BIG DATA

Class Project: YACS

K V Subramaniam
Computer Science and Engineering

BIG DATA

Class Project: YACS - Yet Another Centralized Scheduler

K V Subramaniam
Computer Science and Engineering

BIG DATA

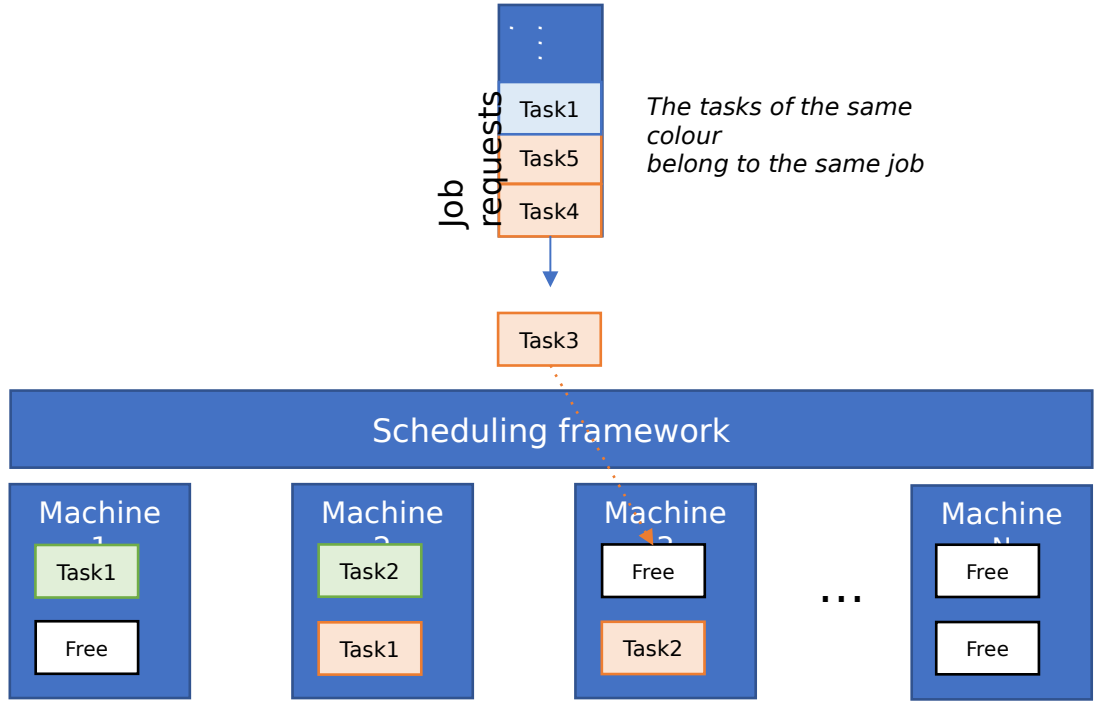
Introduction

Aim

- Simulate the working of a centralized scheduling framework.
- Implement the functioning of 3 different scheduling algorithms.
- Calculate metrics and analyze the results.

