CS322:Big Data

Final Class Project Report

Project (FPL Analytics / YACS coding): YACS Coding Date: 1st December, 2020

SNo	Name	SRN	Class/Section
1	Harish S	PES1201801965	H
2	Suhas R.	PES1201800186	C
3	Vikshith Shetty	PES1201801555	C
4	Prateek Nayak	PES1201800054	C

Introduction

Yet Another Centralized Scheduler (YACS) is as the name suggests a centralized scheduler running on the Master of the distributed system that is responsible for scheduling tasks on the worker nodes. In this implementation, we present a YACS implementation that can schedule task on the workers using the Round Robin Scheduling, the Least Loaded Scheduling and Random Scheduling and compare how they stack against one another in terms of performance and load distribution

Related work

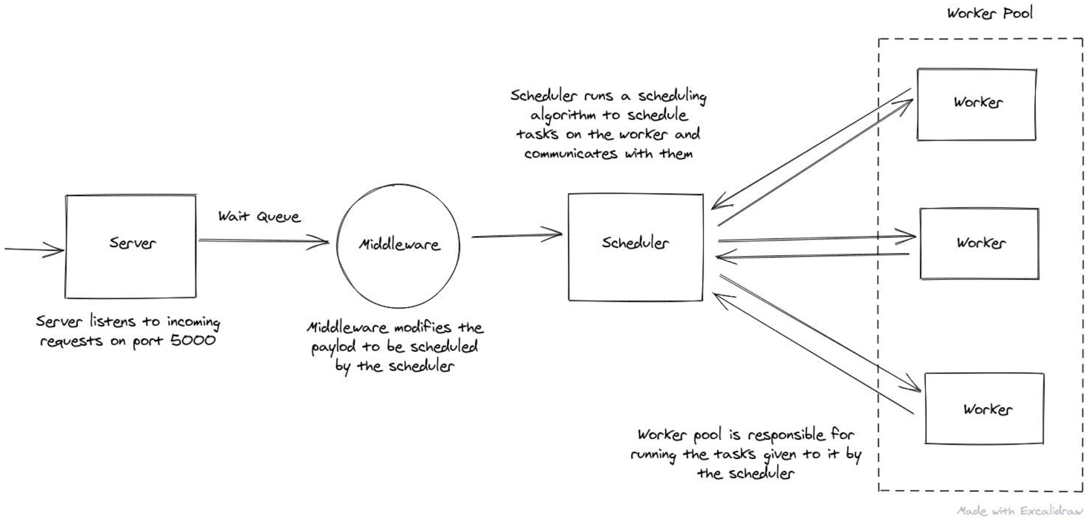
The idea of centralized scheduling is old but to understand the system design better, we referred the the sides of our Big Data course (UE18CS322) and the Jeff Dean's paper covering Map Reduce published in 2004

