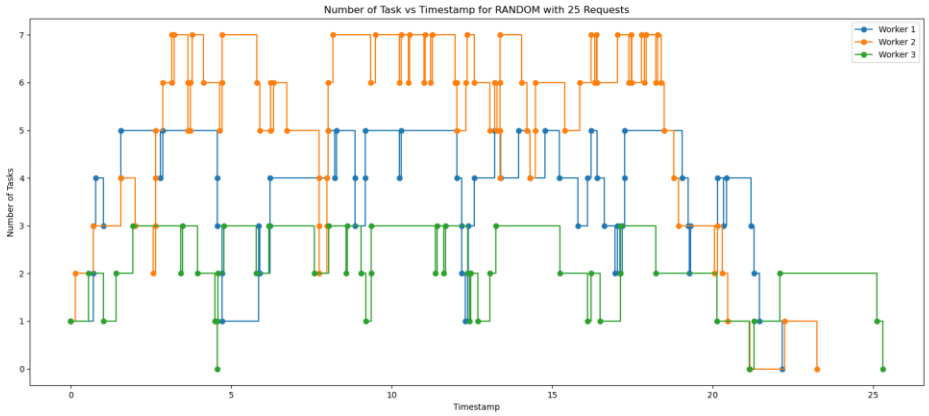
Step Plot for Number of Tasks vs Time. Bar Graph for Number of Tasks on a worker node.

