4/30/2018

Texas A&M University

Cloud Task Scheduling Algorithms

**TABLE OF CONTENTS**

[TIME SPECIFICATIONS & PROJECT MEMBER INFO 1](#_Toc512888571)

[PROJECT SPECIFICATION: INTRODUCTION & PURPOSE 2](#_Toc512888572)

[BACKGROUND RESEARCH 3](#_Toc512888573)

[DESIGN OF SIMULATOR 4](#_Toc512888574)

[ALGORITHMS SELECTION & DESIGN 5](#_Toc512888575)

[PROJECT IMPLEMENTATION & DESIGN OF PARSER 8](#_Toc512888576)

[INSTRUCTIONS TO RUN PROJECT 10](#_Toc512888577)

[ANALYSIS, FINDINGS AND CONCLUSIONS 11](#_Toc512888578)

[REFERENCES 18](#_Toc512888579)

# TIME SPECIFICATIONS & PROJECT MEMBER INFO

The project team was composed of the following people:

* Johana Rueda - UIN: 625007802
* Denishkumar Patel - UIN: 522001683
* Sameer Kumar Behera - UIN: 526004296
* Vishakh Shukla - UIN: 221004930

|  |  |
| --- | --- |
| **Task/Milestone** | **Hours** |
| Research | 20 |
| Analyze Dataset | 30 |
| Design Simulator | 20 |
| Implementation of Algorithms | 40 |
| Analysis of Algorithms | 20 |
| Project Report | 10 |
| **Total Hours** | 140 |

GitHub Link: <https://github.tamu.edu/Sameer-TAMU/689-18-a-P2.git>

# PROJECT SPECIFICATION: INTRODUCTION & PURPOSE

Cloud computing has completely re-shaped the world or technology. Now, for CTOs it is no longer a question about whether or not they should use the cloud, it is more of how they are going to user it, and how quickly they can get to where they need to be as far us utilization of IT resources. Cloud computing is not new though, it dates back to the 1960s, but since the internet started to offer significant bandwidth until the 1990s, cloud computing for the masses has been something that has taken longer to develop.

Cloud computing is a distributed computing model, that enables developers to automatically deploy an application by sharing computing power with other users. The sharing of computing resources is what allows the cloud to provide the amount of benefits and savings that it can provide to its users. In this environment, task scheduling is a key task/problem that must be resolved in order to provide the best service possible. These scheduling algorithms are meant to take into account, priority, arrival time, expected execution time and many more factors into consideration to be able to process the tasks in a manner that starvation is minimized and that important tasks are prioritized.

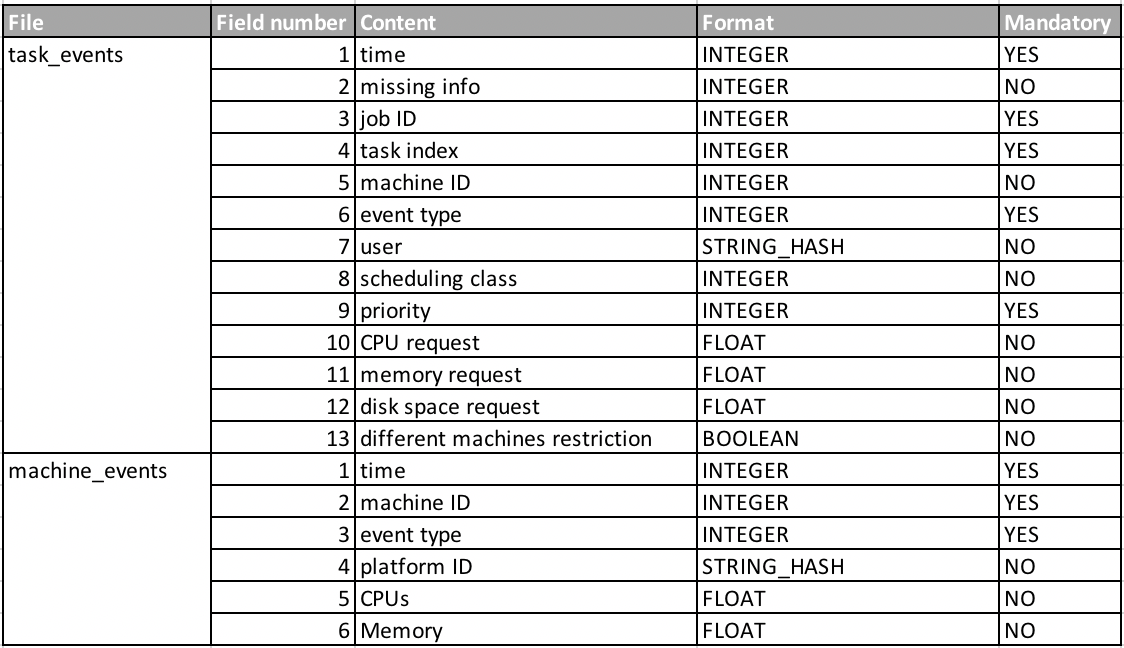
Our project team has researched and developed a couple of the algorithms that have been designed and compared their results. We have also developed a simulator in order to be able to simulate a fake cloud, with fake machines and fake tasks to be able to analyze and compare the performance of the algorithms.

The main objective of our project is to inform ourselves and the reader about the different types of algorithms, a possible design for them, and what the designer of these algorithms have to take into account when developing a solution for the task scheduling problem. As you will see in our conclusion, no algorithm is perfect for every situation, but rather some algorithms perform better than others in certain situations.

# BACKGROUND RESEARCH

The first thing that was researched was the available datasets that we could utilize for the project. The first dataset that was explored was the Microsoft Azure VM dataset from the paper, “Resource Central: Understanding and Predicting Workloads for Improved Resource Management in Large Cloud Platforms” featured in SOSP’17. It was discovered that this dataset did not contain the list of tasks and other task information. Thus, we had to look for other datasets.   
  
The other dataset that we explored was the Google cluster dataset. This data set has 2 versions, the first one was a very simple version but it did not have many details on the tasks and machines. In the other hand, the second version was very complete and had a all the data required for us to do this project.

The following is a summary of the Schema for the data that we used from the second version of the dataset:



Given this schema, we proceeded to design the simulator for the “fake” cloud and the data parser. The following sections will describe the design for each. Note that we did not have a need to use all the fields and picked out the ones we needed.