Design

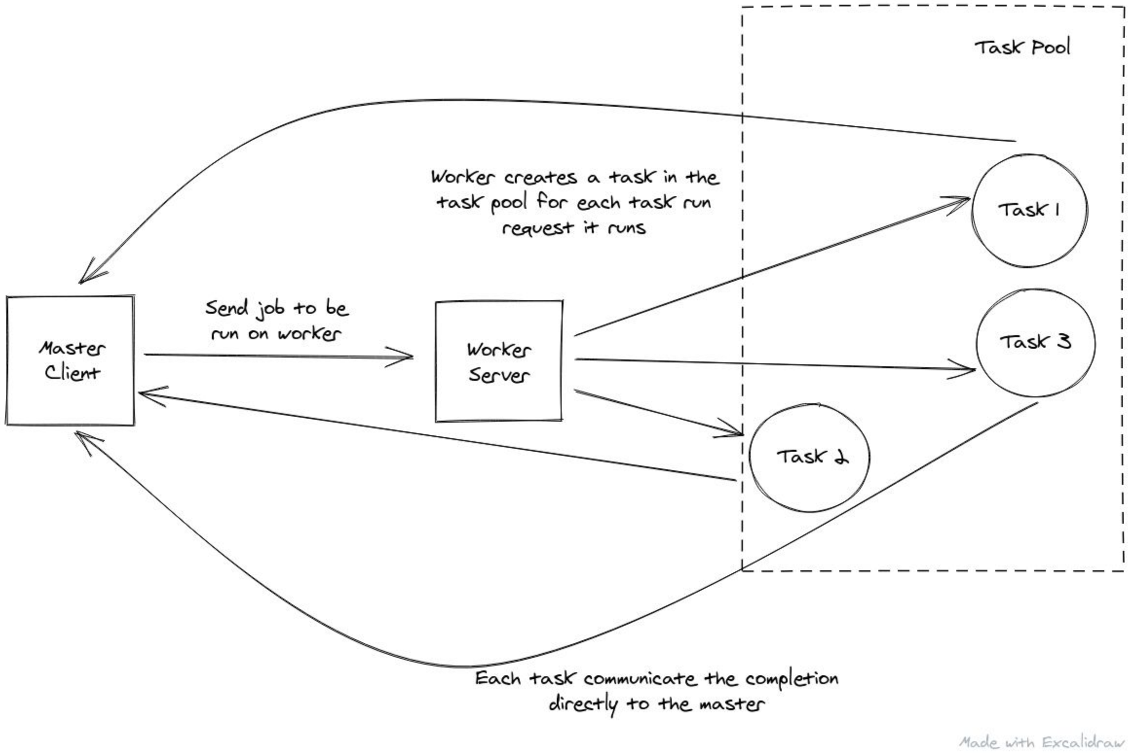
The design is very simple for this minimal implementation. There sits a master on port 5000 listening for any incoming requests. When a request is heard, the payload is retrieved and converted from a JSON string to a Python Dictionary. This dictionary is put on a wait queue which a middleware listens on. The middleware breaks down the request payload into scheduler payload and puts it on the scheduler queue. The scheduler listens to the scheduler queue for any jobs and schedules them on the workers as and when they come based on predefined algorithms:

- Random Scheduling
- Round Robin Scheduling
- Least Loaded Scheduling

The block diagram illustrates the flow of request within our YACS implementation


Worker Pool
Scheduler runs a scheduling algorithm to schedule tasks on the worker and communicates with them
Worker
Wait Queue
Server
Middleware
Scheduler
Worker
Server listens to incoming requests on port 5000
Middleware modifies the paylod to be scheduled by the scheduler
Worker
Worker pool is responsible for running the tasks given to it by the scheduler
Made with Excalidraw

Master Architecture


Task Pool
Worker creates a task in the task pool for each task run request it runs
Task 1
Send job to be run on worker
Master Client
Worker Server
Task 3
Task 2
Each task communicate the completion directly to the master
Made with Excalidraw

