National College of Ireland

An improved task Scheduling Algorithm for Segregating User Requests to different Virtual Machines

MSc Research Project
Cloud Computing

Sreenivasa Prasad Kakumani
Student ID: x18146686

School of Computing
National College of Ireland

Supervisor: Horacio González-Vélez

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Sreenivasa Prasad Kakumani
Student ID:	x18146686
Programme:	M.Sc in Cloud Computing
Year:	2019
Module:	MSc Research Project
Supervisor:	Horacio González-Vélez
Submission Due Date:	12/12/2019
Project Title:	An Improved task Scheduling Algorithm
Word Count:	7639
Page Count:	29

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

I agree to an electronic copy of my thesis being made publicly available on TRAP the National College of Ireland's Institutional Repository for consultation.

Signature:	Sreenivasa Prasad Kakumani
Date:	3rd February 2020

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

An improved task Scheduling Algorithm for Segregating User Requests to different Virtual Machines

Sreenivasa Prasad Kakumani
x18146686
MSc Research Project in Cloud Computing

3rd February 2020

Abstract

Resource Management is key for maintaining trust, improving the user experience by meeting the user requirements and demands within the SLA as agreed by cloud providers. An efficient task scheduling algorithm has to be implemented to overcome problems such as the non-uniform load distribution and the optimal use of resources. An improved task scheduling algorithm based on traditional Longest Job First (LJF) and Shortest Job First (SJF) is developed in this paper in order to allocate the long-running jobs to a pool of Virtual Machines (VMs) and short-running jobs to another pool of VMs parallelly. Three different VM configurations are considered with a pre-defined total VM count and the simulations are executed for every combination of VMs based on total VM count. The evaluated results are then compared to LJF and SJF and observed that the total execution cost, waiting time, completion time are best for the proposed model. The proposed model is successful in finding the best possible combination of VMs to be selected depending on the workload.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

Cloud Resource Management includes different steps in the sequence which includes request submission to request execution process in the cloud. As the demands from the end-user side are increasing, Cloud resource management is gaining significantly high importance and this process aims at emphasizing in meeting the user expectations through effective distribution of cloud resources. It has become a challenge to manage these resources effectively due to heterogeneity nature in physical machines, also huge fluctuations in workload distributions due to unpredictable and uncontrollable loads which leads to interdependent issues in the cloud environment. All these challenges and difficulties in the resource management process can be handled by three cloud models - IaaS, PaaS and SaaS. Moreover, efficient resource management strategies help in achieving the advantage of reduced power consumption which is another biggest problem in cloud computing. In a simplified manner, cloud resource management is a process of effectively and efficiently managing cloud resources by considering user requirements, agreed to SLA and a few other factors in order to improve Quality of Service (QoS) [1].

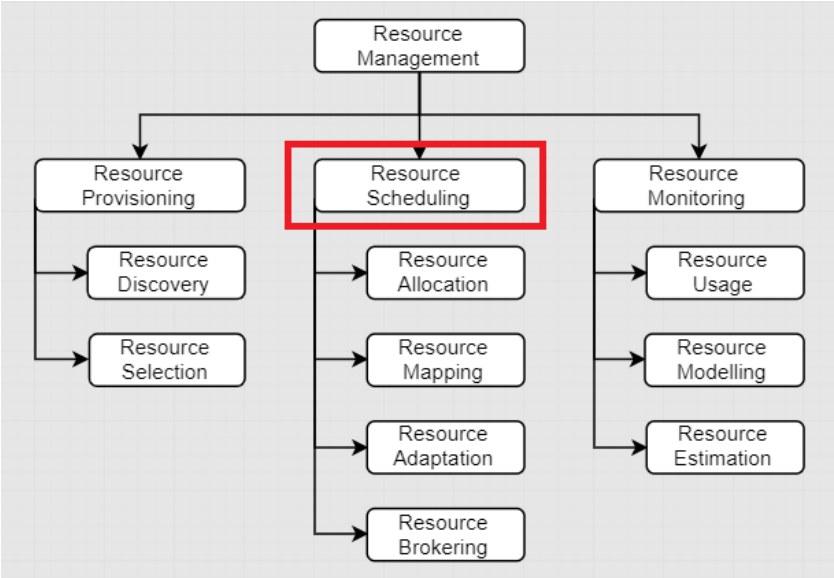

Figure 1: Resource Management Process classification based on [1]

We often deal with three terms during the Resource Management process and these are sub-divided into several functions as shown in [1]

1.1 Resource Provisioning

Resource Provisioning is defined as the process of allocating services to end-users from the cloud providers. The development of provisioning policy is one of the major issues in the field of cloud computing as it is directly related to the enhancement of QoS parameters. The process of provisioning involves two functions:

- **Resource Discovery:** It is the process of discovering the available resources.
- **Resource Selection:** It involves the selection of the best resource, based on different QoS parameters as requested by the cloud user.

1.2 Resource Scheduling

Scheduling is a complex task in the entire Resource Management process because of several reasons like geographical resource distribution, uneven load, varying prices, etc. There are various algorithms proposed in the last two decades but still there exist problems like heterogeneity, dispersion, etc. The process of Scheduling involves four functions:

- **Resource Allocation:** It involves the process of allocating resources among the users economically.
- **Resource Mapping:** It actually manages the mapping between required resources by users and the available resources with the provider.
- **Resource Adaptation:** It is defined as the system's ability to dynamically adjust the available resources depending on the user requirement.
- **Resource Brokering:** It is defined as the negotiation process in which an agent will always ensure that the required resources are available whenever needed to complete the objective of SLA.

1.3 Resource Monitoring

It is important to effectively monitor cloud resource utilization to optimize the performance of the data centre. Monitoring process involves three functions:

- **Resource Usage:** The monitoring system compiles physical resource uses by calculating performance based on CPU and main memory.
- **Resource Estimation:** The number of resources required to run a current application is based on estimating the resources available.
- **Resource Modelling:** Resource Modelling depends on the information provided by elements in the network about the resources.

1.4 Need for Research

Resource Scheduling is a crucial step in the process of Resource Management because two major issues are involved in it. The first one relates to uneven load distribution and the second one being not able to utilize the resources in an effective way. The best way to overcome this issue is to develop an efficient and capable algorithm and these are classified into several types. Some of these include First Come First Serve (FCFS), Shortest Job First (SJF), Longest Job First (LJF), Particle Swarm Optimization (PSO), etc. Though there are several works done in this area still there is a lot of scope for research because of its criticality and the above mentioned two issues. Hence, we developed a new approach for assigning tasks to Virtual Machines (VMs) in the best possible way and compared it with existing algorithms. A brief understanding of the chosen research area and the relevant work done is presented in Section 2. Section 3 describes the choice of methods employed when conducting the research work. Section 4 gives an insight on the architecture and techniques involved in the proposed work. Section 5 describes the implementation of proposed algorithm, tools used and outputs produced. Section 6 analyses the results produced and compare it with existing models with the

inclusion of tables and graphs and finally Section 7 provides a detailed summary of the work done and also the scope for future research work.

1.5 Research Question

1. What is the impact/outcome of scheduling the long-running and short running user tasks to different VMs?

The primary motivation for conducting this research work is to improve the QoS parameters like waiting time, completion time, etc. because the execution costs depends directly on the completion time of tasks irrespective of other parameters. Hence, a new scheduling technique is employed to separate both long jobs and shorter tasks and getting them executed on different VMs at a single time. This work is extended for figuring out the best possible combination of VMs that holds good to execute the dataset i.e chosen irrespective of scheduling algorithm.

2 Related Work

There are several studies that clearly explains the importance of task scheduling techniques and key problems for each of the techniques. In this section, a brief analysis of many such processes involved in the cloud computing paradigm is discussed. Kumar et al. [2] and Arunarani et al. [3] performed a deep comprehensive survey on different scheduling techniques in the field of cloud computing by explaining the importance and current issues related to each of the techniques and each paper provides an insight for effective resource scheduling. These surveys also extended the support for identifying the current issues in scheduling and in turn helped to enhance the performance characteristics of existing algorithms. In computing, Resource Allocation (RA) is defined as the process by which available resources are provided to the requested applications. Resource scheduling in cloud computing is often done in two ways: demand scheduling in which cloud provider randomly allocates the resources and it leads to a problem of allocating tasks to a single resource and the second being reserved scheduling where there exists a problem of being idle not performing any work.

There is another study provided by Singh and Chana [4] to overcome these problems of over provisioning and under provisioning by providing an efficient solution that can understand the workload prior to the scheduling event. The key here is to provide resources to users without violating SLA and upholding user trust. Also, there are several parameters like cost, makespan time, completion times, etc. to be considered before scheduling the tasks to different instances [2]. The main process of categorizing scheduling algorithms involves two parts: dynamic and static scheduling. Dynamic task scheduling is the most effective procedure among the two because it does not require any prior information about the task details whereas in Static scheduling we need to have prior information about the tasks that need to be executed. Each of these scheduling algorithms will be primarily focused on enhancing the performance of some key parameters while not considering other parameters i.e each algorithm has its own advantages and disadvantages based on several parameters. There are hundreds of algorithms that serve many purposes depending on the key metrics on which they are developed. This scheduling scheme is majorly classified as heuristic, meta-heuristic and hybrid scheduling.