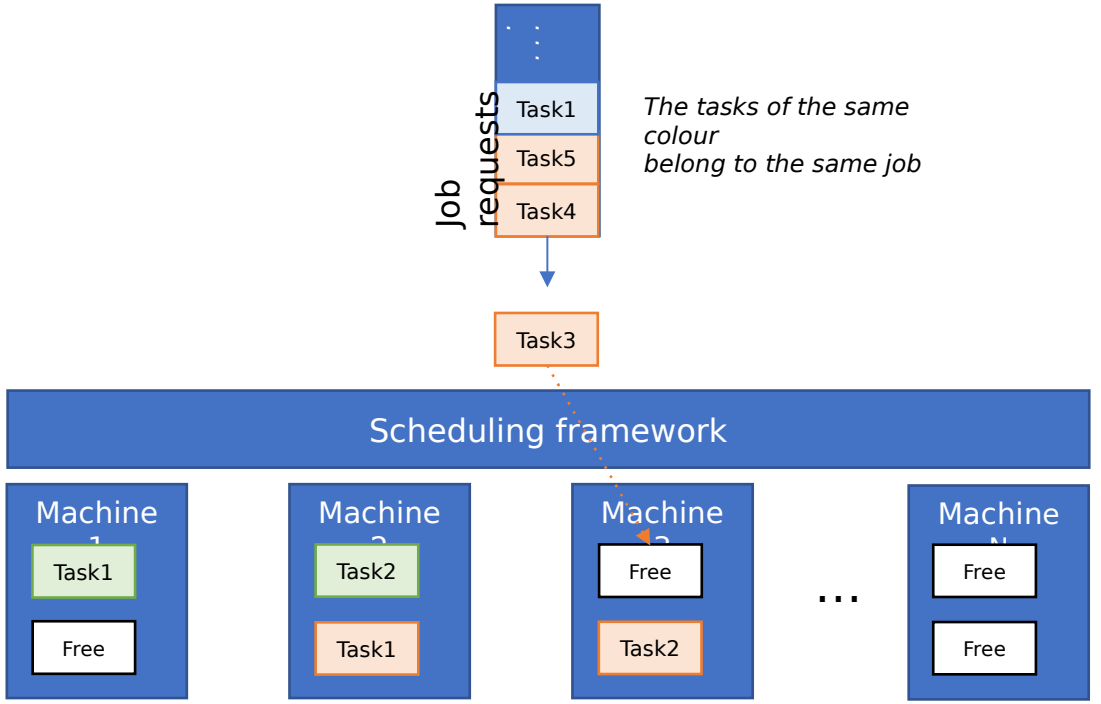
BIG DATA
Overview

- Big data workloads consist of multiple jobs from different applications.
- These workloads are too large to run on a single machine.
- They run on clusters of interconnected machines.
- 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.



