CS322: Big Data

Final Class Project Report

Project (FPL Analytics / YACS coding): YACS Coding **Date: 10/12/2020**

SNo	Name	SRN	Class/Section
1	R. Tharun Raj	PES1201801618	B
2	G Deepank	PES1201800395	B
3	Darshan A Pirgal	PES1201801832	B
4	K. Sreesh Reddy	PES1201801580	E

Introduction

The topic of our project is Yet Another Centralized Scheduler or YACS. In this project we try to simulate a framework in which there exists a central entity known as the Master and a number of Machines to carry out the tasks. The responsibility of the Master is to make used of the different resources available by the workers to schedule jobs from various applications and balance the load. In our case, we have 3 (simulated) machines with worker processes with varying number of slots for the processing of tasks. This job scheduling is implemented using various algorithms such as Random, Round-Robin and Least load algorithms. The main job which we are focused on is the Map-Reduce job, which contains two sets of tasks, the mappers and reducers. The language used for carrying out this framework is Python and various data types such as dictionaries and lists were also used in the project.

Related work

Various websites were referred to for the implementation of the logic and the debugging of the code. Few the websites are referred to below:

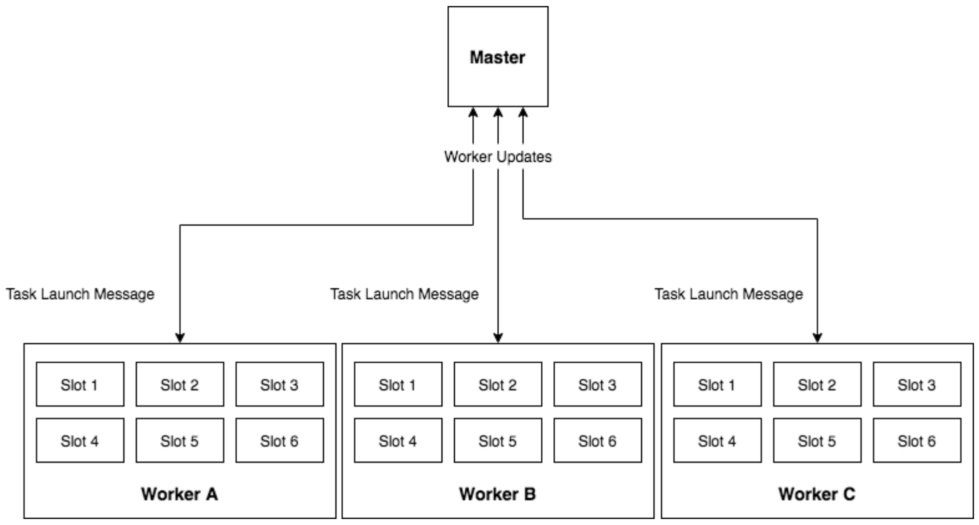
1. https://www.geeksforgeeks.org/read-json-file-using-python/
2. https://stackoverflow.com/questions/36059194/what-is-the-difference-between-json-dump-and-json-dumps-in-python
3. https://www.geeksforgeeks.org/python-select-random-value-from-a-list/
4. https://www.geeksforgeeks.org/multithreading-python-set-1/
5. https://www.programiz.com/python-programming/dictionary-comprehension

Design

The overall design of the system contains a master and three workers. These workers have variable number of slots and the port number through which they talk to the master is also variable. They were predefined in a file called config.json. The content of the files are loaded onto the master in the form of a dictionary called slotsinfo. This dictionary is used to keep tabs the content filled in each of the workers. That is, find the number of resources or slots filled at a particular time. The key of the dictionary is the id of the worker and the value contains list with the first element being a list of the slots, which are initialized to the value None and the first element being the port number of the worker.

Slotsinfo = {worker_id: [[slot1, slot2, slot3,...], port_no]}

Then various functions were used for the initialization of the Master. The connections of the sockets are as follows. The master listened to the requests for job requests through the port 5000 and to the workers for updates on the tasks through port 5001. The three workers listened to the Master for task launch messages through ports 4000, 4001 and 4002. Some functions were implemented with respect to the dictionary such as checking whether there existed an empty slot in the list or not and removing the task from the slot once the simulation of the task had been completed.