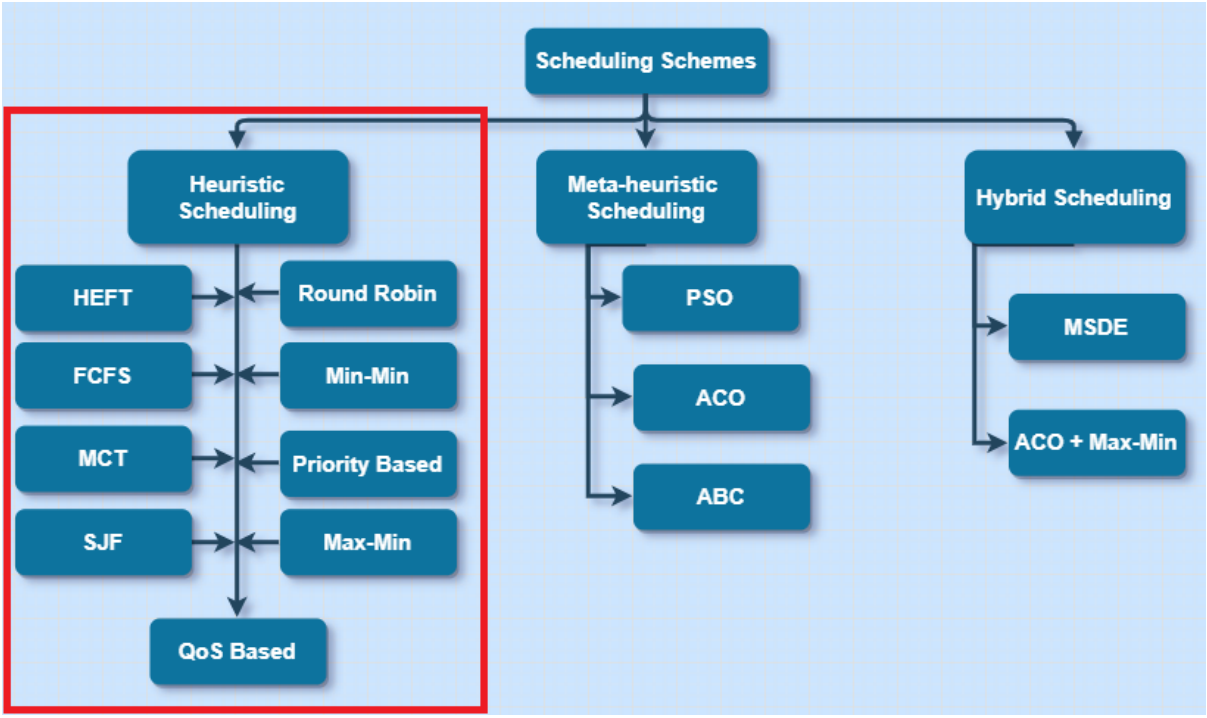

Figure 2: Scheduling Scheme Classification of different Algorithms based on [2]

2.1 Heuristic Scheduling Algorithms

Heuristic algorithms focus on the problem and help in achieving better performance in defined limits of time by providing the exact solution for the specific problem but it yields low performance for other problems. A number of such heuristic algorithms are discussed here based on different categories as shown in Figure 2. Dubey et al. [5] proposed a modified Heterogeneous Earliest Finish Time (HEFT) algorithm in order to overcome the limitations existing in traditional HEFT algorithm and this algorithm is successful in distributing the load to different running VMs by minimizing the makespan time, but it has its own limitations like not able to monitor nodes, not able to share the workload evenly to the resources available. This algorithm is developed based on the concept of calculating individual ranks for all tasks in directed acyclic graphs (DAG). This HEFT algorithm basically works in two phases in which rank is calculated in the first phase whereas assigning processor will be done in the second phase. The rank calculation is done starting with the last node and moving upward in DAG till the root node arrives.

Li et al. [6] proposed a modified Max-Min algorithm for improving resource utilization and decreasing the response time of tasks. The limitation of uneven load distribution in the traditional Max-Min algorithm is the base for this work in which it actually schedules the task with Max task amount and calculates the completion time in each node. Later it checks for the minimum completion time and then assigns the maximum task to this minimum node. The limitation of tasks arriving in the same batch will be executed in traditional Max-Min is taken care of in the modified algorithm by considering consecutive batches. Anousha and Ahmadi [7] developed an improved Min-Min task scheduling algorithm by changing a few steps in the traditional Min-Min algorithm to decrease the makespan time. In Min-Min, all tasks will be sorted i.e tasks with minimum makespan

will be the first in the queue and then it will be assigned to the minimum node to execute it, but in a modified algorithm, before choosing a resource it calculates average completion time and also the standard deviation and based on these results tasks will be assigned to the available resources.

Li and Shi [8] discussed load balancing and proposed an algorithm based on FCFS that every node sends a request and these will be served in the order they arrive in the queue. The tasks have to wait in queue till the previous tasks are completed and they will be executed only in the order they arrive in the queue but this model failed to distribute the load across the available virtual machines. Mondal et al. [9] proposed an efficient algorithm which is based on the criteria of executing the shortest tasks first. This model was compared with the FCFS algorithm and Round Robin algorithms and interestingly this is able to reduce the turnaround time and execution times but there are limitations like the prediction of burst time before CPU actually starts the execution which is not possible. This algorithm failed to enhance QoS parameters because of the uneven distribution of load.

Alworafi et al. [10] introduced an improved model of SJF in which average response times and makespan times are improved when compared to the traditional SJF model, but all these three algorithms [8], [9], [10] failed to achieve stability in distributing the load to virtual machines. None of the above-discussed algorithms considered the priority of tasks and this is crucial with respect to cloud users that they need some tasks to be executed in minimal time. Kumar and Sharma [11] developed a priority aware Longest Job First algorithm in which priority tasks are executed first after sorting them accordingly based on LJF logic and then ordinary tasks will be executed. The results of the proposed algorithm resulted in minimizing execution time and there is an increase in total resource utilization ratio.

Recently, Ponraj [12] developed an optimal VM placement solution that can able to reduce the completion times of job execution for both static and dynamic workloads. This algorithm considers several parameters like QoS, the status of VM, I/O data, and the computing resources depending on the priority-based model to enhance the performance of applications. This model was compared with traditional FCFS and other priority algorithms and interestingly this model resulted in minimizing the overall cost of processing and completion times. The future scope of this paper is focused on enhancing the security measures by considering the profile-based analysis and introducing the work to both hybrid and public cloud environments whereas the present is concentrated only on private cloud.

Panda et al. [13] proposed three algorithms named Minimum Completion Cloud time (MCC), MEdian MAX and Cloud Min-Max normalization (CMMN). In the MCC algorithm, tasks will be allocated to resources by considering the completion time of every task. This algorithm considers the execution times but also considers the load time for assigning tasks to resources available. This results in arbitrary order and every time tasks will be allocated to the best available resource in a queue and this is a major limitation because the best resource is being wasted for smaller tasks. Median MAX is a two-phase scheduling algorithm in which the median will be calculated in the first phase whereas the maximum will be calculated among the available tasks. CMMN is again a two-way scheduling process that is developed to minimize the makespan time. The

results are evidence that the MCC algorithm is able to reduce the makespan time when compared to the other two algorithms but there are certain limitations with each of the three algorithms.

Pradhan et al. [14] modified the existing the Round Robin algorithm to overcome the problem of time quantum. This paper gave the insight to overcome that issue by following dynamic time quantum instead of fixing it initially but it did not provide the optimal time quantum that needs to be maintained but when compared with the existing model it is able to reduce the waiting time. Another impressive work by Devi et al. [15] based on the weighted round robin concept i.e distributing the load circularly but by using time slices. This algorithm is able to enhance the response time and resource utilization ratio but failed to distribute the workload evenly similar to [9], [10]. It is evident from the above literature work done that many of the heuristic scheduling algorithms failed to enhance more than one QoS parameter at a time and to overcome this limitation, [3] surveyed the strategies involved in task scheduling and provided an insight for modeling multi-objective based scheduling algorithms with the combination of two or more existing traditional algorithms.

There are a number of heuristic algorithms developed in the last decade based on several QoS parameters and one of the key parameters includes Energy consumption. Chen et al. [16] developed a novel architecture to overcome the dynamic scheduling issue. Energy-Efficient reactive scheduling (ERECT) algorithm was developed for scheduling tasks to virtual machines by reducing energy consumption, task deadline requirements but this specific algorithm does not consider SLA during the VM migration stage in order to reduce energy consumption. Another study on the energy consumption parameter was done by Chaabouni and Khemakhem [17] by employing the VM migration technique. This algorithm is based on detecting the workload on the host depending on the median and standard deviation which in parallel reduces the energy consumption and overcome the SLA violation observed in [16], but it failed to look other parameters like cost, time and deadline constraint because the migration itself is the time-consuming process. To overcome the SLA limitation observed in [16], Zhou et al. [18] developed two novel energy-aware algorithms based on adaptive three threshold frameworks. This algorithm is successful in minimizing several parameters which include energy, SLA violation, and migration of VM, but failed to consider the priority of tasks similar to [17]. From these three energy-efficient heuristic algorithms [16], [17], [18] it is clear that some of the parameters are violated and not considered while enhancing the energy consumption.

2.2 Meta-Heuristic Scheduling Algorithms

Meta-Heuristic algorithms gained popularity because of their effectiveness while solving complex problems and these are best described as problem independent. These algorithms are the best strategy for eliminating NP-hard optimization issues. These algorithms are often described as the summation of the heuristic approach and the randomization. There are a number of such algorithms developed in the last decade due to their highly reliable nature and some of them include Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), Artificial Bee Colony (ABC), etc. [19], [20], [21]. All these algorithms are mainly focused on obtaining solutions in a limited time. PSO is one such widely used solution works on the nature of birds how they actually discover space searching for food and shelter. Masdari et al. [19] classified the PSO algorithm into several categories.