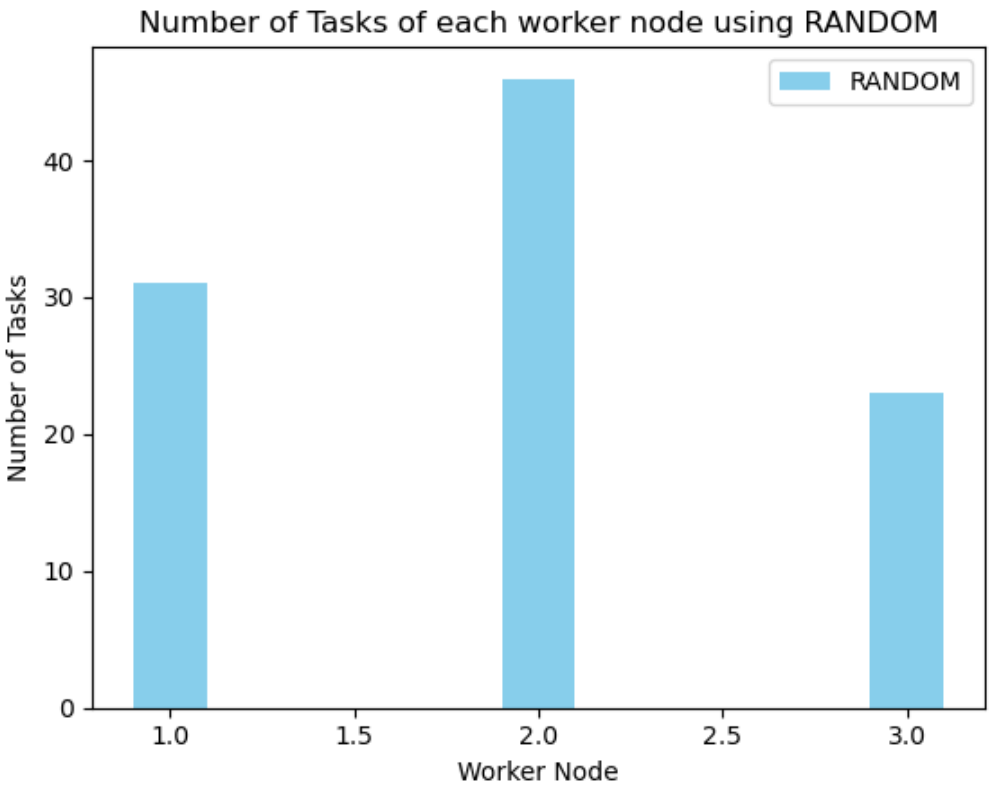
1. Least Loaded



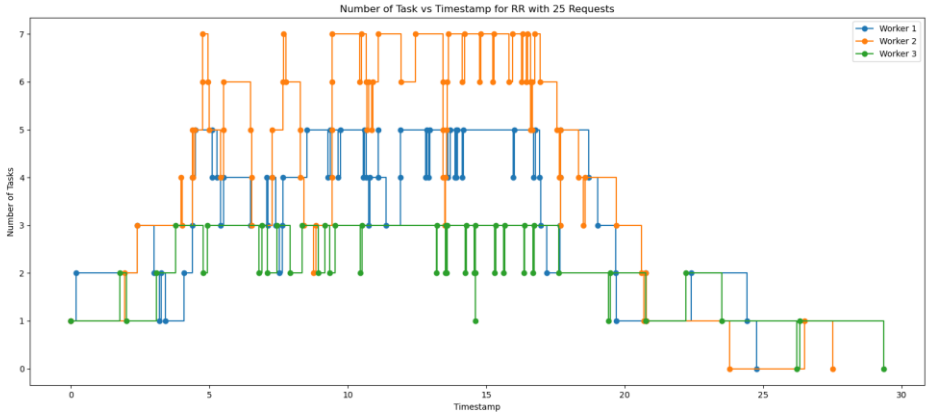


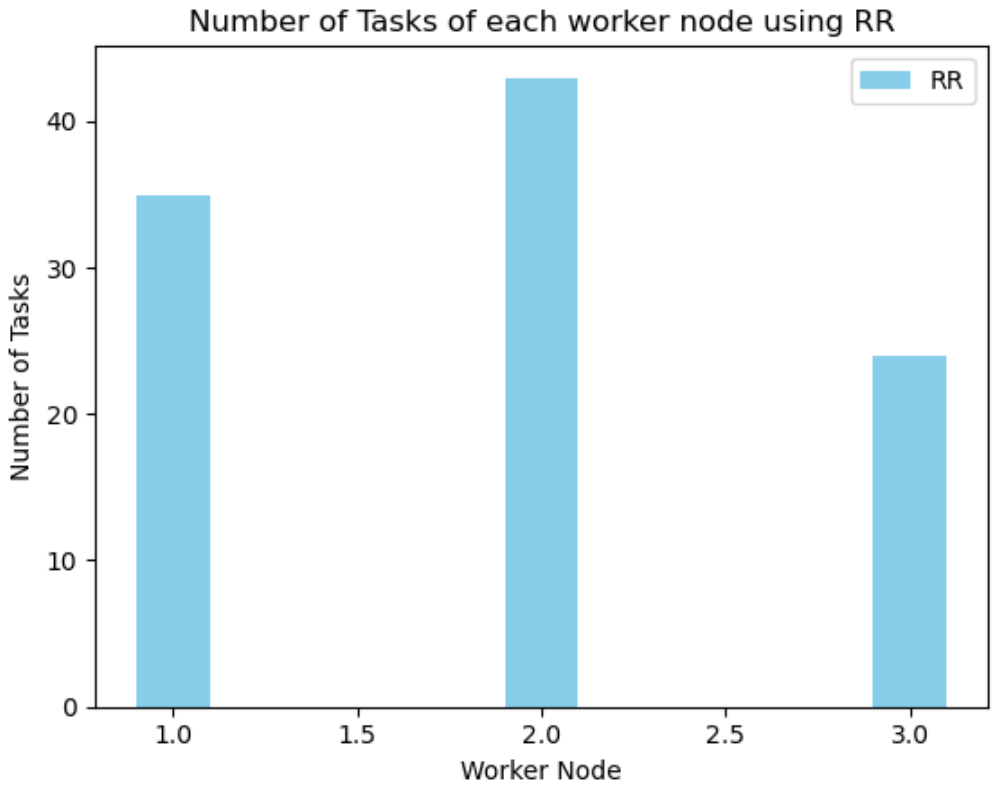
2. Random


Number of Task vs Timestamp for RANDOM with 25 Requests
Worker 1
Worker 2
Worker 3
Number of Tasks
Timestamp


Number of Tasks of each worker node using RANDOM
RANDOM
Number of Tasks
Worker Node
0
10
20
30
40
1.0
1.5
2.0
2.5
3.0

3. Round-Robin





As expected, the tasks were distributed more evenly in the Round Robin implementation. Random had the least mean and time for jobs.

Problems

One of the main problems we faced was with respect to the threading implementation and preventing race conditions that could occur. Finding an optimum method for logging that was convenient and self-sufficient was another hiccup that we faced.

Conclusion

We learnt about the communications between a master and worker by doing which we learnt socket programming and threading. We learnt about the various scheduling algorithms and their use cases. We mainly learnt about the workings and executing of a distributed system and its merits.

EVALUATIONS:

SNo	Name	SRN	Contribution (Individual)
1	Ashwin Alaparthi	PES1201802062	Implemented the connections between the master and workers and formed the main structure of the programs.
2	Raksha Ramesh	PES1201800345	Implemented the scheduling algorithms.
3	Raghav Roy	PES1201800342	Implemented logging for the jobs and tasks.
4	Ritik Hariani	PES1201801558	Implemented the analysis and plotted the graphs.

(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 →)	Done https://github.com/Raksha-Ramesh/BDProject
3.	Instructions for building and running the code. Your code must be usable out of the box.	Done