CS322:Big Data

Project Report

Yet Another Centralised Scheduler

Introduction

YACS: Yet Another Centralised Scheduler

This framework manages the resources and scheduling tasks among machines in the cluster. This framework consists of one master who makes scheduling decisions whereas the 3 workers where each executes tasks and informs the master on completion (execution). Other machines in the cluster have one worker process running on each of them. The master listens to job requests and dispatches the jobs with tasks to machines on the basis of scheduling algorithms.

Scheduling algorithms include

1. Random
2. Round-Robin
3. Least-Loaded

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.

Related work

Any background study material that you may have read and referenced

Websites and external links:

https://realpython.com/intro-to-python-threading/

https://docs.python.org/3/library/datetime.html

https://docs.python.org/3/library/statistics.html

StackOverflow

GeeksForGeeks

Referred books:

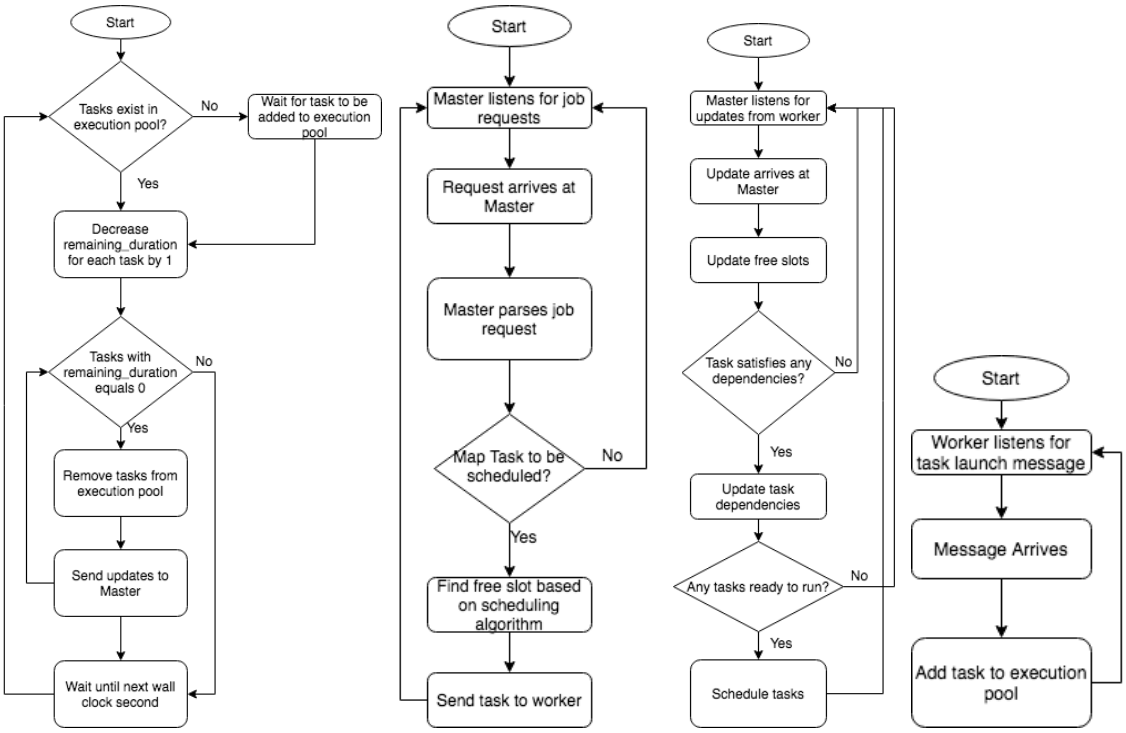
Operating System Concepts by Abraham Silberschatz, Peter B Galvin, Gerg Gagna. Core Python Applications Programming, 3rd edition by Wesley Chun.

Assumptions

We have assumed that there are only 3 workers that communicate to the master.

