CS322:Big Data

Final Class Project Report

Project (FPL Analytics / YACS coding): YACS coding Date: 01-12-2020

SNo	Name	SRN	Class/Section
1	Sriram Subramanian	PES1201800655	5F
2	Pavan A	PES1201800157	5I
3	Kaustubh Milind Kulkarni	PES1201801956	5J
4	Srish Srinivasan	PES1201800051	5G

Introduction

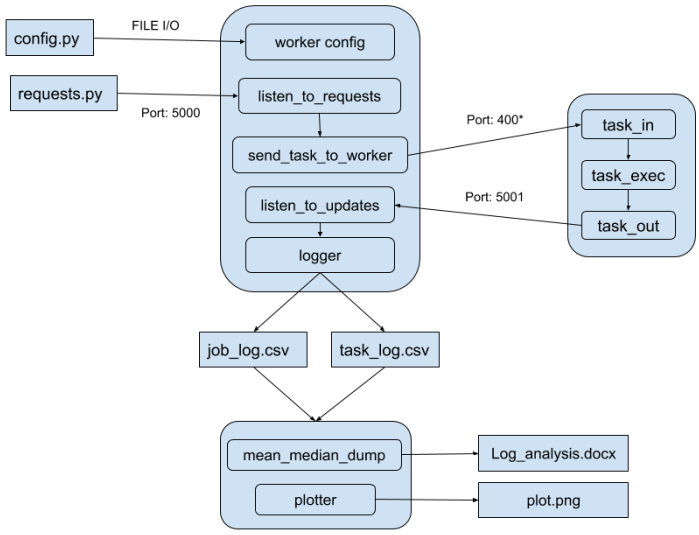
In the modern data driven world, a huge amount of workload consisting of several jobs from a variety of different applications are incapable of running on a single machine. Hence it is required to configure a cluster of interconnected computing machines. This gives us the luxury of processing tasks within each of these jobs parallely. Parallel processing significantly reduces the throughput hence, improving the efficiency and the overall computing capacity of a cluster. All of this is possible with the help of a scheduling framework which efficiently manages and allocates resources to the various tasks that need to be executed. The scheduling framework has two major functions:

1. It allocates the resources to a task that is waiting to be processed in the queue.
2. Once a task has been executed, the scheduling framework must free the resources allocated for the task.

Related work

Multithreading - https://www.geeksforgeeks.org/multithreading-python-set-1/
Basics of socket programming were learnt from the 5th semester Computer Networks course. Implementation and overall design aspect of this project were taken from the guidelines provided to us in the document. Plotting line graphs for depicting the allocation of tasks to their respective workers based on the different scheduling algorithms that were employed - https://www.geeksforgeeks.org/graph-plotting-in-python-set-1/

Design



- Each individual worker, task and job was modeled as a class object.
- All necessary functions are modularised for better understanding and debugging.
- All 3 workers are started first, waiting for tasks to be sent from the master on their respective port numbers i.e 4000, 4001 and 4002. (the block diagram only shows 1 worker for the sake of simplicity) after which the master and request generator fies are launched.

Master

3 threads are used in total:

1. One thread listens to job requests from requests.py, parses the requests and sends all map tasks of each incoming job to the workers.
2. One thread listens to task completion updates from the workers, checks if map tasks of the job have been completed or the entire job has been completed. If only the map tasks of a job are completed then the job is prepped to have its reduce tasks sent to the workers.
3. This third thread sends all reduce tasks of jobs whose map tasks have been completed to the workers.

Shared variables used:

1. One variable to keep track of all jobs.
2. One variable to keep track of all workers.
3. One variable to keep track of indices for scheduling.

Locks/Semaphores used:

1. One semaphore to lock the job tracker.
2. One semaphore to lock the workers variable.
3. The indices used in scheduling are also locked.

Worker

1. One thread keeps listening to the tasks sent by the master and parses each incoming task.
2. Every parsed task is assigned a new thread for further execution.
3. In each newly assigned thread, the task execution is simulated for the entire duration.
4. Once the task execution is 'complete', it is marked as done and the thread responsible for said task sends an update to the master.

