ISLAMIC UNIVERSITY OF TECHNOLOGY

A QOS GUARANTEED RESOURCE ALLOCATION IN CLOUD COMPUTING BASED ON SELECTIVE ALGORITHM

Authors:

Md. Tanbin Rahid Kyser (104426)
Zonayed Ahmed (104429)

Supervisor:

Md. Mohiuddin Khan
Assistant Professor

Department of Computer Science and Technology
Islamic University of Technology

A thesis submitted in partial fulfillments of the requirements for the degree of Bachelor for Science in Computer Science and Engineering

Academic Year: 2013-2014

Department of Computer Science and Engineering
Islamic University of Technology.

A Subsidiary Organ of the Organization of Islamic Cooperation
Dhaka, Bangladesh

Certificate of Research

This is to certify that the work presented in this thesis is the outcome of analysis and investigation carried out by Md. Tanbin Rahid Kyser and Zonayed Ahmed under the supervision of Mr. Mohiuddin Khan in the department of Computer Science and Engineering (CSE), IUT, Gazipur, Bangladesh. It is also declared that neither of this thesis nor any part of this thesis has been submitted anywhere else for any degree or diploma. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given.

Signature of the Head of the Department

_____

Prof. Dr. M.A. Mottalib
Head of the
Department of Computer Science and Engineering (CSE)

Signature of the Supervisor

_____

Md. Mohiuddin Khan
Assistant Professor
Department of Computer Science and Engineering (CSE)

Authors

_____

Md. Tanbin Rahid Kyser
ID: 104426

_____

Zonayed Ahmed
ID: 104429

Abstract

Cloud computing has become a new age technology that has got a huge potentials in enterprises and markets. It involves over a distributed computing over a network where a program or application may run on many connected computers at the same time. As cloud based system has become more and more numerous and dynamic, resource provisioning is become more and more challenging. At the same time energy has become an issue in this days. So, here we discuss resource allocation constraints with the energy and QoS based. Here QoS is the major part and Energy is minor part but we tried to consider both at the same time. Now, Energy and QoS are both depends on resource utilization. This utilization parameter based on two terms but this energy and QoS are reverse proportional. So, if we need to reach an equilibrium state then we need to use some kind of heuristics method. Here we use game theoretic approach for reaching an equilibrium state for getting a QoS and Energy aware system. Basically resource allocation depends on job scheduling. Several existed job scheduling algorithms was implemented. This algorithms are selected according to SLA. This phenomena is known as automated service provisioning for cloud computing. In this paper we provide a well-known Selective Approach for job scheduling including two popular algorithms known as Max-Min and Min-Min scheduling.

Contents

List of Figures

Chapter 1

Introduction

1.1 Overview

Cloud computing specially refers to a computing machine or group of hardware machines who commonly referred as the server connected through a communication network such as internet. Any user who permission to access can run through his application on cloud computing. In common usage cloud is the metaphor of cloud computing.

Cloud computing infrastructures break down the physical barriers inherent in isolated systems, automate the management of the group of resources as a single entity, and provide computational power and data storage facilities to users, ranging from a user accessing a single laptop to the allocation of thousands of computing nodes distributed around the world. Cloud computing is a natural evolution for data and computation centers with automated systems management, workload balancing, and virtualization technologies. Cloud-based services integrate globally distributed resources into seamless computing platforms. Recently, a great deal of applications are increasingly focusing on third-party resources hosted across the Internet and each has varying capacity.

It is known that the underlying cloud computing environment is inherently large scale, complex, heterogeneous and dynamic, globally aggregating large numbers of independent computing and communication resources and data stores. In an open cloud computing framework, scheduling tasks with guaranteeing QoS constrains present a challenging technical problem.

1.2 Problem statement

In our overall thesis we find the problem of cloud computing. At first we came of the points that affect the cloud computing. We came to the point of resource allocation problem in cloud computing. One of the key features of cloud computing is the capability of acquiring and releasing resources on-demand. The objective of a service provider in this case is to allocate and de-allocate resources from the cloud to satisfy its service level objectives (SLOs), while minimizing its operational cost. However, it is not obvious how a service provider can achieve this objective. In particular, it is not easy to determine how to map SLOs such as QoS requirements to low-level resource requirement such as CPU and memory requirements. Furthermore, to achieve high agility and respond to rapid demand fluctuations such as in flash crowd effect, the resource provisioning decisions must be made online.

Designing energy-efficient data centers has recently received considerable attention. This problem can be approached from several directions. For example, energy efficient hardware architecture that enables slowing down CPU speeds and turning off partial hardware components has become commonplace. Energy-aware job scheduling and server consolidation are two other ways to reduce power consumption by turning off unused machines. Recent research has also begun to study energy-efficient network protocols and infrastructures. A key challenge in all the above methods is to achieve a good trade-off between energy savings and application performance.

1.3 Motivation

In the cloud computing environment, task scheduling and resource assignment have been unified managed by providers through virtualized technology. They have been used to hide and complete users' tasks transparently. Task scheduling becomes more complex because of the transparent and dynamic flexibility of cloud computing system, and the different needs for recourses of different applications. Task scheduling strategies only focus on equity or efficiency

will increase the cost of time, space, and throughput and improve the quality of service of the entire cloud computing at the same time. The task scheduling goals of Cloud computing is provide optimal tasks scheduling for users, and provide the entire cloud system throughput and QoS at the same time. Specific goals are load balance, quality of service (QoS), economic principle, and the optimal operation time and system throughput.

