CS322:Big Data

Final Class Project Report

Project (FPL Analytics / YACS coding): YACS   Date: 06-12-2020

SNo	Name	SRN	Class/Section
1	Hari Chandan Sadum	PES1201800397	H
2	Shreyas Vasisht	PES1201800788	D
3	Deepak Dodla	PES1201800361	G
4			

Introduction

YACS(Yet another Centralised Scheduler) is baedon centralised scheduling algorithms. In our project, we implement the simulation of the working of a centralised scheduler. We use two types of programs, master and workers to run the simulation. The master is the program that receives the requests for jobs execution and schedules the job and sends it to the workers. The workers run the tasks and send acknowldgements for the finished tasks. We use TCP sockets for communication between master and workers.

Related work

Implementation of a Centralized Scheduling Algorithm for IEEE 802.15.4e TSCH:

IEEE 802.15.4e Time Slotted Channel Hopping (TSCH) is an important standard in the actual scalable Internet of Things (IoT), to habilitate deterministic low-power mesh networking. In TSCH, scheduling can be distributed or centralized scheduling.

In this paper, they discussed about a centralized scheduler for accesing various networks in the channel. The paper implemets a centralized scheduling algorithm and compares the result with the traditional used algorithm, which shows that the centralized algorithm performs better. The implementation aim is to make a real IoT network running a centralized scheduling algorithm of TSCH which can be effectively applied in the industrial networks and the results proved that.

Design

The design of the system is implemented in three different files, the requests file, master file and the worker files. The requests file generates requests based on the user input of number of files. The job requests are of dictionary form which contains the job id, mapper tasks and the reducer tasks.

<u>MASTER.py :</u>

The master file takes the requests from the requests file through the function 'rcv_requests' which uses a TCP socket to communicate with the requests socket. The job data is then sent to the 'scheduler_selection' which selects the scheduler based on the user input and sends the tasks to the corresponding scheduler. The scheduler takes the tasks in a dictionary format and sends the tasks to the corresponding worker port based on the scheduling algorithm. This is done for all the mapper tasks and then to the reducer tasks(reducer task dependencies).

There are three different schedulers Random, RR and LL. The random scheduler selects a worker by random(random module). The Round Robin scheduler selects the worker in a rcircular fashion starting from the first worker. The port no is set to '0' and then incremented in (n+1)%3 formula for three workers. The Least loaded selects the least loaded worker(find the index of the worker with the least slots and send it to the port of the same index). The function then sends the port number and the task details to the ‘send_task_workers’ function which connects to the specified port and sends the task data to the worker of that port.

The ‘rcv_ack’ function, started at the start of the master file execution, listens for acknowledgemets for the finished tasks and receives the task ids of the finished tasks from the workers and appends them to the ‘ack_task_ids’ list. The ‘update_ports function’ accesses the ‘ack_task_ids’ list and if the task id is in the list, frees the corrsponding port by increasing the slots.

Various threads are used for executing the fucntions. At the start of execution two threads are used to run rcv_requests and rcv_ack functions concurrently. The execution of scheduling algorithms and sending the task data to the workers is done by threads for concurrency. Two locks are used for the threads, one for updating slots and one for updating the ‘ack_task_id’ list which are prone to race condition due to sending and receiving tasks and updating slots concurrently at different points of time.

WORKER.py :

The worker file receives the tasks from the master using the ‘rcv_tasks_master’ function and simulates the working by using time.sleep() for the given task duration. It then sends the task’s id to the ‘send_ack_master’ function which connects to the 5001 port number(port for acks) and sends the task id to the master. It takes port number and worker id as system arguments to run the specified worker.