Scheduling Algorithms

1. RANDOM (Random)

   Randomly choose a worker and check if it has a free slot. If yes, select said worker for the queued task. If no, choose another worker randomly and repeat the process.

2. RR (Round Robin)

   Keep track of the last selected worker. Go in a circular fashion checking for free slots in each encountered worker. If a worker with a free slot is encountered, this worker is selected.

3. LL (Least Loaded)

   Workers are constantly sorted based on the number of free slots. The worker with the most free slots is selected for the queued task. If no free worker is available, there is a second's pause and the workers are sorted again and this process is repeated.

Analysis of logs

1. The log_analyser parses the job logs and task logs dumped by the master after completion. It computes the mean and median completion times for the tasks and the jobs for each of the scheduling algorithms.

2. Once the records are sorted, it goes on to compute the mean and median completion times.
3. After having computed the mean and median completion times, a bar graph depicting the differences in the mean and median job completion times for each of the 3 algorithms is plotted. It also plots bar graphs depicting the mean and median task completion times for each of the 3 algorithms.
4. The log_analyser then plots the tasks scheduled on each worker against the arrival times of the tasks for each of the scheduling algorithms as separate line plots.
5. All the above mentioned plots have been copied onto a docx file for easy viewing.

Results

Jobs were successfully received and tasks were successfully sent to the respective workers based on the scheduling algorithms. More slots resulted in faster throughput of job executions.

On running the log_analyser, the mean and median task and job completion times for each of the scheduling algorithms was computed and the results are mentioned as follows. (The number of requests was set to 50)

RANDOM

- Mean Job Completion time: 6.8376 seconds
- Median Job Completion time: 7.1198 seconds
- Mean Task Completion time: 2.5503 seconds
- Median Task Completion time: 3.0007 seconds

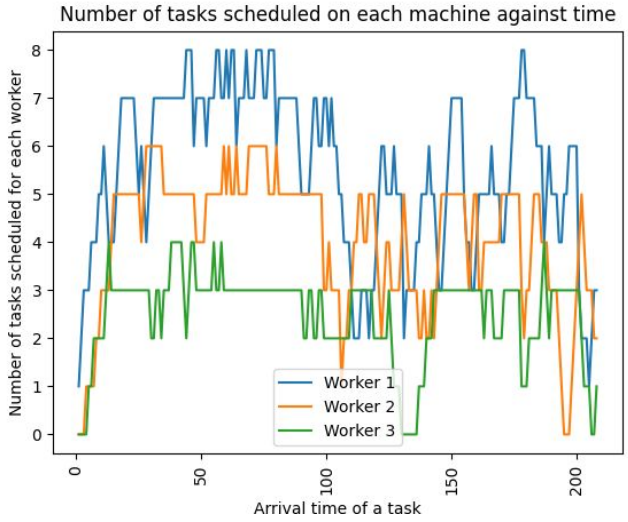
ROUND ROBIN

- Mean Job Completion time: 5.8216 seconds
- Median Job Completion time: 6.1146 seconds
- Mean Task Completion time: 2.3112 seconds
- Median Task Completion time: 2.0023 seconds

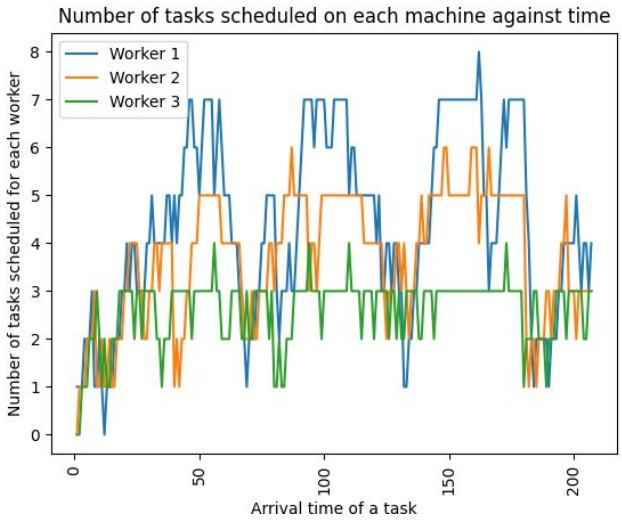
LEAST LOADED

- Mean Job Completion time: 6.4280 seconds
- Median Job Completion time: 6.3062 seconds
- Mean Task Completion time: 2.5423 seconds
- Median Task Completion time: 2.5022 seconds