1.4 Research Challenges

Cloud computing technology provides the method of sharing basic infrastructure. It bring the computing resources and storage resources in different geographical positions into a resource pool through virtual technology. Users need to apply before using it, and we need to release resources after using it so that the resources can be reused. In this way, the cloud computing center can provide high-performance computing resources and huge storage resources which are simple and low cost.

With further development and maturity of cloud computing technology, the features of efficient, flexible, and customizable provide a new way to solve the problems encountered in the operation process of the scientific workflow. When using the cloud platform, researchers need to upload resources data sets to the cloud computing platform. Because the scale of resources set may be very large, there are bandwidth limitations between different parts and some set of resources can only be stored in the specified resource center, researchers cannot upload all the set of data resources to the same resource center or upload all the set of data resources to every resource center.

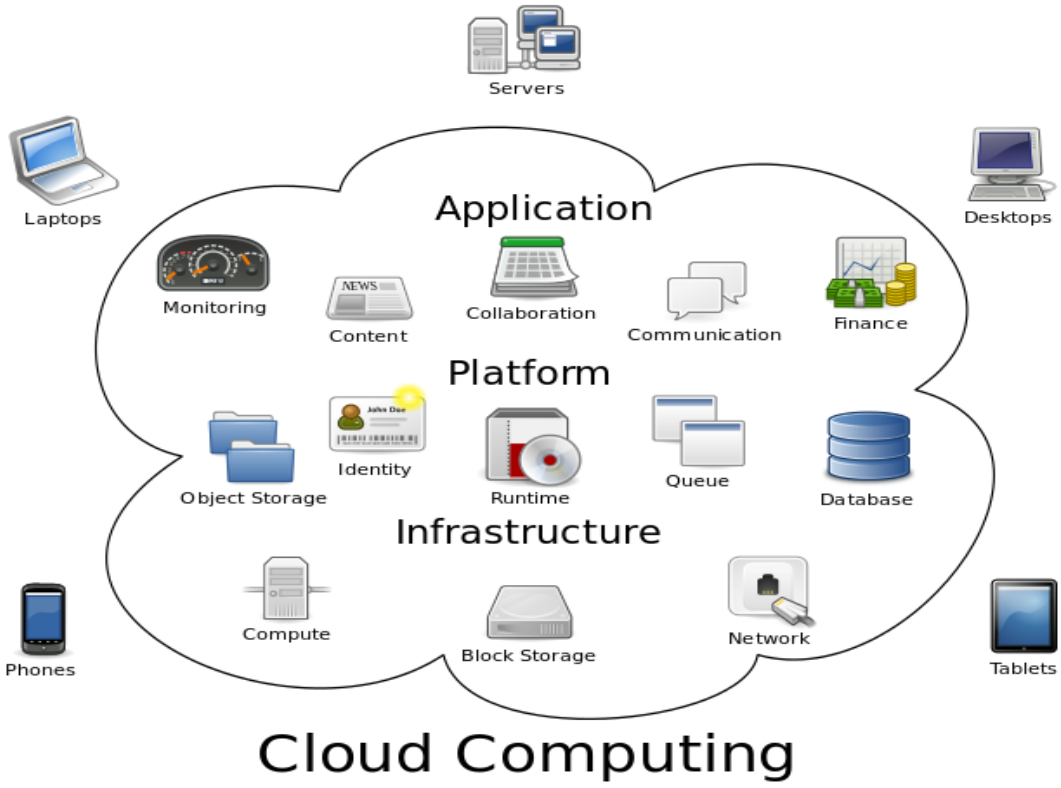
While they need to upload different set of data resources to different data resource centers, so that the tasks of scientific workflow can be executed in parallel. On one hand, it increases the user fees for the use of cloud resources. So studying an effective and fair task scheduling algorithm in a cloud environment is not only important in resources transmission and reducing the transmission of user fees, but also important in enhancing the implementation of the performance and user satisfaction.

Chapter 2

Literature Review

2.1 Cloud Computing

The main idea behind cloud computing is not a new one. John McCarthy in the 1960s already envisioned that computing facilities will be provided to the general public like a utility. The term "cloud" has also been used in various contexts such as describing large ATM networks in the 1990s.



[1]

Figure 3: Users and Services of Cloud Computing

However, it was after Google's CEO Eric Schmidt used the word to describe the business model of providing services across the Internet in 2006, that the term really started to gain popularity. Cloud computing architecture is given below. Generally speaking, the architecture of a cloud computing environment can be divided into 4 layers: the hardware/datacenter layer, the infrastructure layer, the platform layer and the application layer, as shown in Fig.2.