Design

2 threads of the master and worker listen to ports 5000 and 5001 respectively. The master file has an additional thread for scheduling used which helps in handling the incoming tasks efficiently by storing the tasks which are yet to be scheduled.



master.py	worker.py
Thread 1: master.py Master gets requests to port 5000. If map Task is: a. Scheduled: Master continues to listen to job requests b. Not Scheduled: Finds a free slot based on the 3 scheduling algorithms that we have and c. assigns the task to the worker. Thread 2: master.py 1. The master listens to the worker, for updates, which are given on the arrival of free slots. 2. Updates are given if the dependencies and requirements are satisfied by the task and if they are ready to schedule the task. 3. If not, the master still continues to listen for updates. Thread 3: scheduling: master.py	Thread 1: worker.py 1. This thread adds the task to the execution pool on the arrival of the task Thread 2: worker.py 1. The remaining time is reduced by 1, for all tasks in the execution pool.

1. The scheduling algorithm is decided based on the requirement of the user. 2. The task is assigned to one of the workers, decided based on the scheduler. 3. The task is then sent from the queue to the respective worker, and the no. of available slots is decremented accordingly.	2. If the time remaining is 0, for a task, it will be removed from this pool, and this is updated to the master.

Analysis:

Analysis of the various scheduling algorithms was done with the help of the log file generated from the master and worker files. The analysis was done for 10 requests each.

The algorithm used for analysis:

1. The required log file is opened
2. Each line of each file is read sequentially and stored as a list of strings.
3. For each String in the list, the String is split based on the spaces.
4. For each line, the 0th index of the list is checked for the following matches:
    a. TASKS:
        i) Started: The time is extracted, and stored as a dictionary.
        ii) Finished: The start time is extracted from the dictionary, using the ID. The time difference is then calculated.
    b. JOBS:
        i) Received: The time is extracted, and stored as a dictionary.
        ii) Finished: The start time is extracted from the dictionary, using the ID.
5. The time difference is then calculated, and the MEAN and MEDIAN of the same are computed.

Results

We obtained the following metrics for each algorithm, for 10 requests each.

Parameter	Round Robin	Least Loaded	Random
Task Mean	2.8346	2.375	2
Task Median	2	2	2
Job Mean	6.75	6.4	7.666
Job Median	7.0	7	8

We observe that the Job mean for Least Loaded scheduling algorithm is the least, while the Job mean for the Random scheduler is the highest.

The graphs plotted, is a histogram, of the time taken for each task/ job to complete execution.

