CS322:Big Data

Final Class Project Report

Project (FPL Analytics / YACS coding): YACS Date:10/12.2020

SNo	Name	SRN	Class/Section
1	Aditya G Burli	PES1201800034	J
2	Sidharth Pathak	PES1201800142	C
3	Vishnu A S	PES1201800192	J
4	Nishant Tripathy	PES1201801296	C

Introduction

This project is called Yet Another Centralized Scheduler. This project aims to centrally control the scheduling of Map and reduce jobs by the master to its n number of workers. In this case, one master and three workers are used. The Master process makes scheduling decisions and accordingly passes off tasks to the Worker processes and informs the Master when a task completes its execution.
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.In this case, we will work with one master and three workers. The scheduling framework receives job requests and launches the tasks in the jobs on machines in the cluster. Different scheduling algorithms like round-robin, random, least-loaded can be used. The model aims to simulate the central control of task scheduling done by the master and the job execution of the workers as well.

Related work

- Scheduling algorithms implementation
- Race condition prevention using locks
- socket programming
- Multithreading in python

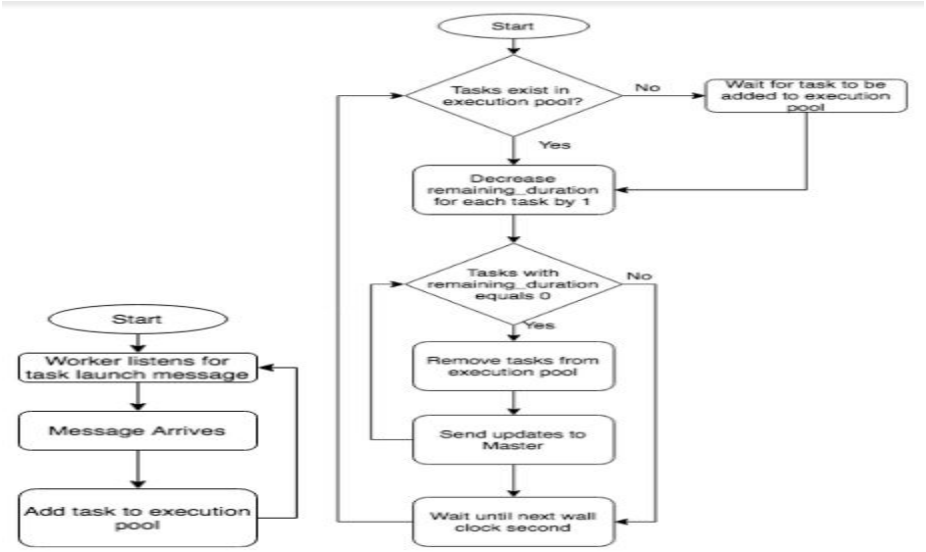
Design

For the worker side, the 'workerID' and 'portno.' are extracted from the command line. The locks for the threads are initialized. A worker class was created where all the worker parameters(workerID, portno., noSlots, avaSlots and a dictionary- slotJobs) are initialized. Another class 'TCPServer' is created which listens for tasks to be executed through a TCP socket which is initialized in the init method.

The 'startserver' function within the class first listens to the master for job requests. It then accesses job parameters by decoding the message and loading it into json. A lock is acquired to perform the tasks of consuming an available slot (avaSlot) and changing the slot parameters(duration, task_id) in the slotJobs dictionary(the execution pool). The lock is released when the job is initiated. The timestamps are sent to a file for analysis.This function is done by one thread.

The other thread is assigned to the 'send_request' function which performs the task of sending a message to the master using the 5001 port. The slot variables upon completion of tasks are updated. The job is added to the completed list and a slot is made available in

'avaSlots' and slot status is made 'available'. The lock is acquired to perform the above task. The timestamps are sent for analysis and the message to the master is sent through portno. 5001. Below flow chart explains and procedure.