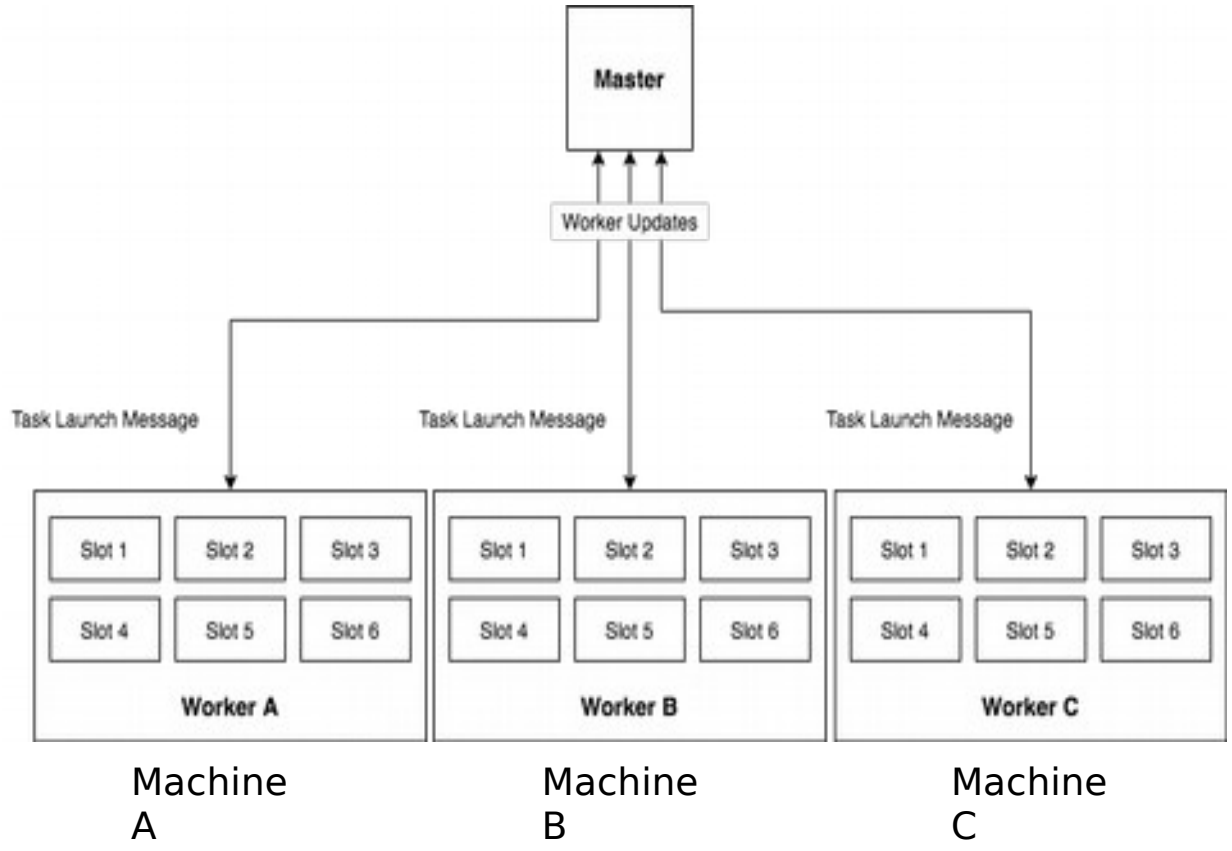
BIG DATA
Overview

- A job is made of one or more *tasks*.
- The scheduling framework receives job requests and launches the tasks in the jobs on machines in the cluster.
- In the figure, Task 3 of the orange job is assigned to the free resources on Machine 3.
- Once the task is allocated resources on a machine, it executes its code.
- When a task finishes execution, the scheduling framework is informed, and the resources are freed.
- The framework can thereafter assign these freed resources to other tasks.



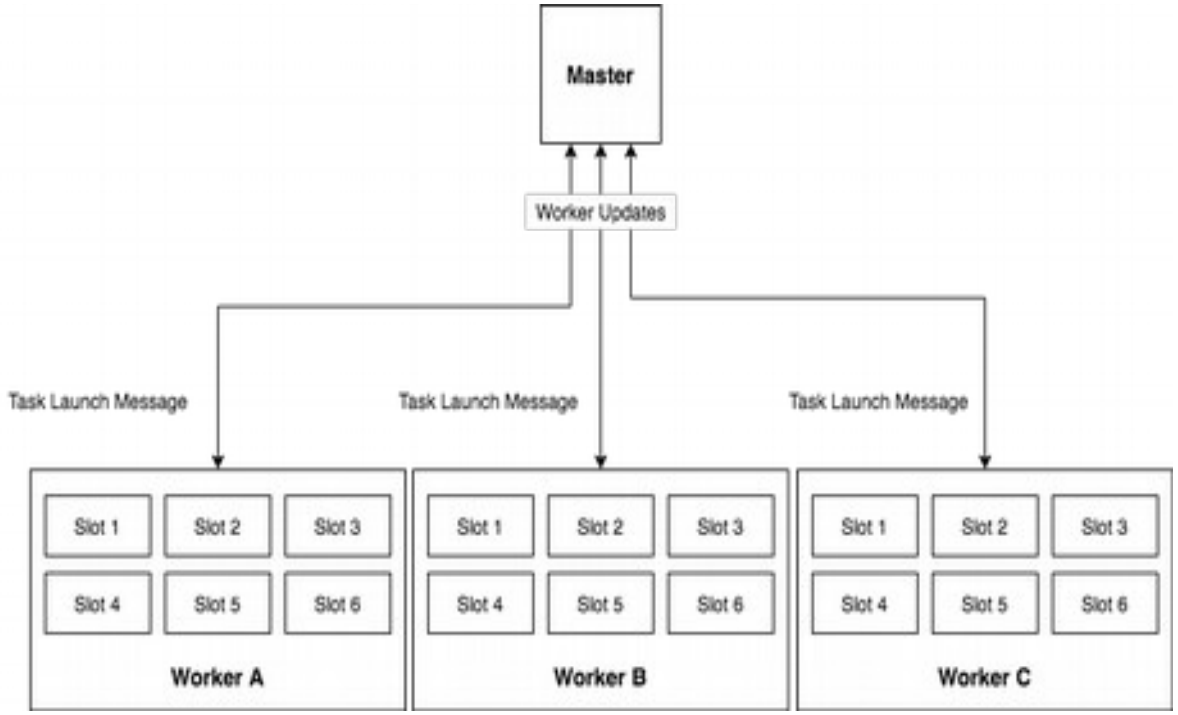
BIG DATA
YACS

- The final project is focused on YACS, a **centralized scheduling framework**.
- 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 process makes scheduling decisions while the Worker processes execute the tasks and inform the Master when a task completes its execution.