# DESIGN OF SIMULATOR

The first phase of the design of the simulator was developing a skeleton of what the application was going to look like. We had 2 types of data, machine and task data. Those 2 became what we called our models. Our machine model has the following methods:

1. Initialization or constructor method- this method sets the id, max\_cpu, max\_mem, for that machine. It also sets the in\_use\_mem, in\_use\_cpus for when we allocate tasks, and an array of the current tasks scheduled for that machine
2. check\_cpu\_available
3. check\_mem\_available
4. check\_num\_curr\_tasks- return number of the current tasks
5. allocate\_mem
6. deallocate\_mem
7. allocate\_cpu
8. deallocate\_cpu
9. add\_task
10. get\_current\_tasks- return the array of the current tasks
11. get\_id- returns the machine ID

Our task model that we developed was a little more complicated than the machine. The following where the methods for this model:

1. get\_task\_id
2. get\_cpu\_requested- gets the CPU requested for the specific task
3. get\_mem-requested- gets the memory requested for the specific task
4. get\_pred\_exec\_time
5. get\_remaining\_time- gets the time remaining for the task to be complete
6. get\_status-
7. get\_started\_time- saves the started time for the task
8. get\_started- boolean value that states if the task started
9. get\_time\_run- this is used for the Round robin algorithm. It compares this number with the quanta to know when to stop the task
10. update\_remaining\_time
11. update\_started\_time
12. update\_arrival\_time
13. update\_to\_finished
14. update\_to\_started
15. update\_time\_run
16. reset\_time\_run

The Simulator is composed of 2 documents. The one titled run.py which is the actual simulator, and the helper\_methods.py. The simulator has the following global variables task\_cue which is the cue for for tasks that have arrived. Global timer it is not in actual seconds but rather iterations of the loop. We also keep a finished tasks array, a cluster array with all the available machines and a task list which are the tasks that are put for the scenario. The simulator is basically a while loop, that only stops when the amount of finished tasks is equal to the amount of tasks arrived. The helper\_methods have the most important methods for the simulator, which are the update\_machine methods. These methods, take the tasks that are currently running in the machine and update the remaining time for that task to be able to finish. When a task is finished, this method is also responsible for releasing the cpu and memory so another task can start. **Our timing is measured in the number of iterations of the global loop since real timing would not make sense in this scenario as we are simulating the tasks.**

The reason why we have 2 machine update methods is because we have one that is specifically for round robin, as this algorithm stops the tasks after the quanta time has been met as we will later discuss.

# ALGORITHMS SELECTION & DESIGN

***1) First Come First Serve Algorithm -*** FCFS is an operating system process scheduling algorithm and a network routing management mechanism that automatically executes queued requests and processes by the order of their arrival. With first come, first serve, what comes first is handled first; the next request in line will be executed once the one before it is complete. FCFS accumulates the tasks and queues them if the resources are bust and then assigns the resources once freed, to the tasks on the basis of their arrival time. FCFS provides an efficient, simple and error-free process scheduling algorithm that saves valuable CPU resources. It uses nonpreemptive scheduling in which a process is automatically queued and processing occurs according to an incoming request or process order.

Let's take a look at how FCFS process scheduling works. Suppose there are three processes in a queue: P1, P2 and P3. P1 is placed in the processing register with a waiting time of zero seconds and 10 seconds for complete processing. The next process, P2, must wait 10 seconds and is placed in the processing cycle until P1 is processed. Assuming that P2 will take 15 seconds to complete, the final process, P3, must wait 25 seconds to be processed. FCFS may not be the fastest process scheduling algorithm, as it does not check for priorities associated with processes. These priorities may depend on the processes' individual execution times.

***2) Priority-based Job Scheduling Algorithm* -** PJS is a method of scheduling processes based on priority. In this method, the scheduler chooses the tasks to work as per the priority, which is different from other types of scheduling. Priority scheduling involves priority assignment to every process, and processes with higher priorities are carried out first, whereas tasks with equal priorities are carried out on a first-come-first-served (FCFS) basis.

Priorities can be either dynamic or static. Static priorities are allocated during creation, whereas dynamic priorities are assigned depending on the behavior of the processes while in the system. To illustrate, the scheduler could favor input/output (I/O) intensive tasks, which lets expensive requests to be issued as soon as possible.

Priorities may be defined internally or externally. Internally defined priorities make use of some measurable quantity to calculate the priority of a given process. In contrast, external priorities are defined using criteria beyond the operating system (OS), which can include the significance of the process, the type as well as the sum of resources being utilized for computer use, user preference, commerce and other factors like politics, etc.

Indefinite blocking, otherwise called starvation, is one of the major issues concerning priority scheduling algorithms. It is a state where a process is ready to be executed, but faces a long wait in getting assigned to the CPU.

It is often possible that a priority scheduling algorithm can make a low-priority process wait indefinitely. For example, in an intensely loaded system, if there are a number of higher priority processes, the low-priority processes may never get the CPU for execution. A remedy to starvation is aging, which is a technique used to gradually increase the priority of those processes that wait for long periods in the system.

***3) Round Robin Scheduling Algorithm -***RRS is a method where each process is assigned a fixed time slot in a cyclic way. (RRS) is a job-scheduling algorithm that is considered to be very fair, as it uses time slices that are assigned to each process in the queue or line. Each process is then allowed to use the CPU for a given amount of time, and if it does not finish within the allotted time, it is preempted and then moved at the back of the line so that the next process in line is able to use the CPU for the same amount of time.

It is mainly used by operating systems and applications that serve multiple clients that request to use resources. It handles all requests in a circular first-in-first-out (FIFO) order and eschews priority so that all processes/applications may be able to use the same resources in the same amount of time and also have the same amount of waiting time each cycle; hence it is also considered as cyclic executive.

It is one of the oldest, simplest, fairest and most widely used scheduling algorithms of all time, partly because it is very easy to implement as there are no complicated timings or priorities to consider, only a FIFO system and a fixed time constraint for each usage of the resource. This also solves the problem of starvation, a problem in which a process is not able to use resources for a long time because it always gets preempted by other processes thought to be more important. The disadvantage of this algorithm is the overhead that the context switching takes.

***4) Minimum Completion Time Algorithm -*** MCT algorithm assigns tasks to VMs or resources based on the best predictable completion time for that task in random order. Each task is assigned to the VM or resource that has earliest completion time. With MCT algorithm, some tasks are allocated to the VMs or resources having no minimum execution time. MCT considers the load of the machine before scheduling a job on that machine. It selects machines for task scheduling on the basis of the expected minimum completion time of tasks among all the machines available. MCT show better results for achieving the degree of imbalance for task scheduling.