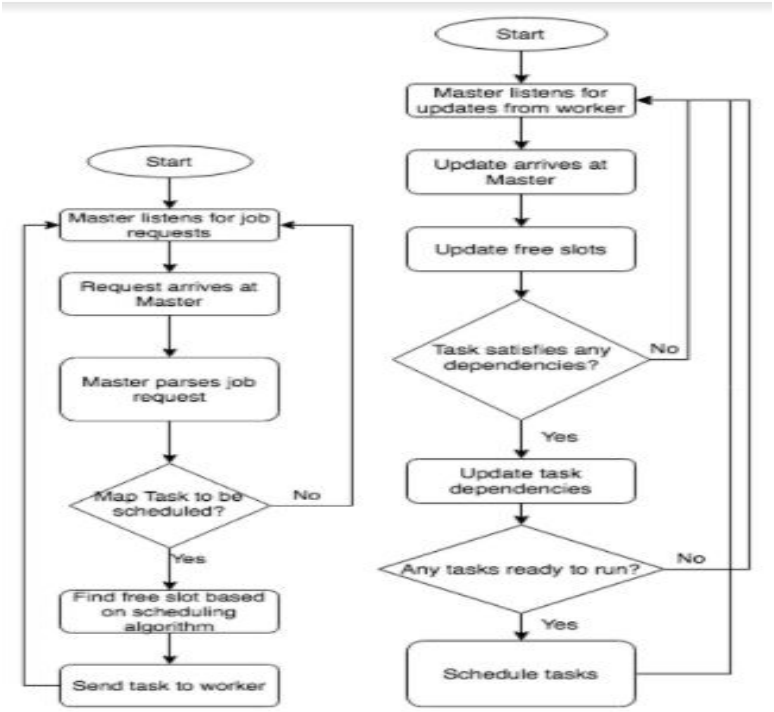


In the design of the master, a different worker class is defined similar to the one in the worker module with its worker parameters. Three different scheduling algorithms are defined in separate functions- Random- which picks a worker at random and assigns the job if it has an available slot, Round Robin -which sorts the worker based on ID and checks for available slot and Least Loaded scheduling- which chooses the worker with most available slots. They are all performed using locks to avoid concurrency.

Another class called 'TCPServer' is defined to perform the main task of receiving and executing (delegating to workers) the job requests. It initializes the parameters- port, ip, socket, jobQueue and also starts listening for incoming requests. The 'startserver' function does the following. First, the job is received, appended to 'jobQueue' and is logged to a file, The message is then decoded and loaded to json. The jobs are sent to workers. Map tasks are first appended(task_id, duration,compornot,sentornot) followed by reduce tasks. The messages from the workers are received. Completed map tasks are updated to the global job list.

Reducer tasks are started using a separate function 'startReducer'. It first checks for any remaining reduce tasks in the 'globalJobContent'. A worker is found based on the scheduling algorithms. When the task is sent then the value is changed to true. 'send_request' function is called.

Another function performs the task of sending requests to the workers. Lock is acquired to change job slots. The slots in the workers are updated and the lock is released. The job is sent to the specified worker and it is logged. Separate threads are created for receiving and processing job requests and getting the message sent by workers and working on them. Below flow chart explains the procedure.



Results

The scheduler was run on a command line interface by running the master file which received the configuration file for the workers, with a certain scheduling algorithm. The requests were generated and their corresponding tasks were completed with the following output received in the terminal -

Running the requests file-

```
D:\fpl_assignment>python requests.py 10
interval:  0
 Job request : {'job_id': '0', 'map_tasks': [{'task_id': '0_M0', 'duration': 3}, {'task_id': '0_M1', 'duration': 1}], 'reduce_tasks': [{'task_id': '0_R0', 'duration'
: 4}]}
interval:  0.104248459482862
 Job request : {'job_id': '1', 'map_tasks': [{'task_id': '1_M0', 'duration': 4}, {'task_id': '1_M1', 'duration': 2}, {'task_id': '1_M2', 'duration': 1}, {'task_id':
'1_M3', 'duration': 4}], 'reduce_tasks': [{'task_id': '1_R0', 'duration': 1}, {'task_id': '1_R1', 'duration': 4}]}
interval:  4.317667187169543
 Job request : {'job_id': '2', 'map_tasks': [{'task_id': '2_M0', 'duration': 4}, {'task_id': '2_M1', 'duration': 3}, {'task_id': '2_M2', 'duration': 2}, {'task_id':
'2_M3', 'duration': 1}], 'reduce_tasks': [{'task_id': '2_R0', 'duration': 3}]}
interval:  0.09655507693360216
 Job request : {'job_id': '3', 'map_tasks': [{'task_id': '3_M0', 'duration': 1}], 'reduce_tasks': [{'task_id': '3_R0', 'duration': 1}]}
interval:  2.6553090649862687
 Job request : {'job_id': '4', 'map_tasks': [{'task_id': '4_M0', 'duration': 4}, {'task_id': '4_M1', 'duration': 1}, {'task_id': '4_M2', 'duration': 2}, {'task_id':
', 'duration': 2}, {'task_id': '4_M3', 'duration': 4}], 'reduce_tasks': [{'task_id': '4_R0', 'duration': 4}]}
interval:  3.7557813264214
 Job request : {'job_id': '5', 'map_tasks': [{'task_id': '5_M0', 'duration': 2}, {'task_id': '5_M1', 'duration': 2}], 'reduce_tasks': [{'task_id': '5_R0', 'duration'
: 4}, {'task_id': '5_R1', 'duration': 3}]}
interval:  0.976612839196545
 Job request : {'job_id': '6', 'map_tasks': [{'task_id': '6_M0', 'duration': 1}], 'reduce_tasks': [{'task_id': '6_R0', 'duration': 2}, {'task_id': '6_R1', 'duration'
: 3}]}
interval:  0.6645861705693596
 Job request : {'job_id': '7', 'map_tasks': [{'task_id': '7_M0', 'duration': 4}], 'reduce_tasks': [{'task_id': '7_R0', 'duration': 2}, {'task_id': '7_R1', 'duration'
: 3}]}
interval:  3.5520111084335366
 Job request : {'job_id': '8', 'map_tasks': [{'task_id': '8_M0', 'duration': 4}, {'task_id': '8_M1', 'duration': 1}, {'task_id': '8_M2', 'duration': 4}], 'reduce_tas
ks': [{'task_id': '8_R0', 'duration': 2}, {'task_id': '8_R1', 'duration': 4}]}
interval:  1.7849187532907975
 Job request : {'job_id': '9', 'map_tasks': [{'task_id': '9_M0', 'duration': 3}, {'task_id': '9_M1', 'duration': 3}, {'task_id': '9_M2', 'duration': 1}, {'task_id':
'9_M3', 'duration': 4}], 'reduce_tasks': [{'task_id': '9_R0', 'duration': 1}]}
```