This paper actually provides the best insight for researchers who are actually working on PSO algorithms because it explains the different categories in PSO algorithms parallelly explaining the advantages and limitations for each of the discussed algorithms.

ACO is another meta-heuristic algorithm for finding an optimal solution to NP-hard problems. This actually works on a similar technique of how the ant finds the shortest path to reach the colonies. Similar to PSO there are many algorithms developed based on traditional ACO. Duan et al. [20] modified existing ACO and developed an algorithm named Pre Ant Policy to overcome the problem of instantaneous peak loads and mapping the tasks to the best available resources and enhancing few QoS parameters. The key motivation for the development of the proposed algorithm is heterogeneity and abundant usage of power and this solution resulted in achieving better makespan time and for maintaining reliability across the system. All these variants of ACO are clearly illustrated with advantages and limitations for solving various problems like scheduling, vehicle routing, etc. in [2], [3].

ABC is another key meta-heuristic scheduling technique inspired by the behavior of bee swarm and this works on the technique similar to bees fly around the space searching for the best solution to a problem. These bees are categorized into three different types based on the behavior they exhibit in search of a food source. Liu et al. [21] proposed a multi-objective ABC algorithm with the help of the traditional ABC algorithm and this model was successful in figuring out the solutions for optimization problems by converting these problems to multi-objective optimization problems. This model was tested against several well-known problems and achieved a positive response for enhanced performance metrics. All the above discussed meta-heuristic algorithms [19], [20], [21] are used in the process of scheduling the tasks to available resources depending on the nature of optimization problems that is to be solved.

2.3 Hybrid Scheduling Algorithms

Hybrid Scheduling Algorithm is another major scheduling approach that is developed from a combination of two or more algorithms that can be either heuristic and meta-heuristic algorithms in order to obtain the appropriate paradigm to solve the scheduling problem. Different combinations of algorithms are selected truly based on the parameters that they are trying to improve. Elaziz et al. [22] developed a hybrid model named Moth Search Algorithm and Differential Evolution (MSDE) by considering parameters like makespan and throughput but the MSDE algorithm majorly focusses on a single objective to reduce makespan time and none of the QoS parameters are considered. This works on the principle of both the Moth Search Algorithm (MSA), which provides exploration and exploitation capabilities based on phototaxis and Levy flight concept and Differential Evolution (DE) algorithm which performs better than MSA for exploitation purpose. Similarly, a number of such hybrid algorithms are developed to serve different purposes but all these algorithms are focused on improving one or two parameters but the future enhancement of these algorithms is focused on improving more QoS parameters.

Based on the reviewed literature, we developed a table as shown in Table 1 to summarize the classification of heuristic, meta-heuristic and hybrid algorithms presented according to their proposed year starting from 2009 to 2019. This table is aimed at providing a quick recap on each algorithm reviewed in our literature along with their

core idea for development, advantages and limitations. In the next section, we will be discussing selected algorithms for our proposed model, simulation methods, techniques we are planning to achieve our goal.

Table 1: Summary of reviewed heuristic, meta-heuristic and hybrid Algorithms

Year	Algorithm	Core Idea	Advantages	Limitations	Ref
2009	FCFS	Serve requests as in the order they come	First come first serve	Not able to distribute load among VMs	[8]
2013 & 2014	Improved min-min & Max-min	To distribute longer task to best available resource	Enhancing utilization rate and minimizing makespan time	Slower, chance of overloading/ under utilization	[6, 7]
2013 & 2016	RR	Allocate tasks in circular manner	Reduced overhead, response time	Failed to decrease makespan, provide efficient distribution strategy	[14, 15]
2015	MCC	Assigning tasks based on minimum completion times	Accounting both execution and resource loads.	Best machine is being used for simple tasks	[14]
2015 & 2016	SJF	Decrease wait time of shorter jobs	Reduced response time	Failed to meet QoS parameters	[9, 10]
2016	Modified LJF	Priority Based LJF reduces the time of VIP longer tasks	Reduced makespan of priority tasks	Failed to meet QoS parameters	[11]
2017	PSO	To obtain approximate results in short time	Minimize cost, processing time	considering only one or two QoS parameters	[19]
2017	ACO	Prediction model that can map tasks accordingly	Better makespan & reliability of system	considering only one or two QoS parameters	[20]
2017	ABC	For balancing workloads	Better energy, cost, response time, utilization rate	No clear idea on cost and time factor	[21]
2017 & 2018	QoS Based	Decrease energy consumption	Reduce count of resources, energy consumption	considering only one or two QoS parameters	[16, 17, 18]
2018	Modified HEFT	Calculate rank for each individual task	Makespan time is reduced	Load distribution is not uniformly done	[5]
2019	Priority Based	Prioritize based on requirements	Reduced completion times & processing costs	Enhanced security features	[12]
2019	MSDE	Single objective Algorithm	Better makespan & throughput	Energy consumption and reliability are not considered	[22]

2.4 Proposed Approach

Based on the reviewed literature, it is clear that there are a number of limitations with all the three different types of scheduling algorithms. The proposed approach is keenly

focused on reducing the cost of execution and analyzing the different workloads to provide the best feasible combination of VMs to execute those tasks. In this paper, a new scheduling policy based on the core idea of traditional SJF and LJF is developed because of the limitations like longer wait times for the shortest jobs in LJF and vice versa. To overcome these limitations and to enhance the QoS parameters by routing both the shorter running jobs and longer running jobs to different VMs parallelly instead of maintaining the same queue for sending all these tasks as it works in traditional methods. Another key aspect of this proposed model is figuring out the best possible combinations of VMs from the available resources based on all the four key metrics discussed in this paper. The proposed approach is aimed at not only routing the tasks to different VMs but also in finding the best machines to execute that tasks based on the algorithmic logic. This parameter helps to bring down the execution costs irrespective of the algorithm that is actually used in scheduling those tasks to different resources available in the host machine.

3 Methodology

Few heuristic scheduling algorithms are identified for developing the proposed model based on the reviewed literature. Some of them include FCFS, SJF, LJF, MCC, Modified SJF and priority based scheduling Algorithm. The proposed approach is mainly concentrated on separating the user tasks depending on the pre-defined parameters such as the number of resources required to execute the task. In this paper, a combination of three different VMs configurations are considered and to test the effectiveness of the algorithm proposed We have considered Google cloud dataset i.e derived from the workload behaviour of google cluster traces and each of these files defined in this workload defines the number of jobs in terms of Million Instructions (MI) [23]. Three different groups of machines based on real-time scenarios of Amazon $EC2$ configurations are defined and the algorithm is tested for all the three combinations of virtual machines for the pre-defined total virtual machine count of 6 in total which are running to complete this job file containing tasks. Another distinctive advantage of our proposed model will be both shorter running jobs as well as longer running jobs will be executed in parallel without waiting for other processes to complete with an enhanced security facility.

A general mathematical derivation for the different combinations possible out of the pre-defined virtual machine count is derived using the below formula:

$$NumberofCombinationsPossible = (n+1)(n+2)/2 \tag{1}$$

Here n = number of VMs considered;

3.1 CloudSim

Cloudsim provides the researcher various offers by cost for validating results and to test different scenarios by bridging the gap between the real-time scenario and the research work. This provides a large scale computing simulation environment by providing various platforms for modeling different components of the cloud. Some of the features of cloudSim include modeling of data centers similar to the real-time scenario in a single computing node, the feasible process of switching between different allocation policies like time-shared and space-shared, creation and management of virtualization engine and

various other features for accelerating the work done by the researchers in the field of cloud computing.

In this paper, a total of 6 virtual machines are considered and its possible to make 28 combinations if we have three different groups of machines by using the above equation. For all the combinations that are possible, tests are executed for proposed algorithm, SJF and LJF. We have identified cloudsim as the potential environment to simulate our results because it provides feasible options for creating a cloud environment [24]. Also, there are various components of cloud computing like scheduling, service brokers, etc. that can be generated using this tool. Various combinations are tested after developing our proposed algorithm according to cloudsim configurations.

3.1.1 CloudSim Architecture & Design

There are different classes in the cloudsim simulation toolkit which are pre-defined to serve different user requests. The below Figure 3 depicts an important hierarchy that is responsible for starting the simulation in cloudsim (Version 3.0.3) [1]. This hierarchy performs a series of operations depending on the cloudlets i.e tasks submitted, entities and the policies defined. This simulation terminates when there are no further events in queues mentioned. Another detailed architecture mentioned below provides a clear understanding of the different options available in the simulation toolkit. Of which, the user code layer is useful for researchers to write their code for testing their models creating their own customized environment. Another layer named CloudSim contains various management interfaces like bandwidth, virtual machines, memory, etc. CloudSim core simulation engine is the base for the start of a simulation process and it contains different entities like queues, event processing.