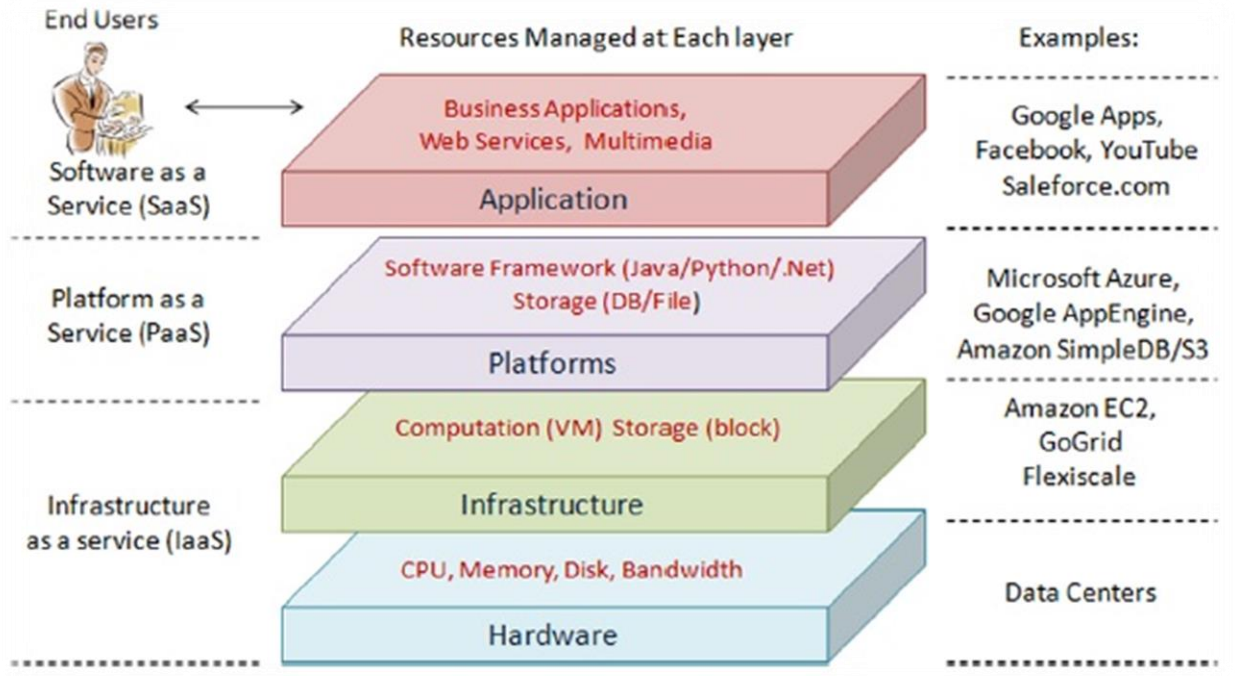


Figure 4: Layers of Cloud Computing

a. Infrastructure layer:

Also known as the virtualization layer, the infrastructure layer creates a pool of storage and computing resources by partitioning the physical resources using virtualization technologies such as Xen, KVM and VMware. The infrastructure layer is an essential component of cloud computing, since many key features, such as dynamic resource assignment, are only made available through virtualization technologies.

b. Platform layer:

Built on top of the infrastructure layer, the platform layer consists of operating systems and application frameworks. The purpose of the platform layer is to minimize the burden of deploying applications directly into VM containers. For example, Google App Engine operates at the platform layer to provide API support for implementing storage, database and business logic of typical web applications.

c. Application layer:

At the highest level of the hierarchy, the application layer consists of the actual cloud applications. Different from traditional applications, cloud applications can leverage the automatic-scaling feature to achieve better performance, availability and lower operating cost.

Types of cloud computing:

There are many issues to consider when moving an enterprise application to the cloud environment. For example, some service providers are mostly interested in lowering operation cost, while others may prefer high reliability and security.

Types of cloud computing is given bellow-

1. Public clouds: A cloud in which service providers offer their resources as services to the general public.
2. Private clouds: Also known as internal clouds, private clouds are designed for exclusive use by a single organization.
3. Hybrid clouds: It is a combination of public and private cloud models that tries to address the limitations of each approach.

2.2 Task scheduling in cloud computing:

Software developers and third-party service providers often deploy applications that exhibit dynamic behavior in terms of workload patterns, availability, and scalability requirements. Typically, Cloud computing thrives on highly varied and elastic services and infrastructure demands.

Traditional task scheduling algorithm focuses on efficiency or cost. It has pertinence for tasks of Specific style and targets for specific types of tasks such as to target the least finishing time, the most optimum availability, and the least cost. These scheduling policies have better efficiency or better cost advantages, but can cause uneven loading, unilateral advantages of efficiency and cost. The expectations of enterprises integrated QoS are not balanced, that means service requirement quality of task scheduling in cloud environment cannot meet the expectations of users. Therefore, it's more important that the efficiency and cost of task scheduling model are balanced. Task scheduling in cloud computing environment should not only meet the balance between efficiency and cost, but also meet the equitable resources distribution.

Divide users 'tasks according to the QoS, users have a clear direction of resources service. The game between efficiency and cost is based on meeting users' benefits or fee requirement and then seeking optimal value or equilibrium point. And finally achieve double-win in user efficiency and cost. We call this process efficiency optimization.

Cloud computing uses virtualized technology to pack resources and then supply for users. These new traits require us to establish a link between users and virtual resources. And we need to develop new applicable task scheduling and resource mapping mechanism.

We would like to discuss two algorithms and evaluation.

(1) Berger Model:

Berger model theory on distributive justice in the field of social distribution was first introduced into the job scheduling algorithm in cloud computing. Through the expansion of CloudSim platform, job scheduling algorithm based on Berger model is implemented. The validity of the algorithm is verified on the extended simulation platform. Berger Model considers two parameters:

(a) Completion time: For real-time demand higher users, the task needs to be completed within as little time as possible.