The biggest drawback of using MCT for task scheduling is that it is too slow and there is no optimization for the selection of the best resource. MCT algorithm also performs poorly in finding the cost makespan and throughput time in both homogeneous and heterogeneous environment with and without workload traces, whereas it performs averagely as compared with other algorithms in finding the degree of imbalance in both scenarios for the task scheduling in IaaS cloud computing.

# PROJECT IMPLEMENTATION & DESIGN OF PARSER

Google cluster data provided a detailed schema report of what the data in the files mean as well as some of the things to look for when parsing in the data. After going through the documentation we narrow down the parsing to just 2 main files which ‘task\_events’ and ‘machine\_events’. From machine events we created a machine class and parsed in the following data:

**class** Machine:

**def** \_\_init\_\_(self, id, max\_cpu\_cores, max\_mem):

self.id = id

self.max\_cpu = int(max\_cpu\_cores) **if** int(max\_cpu\_cores) > 0 **else** 1

self.max\_mem = float(max\_mem)

self.in\_use\_mem = 0

self.in\_use\_cpus = 0

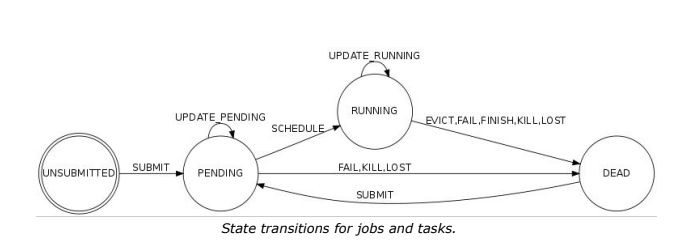
self.current\_tasks = []

self.scheduled\_tasks = []

self.number\_of\_finished = 0

The if condition prevents a machine that does not have cpu, the reason for this is because google normalized there data using some hash method so the actual values are unknown. We used a multiplier in constants.py to make the values more sensible than what is present in the files.

The tasks we also parsed in a similar way. The dataset has task events which can be in one of the states below:



● SUBMIT (0): A task or job became eligible for scheduling.

● SCHEDULE (1): A job or task was scheduled on a machine. (It may not start running

immediately due to code-shipping time, etc.) For jobs, this occurs the first time any

task of the job is scheduled on a machine.

● EVICT(2): A task or job was descheduled because of a higher priority task or job,

because the scheduler overcommitted and the actual demand exceeded the machine

capacity, because the machine on which it was running became unusable (e.g. taken

offline for repairs), or because a disk holding the task’s data was lost.

● FAIL(3): A task or job was descheduled (or, in rare cases, ceased to be eligible for

scheduling while it was pending) due to a task failure.

● FINISH(4): A task or job completed normally.

● KILL(5): A task or job was cancelled by the user or a driver program or because

another job or task on which this job was dependent died.

● LOST(6): A task or job was presumably terminated, but a record indicating its

termination was missing from our source data.

● UPDATE\_PENDING(7): A task or job’s scheduling class, resource requirements, or

constraints were updated while it was waiting to be scheduled.

● UPDATE\_RUNNING(8): A task or job’s scheduling class, resource requirements, or

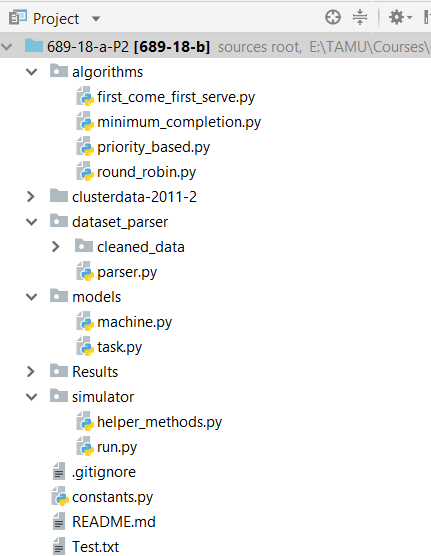
constraints were updated while it was scheduled.

The dataset has a column that is integer values as above and we used these values to get the correct data out of the dataset. We also used these events to calculate **execution time**. We used a dictionary compress the data using the job\_id+task\_id to represent a unique task id and as we parse in the data we kept track of what events were happening. we than simply too the scheduled event and the finish event and the difference of those timestamp was the execution time. The task events table also contained information such as cpu requested, memory requested, etc. we parsed those data by transforming the normalized values and than creating objects using our task class.

All the code for parser is inside of the dataset\_parser/parser.py file.

# INSTRUCTIONS TO RUN PROJECT

The following is the structure of the project:



To Run the Project, kindly follow the instructions below:

* Clone the project and retain the structure.
* Download python: <https://www.python.org/downloads/release/python-343/>
* Instructions to set path on windows: <https://docs.python.org/3/using/windows.html>
* Navigate to root directory '689-18-a-P2'
* You might have to do export PYTHONPATH=.
* You must run code from 689-18-a-P2 directory and not any sub directory.
* The command to run the project is as follows:

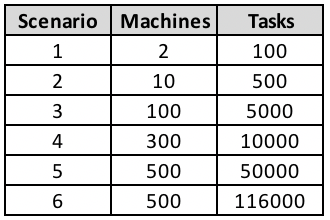
**python simulator/run.py <algo\_name> <num\_of\_machines> <num\_of\_tasks>**

* <**algo\_name**> = RR (Round Robin), PJS (Priority Job), MCT (Minimum Completion Time), FCFS (First Come First Serve)
* <**num\_of\_machines**> = Integer in the range 1 - 12555
* <**num\_of\_tasks**> = Integer in the range 1 - 116000
* Example of the Command:

**python simulator/run.py FCFS 500 50000**  
  
 - This command is for first come first serve for 500 machines and 50000 tasks.