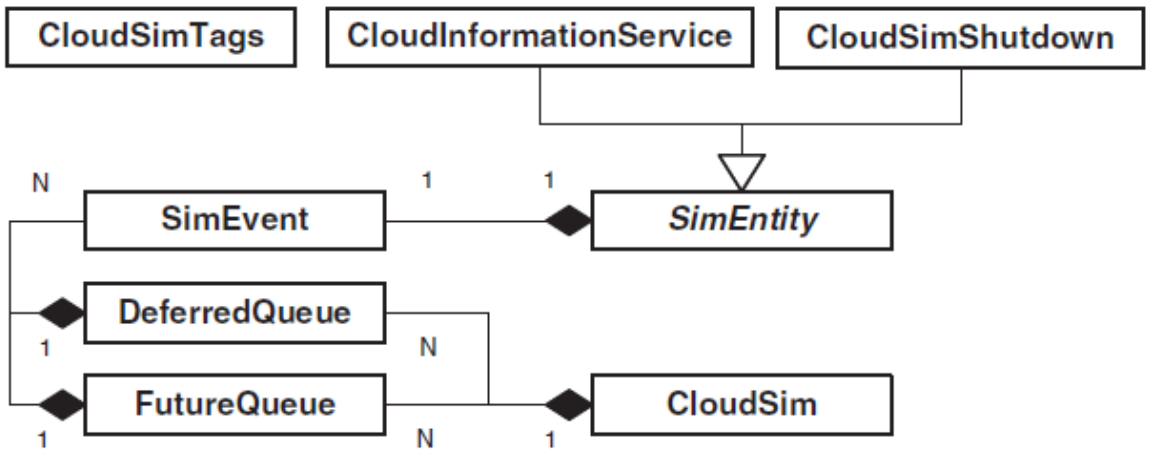

Figure 3: Cloudsim call hierarchy of different classes

3.2 Complexity of Proposed Algorithm

The proposed algorithm starts with sorting the tasks either in ascending or descending order and for this purpose linear sort technique is employed i.e all tasks that are present

[1]cloudsim: https://www.cloudsimtutorials.online/cloudsim-simulation-toolkit-an-introduction/

in the array will be sorted and then from the top of the array jobs will be sent to half the available virtual machines and then from the bottom of the array jobs are allocated to next half of the virtual machines. In this way, all the shorter running jobs will be sent to one set of virtual machines whereas all longer running jobs are sent to another set of machines available in the pool. There are various sorting techniques available which proved to be better than Linear sorting but the proposed algorithm is compared with SJF and LJF and both these algorithms involve the sorting procedure and the same sorting is applied for both. Hence, there are no differences involved using this technique for sorting but for real-time scenarios, this can be replaced with the best possible sorting technique.

3.2.1 Time Complexity

For the first line in the proposed algorithm, Consider there are n tasks in the array, so the loop executes for $(n-1)$ times. The initialization takes only one primitive operation and the loop will be executed one more time than the required number of operations because that extra one time decides that the condition is failed. The increment will take place as many times as loop executes i.e $(n-1)$ times.

for$(i = 0; i < n-1; i++) => 1 + (n-1+1) + (n-1) = 2n$
$SmallJob \leftarrow i => (n-1)$
for$(i^* = i+1; i < n; j++) => (n-1) + (x+1) + x = 2x + n$
Here; $x = 1 + 2 + 3... + n - 1 = (n-1)(n-1+1)/2$
if$(k(i^*) < k(smalljob)) => 3x$
$Smalljob = i^* => x$
$temp = k(i) => 2(n-1)$
$k(i) = k(smalljob) => 2(n-1)$
$k(smalljob) = temp => 2(n-1)$
$return => 1$

$$T(n) = 6x + 10n - 4 \tag{2}$$

After substituting x in equation(2);

$$T(n) = 3n^2 + 7n - 4 \tag{3}$$

Here $T(n)$ referred as time complexity;

The value of n^2 denotes the time with respect to n tasks. So, $T(n)$ is $O(n^2)$.

3.2.2 Space Complexity

The value of space complexity of selection sorting technique used is $O(1)$ because it follows in-place comparison i.e a specific array position is replaced by the smallest element.

3.3 Proposed Algorithm

In this section, pseudo code of the proposed algorithm is explained and the execution of algorithm actually starts with by considering the tasks presented in the dataset. These tasks are sent to the broker and then they will be sorted using linear sort technique and it is done in the Step 1 of the below mentioned Algorithm 1. These sorted tasks are then sent to the VMs and before passing these tasks to the VMs, Step 2 of the Algorithm 1 will

be executed. And then the elements in the sorted array are assigned to the VMs in such a way that the elements from the top of the queue will be sent to the first half of VMs and the elements from bottom of the array are sent to next half of VMs at a single time till the time that there are tasks left in the sorted array list. Once execution of Step 3 is finished, parameters like waiting time, completion time, execution cost and turnaround time are calculated using the standard formulas for each and every combination of VMs that are possible from the defined count of machines. These results are then compared with existing LJF and SJF algorithms for all the combinations of VMs to check the best combination of VMs for every algorithm.

Algorithm 1 Proposed Algorithm

$i = (1, 2, ...N)$ ▷ number of tasks
$j = (1, 2...M)$ ▷ number of virtual machines
Step 1 : *Sort the tasks* $i = (1, 2, ...N)$
for $i = (1, 2, ...N)$ **do**
  Sort the tasks $i = (1, 2, ...N)$
  $SmallJob \leftarrow i$
  for $i^* = (i + 1, i + 2, ...N)$ **do** ▷ Updated number of tasks
    if $k(i^*) < k(smalljob)$ **then**
      $k \leftarrow tasklength$
      $Smalljob = i^*$
    end if
  end for
  $temp = k(i)$
  $k(i) = k(smalljob)$
  $k(smalljob) = temp$
end for
Step 2 : *Sort the virtual machines* $j = (1, 2, ...M)$
for $j = (1, 2.....M)$ **do**
  Sort the Virtual Machines based on MIPS
end for
Step 3 : *Assign the sorted tasks* $i = (1, 2, ...N)$ *to Virtual machines* $j = (1, 2....M)$
if $(N \pmod 2 == 0 \wedge M \pmod 2 == 0)$ **then**
  Assign $\frac{N}{2}$ *tasks to* $\frac{M}{2}$ *machine*
  $\wedge$ *rest* $\frac{N}{2}$ *tasks to remaining* $\frac{M}{2}$ *machine*
else if $(N \pmod 2 == 0 \wedge M \pmod 2 == 1)$ **then**
  Assign $\frac{N}{2}$ *tasks to* $\frac{M-1}{2}$ *machine*
  $\wedge$ *rest* $\frac{N}{2}$ *tasks to* $\frac{M+1}{2}$ *machine*
else if $(N \pmod 2 == 1 \wedge M \pmod 2 == 0)$ **then**
  Assign $\frac{N-1}{2}$ *tasks to* $\frac{M}{2}$ *machine*
  $\wedge$ $\frac{N+1}{2}$ *tasks to remaining* $\frac{M}{2}$ *machine*
else if $(N \pmod 2 == 1 \wedge M \pmod 2 == 1)$ **then**
  Assign $\frac{N-1}{2}$ *tasks to* $\frac{M-1}{2}$ *machine*
  $\wedge$ $\frac{N+1}{2}$ *tasks to* $\frac{M+1}{2}$ *machine*
end if

4 Design Specification

Resource Scheduling is the process in which incoming tasks from cloud users will be assigned to specified resources based on the agreed SLA and to gain the user trust by minimizing the waiting time of tasks, completing the tasks with an increased rate of utilization etc. Several Algorithms are developed depending on the key parameters that researchers aim at primarily and it differs based on various policies and requirements. The proposed algorithm is the enhancement of traditional SLA and LJF algorithms because longer jobs have to wait for more time in LJF, on the other side shorter jobs have to wait for more time in LJF. Hence, the proposed algorithm minimizes the wait times of both the jobs by allocating them to resources parallelly. The problem of under-provisioning and over-provisioning is also not handled well because of ample variations.

4.1 Proposed Architecture

There are several limitations in the Resource Scheduling process though several algorithms are developed in the last decade. In order to reduce waiting time, turnaround time, completion time and the execution cost a new task scheduling algorithm has been proposed with the core idea of separating both shorter running jobs and longer running jobs to different VMs by creating two different pools based on the resource availability and the task nature. This actually reduces the waiting time and completion times of both shorter and longer running jobs. A detailed explanation of the proposed architecture is depicted in Figure 4 and the architecture is classified into three parts as explained below:

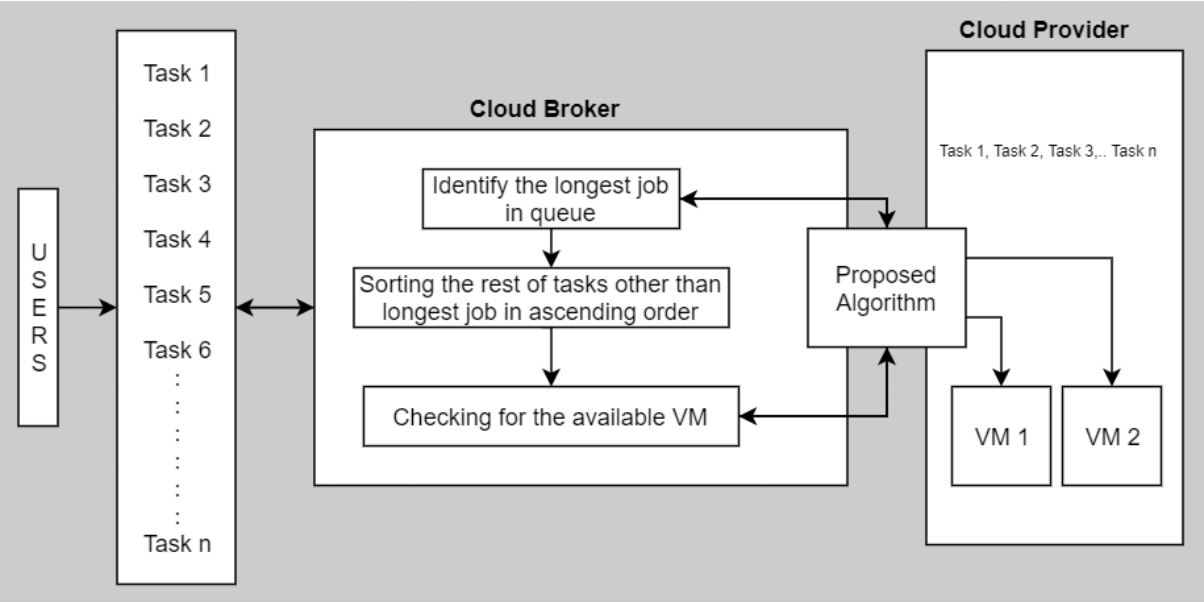

Figure 4: Proposed Architecture for allocating user tasks to different VMs

1. **User side tasks** - This denotes the tasks that are to be executed as requested by the cloud user and each task will have its own specifications like length, resource requirement, etc.