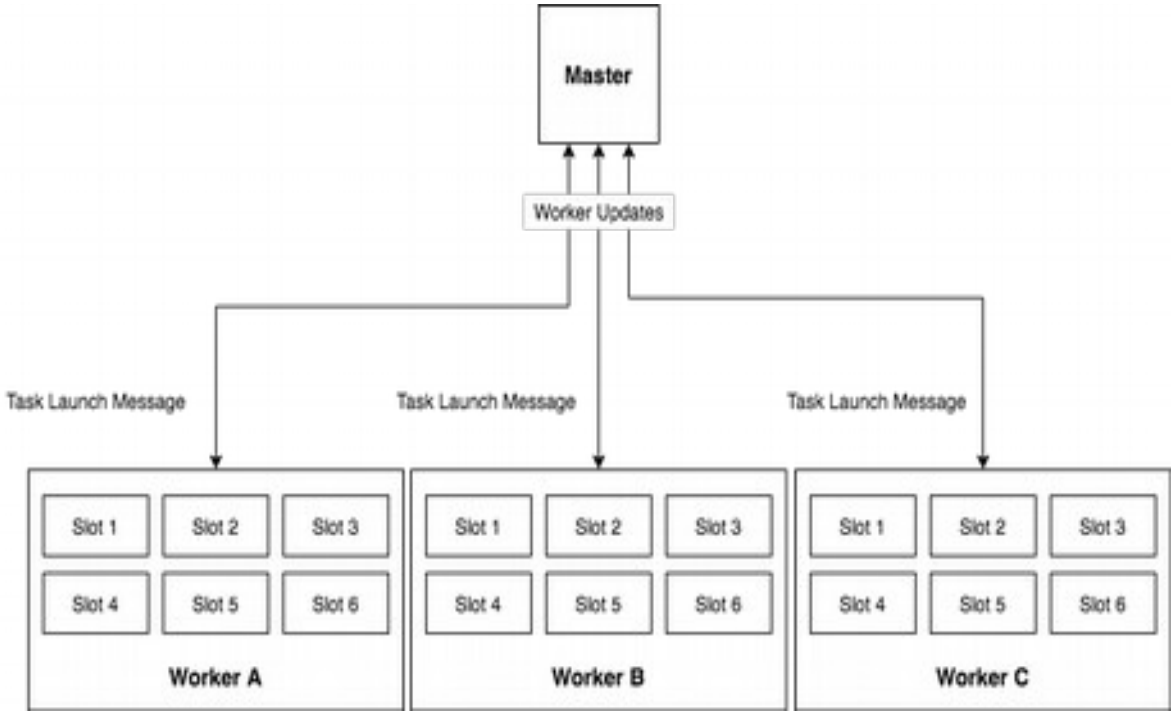
BIG DATA
YACS

- The Master listens for job requests and dispatches the tasks in the jobs to machines based on a *scheduling algorithm*.
- Each machine is partitioned into equal-sized resource encapsulations (E.g. 1 CPU core, 4GB of memory, 1TB of disk space) called **slots**.
- The number of slots in each machine is fixed. Each slot has enough resources to execute one task at a time.
- Machines may have different capacities (in terms of amount of total memory, total number of cores, etc.); the number of slots may differ from machine to machine.
- Maximum number of tasks a specific machine can run is equal to the number of slots in the machine.



BIG DATA
YACS

- 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.



BIG DATA

Part 1: Simulation

Simulation - Overview

- Simulate YACS with **1 Master process and 3 Worker processes**.
- All processes will run on the same PC.
- However, must behave as though each is a component deployed on a dedicated machine.
- Simulated machine has nothing to do with the PC on which you will be running the simulation.
- The number of slots in each simulated Worker machine is as per the config passed to the Master process.

BIG DATA

Simulation – Machine resources (Slots)

- In your implementation, *slots* are only an abstraction.
- For instance, the value of available slots for a simulated machine is only a number.
- It is decremented when a slot is said to have been allocated to a task
- It is incremented when a slot is said to have been freed on a task’s completion

Simulation – Worker process and task execution

- The Worker process listens for task launch message from the Master.
- When it receives a task launch message, it adds the task to its execution pool.
- The Worker process must *simulate* the running of the tasks in the execution pool.
- It does so by decrementing the *remaining_duration* value of each task, every second, until it reaches 0.
- Once the *remaining_duration* of a task reaches 0, the Worker removes the task from its execution pool and reports to the Master that the task has completed its execution.