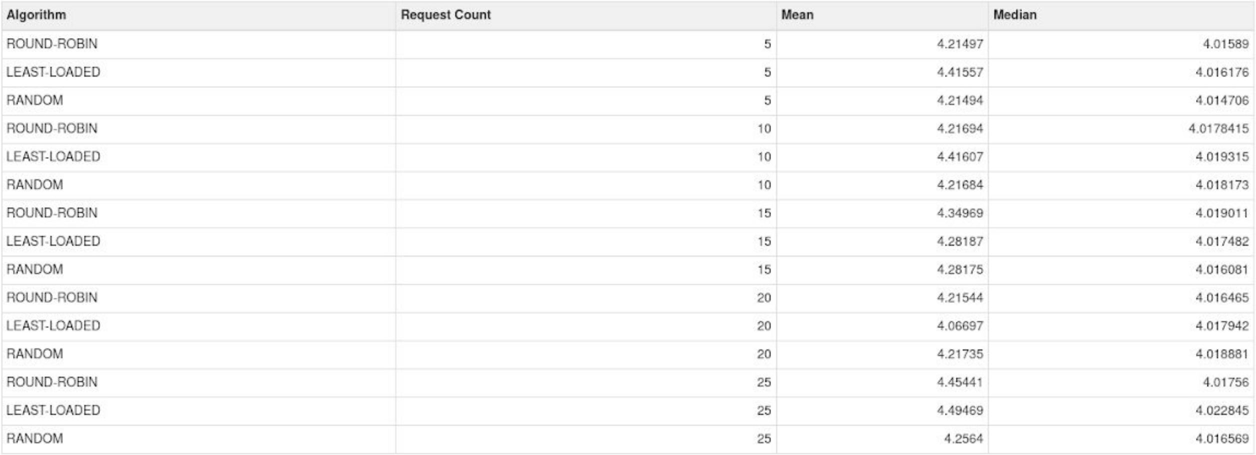
Worker Architecture

Results

The result we got are as follows

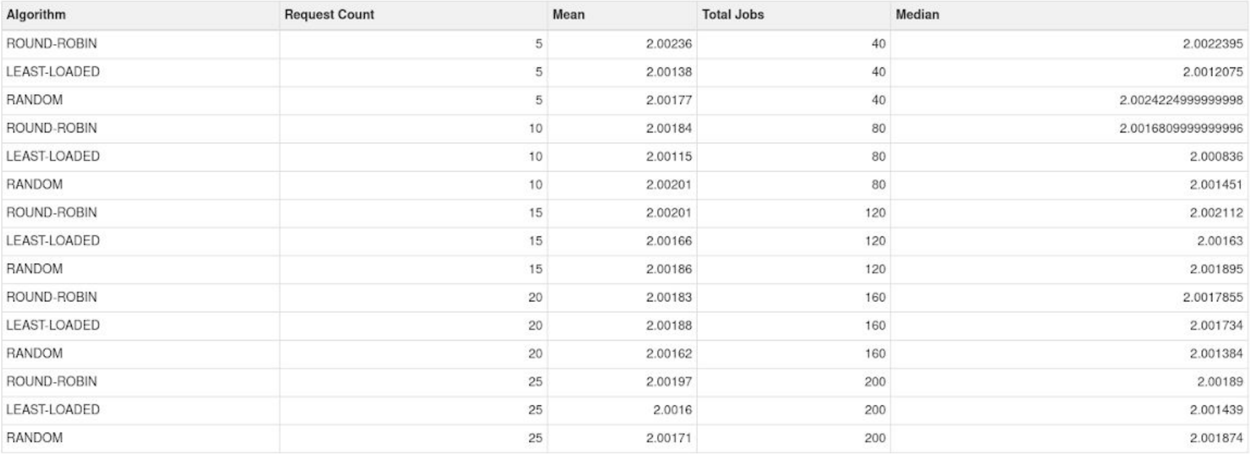
Result 1:

For the Master



Algorithm	Request Count	Mean	Median
ROUND-ROBIN	5	4.21497	4.01589
LEAST-LOADED	5	4.41557	4.016176
RANDOM	5	4.21494	4.014706
ROUND-ROBIN	10	4.21694	4.0178415
LEAST-LOADED	10	4.41607	4.019315
RANDOM	10	4.21684	4.018173
ROUND-ROBIN	15	4.34969	4.019011
LEAST-LOADED	15	4.28187	4.017482
RANDOM	15	4.28175	4.016081
ROUND-ROBIN	20	4.21544	4.016465
LEAST-LOADED	20	4.06697	4.017942
RANDOM	20	4.21735	4.018881
ROUND-ROBIN	25	4.45441	4.01756
LEAST-LOADED	25	4.49469	4.022845
RANDOM	25	4.2564	4.016569