2. **Cloud Broker** - This is a crucial part of the scheduling process because it acts as

a bridge for assigning user tasks to the available resources that are managed by a cloud provider by using the defined algorithm.

3. **Cloud provider** - Based on the requests from cloud users, the provider creates the resources and each of these resources created may or may not have similar configurations.

The basic idea behind the proposed algorithm is to reduce the cost of executing the tasks, waiting time, completion time and turnaround time through the allocation of tasks to the VMs based on the logic written in the proposed algorithm. The proposed algorithm initially sorts the list of tasks and then VMs will be created to two different pools based on the count of resources. These sorted elements from the top of the array list are then sent to the first pool of VMs created and parallelly tasks from the bottom of the array are sent to other pool of VMs available till there are no empty tasks left in the array. This logic actually separates both shorter running jobs and longer running jobs from each other and are allocated to different machines, hence reducing the wait time, the completion time of both shorter and longer running jobs. This is an enhancement for the two traditional heuristic algorithms SJF and LJF. This work can be extended to figure out DDoS or DoS attacks because those are concentrated mainly on longer running jobs and we have already separated these jobs from shorter running jobs.

The proposed model not only allocates shorter running jobs to a pool of VMs and longer jobs to another VMs but also reduces total cost, waiting and completion times when compared to SJF and LJF algorithms. Traditional SJF algorithm reduces the wait time of shorter running jobs which eventually reduces the total completion time of tasks if the dataset contains more number of shorter jobs and arrive at the end. Similarly, LJF algorithm reduces longer running jobs waiting time and turnaround time. In addition, none of the above discussed algorithms consider the overall cost of executing the tasks but the proposed algorithm is focused on selecting the viable resources with a specific combination where that particular algorithm holds good. For this specific reason, a mathematical derivation which denotes the number of different combinations possible for a chosen number of resources is derived and the simulation is run for all the different combinations and calculated the cost parameter to check where the algorithm fits well and it is also useful for allocating the resources prior based on the behaviour and specifications of tasks.

4.2 Instance Configurations

Below are the different group of machines considered while performing the analysis [2]. All these configuration details are shown in Table 2. As per the proposed approach, jobs will be allocated to these combination of machines based on length of the tasks i.e small tasks will be sent to first three machines in the pool whereas the larger tasks will be sent to rest of the three machines available because while implementing a group of six VMs are considered. Also, cost factor is considered while analysing the output and the total cost of executing the jobs in the dataset is calculated for three different algorithms. All these configurations are created in cloudsim because this provides a feasible approach to compare the results with traditional algorithms.

[2]Instance Pricing: https://instaguide.io/

Table 2: Virtual Machines Configurations used for Analysis

Instance Type	MIPS	CPU	RAM (GiB)	Cost Per Sec ($)
c5.large	6000	2	4	0.0016
c5.xlarge	6000	4	8	0.0032
c5.2xlarge	6000	8	16	0.0064

5 Implementation

There are various versions of cloudsim that are made available through GitHub and there are few basic pre-requisites required to make it available on a local desktop. Initially, the required version of the .zip format has to be downloaded [3]. Also, there should be a java JDK downloaded to the machine where we need it to run because the entire code of CloudSim is written in Java. For better convenience, it is good to have Java IDE installed and this can be downloaded through Eclipse foundation. Most importantly, cloudsim provide support for calculations and to efficiently run this we need to have a math library function that can be downloaded through Apache's website.

For evaluating, we have considered six virtual machines of three different sizes with all possible combinations and the output results are compared with traditional SJF and LJF algorithms. A total of 28 different combinations are tested for each of the three algorithms which include the proposed algorithm. According to the proposed algorithm, all the tasks are executed in such a way that tasks requiring less time for execution are being served by one pool of virtual machines and the tasks that require more time to complete will execute on other pool of machines available i.e in a sorted list of array, tasks from top of the queue will be assigned to virtual machines array list starting from the smallest machine in pool and then tasks from bottom of the queue will be executed starting from largest machine available in queue parallelly till there are no virtual machines available in queue.

5.1 Dataset Description

Google Cloud Jobs Dataset (GoCJ) contains data about size of jobs in terms of Millions of Instructions (MI). The chosen GoCJ dataset is comprised of 21 different text files and is named after "`GoCJ_Dataset_XX.txt`", where XX denotes the number of jobs present in each file [4]. Each file contains different set of rows equivalent to the number of jobs denoting size of each job in MI and these values ranges between 15K MI to 900K MI. These values are analyzed based on Google Cluster traces over a period of 29 days. The GoCJ dataset is generated using Monte Carlo (MC) simulation method and by using this simulation method we can create any number of jobs. The actual time to complete a job depends on the machine MIPS.

$$ETC_{Sec} = Job_{MI}/Machine_{MIPS} \tag{4}$$

Here ETC = Time to complete a job in Seconds;

[3] GitHub: https://github.com/Cloudslab/cloudsim/releases/
[4] GoCJ Dataset: https://data.mendeley.com/datasets/b7bp6xhrcd/1

A text file based on GoCJ dataset containing 1000 tasks is used to test this scenario. This dataset contains the MI value which denotes task length and these tasks are sent to different virtual machines available in data centre based on proposed logic. Here, to make it more realistic we considered that all these tasks did not arrive at the same time instead we assumed that there occurs some delay in the arrival of those tasks and this is sufficed using some random function starting with delay value of 10 Seconds for the first task because minimum time to complete shortest individual task takes 2.5 Seconds and the longest job takes 2.5 Minutes and the average time to complete each task working on a single core takes nearly 76.25 Seconds. Also the dataset contains 60%-70% longer running jobs, hence a delay of 10% based on average completion time of individual task is considered and then a random function is used for rest of the tasks. For each individual task completion time, waiting time, turn around time and the total cost is calculated for the proposed algorithm and for each possible combinations of those six virtual machines of three different combinations. Finally, the total cost for executing all the tasks in the dataset for each combination is calculated and compared with SJF and LJF algorithms for similar combinations. A detailed analysis of test results is evaluated in the next section by depicting graphs for average waiting time, total completion time and total cost after running all tasks for the three algorithms.

6 Evaluation

To verify the effectiveness of the proposed algorithm a series of tests are conducted using the Cloudsim simulation toolkit. A total of 28 different combinations of VMs are selected from a selected count of six VMs and several parameters like average waiting time, completion time, turn around time and total cost of execution are calculated individually for each combination by allocating tasks to VMs based on the proposed model. We have used GoCJ dataset containing 1000 tasks to test the algorithm. These results are then quantitatively compared against similar combinations of VMs by using traditional SJF and LJF algorithms. A brief explanation of the parameters evaluated in this paper are discussed below:

- **Arrival Time (AT):** It is defined as the time at which the entire process is in ready state.
- **Burst Time (BT):** It is the total time required for a job to execute.
- **Completion Time (CT):** It is defined as the actual time that a job is completed.
- **Turnaround Time (TAT):** It is the actual time that a specific job spent on the system.
$$TAT = CT - AT \tag{5}$$
- **Waiting Time (WT):** It is the total time that a task is in queue waiting for resources.
$$WT = TAT - BT \tag{6}$$

For calculating the total cost of execution, the completion time of tasks for each combination and for every algorithm, we assumed all the six virtual machines are running till the execution is completed. For example, if we run a test with six c5.large

instances alone, then the total cost of execution will be the result of $completionTime * 6 * InstanceCost/sec$. It is also observed that all the shorter jobs are sent to the first three VMs and the longer jobs are sent to the last three VMs, which is the core idea of this research work. As we considered a random function for delaying the tasks to arrive, the simulation results vary a bit when running multiple times. Hence, for maintaining stability across the output test i.e for each combination of VMs under different algorithms, a simulation was carried out six times and the average of the outputs are recorded and tabulated as shown in Table 3.