BIG DATA

Simulation – Inter-task dependency

- The framework will have to respect the map-reduce dependency in the jobs.
- When a map task completes execution, the Master will have to check if it satisfies the dependencies of any reduce tasks and whether the reduce tasks can now be launched.
- A job is said to have completed execution only when all the tasks in the job have finished executing.
- All jobs have 2 stages only - The first stage consists of map tasks and 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.

BIG DATA

Scheduling Algorithms

BIG DATA

Scheduling Algorithms

- **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.
- **Round-Robin:** The machines are ordered based on worker_id of the Worker running on the machine. The Master picks a machine in round-robin fashion. If the machine does not have a free slot, the Master moves on to the next worker_id in the ordering. This process continues until a free slot is found.
- **Least-Loaded:** The Master looks at the state of all the machines and checks which machine has 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.

BIG DATA

Workflow

BIG DATA

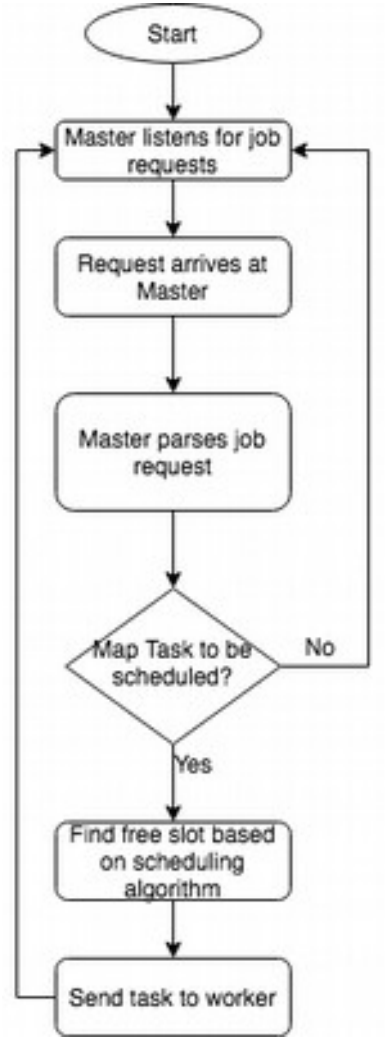
Workflow- overview

1) The Master will need to have at least 2 threads:
   a. to listen for job requests
   b. to listen for updates from Workers.
2) Each Worker will need to have at least 2 threads:
   c. for listening for task launch messages from Master
   d. to simulate the execution of the tasks and to send updates to the Master.

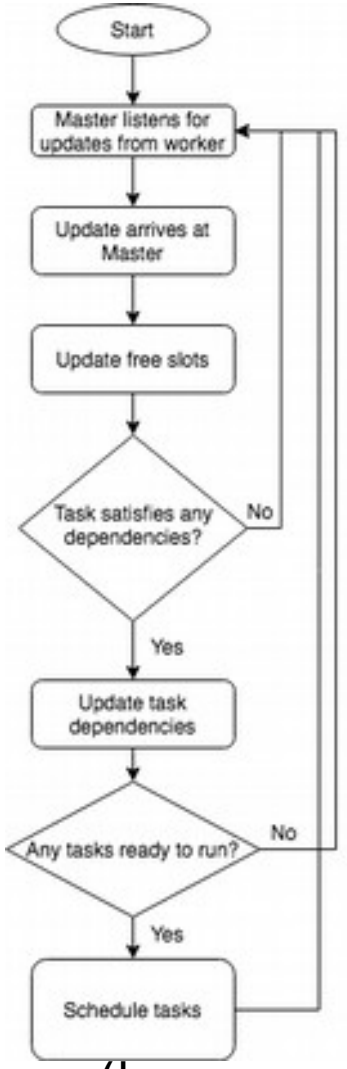
The workflows of the threads a, b, c, and d are depicted in the next slide.