(2) Bandwidth: If user requests need higher communication bandwidth, the bandwidth requirements need to give priority to.

To measure user satisfaction according to different QoS parameters, different quantification evaluation criteria need to be established for different QoS parameters.

The cloud computing uses virtualization technology to map host resources to the virtual machine layer, therefore the job scheduling in cloud computing is implemented in application layer and virtual machine layer. Scheduling is to map task to resources according to a certain principle of optimization. Cloud computing makes required resources of task manifest in the form of a virtual machine. The resource search process is converted the search process of a virtual machine.

Suppose the resource characteristics set of the virtual machine VM_i is

$C_i = [C_{i1}, C_{i2}, C_{ir}]$; r=1;2;3

Where C_{i1}, C_{i2}, C_{i3} denotes CPU, memory and bandwidth respectively. The performance vector of VM_i is:

$VM_i = [EC_{i1}, EC_{i2}, EC_{i3}]$

Where EC_{ir} is the corresponding performance value of characteristics C_{ir} of VM_i

So, the general expectations vector of the type i task is:

$e_i = [e_{i1}, e_{i2}, e_{i3}]$

And,

$$\sum_{j=1}^{3} e_{ij} = 1$$

Where the first dimensional e_{i1} denotes weight for CPU number, the second dimensional e_{i2} denotes weight for memory; the third dimensional e_{i3} denotes weight for bandwidth.

Pseudo code of resource selection process as follows:

Begin
Define initial vector of general expectation e_i
for Ti, i = 1 to m {
    Select VM_i according to the require of T_i;
}
for i = 1 to t{//normalization
    for j = 1 to 3{
        W_i = Computing GX_{IJ};
    }
}
for i = 1 to t{
    eucDis[i] = Computing Euclidean distance of W_i and e_i
}
Select minimum Euclidean distance D_o from eucDis[i];
Binding T_i to D_o;
End;

The actual total completion time T_f ($T_f = T_{wait}+T_{exec}+T_{Trans}$) of $task_{Ti}$ is from submit the task to return results. The expectations completion timeT_{expt} of $task_{Ti}$ is assigned by user. Thus, the Eq.is:

$$J_i = \theta \ln T_f / T_{expt}$$

We can use this equation to calculate bandwidth:

$$J_i = \theta \ln BW_{vm} / BW_{user}$$

When user task needs a variety of QoS requirements, we should use integrated general expectation.

$$\bar{e}_{ij} = 1/n \sum_{j=1}^{n} e_{ij}, \quad (n = 1,2,3)$$

The equation gives us the integrated expectation where e_{ij} denotes the j expectation value of task T_i

So, $J_i = \theta \ln AR_i/ER_i$ equation converts to

$$J_i = \sum_{j} |\ln AR_j/ER_j|, \quad j = 1, \ldots, 4$$

As to the adjustment on the initial vector of general expectations, it can be constrained by defining threshold value $|J_i|>1$. When $|J_i| > 1$, the system automatically adjusts the vector of general expectations. Tasks can be repeated to obtain a reasonable vector of general expectations.

Based on completion time, the algorithm is:

(1) According to QoS classification, the general expectation of tasks, which acts as the fairness constraints for selection and allocation of resource, is established.

(2) According to parameterized task characteristics and their corresponding general expectation constraint, select the better resources of virtual machine to run the task.

(3) Calculate the value of fairness justice function based on the results of resource allocation. Statistics user satisfaction and adjust model.

Disadvantage:

As to the initial value of general expectation vector, what the Berger model gives is empirical value. We need to build a fuzzy neural network of QoS feature vector of task and parameter vector of resource based on the non-linear mapping relationship between QoS and resource. Through learning, more accurate vector value of the general expectation can be obtained.

The flowchart of the algorithm is given below:

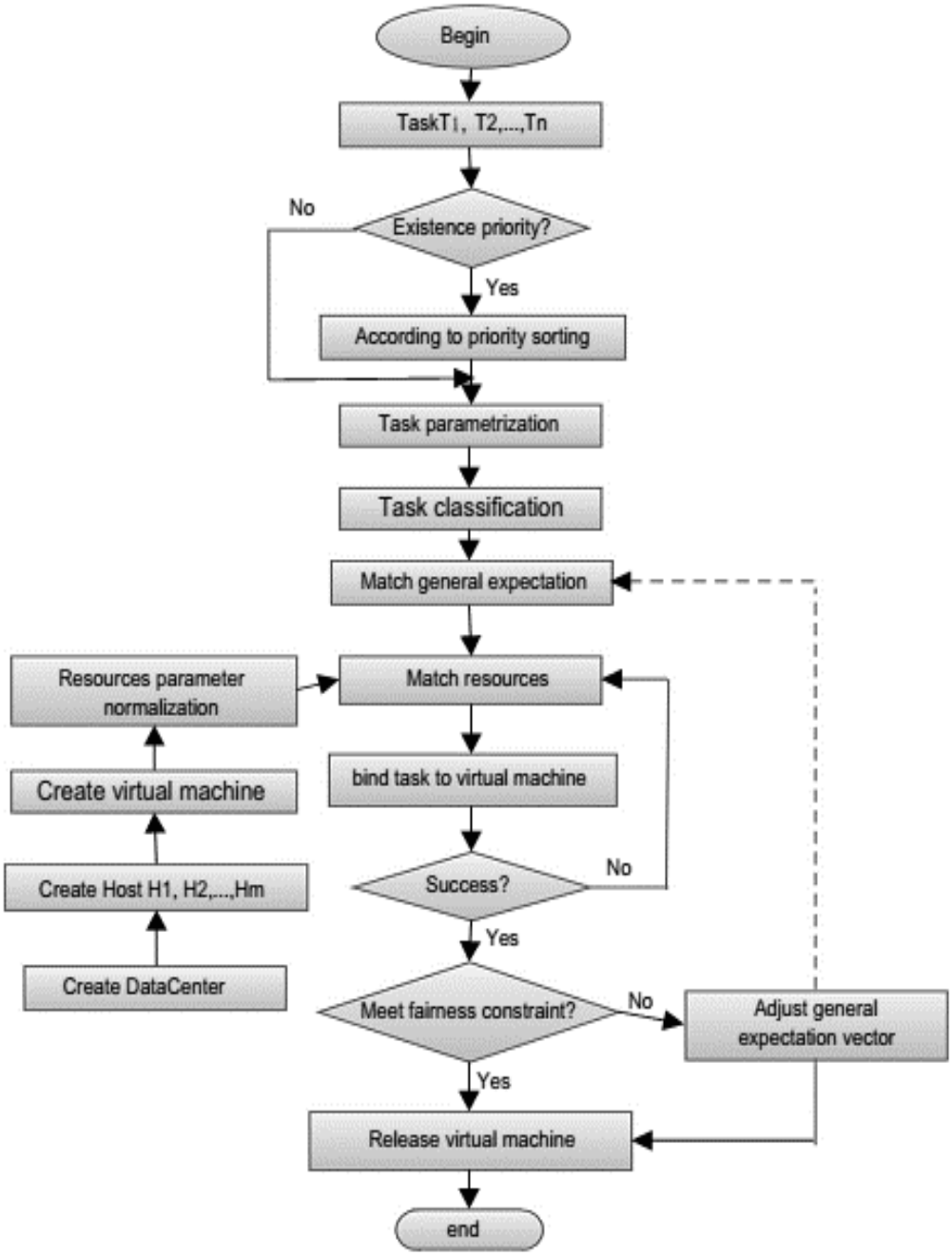


Figure 5: Algorithm flowchart of Berger Model Algorithm

(2) Li et al. (CMMS):

Li et al. uses CMMS (Cloud Min-Min Scheduling) for static resource allocation in cloud computing. Min-min is a popular greedy algorithm. The original min-min algorithm does not consider the dependencies among tasks. So in the dynamic min-min algorithm, we need to update the mappable task set in every scheduling step to maintain the task dependencies. Tasks in the mappable task set are the tasks whose predecessor tasks are all assigned.

The algorithm is:

Require: A set of tasks, different clouds,Mmatrix
Ensure: A schedule generated by CMMS.
1: form a mappable task set P
2: while there are tasks not assigned do
3: update mappable task set$P$
4: for i: task$v_i \in P$do
5: Send task check requests ofvi to all other schedulers
6: Receive the earliest resource available time responses from all other schedulers
7: Find the cloud$C(vi)$ giving the earliest finish time of vi, assuming no other task preempts vi
8: end for
9: Find the task-cloud pair(vk,$Cmin(vk)$)with the earliest finish time in the pairs generated in for-loop
10: Assign taskvk to cloud $D(vk)$
11: Remove$vk$from P
12: update the mappable task set$P$
13: end while

Disadvantage:

Sometimes only min-min algorithm can't perform better than min-max. So, we need to use both the algorithms to ensure maximum utility.

2.3 Energy consumption in cloud computing:

Improving energy efficiency is another major issue in cloud computing. It has been estimated that the cost of powering and cooling accounts for 53% of the total operational expenditure of data centers. In 2006, data centers in the US consumed more than 1.5% of the total energy generated in that year, and the percentage is projected to grow 18% annually. Hence infrastructure providers are under enormous pressure to reduce energy consumption. The goal is not only to cut down energy cost in data centers, but also to meet government regulations and environmental standards.

Designing energy-efficient data centers has recently received considerable attention. This problem can be approached from several directions. For example, energy efficient hardware architecture that enables slowing down CPU speeds and turning off partial hardware components has become commonplace. Energy-aware job scheduling and server consolidation are two other ways to reduce power consumption by turning off unused machines.

Recent research has also begun to study energy-efficient network protocols and infrastructures. A key challenge in all the above methods is to achieve a good trade-off between energy savings and application performance. In this respect, few researchers have recently started to investigate coordinated solutions for performance and power management in a dynamic cloud environment.

Lack of energy-conscious provisioning techniques may lead to overheating of Cloud resources (compute and storage servers) in case of high loads. This in turn may result in reduced system reliability and lifespan of devices. Another related issue is the carbon dioxide (CO2) emission that is detrimental to the physical environment due to its contribution in the greenhouse effect. All these problems require the development of efficient energy-conscious provisioning policies at resource, VM, and application level.

Chapter 3

Proposed Approach

From the above knowledge of task scheduling due to resource allocation in cloud computing. We can define that task scheduling problem is related with Knapsack problem. Because if we can see the analogies between them in knapsack problem there are some boxes with weight and cost so does in resource allocation problem we can see that here resources with their cost and capabilities moreover the execution time and capability to run a job.