Table 3: Test results from cloudsim using different algorithms for GoCJ dataset

Instances			Proposed Model				SJF				LJF			
c5. large	c5. xlarge	c5. 2xlarge	WT	CT	TAT	COST	WT	CT	TAT	COST	WT	CT	TAT	COST
6	0	0	874.496	2722.31	21.613	26.134	840.095	1881.04	21.611	18.058	914.18	1831.01	21.6135	17.577
5	1	0	775.959	2721.763	21.613	30.483	793.266	1859.65	21.611	20.828	852.033	1834.593	21.6126	20.54
5	0	1	730.104	2714.956	21.612	39.146	758.439	1859.09	21.612	26.7709	787.492	1829.509	21.6129	26.344
4	2	0	687.805	2675.539	21.613	34.246	709.15	1883.73	21.612	24.11	749.444	1829.37	21.611	23.4159
4	1	1	618.9376	2678.76	21.613	42.86	653.175	1885.31	21.612	30.165	712.2308	1829.593	21.6135	29.2734
4	0	2	538.395	2679.76	21.613	51.4513	633.649	1894.016	21.611	36.3651	675.716	1829.793	21.6134	35.132
3	3	0	549.196	1370.513	21.614	19.735	607.193	1864.543	21.612	26.849	696.85	1831.483	21.6133	26.3733
3	2	1	500.894	1365.04	21.613	24.024	581.791	1833.04	21.6124	32.2615	640.091	1829.81	21.6136	32.204
3	1	2	424.879	1350.7	21.614	28.094	536.176	1891.26	21.611	39.338	598.015	1832.479	21.6145	38.1155
3	0	3	374.322	924.093	21.615	22.178	519.41	1882.62	21.614	45.182	551.701	1832.79	21.614	43.9869
2	4	0	510.867	1370.2	21.614	21.923	566.86	1861.07	21.612	29.777	582.49	1830.84	21.6143	29.2935
2	3	1	458.3	1373.93	21.6142	26.37952	522.85	1877.849	21.612	36.0547	577.138	1831.426	21.613	35.1633
2	2	2	398.4252	1349.753	21.615	30.2344	486.383	1847.79	21.613	41.39	491.31	1830.233	21.614	40.997
2	1	3	338.674	926.566	21.616	23.72	445.494	1875.763	21.613	48.0195	498.129	1832.43	21.6149	46.91
2	0	4	319.865	924.403	21.615	26.622	410.095	1880.646	21.6136	54.162	455.361	1832.459	21.6141	52.774
1	5	0	481.077	1398.81	21.614	24.619	487.382	1876.37	21.613	33.024	532.284	1832.67	21.615	32.255
1	4	1	424.264	1366.229	21.615	28.4175	442.562	1811.846	21.614	37.6864	500.38	1830.67	21.6156	38.0779
1	3	2	365.16	1379.709	21.616	33.113	410.728	1810.59	21.613	43.454	444.66	1832.82	21.614	43.987
1	2	3	297.073	924.29	21.617	25.1406	386.059	1814.21	21.612	49.3465	420.1	1832.426	21.615	49.842
1	1	4	281.033	925.28	21.616	28.128	346.08	1813.736	21.614	55.1375	373.629	1832.15	21.616	55.6973
1	0	5	265.192	924.846	21.615	31.074	292.424	1858.79	21.615	62.455	316.915	1831.45	21.616	61.536
0	6	0	432.301	1369.78	21.614	26.299	419.457	998.456	21.613	19.17	449.83	977.37	21.616	18.765
0	5	1	380.635	1368.873	21.615	30.662	368.988	998.789	21.612	22.372	410.406	942.76	21.615	21.117
0	4	2	326.478	1351.73	21.615	34.604	325.723	959.51	21.613	24.5635	366.783	966.453	21.616	24.741
0	3	3	265.001	758.07	21.617	21.832	289.043	1004.82	21.615	28.938	323.27	996.93	21.616	28.711
0	2	4	242.035	726.016	21.616	23.2325	243.121	1003.98	21.616	32.127	294.601	924.37	21.618	29.579
0	1	5	223.675	738.76	21.6173	26.0045	230.918	1011.23	21.615	35.595	242.337	988.26	21.619	34.7867
0	0	6	212.14	732.456	21.6173	28.1263	189.17	574.733	21.617	22.0697	200.162	545.116	21.62	20.932

From the test results Table 3 it is evident that the proposed algorithm is performing well for the GoCJ dataset for a combination of 0*c5.large, 3*c5.xlarge and 3* c5.2xlarge instances. For this combination of VMs total cost of execution, waiting time, completion time and the turnaround time is exceptionally good compared to other combinations. The traditional SJF and LJF algorithms a combination of 6*c5.2xlarge instances is the ideal choice to execute the GoCJ dataset because all the four metrics presented in the above table are performing better than the rest of the combinations.

From the below Figure 5, a quantitative evaluation of the average waiting time of tasks for three algorithms and for each combination of VMs are plotted. It is easy to predict from the plot that for any combination (all 28 combinations) of VMs the proposed algorithm is performing better than the SJF and LJF algorithms. Amongst the rest two algorithms, SJF has reduced waiting times compared to the LJF algorithm. So, to conclude the combination with the lowest average waiting time will be the best choice while considering any algorithm and in this case, the proposed algorithm is performing better than the rest two traditional heuristic algorithms. For any combination, nearly 10-20% of the average waiting time is reduced by our proposed algorithm.

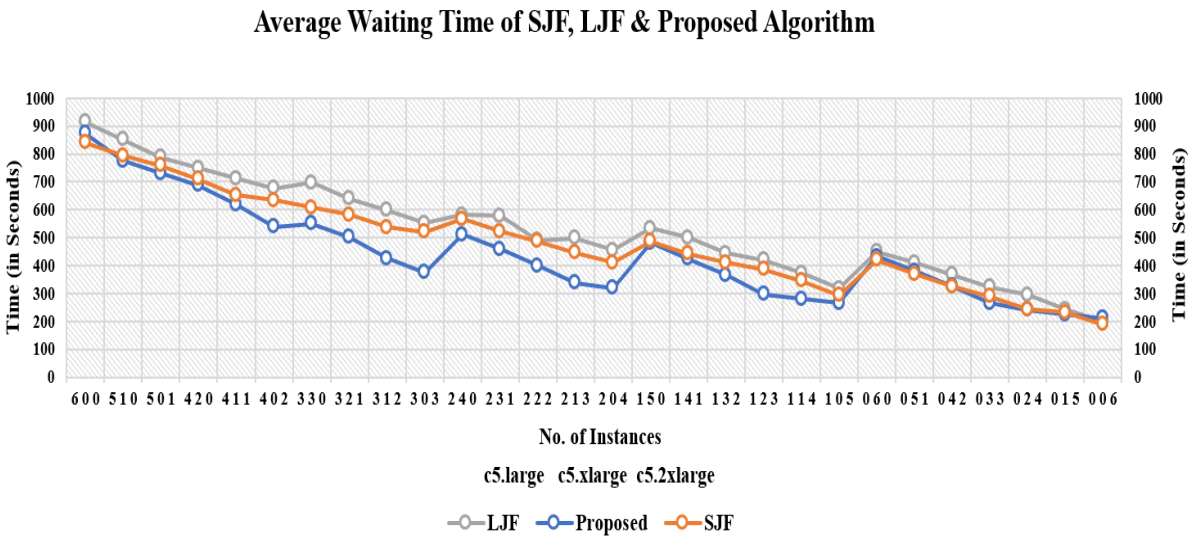

Figure 5: Average Waiting Time of SJF, LJF & Proposed Algorithm

The total cost of execution is the key metric that both the cloud provider and the cloud user look for while executing tasks on the resources. Hence, a plot between the proposed algorithm, SJF and LJF is drawn for all 28 combinations that are possible from the six VMs selected. It is evident from Figure 6 that the total cost of execution is better for the proposed algorithm in most of the combinations when compared to SJF and LJF but if the c5.large instances are considered more than c5.xlarge and c5.2xlarge, then the total cost for executing the tasks will be less in SJF and LJF algorithms. It is because of the presence of more number of longer tasks present in the dataset. To conclude from Figure 6, the proposed algorithm is reducing nearly 30-40% of the costs for nearly 19 combinations and another interesting fact from the plot is it is better to execute the GoCJ dataset with c5.large instances and proposed algorithm with c5.xlarge instances for huge cost savings.

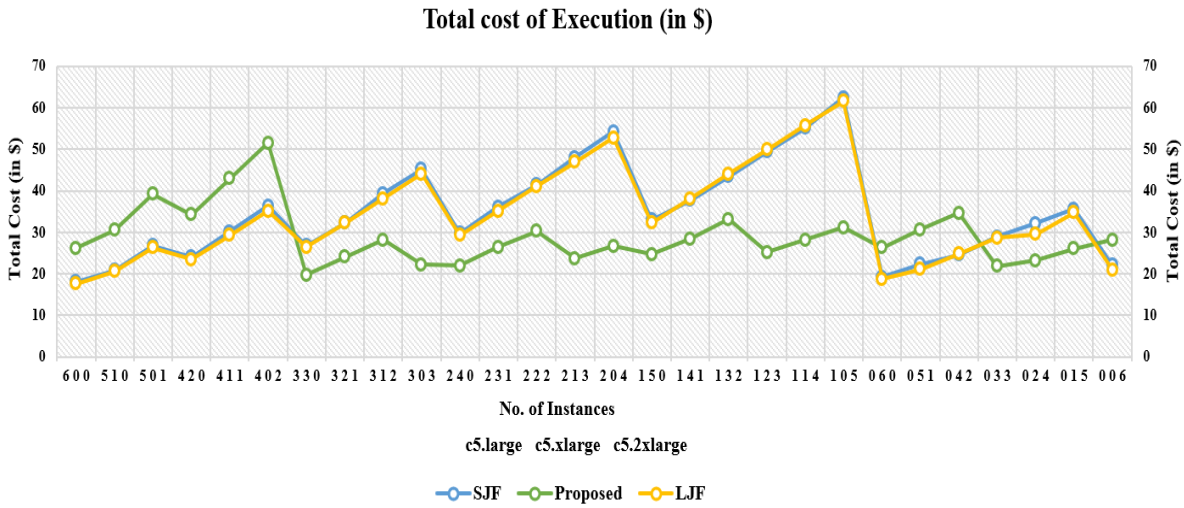

Figure 6: Total cost of execution of SJF, LJF & Proposed Algorithm

In the below Figure 7, the completion time of tasks when executing under three different algorithms and the different combination of VMs is plotted. The completion time of tasks is directly proportional to the execution cost because completion time is the parameter that decides the overall cost of the service. Similar to execution costs, for the majority of combinations the proposed algorithm is performing nearly 40-50% better than the traditional SJF and LJF algorithms. As discussed above, instances with more c5.large combinations are better for the chosen dataset in terms of both cost and execution time and for the rest 19 combinations, our proposed algorithm is performing better than the traditional algorithms. But, waiting time will be more though the cost and completion time is less for a few combinations in the series for both SJF and LJF algorithms compared to the proposed algorithm.