# ANALYSIS, FINDINGS AND CONCLUSIONS

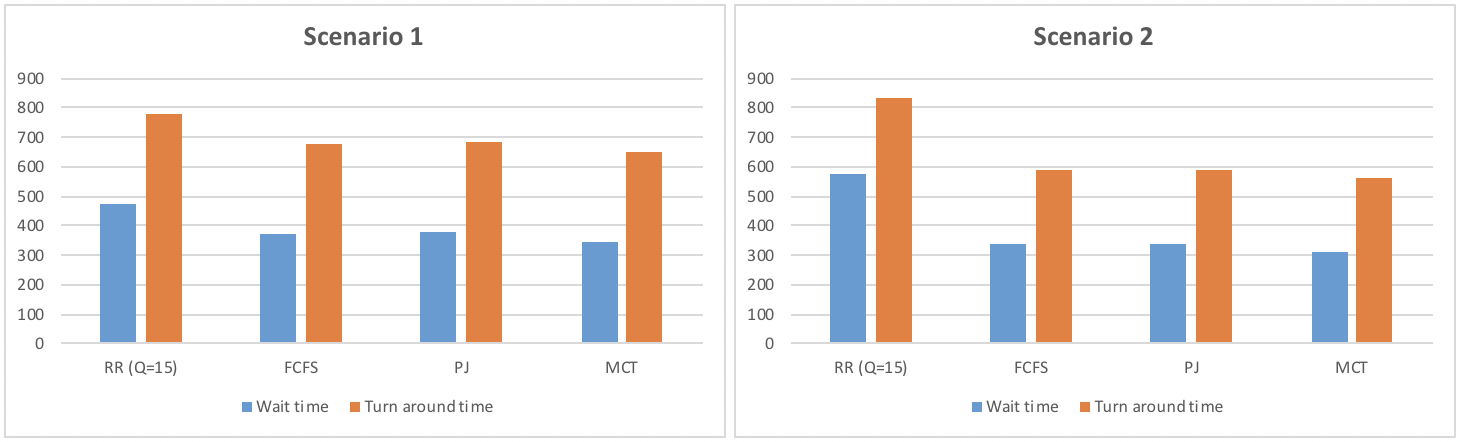
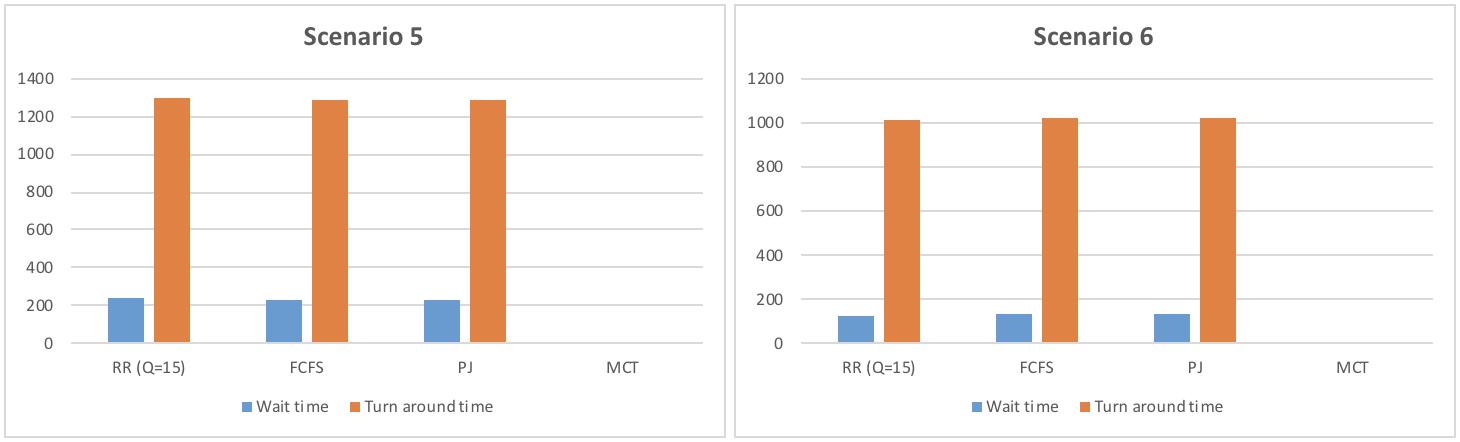
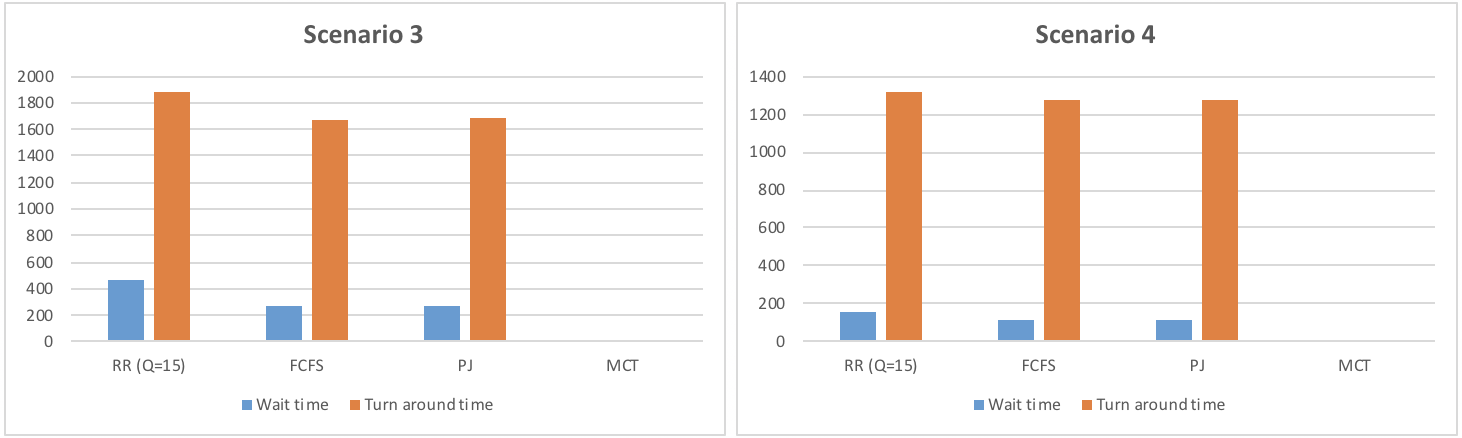
To analyze the algorithms designed we took 6 scenarios, each one took a number of machines and a number or tasks from the google dataset, then we ran it with each of the algorithms and recorded the results. The six scenarios were as follows:



We analyzed the following aspects for each of the scenarios:

1. Average wait time for each task
2. Turn Around time for each task, also known as response time
3. Total time to finish all the tasks
4. And average number of tasks that are completed per iteration

The following are the comparison charts between wait time and turnaround time:



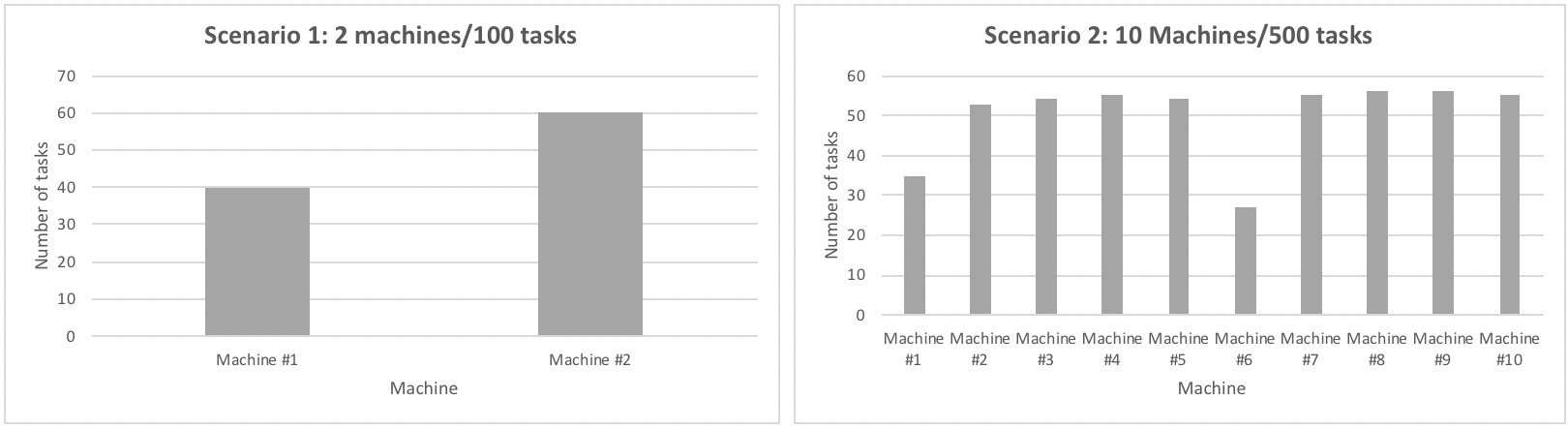
The following table summarizes all the algorithms with the metrics chosen to evaluate them:

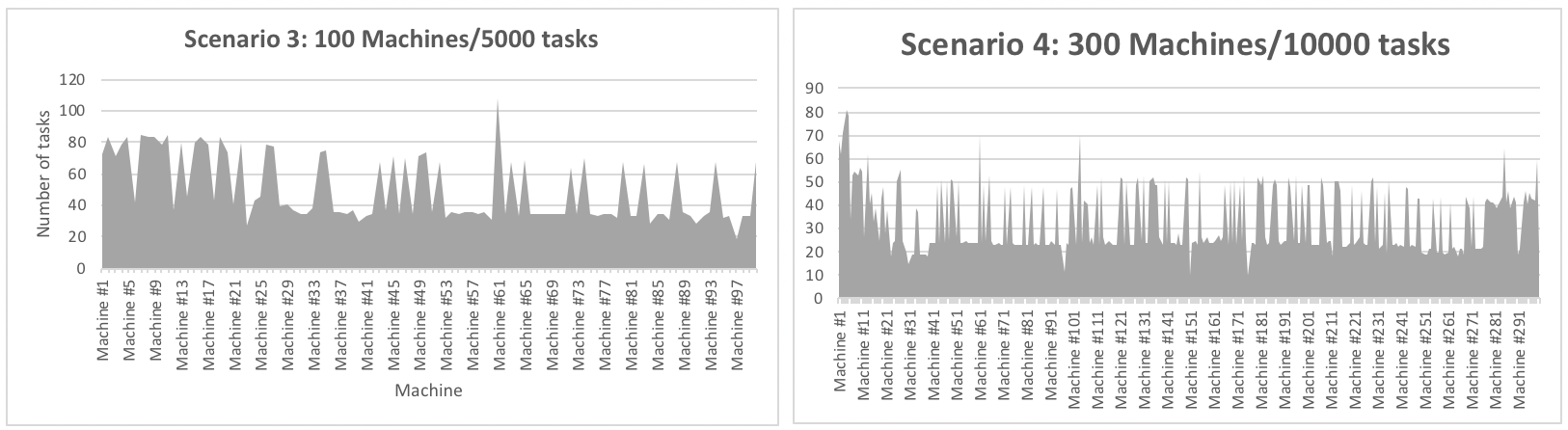
****

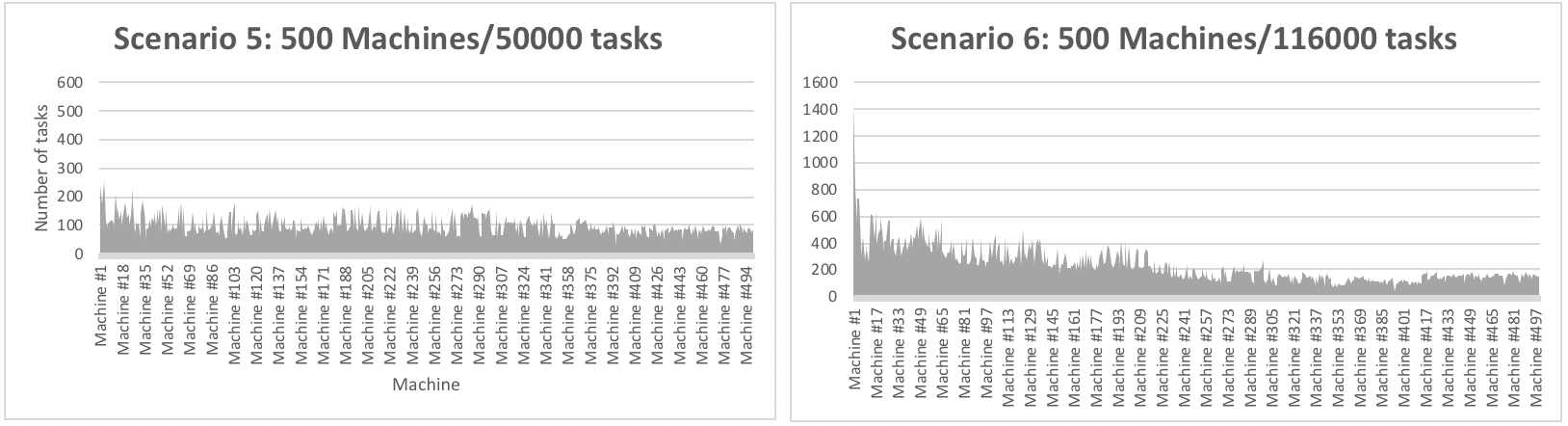
**Table comparing wait time, turnaround time, and total time taken by each algorithm.**