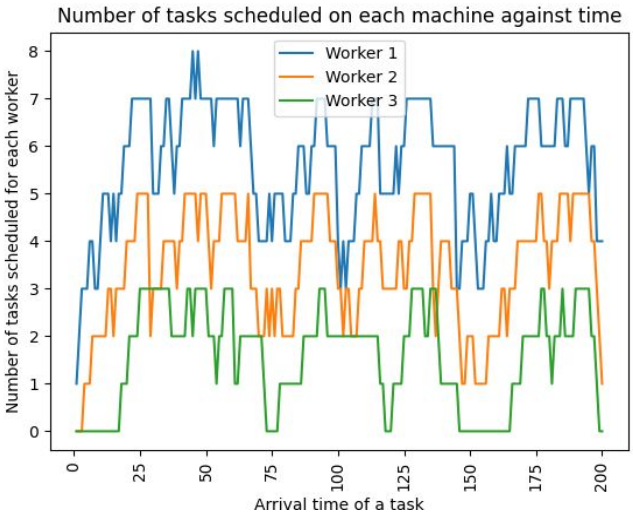
A graph depicting the distribution of tasks across workers at every point in time when a task arrives has also been plotted for each of the three scheduling algorithms.



RANDOM



ROUND ROBIN



LEAST LOADED

Note :

For a set of 50 randomly generated job requests, it is observed that the Round Robin scheduling algorithm has the least mean and median job completion times and the Random scheduling algorithm has the largest mean and median job completion times.

Problems

One of the major problems we faced was that, while receiving task updates from the worker on port 5001, which was handled by a seperate thread, we tried to immediately check if all map tasks were done and start scheduling reduce tasks in the same thread. Hence when the thread running in the listen_to_updates started scheduling the map tasks for other jobs, there was a deadlock stage wherein, the algorithm got caught in an infinite loop trying to find a worker with free slot, at the same time when no updation of any sort was happening to the worker slot after receiving a task update from the worker.

Since the program design was extremely modular and readable, detailed debugging of every task sent and updated, we figured out this was the error.

This was solved by using a seperate thread to schedule the reduce tasks, which were originally appended in the listen_to_updates function after checking that all the map tasks of that particular job were done.

Conclusion

- We learnt how to practically implement threads and lock critical sections using locks and semaphores.
- We learnt more about socket programming and its practical implementation.
- We got a glimpse of the overall working of centralised job schedulers.
- We got more experience doing object oriented programming.
- We learnt how scheduling algorithms can prove to be effective for time and resource management in a distributed framework.
- Last but not least, everyone poured in their contributions to make this project a successful one in the end.

EVALUATIONS:

SNo	Name	SRN	Contribution (Individual)
1	Pavan A	PES1201800157	- Design of classes for representing jobs, workers and tasks in master and worker. - Communication from master to worker - Handling of sockets - Log output contents
2	Sriram Subramanian	PES1201800655	- Design and working of threads - Handling of locks and semaphores - Functioning of worker - Structure of code
3	Kaustubh Milind Kulkarni	PES1201801956	- Design of scheduling algorithms to allocate tasks to certain workers. - Simulation of execution of tasks in worker - Worker threading - Structure of code
4	Srish Srinivasan	PES1201800051	- Communication from worker to master - Logging of jobs - Detailed log analysis

(Leave this for the faculty)

Date	Evaluator	Comments	Score

CHECKLIST:

SNo	Item	Status
1.	Source code documented	Yes
2.	Source code uploaded to GitHub – (access link for the same, to be added in status)	https://GitHub.com/KulkarniKaustubh/BigData/tree/master/BD_YACS/submittables
3.	Instructions for building and running the code. Your code must be usable out of the box.	Readme is present in the above GitHub repo link for running the code.