CS322:Big Data

Final Class Project Report

Project (FPL Analytics / YACS coding): YACS coding **Date: 1/12/2020**

SI No	Name	SRN	Contribution (Individual)
1	Abhishek Shyam	PES1201801743	General YACS design and working on master and worker scripts.
2	H M Thrupthi	PES1201801987	Designing of Master and Report
3	Hemanth Alva R	PES1201801937	Visualization using Graphs and Analysis
4	Sreekanth R Gunishetty	PES1201801467	Report and Implementation of Master to do analysis

YACS: Yet Another Centralized Scheduler

Introduction

At a high level, a big data strategy is a plan designed to help oversee and improve the way to acquire, store, manage, share and use data within and outside of the organization. Big Data workloads consist of multiple jobs from different applications. A scheduling framework is used to manage and allocate the resources of the cluster (CPUs, memory, disk, network bandwidth, etc.) to the different jobs in the workload.

YACS is the centralized scheduling framework consisting of one Master, which runs on a dedicated machine and manages the resources of the rest of the machines in the cluster. The other machines in the cluster have one Worker process running on each of them. The Master process makes scheduling decisions while the Worker processes execute the tasks and inform the Master when a task completes its execution.

Related work

Earlier studies already demonstrated the superior convergence rates of YACS compared to modern commercial 3D solvers. [1] Whereas, with the most recent revision of YACS the emergence of spurious modes, when solving for eigenmodes with an azimuthal mode number m 0 and utilizing higher order basis functions for the field or geometry discretization, could be successfully suppressed.

The paper on Scheduling algorithms summarizes the state of the real-time field in the areas of scheduling and operating system kernels.[2] Given the vast amount of work that has been done by both the operations research and computer science communities in the scheduling area, they discussed four paradigms underlying the scheduling approaches. The four paradigms are: static table-driven scheduling, static priority preemptive scheduling, dynamic planning-based scheduling, and dynamic best effort scheduling.

Design of YACS

In the YACS, the framework consists of one Master, which runs on a dedicated machine and manages the resources of the rest of the machines in the cluster. The other machines in the cluster have one Worker process running on each of them. The Master listens for job requests and dispatches the tasks in the jobs to machines based on a scheduling algorithm. The Master is informed of the number of machines and the number of slots in each machine with the help of a config file.

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. The execution pool consists of all currently running tasks in the machine.

We have designed the master script in our YACS as follows:

1) We have three threads to carry out the working of a master.
2) The first one deals with intercepting incoming job requests.
3) The second one deals with iterating over the requests and scheduling them.
4) The third one deals with getting an update back from the worker script, this also helps us sequence operations from mapper to reducer.

We have designed the Worker script in our YACS as follows:

1) The first thread accepts task requests for execution
2) The second thread processes the requests and sends the completed task with start time and end time.