3.1 Strategy Selection & Optimization:

As knapsack problem is an np-complete problem but its optimization is np-hard. Due to the np-hard problem we can see that here no optimal solution is existed. But still we can implement the optimal strategy by using some kind of greedy approach. But at the same time if we see the equation of energy and scheduling in cloud. We can see they are almost inversely proportional but there is no linear equation which we can draw between them. So, we cannot implement greedy strategy.

Greedy strategy cannot be implemented because there is no linear equation possible between QoS and energy. As we are increasing the QoS means we are increasing the compute then the power is increasing at the same time. On the other hand the amount of energy is also increasing at the same time. So, whenever we are increasing the QoS the energy consumption is decreasing. For QoS parameter we considered the Mean execution time according to SLA. According SLA which have higher priority will be executed with the min execution time. Due to considering energy consumption we must consider the

computation power. So, if we have capabilities of computational devices we will migrate VM's according to the SLA. At the same time we can consider the energy parameters and shut down the hosts which is not considerably well for us.

3.2 Scenario:

Here in this scenario we have implemented a Selective algorithm to choose between 2 algorithms which is very popular in names of MAX-MIN and MIN-MIN algorithm.

The reason behind the choosing this algorithm is this is a game theoretic approaches which is verily known as Fairness in a game or No Sum Game. So, here we get the idea of completing the task within time.

The main goal of our algorithm is to provide a better execution time to the makespan of tasks on machines and provide better quality of service we design an algorithm that assigns tasks to best machines in such a way that it provides satisfactory performance to both, cloud users and providers. The algorithm is designed to make a choice between Min-Min and Max-Min scheduling algorithm based on certain criteria.

Min-Min algorithm follows the following procedure:

Phase1: It first computes the completion time of cloudlets on each VM and then for each cloudlet it chooses the VM which processes the cloudlet in minimum possible time.

Phase2: Then among all the cloudlets in MT the cloudlet with minimum overall completion time is chosen and is allocated to VM on which minimum execution delay is expected. The cloudlet is removed from the list of MT and the process continues until MT list is empty.

Max-Min algorithm works similarly like Min-Min in phase 1. Unlike Min-Min, Max-Min allocates the cloudlet with maximum expected completion time to VM (i.e. Longer task is first allocated a VM). Max-Min algorithm allocates considerably long task (cloudlet) (long in comparison to other given set of tasks) to one VM and smaller tasks to another VM based on their completion time on each VM. Min-Min on the other hand is useful when all the tasks are almostnof same size. When all the tasks are of similar sizes then min-min allocates the cloudlet to a VM on which it executes in minimum possible time. This not only ensures that overall makespan of tasks on VM is reduced but also provide minimum delays in processing of tasks.