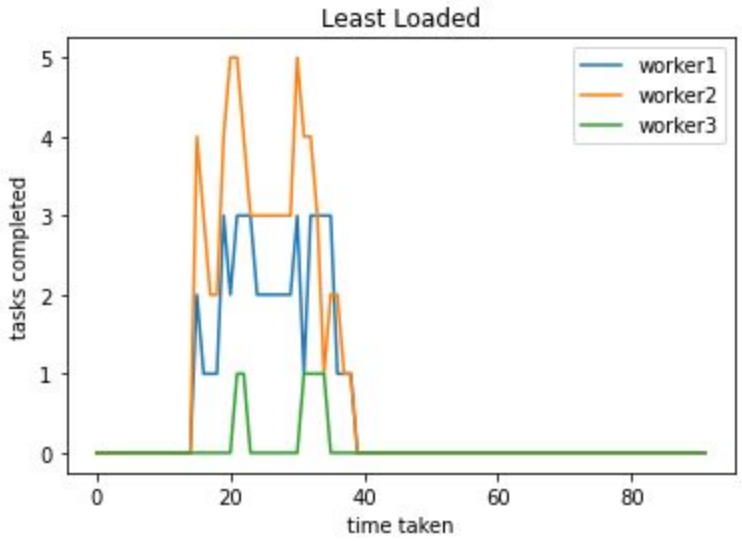
Running the master file-

```
D:\fpl_assignment>python git_master.py "config.json" RR
waiting for connection at port number :5000
started thread <Thread(Thread-1, started 9580)>
waiting for connection at port number :5001
started thread <Thread(Thread-2, started 11324)>
started thread <Thread(Thread-3, started 18356)>
started thread <Thread(Thread-4, started 4628)>
job b'{"job_id": "0", "map_tasks": [{"task_id": "0_M0", "duration": 3}, {"task_id": "0_M1", "duration": 1}], "reduce_tasks": [{"task_id": "0_R0", "duration": 4}
eceives from ('127.0.0.1', 49962) through port 5000
Available Slots=  5
sending {"task_id": "0_M0", "duration": 3} to 4000
Available Slots=  4
sending {"task_id": "0_M1", "duration": 1} to 4000
waiting for connection at port number :5000
job b'{"job_id": "1", "map_tasks": [{"task_id": "1_M0", "duration": 4}, {"task_id": "1_M1", "duration": 2}, {"task_id": "1_M2", "duration": 1}, {"task_id": "1_M
duration": 4}], "reduce_tasks": [{"task_id": "1_R0", "duration": 1}, {"task_id": "1_R1", "duration": 4}]}' receives from ('127.0.0.1', 49966) through port 5000
Available Slots=  3

sending {"task_id": "1_M0", "duration": 4} to 4000
Available Slots=  2
sending {"task_id": "1_M1", "duration": 2} to 4000
Available Slots=  1
Available Slots=  0
job b'{"worker_id": 1, "avaSlots": 2, "slotJobs": [true, 0, ""], "slot_id": 2, "jobCompleted": "0_M1"}' receives from ('127.0.0.1', 49969) through port 5001

waiting for connection at port number :5001
Available Slots=  7
sending {"task_id": "1_M2", "duration": 1} to 4001
Available Slots=  2
sending {"task_id": "1_M3", "duration": 4} to 4000
waiting for connection at port number :5000
job b'{"job_id": "2", "map_tasks": [{"task_id": "2_M0", "duration": 4}, {"task_id": "2_M1", "duration": 3}, {"task_id": "2_M2", "duration": 2}, {"task_id": "2_M
duration": 1}], "reduce_tasks": [{"task_id": "2_R0", "duration": 3}]}' receives from ('127.0.0.1', 49970) through port 5000
Available Slots=  1
Available Slots=  0
```