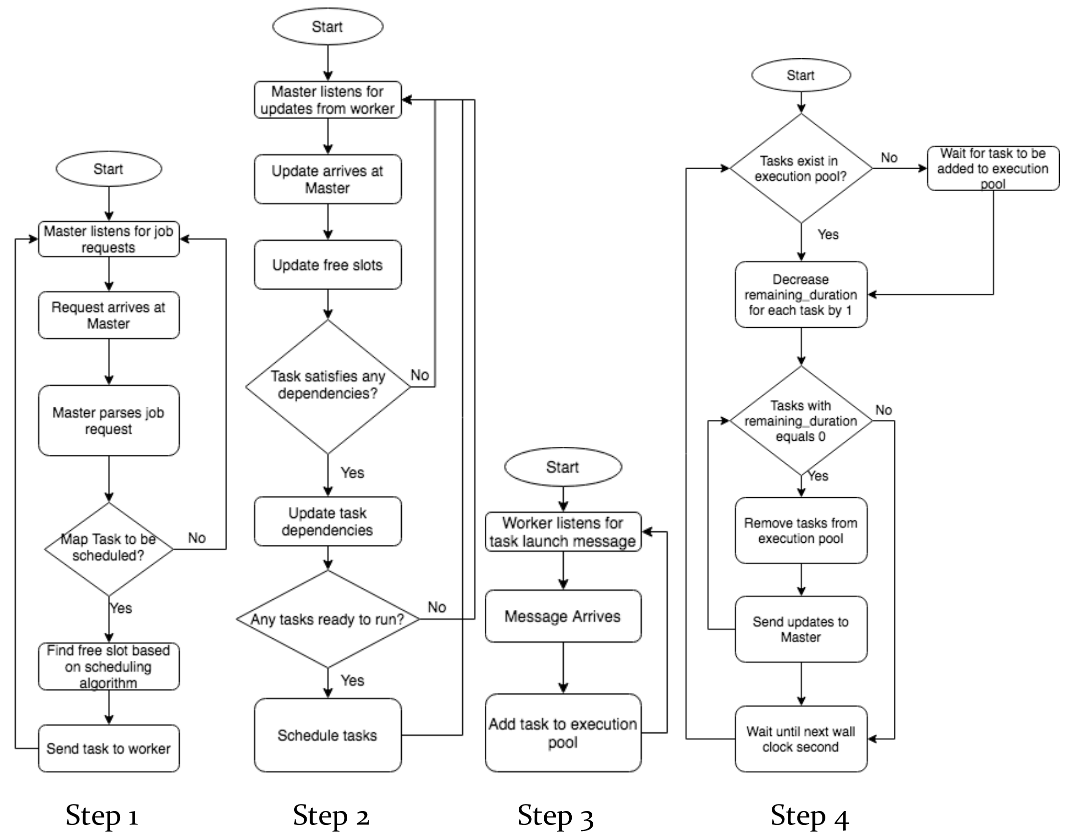


For the simulation of the insertion of the tasks in the slots, we use a queue (list) in which the tasks are waiting. The way in which the function in the Master stores the job requests is a temporary list, in which a dictionary full of mapper tasks is appended.

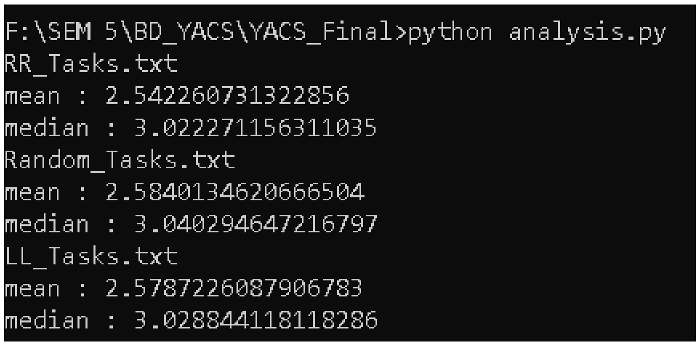
Task_queue = [(task_id, duration),..]