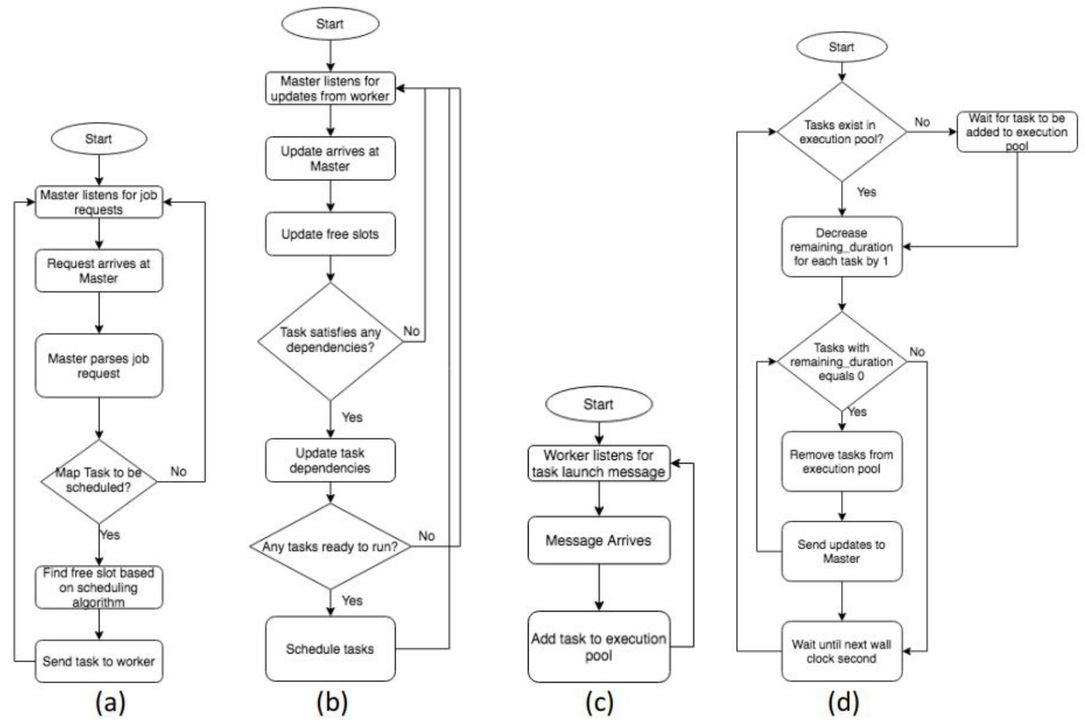
Implementation of YACS Model

In the functioning of YACS simulated framework that gets created, there is 1 Master process and 3 Worker processes. All the processes run on the same PC, however, they must behave as though they are on separate dedicated machines.

All jobs have 2 stages only.

- ✦ The First stage consists of Map tasks
- ✦ The Second stage consists of Reduce tasks.

The reduce tasks in a job can only execute after all the map tasks in the job have finished executing. There is no ordering within Reduce tasks, or within map tasks. All map tasks can run in parallel, and all reduce tasks can run in parallel.


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
(a)
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
(b)
Start
Worker listens for task launch message
Message Arrives
Add task to execution pool
(c)
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
(d)

Each Worker's machine will be configured with a fixed number of slots. The Master needs to keep track of the number of slots available in each machine. The Master and the Workers need to maintain a log of important events.

The Master will need to have at least 2 threads in order to listen for job requests and updates from Workers. Each Worker will need to have at least 2 threads for listening for task launch messages from Master and to simulate the execution of the tasks and send updates to the Master. 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. The execution pool consists of all currently running tasks in the machine.

When a task completes execution, the Worker process on the machine informs the Master. The Master then updates its information about the number of free slots available on the machine.

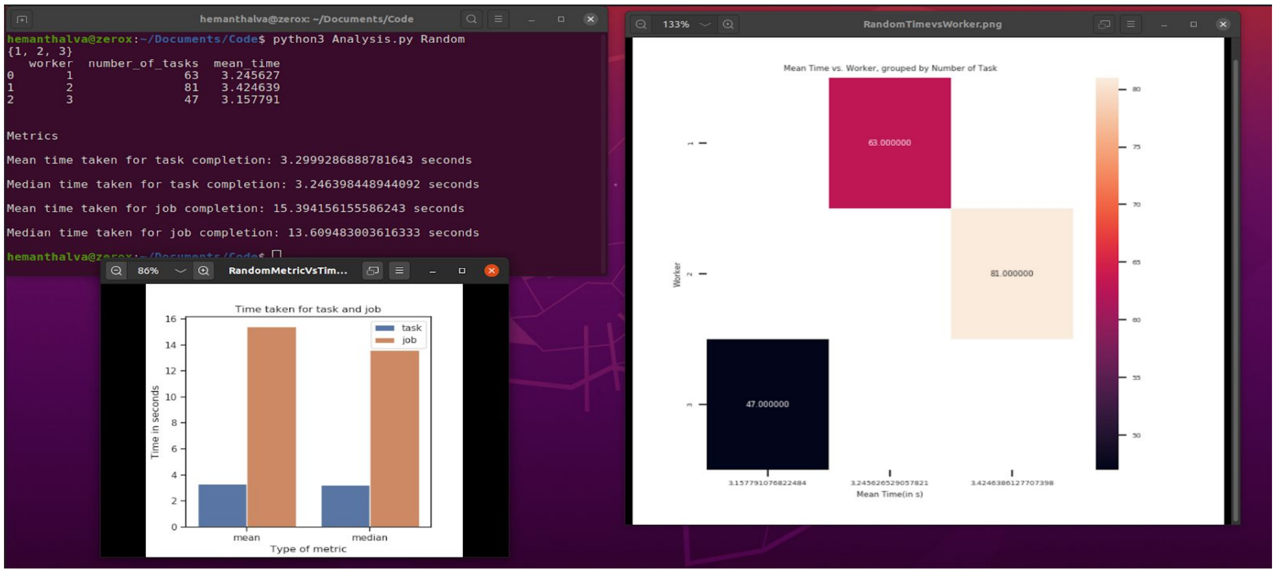
Out Implementation deals with YACS as python scripts with one master and one worker script. The data transfer happens using socket programming.