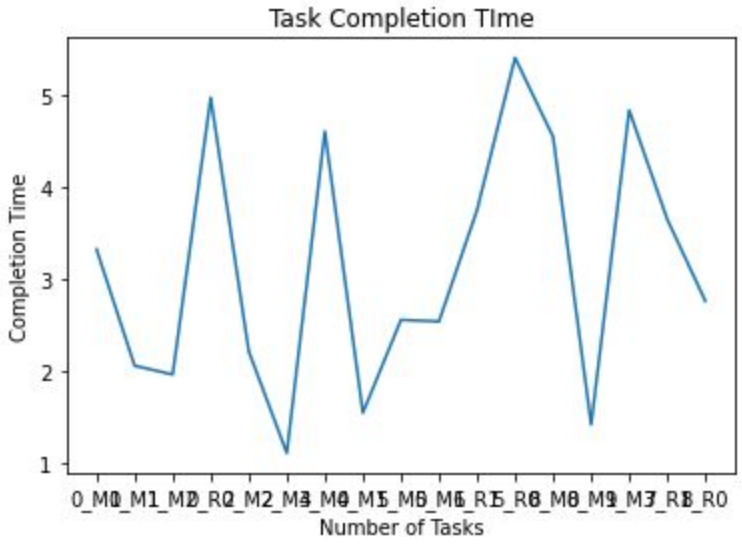
Running one of the workers-

```
D:\fpl_assignment>python worker.py 4000 1
started thread <Thread(Thread-1, started 5932)>
waiting for connection at 4000
started thread <Thread(Thread-2, started 13088)>
port number 4000 receives -> b'{"task_id": "0_M0", "duration": 3}'
received:0_M0,1607580744335.0562

waiting for connection at 4000
port number 4000 receives -> b'{"task_id": "0_M1", "duration": 1}'
received:0_M1,1607580744467.157

waiting for connection at 4000
[False, 3, '0_M0']
[False, 1, '0_M1']
port number 4000 receives -> b'{"task_id": "1_M0", "duration": 4}'
received:1_M0,1607580745040.492

waiting for connection at 4000
port number 4000 receives -> b'{"task_id": "1_M1", "duration": 2}'
received:1_M1,1607580745180.706

waiting for connection at 4000
[False, 2, '0_M0']
sending {'worker_id': 1, 'avaSlots': 2, 'slotJobs': [True, 0, ''], 'slot_id': 2, 'jobCompleted': '0_M1'} to 5001
[False, 4, '1_M0']
[False, 2, '1_M1']
port number 4000 receives -> b'{"task_id": "1_M3", "duration": 4}'
received:1_M3,1607580746467.4348

waiting for connection at 4000
[False, 1, '0_M0']
[False, 4, '1_M3']
[False, 3, '1_M0']
[False, 1, '1_M1']
sending {'worker_id': 1, 'avaSlots': 2, 'slotJobs': [True, 0, ''], 'slot_id': 1, 'jobCompleted': '0_M0'} to 5001
[False, 3, '1_M3']
[False, 2, '1_M0']
sending {'worker_id': 1, 'avaSlots': 3, 'slotJobs': [True, 0, ''], 'slot_id': 4, 'jobCompleted': '1_M1'} to 5001
```

The analysis for LL scheduling algorithm also shows the following-



The analysis for completion time of the worker is shown below-



Problems

We faced the following problems

- creating the sockets for send and receiving of data were often giving “Connection refused error”
- creating and joining the threads
- Designing the framework in such a way that map tasks for each job is finished before reduce tasks was challenging
- knowing when to acquire and release the locks

Conclusion

We were able to take away many learnings from this YACS project.

- An idea of how the master-slave architecture of Hadoop/Spark works was learnt.
- The way in which the map tasks and reduce tasks must be synchronously processed.
- The all-important two way communication between master and its workers.
- The effect of three different scheduling algorithms on the job scheduling of the master.
- How threads can be used to parallelize the different processes of the worker and master.

EVALUATIONS:

SNo	Name	SRN	Contribution (Individual)
1	Aditya G Burli	PES1201800034	Report, Worker
2	Sidharth Pathak	PES1201800142	
3	Vishnu A S	PES1201800192	
4	Nishant Tripathy	PES1201801296	

(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.	