3.3 Flowchart:

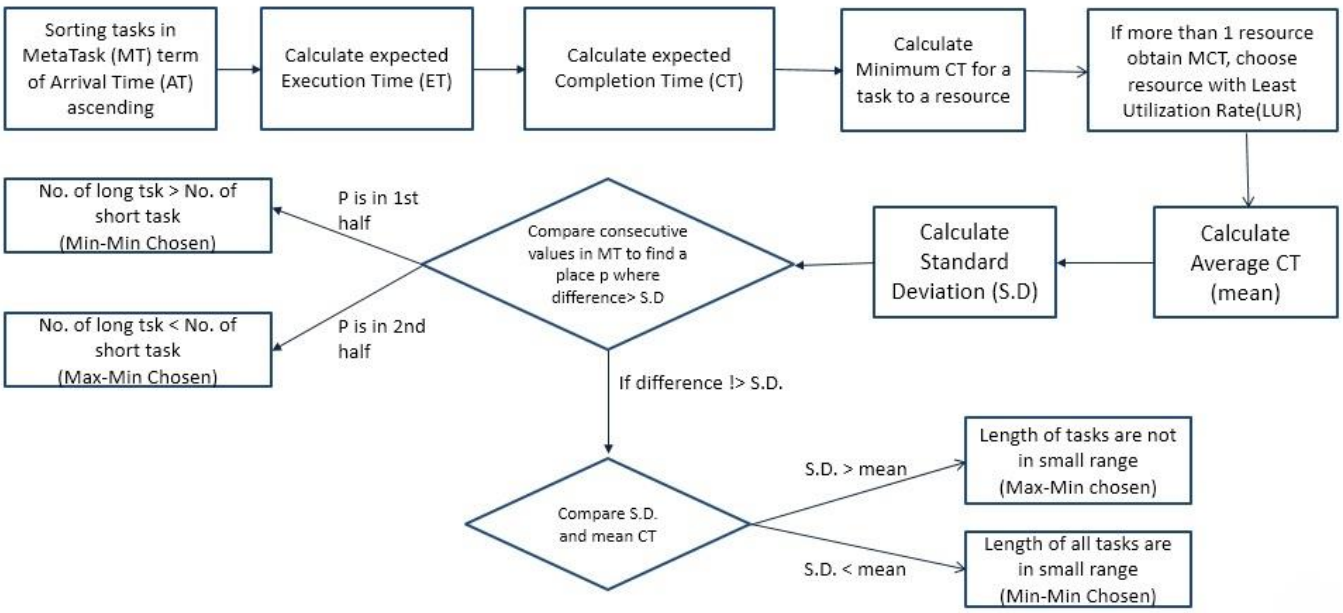


Figure 6: Flowchart of Proposed Method

3.4 Algorithm:

(1) Sort tasks in meta-task MT ascending.
(2) While there are tasks in MT
(3) for all tasks t_i in MT
(4) for all machines m_j
(5) $Ct_{ij} = Et_{ij} + r_j$
(6) for all tasks t_i in MT
(7) Find minimum CT_{ij} and resource m_j that
(8) If there is more than one resource that obtain
(9) Select resource with least utilization rate
(10) Calculate average CT (mean)
(11) Calculate standard deviation (S.D.)
(12) Find place p in MT where difference of two consequence CT_{ij} > S.D.
(13) If p<= s/2 or S.D. > mean then
(14) Assign t_1 to resource ml that obtains CT_{1l}
(15) else
(16) Assign t_s to resource ml that obtains CT_{sl}
(17) Delete assigned task from MT
(18) End While

3.5 Necessary Equations:

- Setup an environment in CloudSim.
- Find the expected execution time,

  E = FileSize (Cloudlets)/ MIPS (VM)

- Find the completion time by considering that all the tasks in a first Serve First Come manner to the resources

  C = E + R; Where, R = waiting time for a resource. For the first resource it is 0.

- Least Utilization Rate,

$$ru_j = \frac{\sum_{i \text{ where } t_i \text{ has been executed on } m_j} (te_i - ts_i)}{T}$$

Where, te_i = finishing time of executing t_i on resource m_j

ts_i = starting time of executing t_i on resource m_j

T = the total application execution time so far;

$$T = \max_{i \text{ where } t_i \text{ is executed till now}} (te_i - ts_i)$$

- Then find the mean for the particular resources.

Mean = $(C_1 + C_2 + \ldots\ldots\ldots\ldots + C_n)/ n$.

- Then find the Standard Deviation.

S.D. = $\sqrt{(C_n - \text{Mean})^2}/ \sqrt{n}$

3.6 Description of the algorithms:

(a) Max-Min algorithm:

Here we have an example to explain the MAX-MIN algorithm. For example considering the following scenario.

	VM 1	**VM 2**
Task 1	1.6	0.8
Task 2	6.6	3.3
Task 3	9.6	4.8

Table1: Scenario of tasks and resources (VMs)

From the table we need to find the minimum value of the tasks and then make a set then find the highest value from the set then put that task to that VM. And remove that row.

After updating the table looks alike this-

	VM 1	**VM 2**
Task 1	1.6	5.6
Task 2	6.6	8.1

Then find the minimum values again and put the values to a set and find the maximum again and put that to the resource.

And update the table remove the task.

	VM1	**VM2**
Task1	8.2	5.6

Then assign the final task to the resource.

(b) <u>Min-MIN algorithm:</u>

Min-Min algorithm uses a similar technique like the previous MAX-MIN algorithm. Here in this section at first we need to find the minimum value instead of creating a set of minimum values.

So, at first find the minimum value here we can see that Task 1 on VM 2 is the minimum value so allocate VM2 to Task 1.

	VM 1	VM 2
Task 1	1.6	0.8
Task 2	6.6	3.3
Task 3	9.6	4.8

Now, here update the table after allocating task 1 to VM 2.

	VM 1	VM 2
Task 2	6.6	4.1
Task 3	9.6	5.6

After updating the table here we need find the minimum value and allocate that to the VM. Now, here we can see that minimum value is 4.1 so assign the task to that VM.

After that again update the table and continue the procedure.

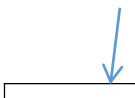
	VM 1	VM 2
Task 3	9.6	8.9

3.7 Case Study:

Case 1 (min-min better than max-min):

Assume that minimum CT_{ij} for each task is found and S.D. is calculated too (i.e. lines 1 to 11 of the algorithm is executed).

Index	**1**	**2**	**3**	**4**	**5**	**6**
CT_{ij}	1	50	60	80	100	120



p = 1< s/2= 1<3

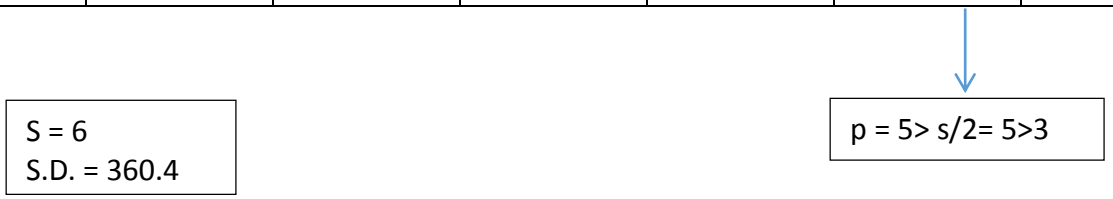
S = 6
S.D. = 38.18

In this case, there exists one short task and five long tasks. So, according to our algorithm, Min-Min outperforms Max-Min. As it can be seen, occurrence of the place of difference is in the first half, so Min-Min is selected to assign next task.

Case 2 (max-min better than min-min):

Assume that minimum CT_{ij} for each task is found and S.D. is calculated too (i.e. lines 1 to 11 of the algorithm is executed).

Index	1	2	3	4	5	6
CT_{ij}	10	20	30	50	60	1000



In this case, there exists one long task and five short tasks. So, according to our algorithm, Max-Min outperforms Min-Min. As it can be seen, occurrence of the place of difference is in the second half, so Max-Min is selected to assign next task.