**MCT took too long in python to finish for the last 4 cases.**

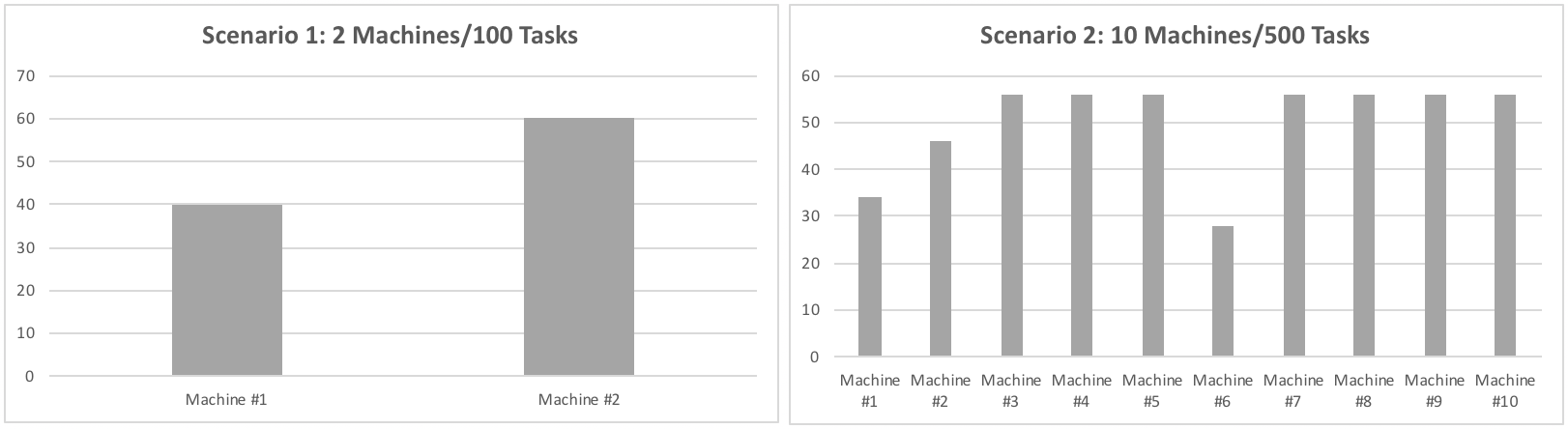
**The following is the FCFS Load Performance for the 6 Scenarios:**