MASTER

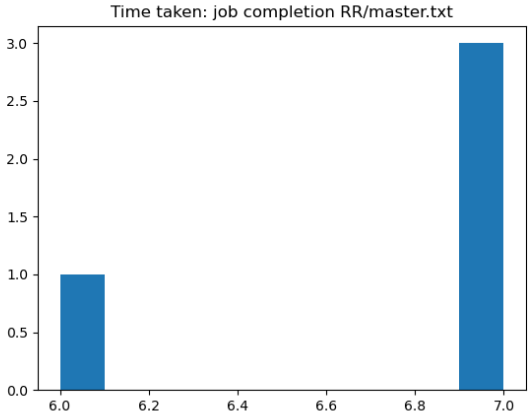


Fig 1: Round Robin

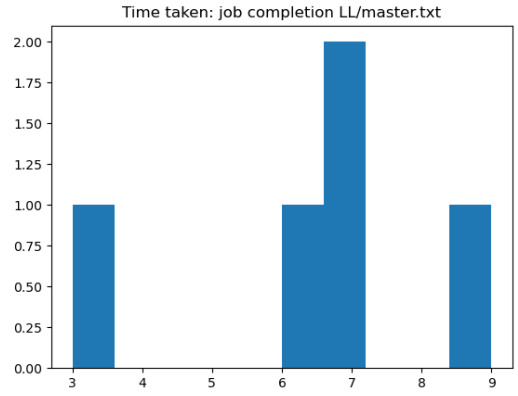


Fig 2: Least Loaded

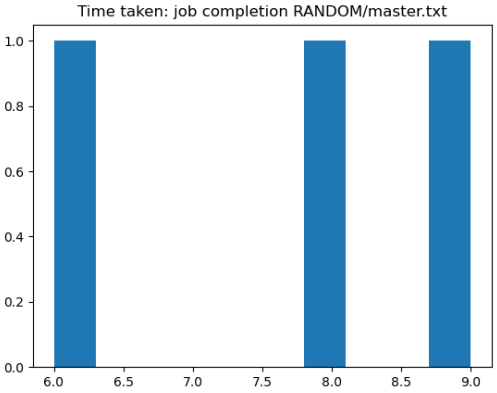


Fig 3: Random Scheduler

WORKER

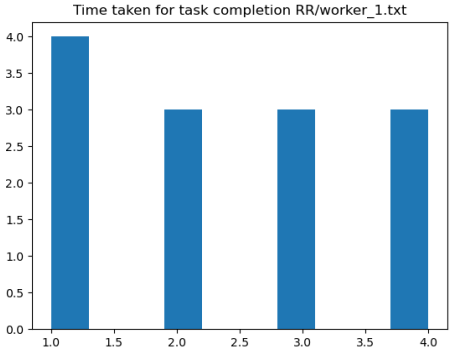


Fig 4: Round Robin

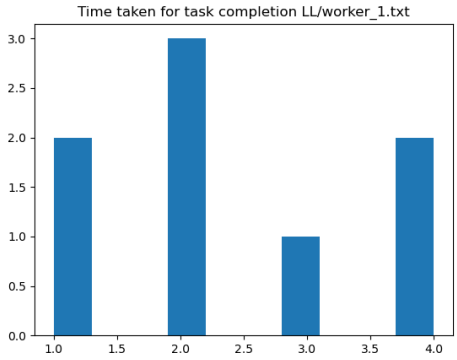


Fig 5: Least Loaded

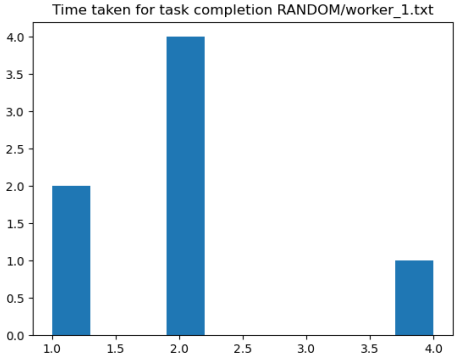


Fig 6: Random

STATISTICS

```
========== RESTART: C:\Users\Surabhi\Downloads\YACS\analysis.py ======
TASK_MEAN:
2.3846153846153846
TASK_MEDIAN:
2
RR/master.txt
JOB_MEAN:
6.75
JOB_MEDIAN:
7.0
------------------------------------------------------------------------
TASK_MEAN:
2.375
TASK_MEDIAN:
2.0
LL/master.txt
JOB_MEAN:
6.4
JOB_MEDIAN:
7
------------------------------------------------------------------------
TASK_MEAN:
2
TASK_MEDIAN:
2
RANDOM/master.txt
JOB_MEAN:
7.666666666666667
JOB_MEDIAN:
8
------------------------------------------------------------------------
```

Problems

The problems we faced were:

1. The right choice of data structure for every application/class.
2. Deciding when to acquire and release locks to avoid deadlocks and race conditions.
3. We had issues in figuring out the right method to log into the file, tried using regex initially, then figured out an easier method to handle the issue, which we finally implemented.
4. Due to deadlocks, reducers weren't being assigned to the workers.
5. Due to various runtime and compile-time errors, the connection was not being established between different machines.

Conclusion

What was your main learning from this project?

- This mini-project served as a great learning experience, as we understood the concepts of both YARN and Operating Systems.
- Understood the basics of the various scheduling algorithms, and how they are internally implemented.
- Better understanding of socket programming.