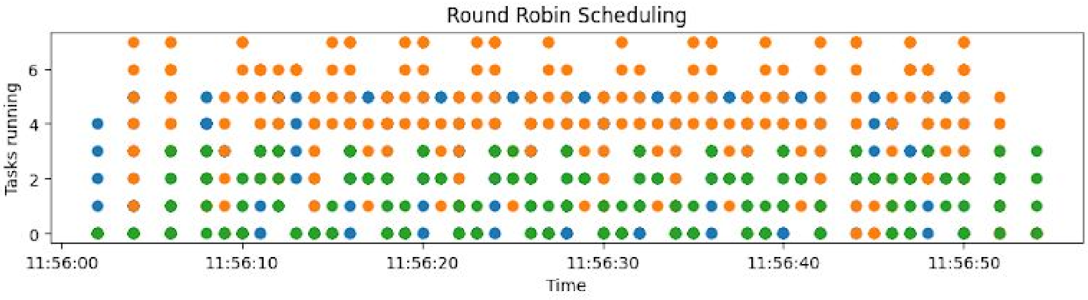
For the Worker we have

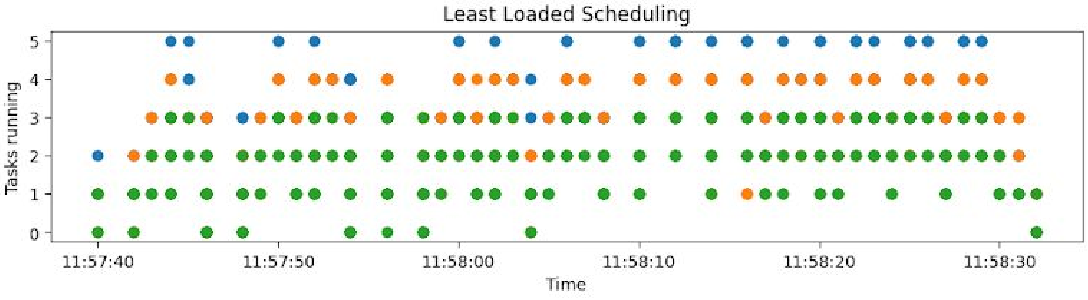


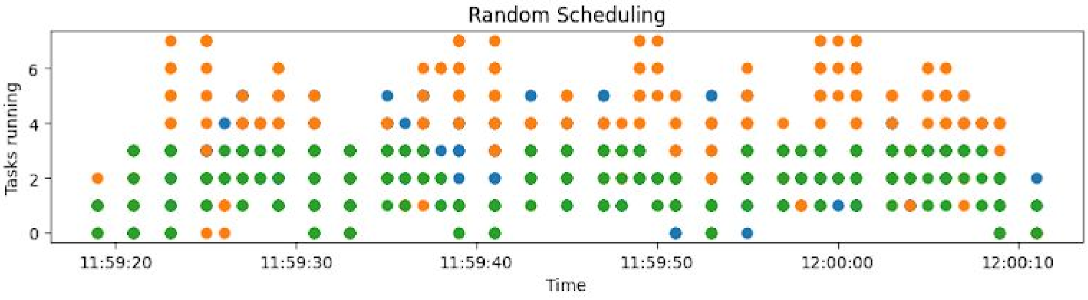
Algorithm	Request Count	Mean	Total Jobs	Median
ROUND-ROBIN	5	2.00236	40	2.0022395
LEAST-LOADED	5	2.00138	40	2.0012075
RANDOM	5	2.00177	40	2.0024224999999998
ROUND-ROBIN	10	2.00184	80	2.0016809999999996
LEAST-LOADED	10	2.00115	80	2.000836
RANDOM	10	2.00201	80	2.001451
ROUND-ROBIN	15	2.00201	120	2.002112
LEAST-LOADED	15	2.00166	120	2.00163
RANDOM	15	2.00186	120	2.001895
ROUND-ROBIN	20	2.00183	160	2.0017855
LEAST-LOADED	20	2.00188	160	2.001734
RANDOM	20	2.00162	160	2.001384
ROUND-ROBIN	25	2.00197	200	2.00189
LEAST-LOADED	25	2.0016	200	2.001439
RANDOM	25	2.00171	200	2.001874

Result 2

The plot of number of tasks running on each worker for each algorithms is as follows


Round Robin Scheduling


Least Loaded Scheduling


Random Scheduling

Problems

Some of the problems we faced were:

- Using Asyncio to handle requests asynchronously
- Evaluating correctness of algorithm
- Analyzing log file using regular expression

Conclusion

Some of learning we have taken away from the project are:

1. Asynchronous programming is better when it comes to event driven program
2. We learnt the basic syntax of asyncio and hacks to maintain and cleanup the task pool
3. We learnt the difficulties of asynchronous code with respect to race condition and data races they can cause
4. We briefly learnt how task scheduling works on a distributed system with a centralized master
5. Listening to class can help one undertake projects better.

EVALUATIONS:

SNo	Name	SRN	Contribution (Individual)
1	Harish S	PES1201801965	Analysis, Scheduling
2	Suhas R.	PES1201800186	Scripting, Worker Architecture
3	Vikshith Shetty	PES1201801555	Scripting, Scheduling
4	Prateek Nayak	PES1201800054	Server Architecture, Task Synchronization

(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.	