Then the dictionary full of reducer tasks is appended. This temporary list is then used as a value to another dictionary whose key will be the job id. This dictionary is used to check the dependencies between the mapper-reducer tasks. Since the reducer task of a job cannot be executed unless all the mapper tasks of the same mentioned job id are done. All the mappers are appended to the task queue once for execution. After the tasks have reached the slots and completed their executions, the mapper tasks are deleted from the queue and the dictionary. Then the conditions are checked to see if the dictionary containing the mappers of that job id have been deleted. The reducer tasks are then inserted to the beginning of the queue to finish the whole job as soon as possible. The assigning of the tasks to their slots are done using the algorithms. The round robin was implemented by checking in a cyclic manner when an empty slot is encountered first. After that, the slot is filled with the first value in the queue. For the random algorithm, any integer from 1 to 3 was chosen at random using the random module. This worker id was used to fill the empty slot in that worker with the task present in queue. Our implementation of the least load algorithm was to find the worker with the maximum number of empty slots. If nothing is found, then wait until 1 second and repeat the tasks again. Threading was used to carry out multiple parts of the processes parallelly. Locks were used to prevent race conditions. In the worker.py file, only the simulation of the execution of the task was done by using time.sleep().

For our analysis we logged the time taken to complete each task and the completion of each job and storing these in a json format in txt files. These were later used to find out the mean and the median of each algorithm. We ask the user to input the algorithm and based on that, txt files according to the algorithms are made and the details are stored in the respective files. For plotting the matplotlib library was used and the plotting was done for number of slots filled against time in seconds for 100 job requests. This was done using the plt.plot() function and the plot contained the slots for all the three workers. For calculating the mean and the median, the json type were converted to a list and then into a numpy array. Using this numpy array, calculation of mean and median was straightforward using the np.mean() and the np.median() function instead of calculating the middlemost element for median and the sum of the whole list for the mean.


Start
Master listens for job requests
Request arrives at Master
Master parses job request
Map Task to be scheduled?
No
Yes
Find free slot based on scheduling algorithm
Send task to worker
Step 1
Start
Master listens for updates from worker
Update arrives at Master
Update free slots
Task satisfies any dependencies?
No
Yes
Update task dependencies
Any tasks ready to run?
No
Yes
Schedule tasks
Step 2
Start
Worker listens for task launch message
Message Arrives
Add task to execution pool
Step 3
Start
Tasks exist in execution pool?
No
Wait for task to be added to execution pool
Yes
Decrease remaining_duration for each task by 1
Tasks with remaining_duration equals 0
No
Yes
Remove tasks from execution pool
Send updates to Master
Wait until next wall clock second
Step 4

Results



```
F:\SEM 5\BD_YACS\YACS_Final>python analysis.py
RR_Tasks.txt
mean : 2.542260731322856
median : 3.022271156311035
Random_Tasks.txt
mean : 2.5840134620666504
median : 3.040294647216797
LL_Tasks.txt
mean : 2.5787226087906783
median : 3.028844118118286
```

The results obtained were collected for 10 job requests provided to each algorithm. The RR_Tasks.txt represents the mean and median for the Round Robin Algorithm, similarly the LL_Tasks.txt represents for the Least Load, and Random_Tasks.txt for the Random algorithm. It can be observed from the above picture that the Round Robin algorithm gave the best results in terms of performance measured using time. Although the means and the medians do not differ by much, Round Robin trumps the other two by a pretty short margin.

Problems

There was plenty of debugging which had to be done for the smooth workflow of the project. Some of the problems were to extinguish the race conditions and the deadlocks which were very hard to detect. This was taken care of using mutex lock mechanisms to make any process mutually exclusive so that other threads were not interfering at the same time the current process was making changes. The algorithms too were understood in theory but were challenging to implement. The outputs of the worker slots were constantly monitored to verify the correctness of our algorithms. The logging part was also corrected by comparing the times stored in our logs to the time provided in the request commands. But the main obstacle we had to overcome was deciding on the data structures to store the jobs, tasks and the workers to verify the dependencies and the slot verification.

Conclusion

This final project was an interesting and enriching experience since we had to make use of various concepts in the CS field such as socket programming from Computer Networks and multithreading from Operating Systems. We learned how a scheduling framework operates hands-on. The working of the different scheduling algorithms such as Random, Round Robin, and Least Load were also visualized in the filling of the slots. We also encountered the necessity of knowing which algorithm to implement for the best load distribution of the jobs by the Centralized Scheduler.

EVALUATIONS:

SNo	Name	SRN	Contribution (Individual)
1	R. Tharun Raj	PES1201801618	Worked on master, worker
2	G Deepank	PES1201800395	Worked on algorithms, plotting
3	Darshan A Pirgal	PES1201801832	Worked on worker, plotting
4	K. Sreesh Reddy	PES1201801580	Worked on master, debugging

(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.	