BIG DATA

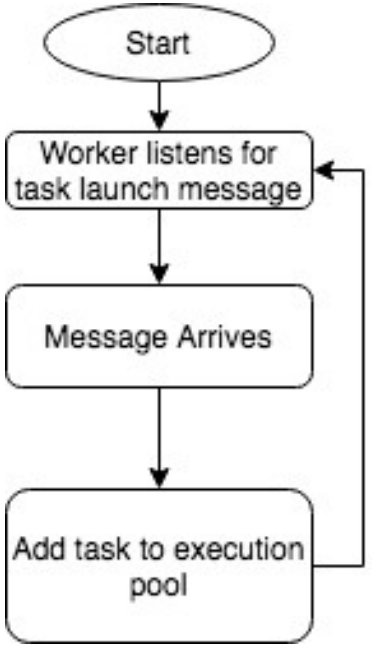
Workflow- overview



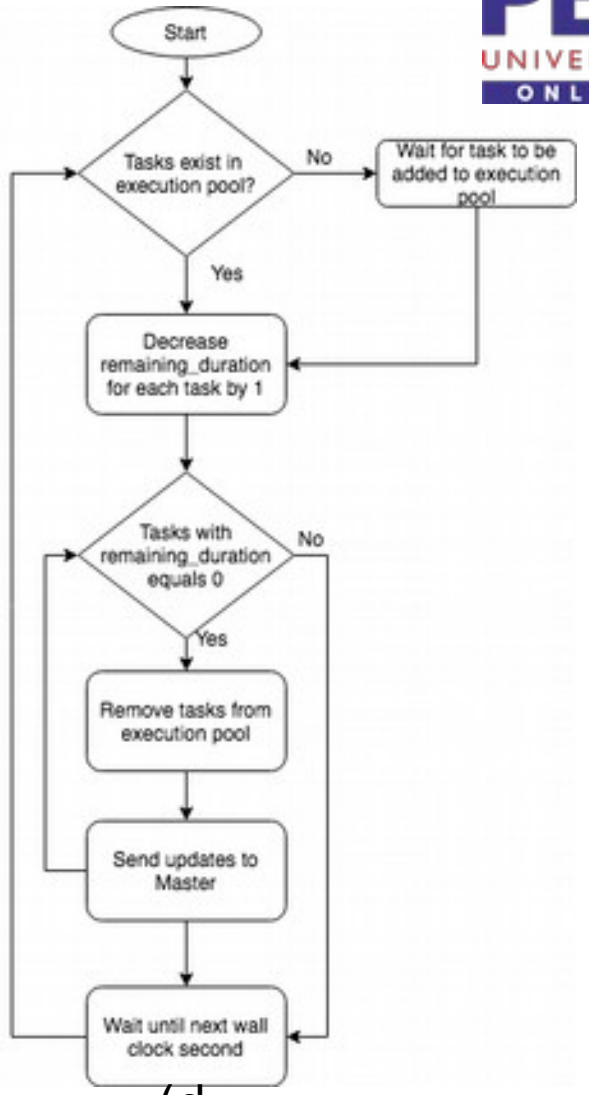
(a



(b



(c



(d

BIG DATA

Implementation

BIG DATA

Implementation - Overview

- You will be using sockets for all communications between the Master and Workers
- To prevent race conditions, make sure to use locks for data structures that are shared between the threads in a component.
- *Even though you are running the Master and Worker processes on the same machine, do not make any assumptions which prevent the Master or any of the Workers from being deployed on separate machines*.
- Ensure that all data sharing between the various components takes this aspect into account. Do not rely on methods such as shared files or global variables for communicating between the Workers and Master.

BIG DATA

Format of Job Request Message (JSON):


```
{   "job_id":<job_id>,
    "map_tasks":[
        {"task_id":"<task_id>","duration":<in seconds>},
        {"task_id":"<task_id>","duration":<in seconds>}
        ...
    ],
    "reduce_tasks":[
        {"task_id":"<task_id>","duration":<in seconds>},
        {"task_id":"<task_id>","duration":<in seconds>}
        ...
    ]
}
```

BIG DATA

Format of config file for Master (JSON):

```
{
      "Workers": [ //one worker per machine
        {
          "worker_id": <worker_id>,
          "slots": <number of slots>, // number of slots in the machine
          "port": <port number> // port on which the Worker process listens for task launch messages
        },
        {
          ”worker_id": <worker_id>,
          "slots": <number of slots>,
          "port": <port number>
        },
        ...
      ]
}
```