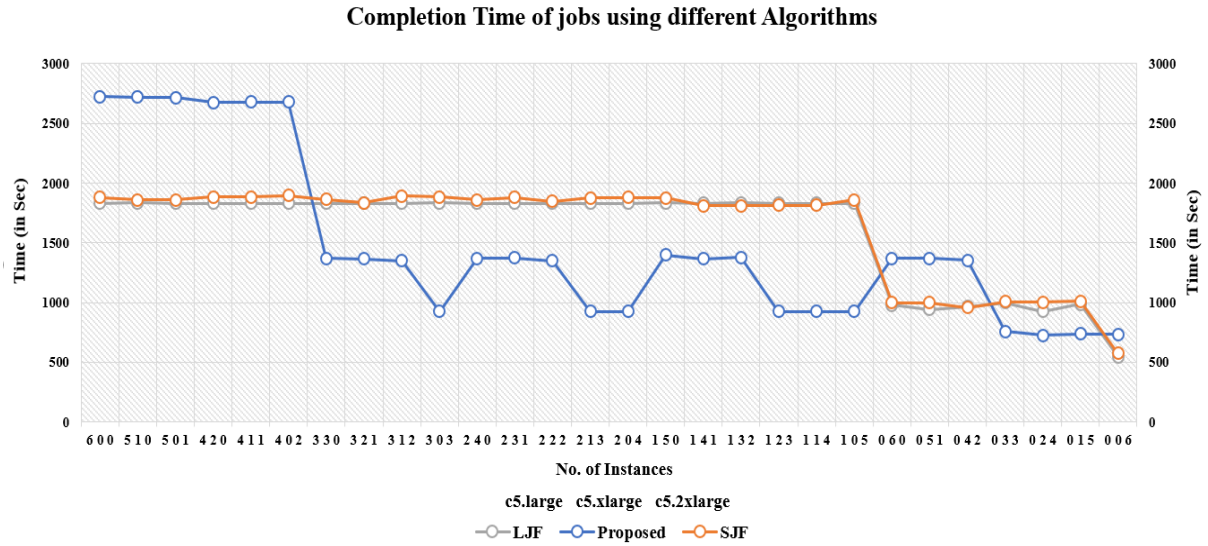

Figure 7: Completion Time of jobs for each combination of VMs using SJF, LJF & Proposed Algorithm

Another key metric from the simulation is the turnaround time but interestingly it is observed that there is no difference in the average turnaround times for all the three algorithms. It is also observed that there exists difference in turnaround times for individual tasks but when it is calculated for an average of 1000 tasks it remains the same and if we run the experiment with 100 tasks there is some slight difference in turnaround times. To conclude, average turnaround time depends on the kind of workload used, VMs configurations and number of tasks present in dataset.

7 Conclusion and Future Work

In this paper, a detailed review of different scheduling algorithms along with their advantages and limitations are discussed. Based on the literature reviewed, we have identified SJF and LJF algorithms for the enhancement of the proposed algorithm. Because of the uneven load distributions, not utilizing the resources efficiently there are several problems like an increase in the execution cost, taking more than the expected time to finish the

jobs, etc. To overcome these issues, the proposed model eliminates the issues in traditional SJF and LJF algorithms by routing the tasks to two different pools of machines based on the length of the tasks i.e shortest jobs are sent to one pool and longest jobs are sent to another pool for execution. This actually reduces the wait time of both the shortest tasks and the longest tasks thus decreasing the overall wait time when compared to SJF and LJF algorithms. Another distinctive advantage of the proposed model is the minimization of execution cost and completion time of the tasks.

The simulation is conducted for different combinations of VMs i.e for the total count of 6 VMs with three different configurations there occur 28 combinations and this is helpful for identifying the best suitable combination of VMs for executing the considered GoCJ dataset. Individually for each algorithm, the best combination of VMs to be considered are identified based on the key metrics evaluated. The results obtained shows that the proposed algorithm is performing better than SJF and LJF algorithms regarding waiting time and also the execution cost and completion times for 70% of the combinations are performing better in the proposed algorithm and saves nearly 20-30% of average cost and time when compared to SJF and LJF and another interestingly study on turnaround time shows it is same for all the three different algorithms. In the future, the proposed work is extended for dynamically checking the best combination of VMs that are to be scheduled for any algorithm depending on the workload.

References

[1] B. Belén, F. Sonja, J. Carlos, G. Beatriz, and G. Carlos, "Improving the energy efciency in cloud computing data centres through resource allocation techniques," in *Research advances in cloud computing* (S. Chaudhary, G. Somani, and R. Buyya, eds.), pp. 211–235, Singapore: Springer, 2017. ISBN 978-981-10-5026-8.

[2] M. Kumar, S. Sharma, A. Goel, and S. Singh, "A comprehensive survey for scheduling techniques in cloud computing," *Journal of Network and Computer Applications*, vol. 143, pp. 1 – 33, 2019. Impact-Factor=3.991.

[3] A. Arunarani, D. Manjula, and V. Sugumaran, "Task scheduling techniques in cloud computing: A literature survey," *Future Generation Computer Systems*, vol. 91, pp. 407–415, 2019. Impact-Factor=4.639.

[4] S. Singh and I. Chana, "Resource provisioning and scheduling in clouds: Qos perspective," *The Journal of Supercomputing*, vol. 72, no. 3, pp. 926–960, 2016. Impact-Factor=1.532.

[5] K. Dubey, M. Kumar, and S. Sharma, "Modified heft algorithm for task scheduling in cloud environment," *Procedia Computer Science*, vol. 125, pp. 725–732, 2018. Impact-Factor=0.79.

[6] X. Li, Y. Mao, X. Xiao, and Y. Zhuang, "An improved max-min task-scheduling algorithm for elastic cloud," in *2014 International Symposium on Computer, Consumer and Control*, pp. 340–343, IEEE, 2014. No. of citations: 32 as on 23/11/2019.

[7] S. Anousha and M. Ahmadi, "An improved min-min task scheduling algorithm in grid computing," in *International Conference on Grid and Pervasive Computing*, pp. 103–113, Springer, 2013. Core Rank = C.

[8] W. Li and H. Shi, "Dynamic load balancing algorithm based on fcfs," in *2009 Fourth International Conference on Innovative Computing, Information and Control (ICICIC)*, pp. 1528–1531, IEEE, 2009. No. of citations: 27 as on 23/11/2019.

[9] R. K. Mondal, E. Nandi, and D. Sarddar, "Load balancing scheduling with shortest load first," *International Journal of Grid and Distributed Computing*, vol. 8, no. 4, pp. 171–178, 2015. No. of citations: 18 as on 23/11/2019.

[10] M. A. Alworafi, A. Dhari, A. A. Al-Hashmi, A. B. Darem, *et al.*, "An improved sjf scheduling algorithm in cloud computing environment," in *2016 International Conference on Electrical, Electronics, Communication, Computer and Optimization Techniques (ICEECCOT)*, pp. 208–212, IEEE, 2016. No. of citations: 10 as on 23/11/2019.

[11] M. Kumar and S. C. Sharma, "Priority aware longest job first (pa-ljf) algorithm for utilization of the resource in cloud environment," in *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, pp. 415–420, March 2016.

[12] A. Ponraj, "Optimistic virtual machine placement in cloud data centers using queuing approach," *Future Generation Computer Systems*, vol. 93, pp. 338–344, 2019. Impact-Factor=4.639.

[13] S. K. Panda and P. K. Jana, "Efficient task scheduling algorithms for heterogeneous multi-cloud environment," *The Journal of Supercomputing*, vol. 71, no. 4, pp. 1505–1533, 2015. Impact-Factor=1.532.

[14] P. Pradhan, P. K. Behera, and B. Ray, "Modified round robin algorithm for resource allocation in cloud computing," *Procedia Computer Science*, vol. 85, pp. 878–890, 2016. Impact-Factor=0.79.

[15] D. C. Devi and V. R. Uthariaraj, "Load balancing in cloud computing environment using improved weighted round robin algorithm for nonpreemptive dependent tasks," *The scientific world journal*, vol. 2016, 2016.

[16] H. Chen, G. Liu, S. Yin, X. Liu, and D. Qiu, "Erect: energy-efficient reactive scheduling for real-time tasks in heterogeneous virtualized clouds," *Journal of computational science*, vol. 28, pp. 416–425, 2018. Impact-Factor=1.925.

[17] T. Chaabouni and M. Khemakhem, "Energy management strategy in cloud computing: a perspective study," *The Journal of Supercomputing*, vol. 74, no. 12, pp. 6569–6597, 2018. Impact-Factor=1.532.

[18] Z. Zhou, J. Abawajy, M. Chowdhury, Z. Hu, K. Li, H. Cheng, A. A. Alelaiwi, and F. Li, "Minimizing sla violation and power consumption in cloud data centers using adaptive energy-aware algorithms," *Future Generation Computer Systems*, vol. 86, pp. 836–850, 2018. Impact-Factor=4.639.

[19] M. Masdari, F. Salehi, M. Jalali, and M. Bidaki, "A survey of pso-based scheduling algorithms in cloud computing," *Journal of Network and Systems Management*, vol. 25, no. 1, pp. 122–158, 2017. Impact-Factor=1.750.