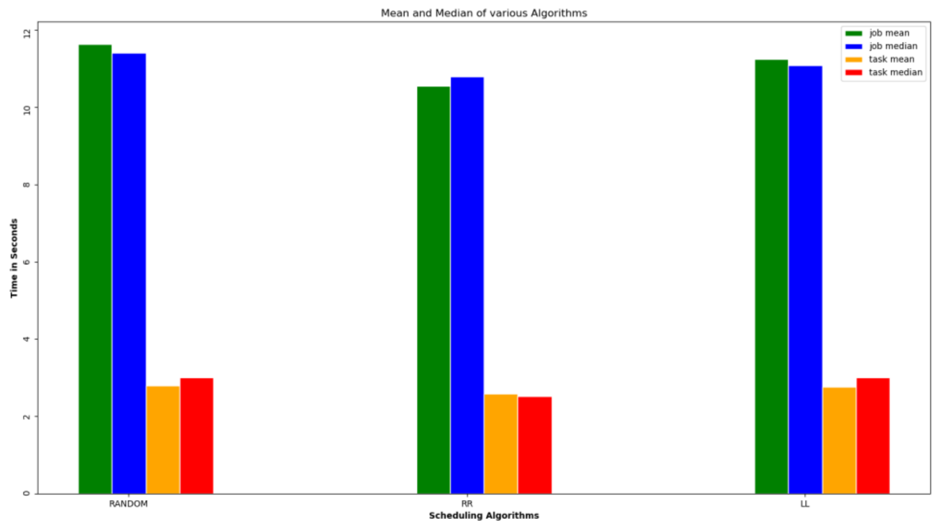
Results

The Results are analysed in two different graphs and three files, Analysis_part1.py, Analysis_part2_1.py, Analysis_part2_2.py. The Analysis_part_1.py file takes the job log and task log data of three algorithms and computes the mean and median of jobs and tasks execution time and writes the data into analysis files. It takes the scheduling algorithm as system argument.

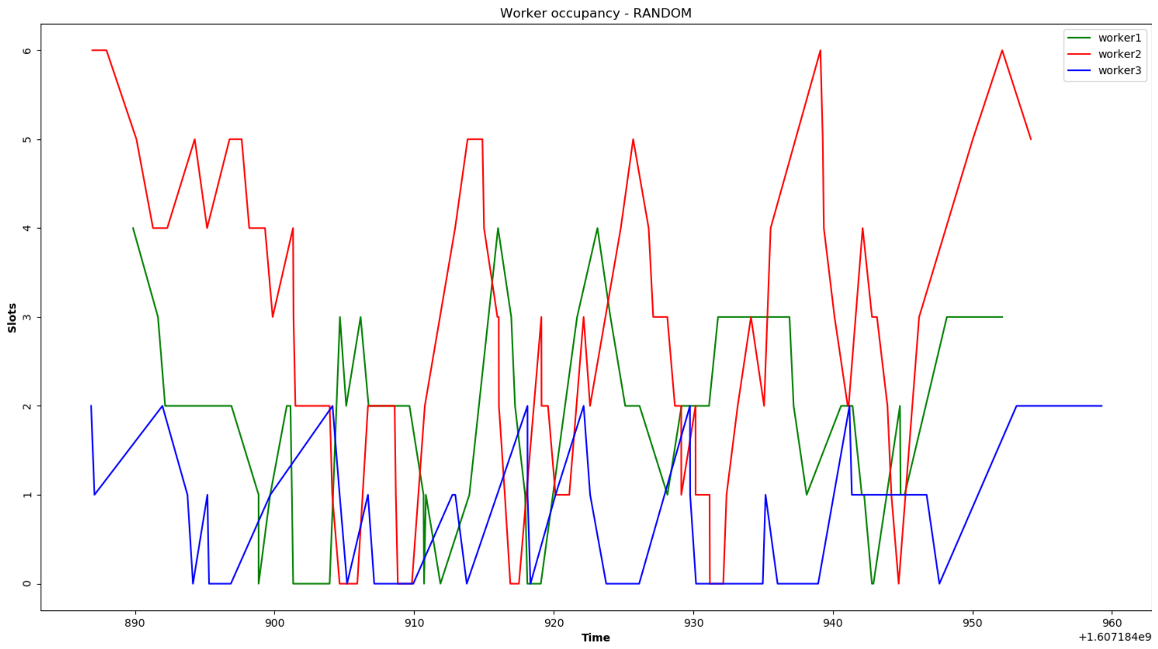
The Analysis_part2_1.py file collects the data from all the analysis files and plots a grouped bar graph for the mean and median for various algorithms.

The Analysis_part2_2.py file collects the worker logs for the algorithms and plots a line graph of free slots against time of execution of tasks. The algorithm is given as system argument.