****

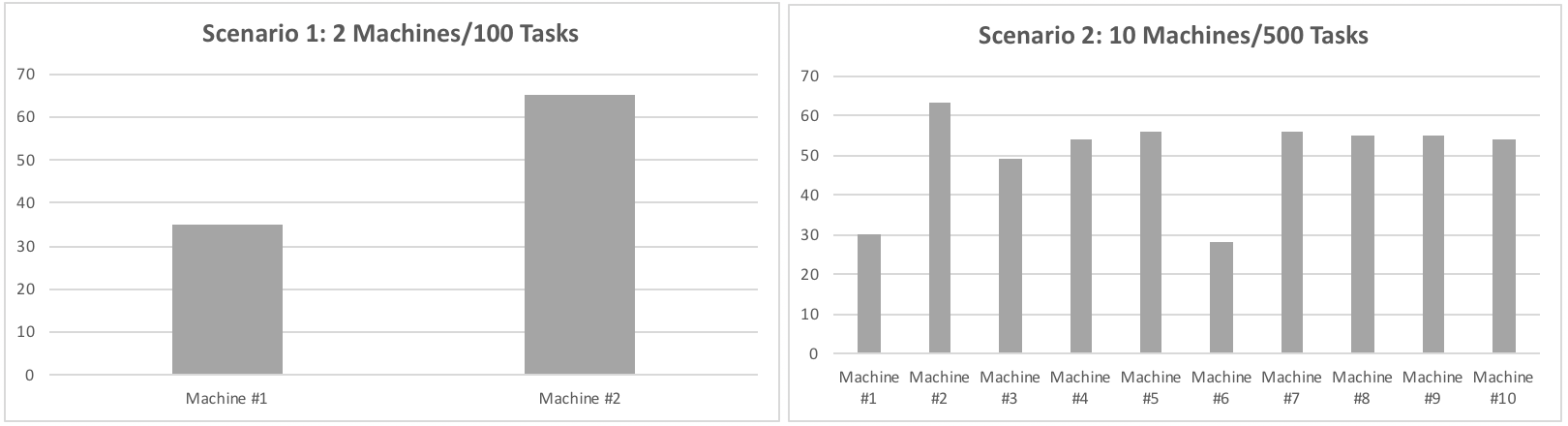
****

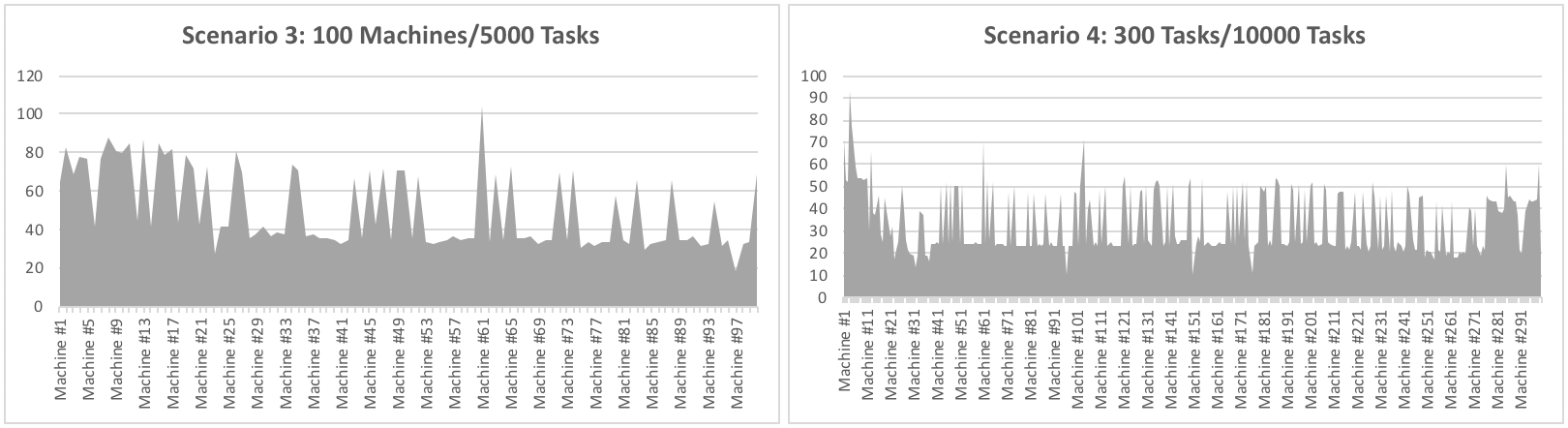
****

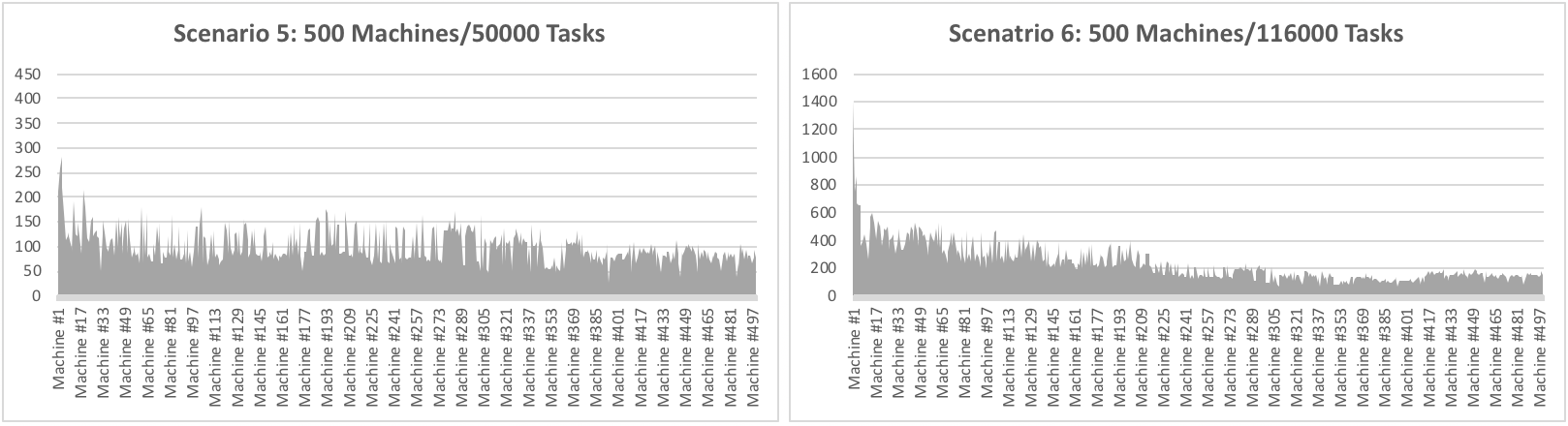
**The following is the MCT Load Performance for the 2 Scenarios:**

****

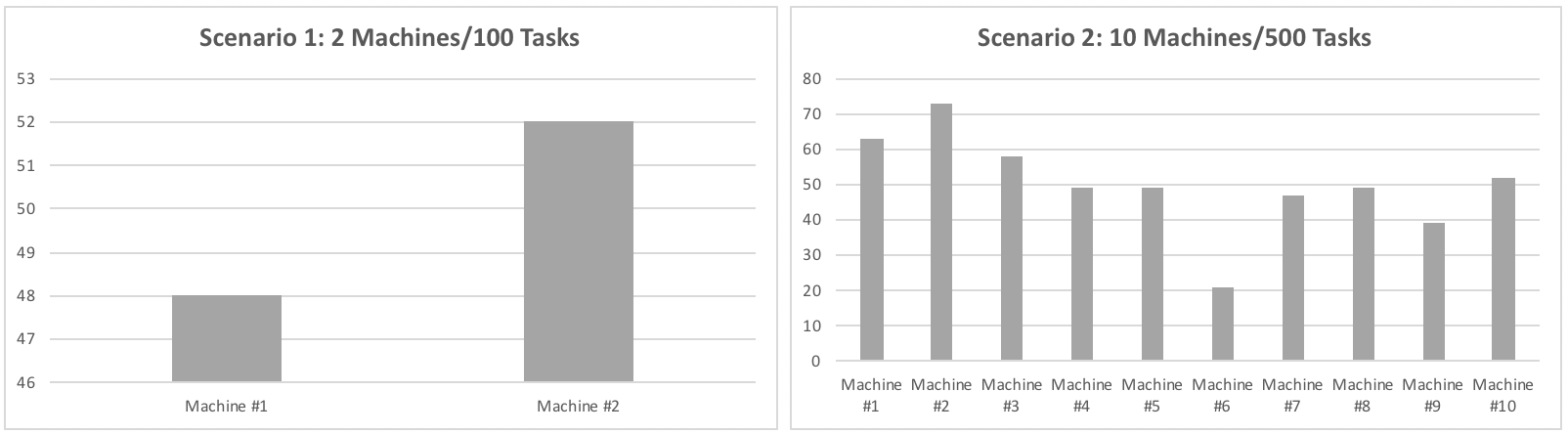
**The following is the PJS Load Performance for the 6 Scenarios:**