[20] H. Duan, C. Chen, G. Min, and Y. Wu, "Energy-aware scheduling of virtual machines in heterogeneous cloud computing systems," *Future Generation Computer Systems*, vol. 74, pp. 142–150, 2017. Impact-Factor=4.639.

[21] L. Foxiang, S. Yuehong, L. Yanhui, and W. Tingting, "An artificial bee colony algorithm based on multiobjective and nondominated solution replacement mechanism for constrained optimization problems.," *Numerical Mathematics: Theory, Methods & Applications*, vol. 12, no. 3, pp. 797 – 823, 2019. Impact-Factor=0.695.

[22] M. A. Elaziz, S. Xiong, K. Jayasena, and L. Li, "Task scheduling in cloud computing based on hybrid moth search algorithm and differential evolution," *Knowledge-Based Systems*, vol. 169, pp. 39–52, 2019. Impact-Factor=4.396.

[23] A. Hussain and M. Aleem, "Gocj: Google cloud jobs dataset for distributed and cloud computing infrastructures," *Data*, vol. 3, no. 4, p. 38, 2018. No. of citations: 3 as on 03/12/2019.

[24] R. Buyya, R. Ranjan, and R. N. Calheiros, "Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities," in *2009 international conference on high performance computing & simulation*, pp. 1–11, IEEE, 2009. Core Rank = B.

A List of Acronyms

Glossary

GoCJ Google Cloud Jobs Dataset.

CMMN MEdian MAX and Cloud Min-Max normalization.

MSA Moth Search Algorithm.

SVM support vector machine.

CD compact disk.

SLA Service Level Agreement.

QoS Quality of Service.

FCFS First Come First Served.

SJF Shortest Job First.

LJF Longest Job First.

DoS Denial of Service.

MCC Minimum Cloud Completion Time.

RA Resource Allocation.

CPOP Critical Path on Processor.

VMs Virtual Machines.

VM Virtual Machine.

RPS Resource Provisioning with Scheduling.

PSO Particle Swarm Optimization.

HEFT Heterogeneous Earliest Finish Time.

DAG Directed Acyclic Graphs.

CLS Cloud list Scheduling.

CMMS Cloud MinMin Scheduling.

ACO Ant Colony Optimization.

ABC Artificial Bee Colony.

MSDE moth search and differential evolution.

GA Genetic Algorithm.

PSSF previous-selected-server-first-policy.

DDoS Distributed Denial of Services.

AT Arrival time.

BT Burst time.

CT Completion time.

WT Waiting time.

TAT Turnaround time.

IaaS Infrastructure as a Service.

PaaS Platform as a Service.

SaaS Software as a Service.

B Configuration Manual

B.1 Introduction

A detailed step-by-step procedure for implementing the proposed algorithm is clearly explained along with the configurations that are used while conducting the experiment. In the following sections, set of cloudsim environment and the dependencies involved for executing the project.

B.2 Implementation Steps

B.2.1 Cloudsim Installation

This is the first step of the implementation process. It requires few mandatory files to be downloaded along with the .zip file.

Table 4: CloudSim Pre-Requisites

CloudSim Pre-Requisites	Version
cloudsim-3.0.3.zip	3.0.3
Java JDK	13.0.1
Java IDE	2019-09
commons-math3-3.6.1-bin.zip	3.6.1

B.2.2 Downloading Dataset

For conducting the experiment, we used GoCJ dataset and this can be downloaded from the Mendeley data repository [5] and the dataset looks the way as shown in below Figure 8.

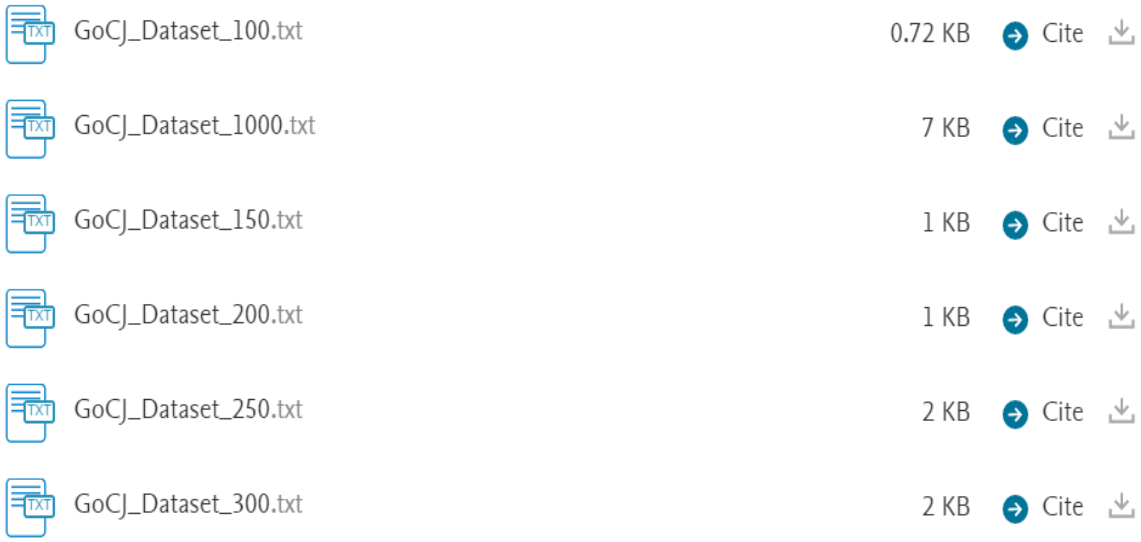

Figure 8: GoCJ Dataset

B.3 Virtual Machine Configurations

Below are the virtual machine configurations created in a Constants.java file as shown in Figure 9.

```
public class Constants {

    //Vm configuration for the c5.large

    public static final long smallSize=2000;
    public static final int smallRam=4096;
    public static final int smallMips=6000;
    public static final long smallBw=100000;
    public static final int smallPesNumber=2;
    public static final String smallVmm="Xen";
    public static final String smallType="small";

    //Vm configuration for the c5.xlarge

        public static final long mediumSize=4000;
        public static final int mediumRam=8192;
        public static final int mediumMips=6000;
        public static final long mediumBw=100000;
        public static final int mediumPesNumber=4;
        public static final String mediumVmm="Xen";
        public static final String mediumType="medium";

        //Vm configuration for the c5.2xlarge

            public static final long largeSize=4000;
            public static final int largeRam=16384;
            public static final int largeMips=6000;
            public static final long largeBw=100000;
            public static final int largePesNumber=8;
            public static final String largeVmm="Xen";
            public static final String largeType="large";

            public static final double SmallCostPerSec= 0.0016;
            public static final double MediumCostPerSec= 0.0032;
            public static final double LargeCostPerSec= 0.0064;
```

Figure 9: VM Configurations

[5] GoCJ: https://data.mendeley.com/datasets/b7bp6xhrcd/1

B.4 Code Development

Entire cloudsim code is written in java, so based on the above configurations we need to develop a data centre environment in cloudsim and identify the necessary classes where we need to change the methods. Below are the two classes where code is changed in the existing cloudsim toolkit.

- DatacenterBroker.java
- cloudlet.java

B.5 Test Results

After running the .java file, below is the output that will be generated in console as shown in

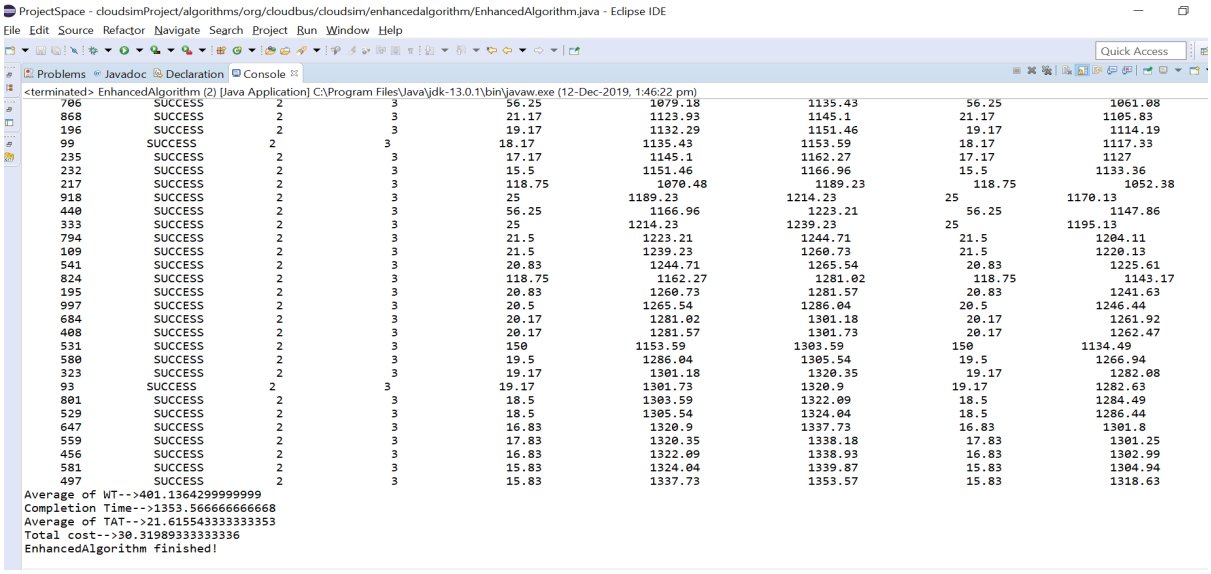

Figure 10: Test Results from console