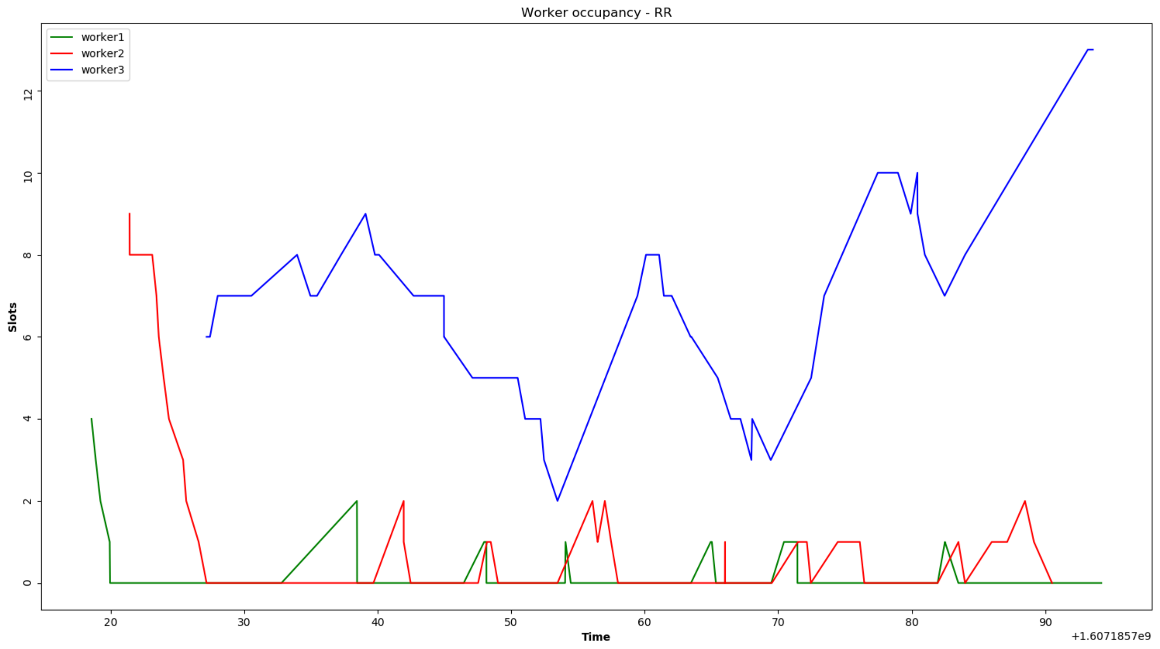
1. Bar graph depicting the mean and median of jobs and tasks for various algorithms



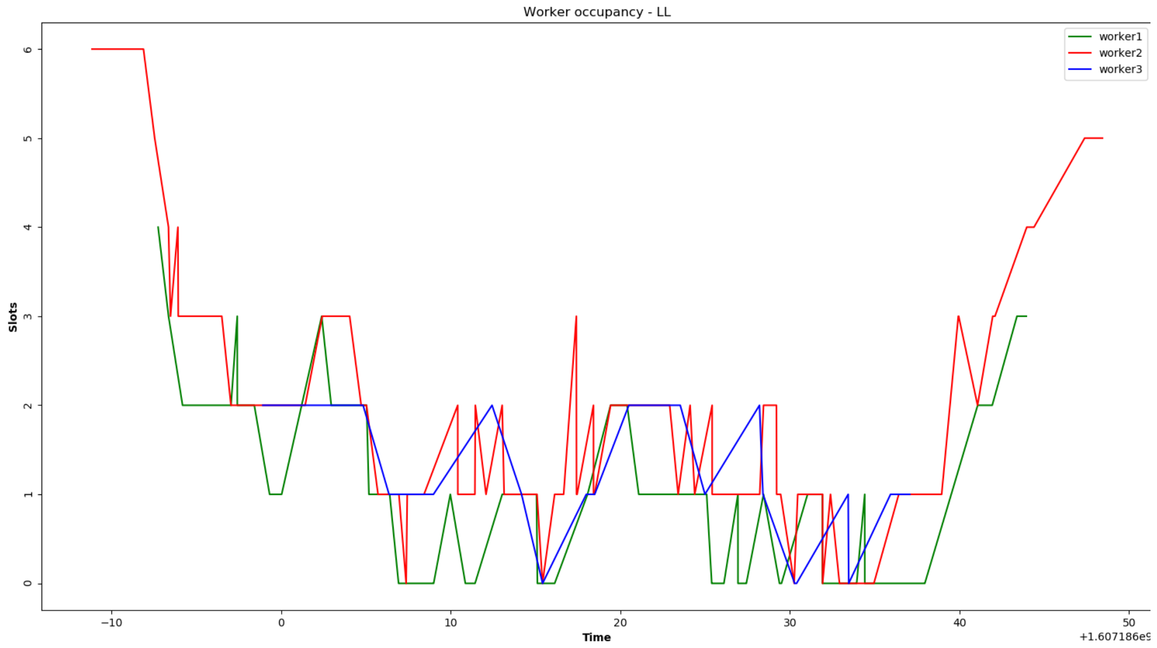
2. Line graph for Random algorihtm:



3. Line graph for Round robin algorithm:



4. Line graph for Least loaded algorithm:



Problems

The problems faced were mainly of threads and checking for map-reduce task dependencies. The race condition needs to be checked and the locks are to be properly handled. If there is a lock which wasn't released, there would be deadlock. The question of when to start sending reduce tasks and knowing when the map tasks would end was a bit tricky.

Conclusion

We learnt the advantage of having a centralized scheduler and how it works. The importance of threading and locks were also conveyed. The main outcome of this project was to implement an ideal scheduler for handling jobs concurrently with high time efficiency and work load distribution.

EVALUATIONS:

SNo	Name	SRN	Contribution (Individual)
1	Hari Chandan Sadum	PES1201800397	75(Execution part)
2	Shreyas Vasisht	PES1201800788	25(Analysis part)
3	Deepak Dodla	PES1201800361	0
4			

(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.	