We receive the data from job requests file to master in port 5000 and communication between worker and master happens in port 5001. Ports dealing with workers are set in the config file. Input for the particular scheduling algorithm is taken : Round Robin, Random and Least Load scheduling Algorithms. We use a separate thread for scheduling the tasks based on user requirement.

Scheduling Algorithms

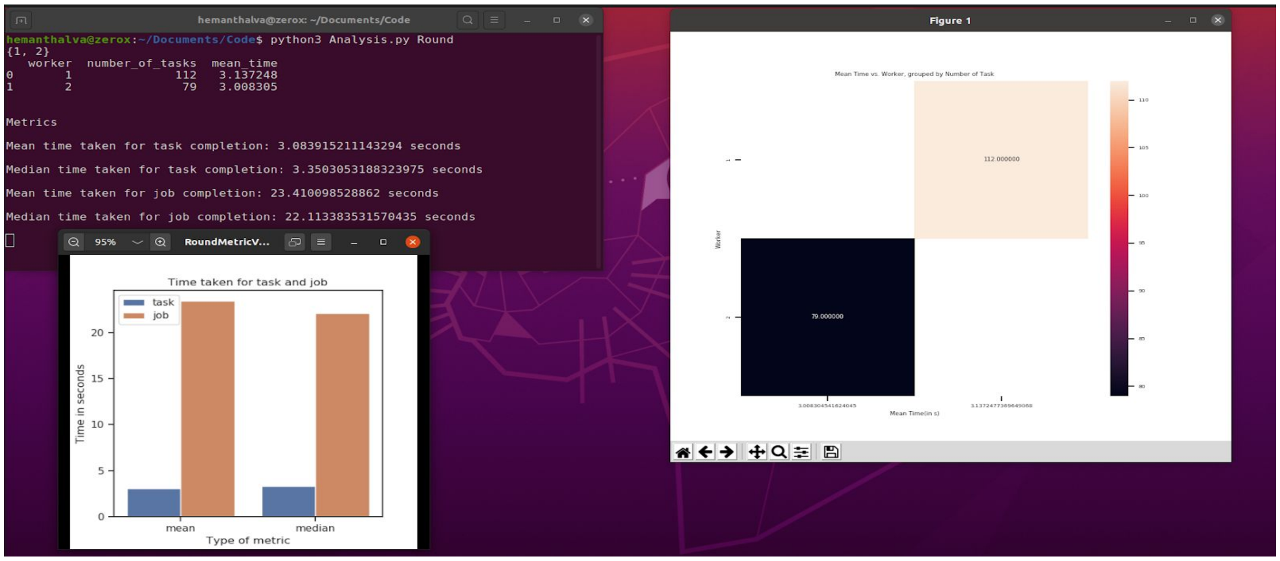
The Master listens for job requests and dispatches the tasks in the jobs to machines based on a ***scheduling algorithm***.

1) **Random:** The Master chooses a machine at random. It then checks if the machine has free slots available. If yes, it launches the task on the machine. Else, it chooses another machine at random. This process continues until a free slot is found.