BIG DATA

Setting up Scheduling Framework and running the simulation

- Implement the Master and the Workers as two separate python programs - one program for the Master, and one for the Worker.
- The Master will be listening for incoming jobs on port **5000**.
- The path to the config file, and the scheduling algorithm (*RANDOM, RR, LL*) are given to the Master as command line arguments.
  - E.g. python Master.py /path/to/config RR
- The Master will listen for updates from Workers on port **5001**.
- Each Worker will be listening for messages from the Master.
- The port and worker_id are supplied as command line arguments to each Worker program.
- These settings need to be made as per the config file supplied to the Master.
  - E.g. python Worker.py 4000 1

BIG DATA

Setting up Scheduling Framework and running the simulation

- Run the requests.py file to generate job requests with exponentially distributed inter-arrival times and send them to the Master.
- The program takes as command line argument the number of requests to be sent to the Master.
  - E.g. python requests.py 10
- The program requires the following python modules to be installed:
  - json
  - socket
  - time
  - sys
  - random
  - numpy
- The Master and the Workers need to maintain a log of important events.

BIG DATA

Part 2: Results to be analyzed from logs:

BIG DATA
Analyzing results

Write a program that does the following:

- Calculates the mean and median task and job completion times for the 3 scheduling algorithms.
  - task completion time:

    (end time of task in Worker)–(arrival time of task at Worker)

  - job completion time:

    (end time of the last reduce task)–(arrival time of job at Master)
- Plots the number of tasks scheduled on each machine, against time, for each scheduling algorithm.
- Note: During evaluation, you may be asked to modify the config and run the simulation for the new config, and also generate the results again.

BIG DATA

Hints

BIG DATA

Suggested order of implementation:

To always have a working project, and to get partial marks even if some of the features are incomplete, you could follow this order:

- Write simple code for both Master and Workers to just listen for incoming requests.
- Decide on data structures needed in both, and which ones need to be shared between threads.
- Implement map task launch in Master for any one scheduling algorithm
- Implement simulation of task execution in Worker.
- Implement Master logic to take care of reduce task dependencies.
- Implement the remaining scheduling algorithms
- Check what all events need to be logged to obtain results described in part 2 and implement the logging in a consistent format.
- Write code that takes in logs from the Workers and Master to generate the results.

BIG DATA
Debugging Tips

Since you are dealing with multiple parallel components, each of which is multithreaded, you could use the following suggestions, apart from logging, to help you debug any issues:

- Send custom job requests to the server to trace the functioning of your scheduling algorithm.
- To verify the correctness of your code, work out a few examples by hand (for RR and LL) and see if the system functions the same way.
- To examine the state of variables at any point of time:
  - create a thread in each component that accepts variable names from standard input and prints the value of the corresponding variable.
  - This will help you debug race conditions and deadlocks (one or more threads may hang).
  - To debug Worker-related issues, examine variables that store the number of slots available, or the state of the tasks executing on the Worker.
  - For the Master, variables that store counters for RR or the state of the Workers in the system may give you valuable insight.
  - Additionally, you could take as input a keyword “all” that dumps the entire internal state of the component.

BIG DATA

Evaluation

BIG DATA

Marks Breakup

S. No	Feature	Marks
1	Master listens on port 5000 and receives job requests	1
2	Workers started according to config	1
3	Scheduling Algorithm in Master	3 (1 each)
4	Tasks received by Worker	1
5	Task execution simulated	2
6	Task completion update processed by Master	1
7	After map tasks, dependent reduce tasks launched	3
8	Part 2, result 1	2
	Part 2 result 2	2
9	Report – illustrating design, assumptions made, and a list of all challenges your team had to solve	2
10	Viva and Demo	2

BIG DATA
Submission

- Due Date: 01/12/2020
- Final Report: the submission has to be accompanied by a final report which should outline
  - your design
  - assumptions made during design
  - the results of your experiments and analysis.
  - challenges faced by team
- Note: During evaluation, you may be asked to modify the config and run the simulation for the new config, and also generate the results again.

THANK YOU

K V Subramaniam
Computer Science and Engineering
subramaniamkv@pes.edu
+91 80 6666 3333 Extn 877