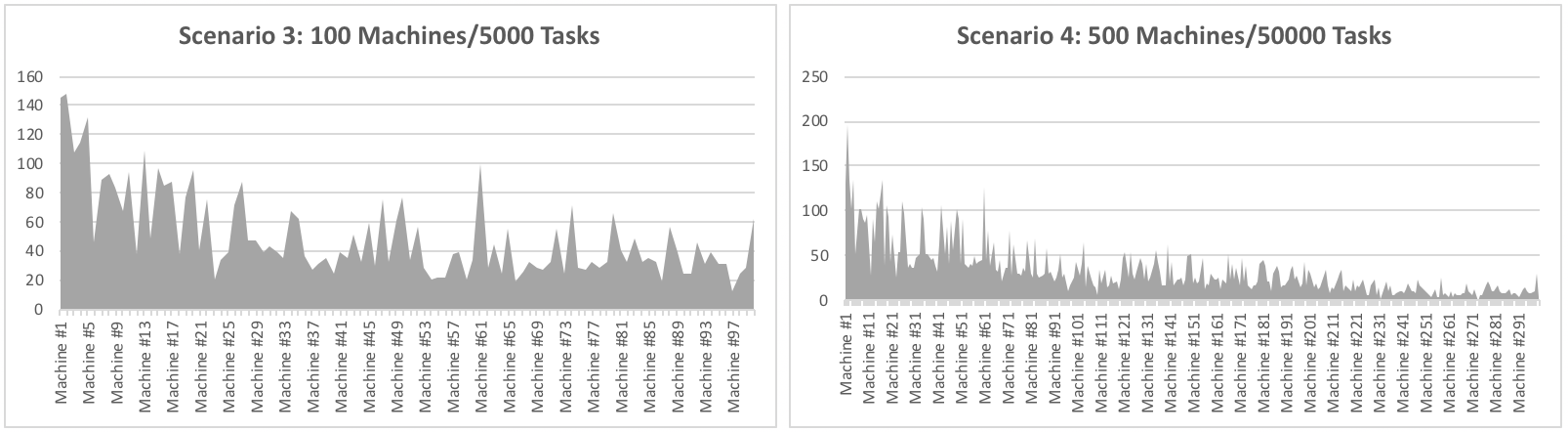
****

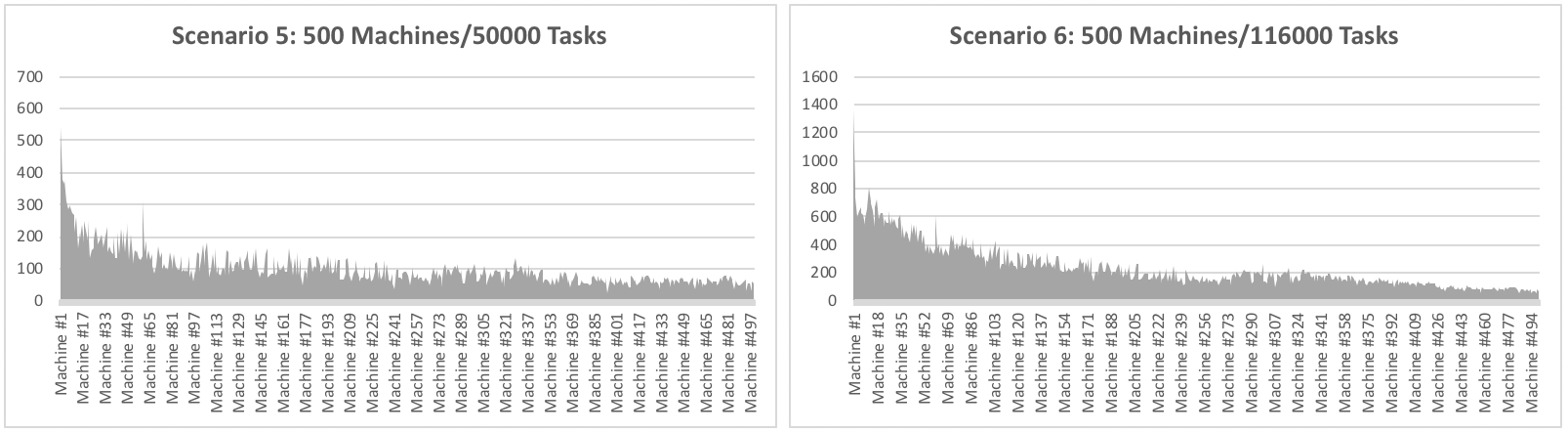
****

****

**The following is the RR Load Performance for the 6 Scenarios:**

****

****

****

As stated earlier, cloud computing has completely re-shaped the computing world. To meet the cloud needs of enterprises and other entities, it is vital that that cloud resources are efficiently and effectively used. For this, there are several task scheduling algorithms that are widely available. However, each of these algorithms has their advantages and disadvantages. In this project, we did a study of several algorithms namely First Come First Serve (FCFS), Round Robin (RR), Priority Based Job Scheduling (PJS), and Minimum Completion Time Algorithm (MCT). These scheduling algorithms use metrics such as task priority, arrival time, expected execution time, and many other factors.

For analyzing and studying these algorithms, our team researched and developed these algorithms and compared their results. For simulating these algorithms and analyzing these algorithms, we also developed a simulator to simulate a fake cloud and we also invoked fake tasks and machines. Our main objective in this project was to compare the performance of these algorithms.

We found that FCFS was a good baseline measurement for other algorithms. The other algorithms do comparably well to FCFS. RR does better in some cases, for example case 1 and case 6, and does a little bit worse or just as good in the other cases. RR depends on the quantum so depending on the scenario it could be better than FCFS in some cases while it could end up just adding overhead in other cases which can be seen in our results. MCT does better in just about in all cases however the complexity on MCT limited to what test scenarios we can run with Python as we can see that we were only able to complete the run in the 2 small cases while the other cases took way too long and didn’t have time to finish the runs. MCT would be the most optimal in all of this algorithm as it’s also doing load balancing. We can also judge the algorithms on other metrics such as turnaround time, and wait time.

We also found that FCFS was an easy algorithm to implement. FCFS used arrival time as the only input for task scheduling, no other logic is used when the task is being scheduled. FCFS is therefore the simplest of the algorithms we developed. The RR algorithm used arrival time and quantum time for the logic in scheduling. This algorithm prevents starvation and gives the illusion of a shared, continuous progress (depending on your quantum time), and balances the load. However, preemption is required for this algorithm, and that adds overhead, which in some cases is not worth it. For MCT, we used expected completion time to schedule the tasks. The disadvantage of MCT was that the algorithm was difficult to implement. The other disadvantage was that it does not use any optimization for selecting the best resource. However, the advantage of MCT is that it also uses load balancing and it is the more efficient algorithm than the others we analyzed. However, as stated, we were only able to conduct experiments with smaller input data because of Python’s inefficiency to perform millions of operations, this would have been easier in C++. Lastly, PJS took into consideration the priority of tasks. The disadvantage of PJS is that it uses priority of tasks which isn’t ideal in some scenarios.

# 

# REFERENCES

<http://ieeexplore.ieee.org/document/7855977/>

<https://github.com/google/cluster-data>

<http://journals.plos.org/plosone/article/file?id=10.1371/journal.pone.0176321&type=printable>

<https://www.geeksforgeeks.org/gate-notes-operating-system-process-scheduling/>

<https://www.tutorialspoint.com/operating_system/os_process_scheduling_algorithms.htm>

<https://www.researchgate.net/publication/257726215_A_Priority_Based_Job_Scheduling_Algorithm_in_Cloud_Computing>

<https://stackoverflow.com/questions/20864641/finding-minimum-completion-time-of-scheduled-tasks-with-topological-sort>

<https://www.researchgate.net/publication/254052145_Minimum_Completion_Time_for_Power-Aware_Scheduling_in_Cloud_Computing>

<https://www.techopedia.com/definition/23455/first-come-first-served-fcfs>

<https://www.studytonight.com/operating-system/round-robin-scheduling>