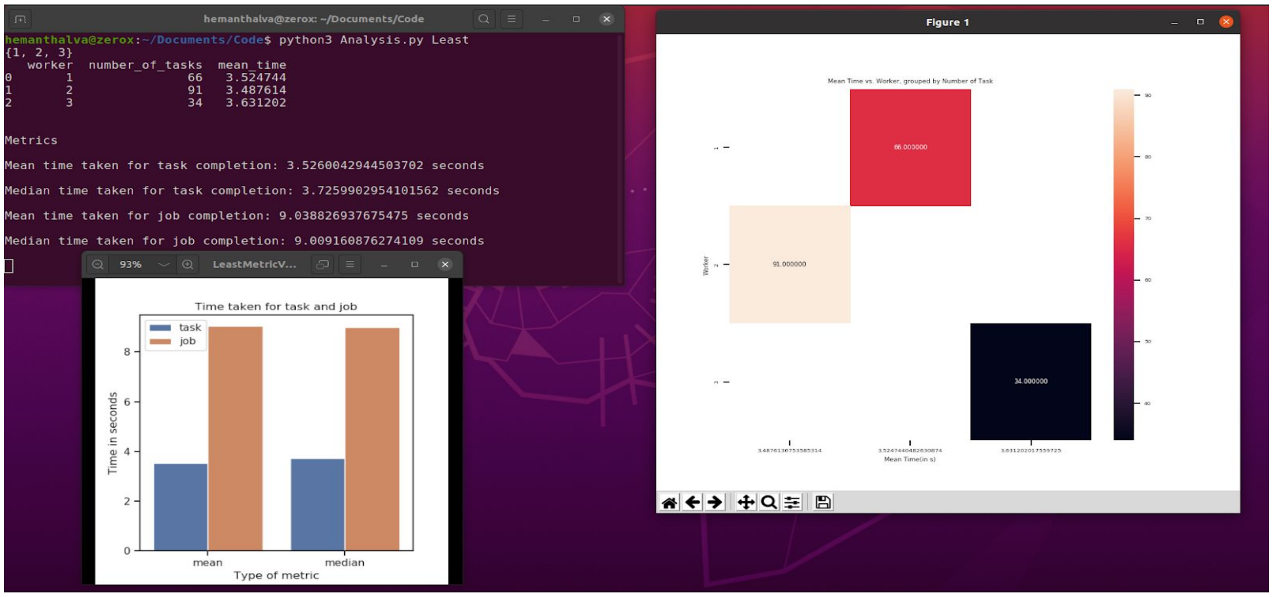
```
hemanthalva@zerox:~/Documents/Code$ python3 Analysis.py Random
{1, 2, 3}
   worker  number_of_tasks  mean_time
0       1               63   3.245627
1       2               81   3.424639
2       3               47   3.157791


Metrics

Mean time taken for task completion: 3.2999286888781643 seconds

Median time taken for task completion: 3.246398448944092 seconds

Mean time taken for job completion: 15.394156155586243 seconds

Median time taken for job completion: 13.609483003616333 seconds

hemanthalva@zerox:~/Documents/Code$
```

2) **Round Robin**: The Master comes to each of the workers based on their ids one by one. It checks if the first worker is free, if not goes to the second and goes on a circular mission to find a free slot. When it finds a free slot and identifies a worker, it schedules the task on the worker.



```
hemanthalva@zerox:~/Documents/Code$ python3 Analysis.py Round
{1, 2}
   worker  number_of_tasks  mean_time
0       1              112   3.137248
1       2               79   3.008305


Metrics

Mean time taken for task completion: 3.083915211143294 seconds

Median time taken for task completion: 3.3503053188323975 seconds

Mean time taken for job completion: 23.410098528862 seconds

Median time taken for job completion: 22.113383531570435 seconds
```

3) **Least-Loaded:** The Master looks at the state of all the machines and checks which machine has the most number of free slots. It then launches the task on that machine. If none of the machines have free slots available, the Master waits for 1 second and repeats the process. This process continues until a free slot is found.

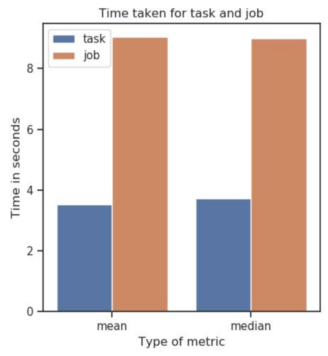


```
hemanthalva@zerox:~/Documents/Code$ python3 Analysis.py Least
{1, 2, 3}
   worker  number_of_tasks  mean_time
0       1               66   3.524744
1       2               91   3.487614
2       3               34   3.631202


Metrics

Mean time taken for task completion: 3.5260042944503702 seconds

Median time taken for task completion: 3.7259902954101562 seconds

Mean time taken for job completion: 9.038826937675475 seconds

Median time taken for job completion: 9.009160876274109 seconds
```

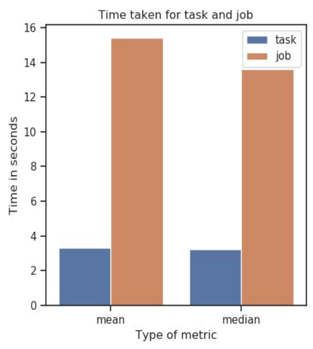
Results

Scheduling Algorithm	Mean time for Task Completion	Median time for Task Completion	Mean time for Job Completion	Median time for Job Completion
Random	3.29	3.24	15.39	13.60
Round Robin	3.08	3.35	23.41	22.11
Least-Loaded	3.52	3.72	9.03	9.00

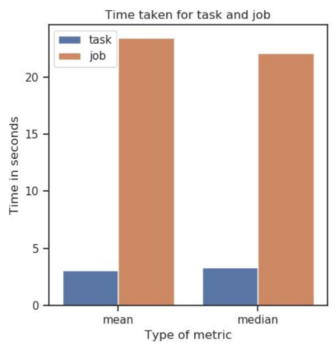
Least-Loaded Vs Time

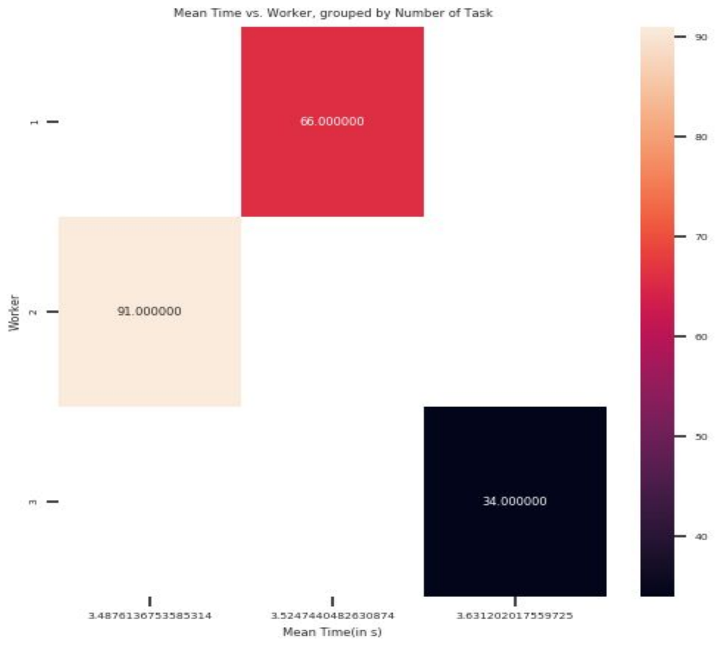


Random Metric Vs Time

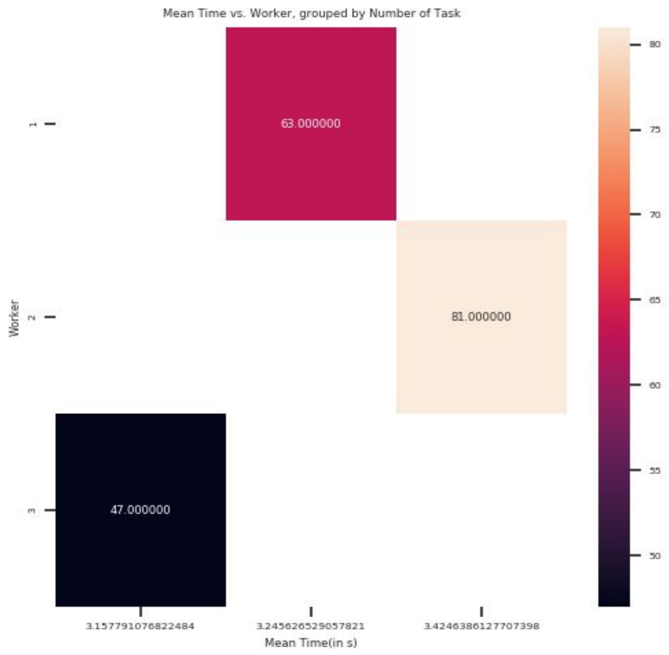


Round Robin Vs Time

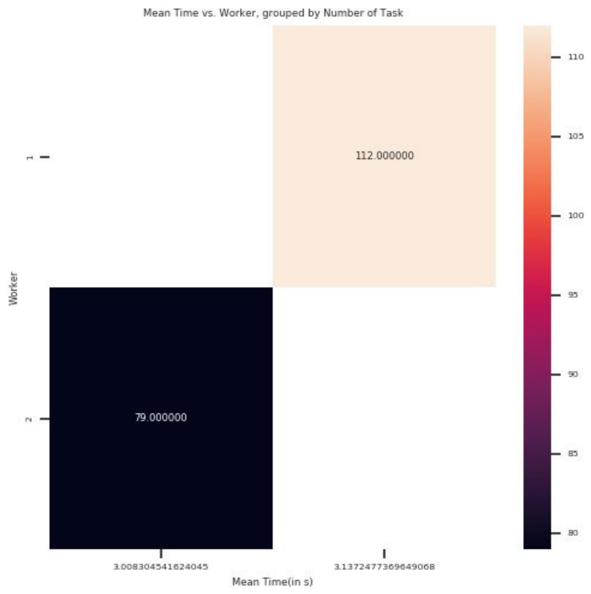




Least -Loaded Algo



Random Algorithm



Round Robin Algorithm

Challenges

The Challenges we faced in implementation are:

a) Initially we faced problems with the design, on the threads and their allotment. We finally decided on three threads for master and two threads for workers. The three threads in worker are for accepting requests, scheduling and receiving from workers.
b) We ran into some problems while using variables that span more than a single thread, and using locks properly worked like a charm.
c) We faced problems while trying to gracefully exit the master. We then set a general timeout that made sense and made sure to end the threads there and output the logs.

Conclusion

From our implementation and the results we can fairly assume that the least loaded way of scheduling the tasks gives the best mean time for job completion. With more number of workers, the mean time decreases as the work gets distributed more parallely. However, we notice the drawback of Least Loaded when there are a lot of workers and we have to iterate over all the slots of all the workers to schedule a task. Round Robin does not suffer with this problem but it leads to starvation for the last few workers when we schedule by sorting by worker id and scheduling even if one slot gets available. Random generally works better than Round robin in the general sense while also taking no time (O(1)) to decide which worker to give work to.

We have learnt an implementation of the YACS Model and its importance. The Scheduling Algorithms has processed on YACS and obtained various mean time and median time for both Job and Task completion, which impacted on us to choose the appropriate algorithm for given real time applications.

References

1) Isbarn, B. D., et al. "Cavity Characterization Studies With the Latest Revision of YACS." *9th Int. Particle Accelerator Conf.(IPAC'18), Vancouver, BC, Canada, April 29-May 4, 2018*. JACOW Publishing, Geneva, Switzerland, 2018.
2) Ramamritham, Krithi, and John A. Stankovic. "Scheduling algorithms and operating systems support for real-time systems." *Proceedings of the IEEE* 82.1 (1994): 55-67.
3) YARN studied in class.
4) Discussion with friends.

EVALUATIONS:

SNo	Name	SRN	Contribution (Individual)
1	Abhishek Shyam	PES1201801743	General YACS design and working on master and worker scripts.
2	H M Thrupthi	PES1201801987	Designing of Master and Report.
3	Hemanth Alva R	PES1201801937	Visualization using Graphs and Analysis.
4	Sreekanth R Gunishetty	PES1201801467	Report and Implementation of Master to do analysis.

(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.	