Chapter 4

Dataset & Evaluation

For our working purpose we use the CloudSim toolkit. CloudSim is a support model for simulating cloud frameworks or cloud environments.

4.1 Introduction to CloudSim

CloudSim is a simulation toolkit made by University of Melbourne's cloud lab. For a formal definition of CloudSim is given below.

> ***A suitable alternative is the utilization of simulations tools, which open the possibility of evaluating the hypothesis prior to software development in an environment where one can reproduce tests. Specifically in the case of Cloud computing, where access to the infrastructure incurs payments in real currency, simulation-based approaches offer significant benefits, as it allows Cloud customers to test their services in repeatable and controllable environment free of cost, and to tune the performance bottlenecks before deploying on real Clouds.***

The CloudSim simulation layer provides support for modeling and simulation of virtualized Cloud-based data center environments including dedicated management interfaces for VMs, memory,

storage, and bandwidth. The fundamental issues, such as provisioning of hosts to VMs, managing application execution, and monitoring dynamic system state, are handled by this layer.

There is a clear distinction at this layer related to provisioning of hosts to VMs. A Cloud host can be concurrently allocated to a set of VMs that execute applications based on SaaS provider's defined QoS levels. This layer also exposes the functionalities that a Cloud application developer can extend to perform complex workload profiling and application performance study.

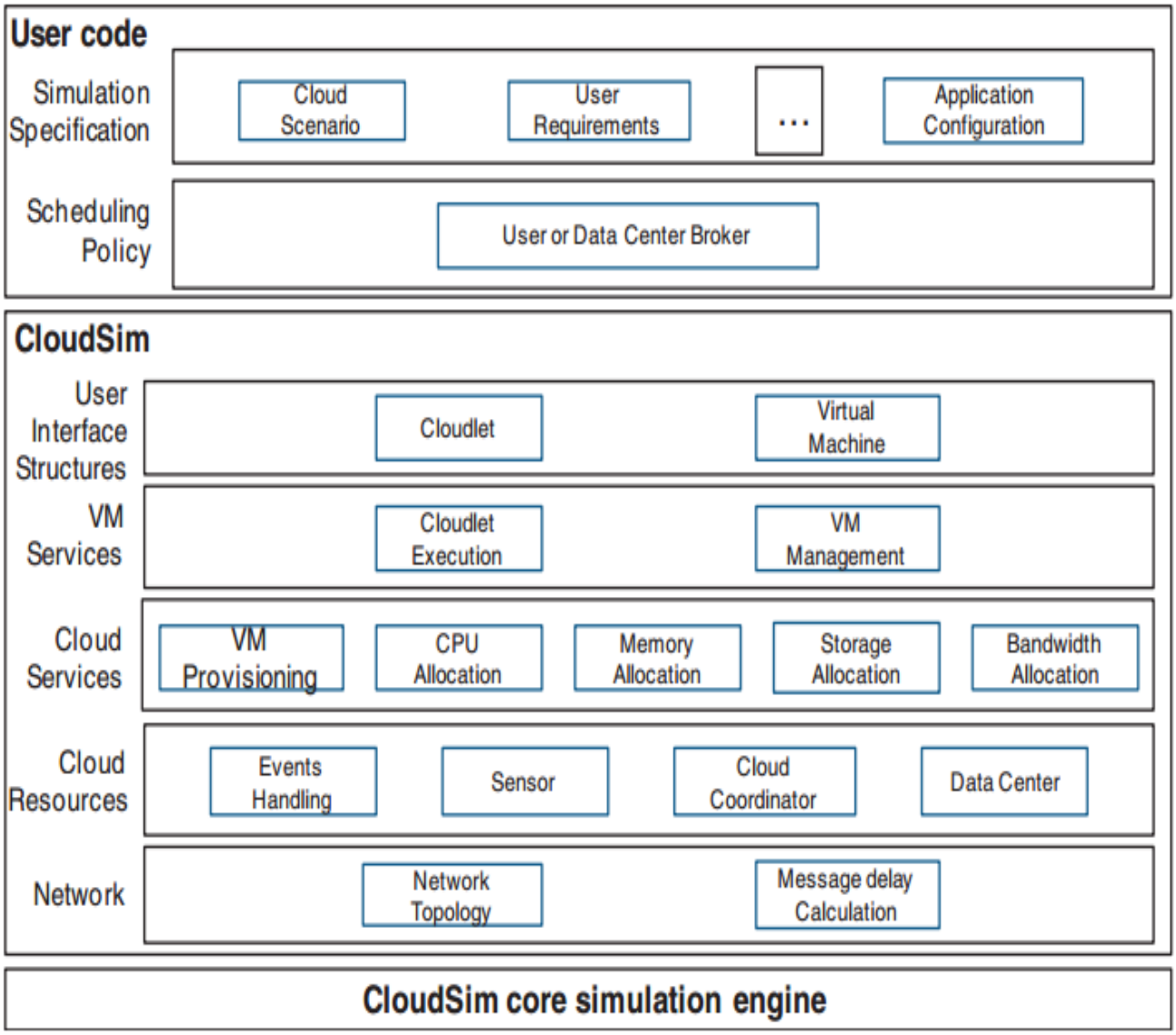


Figure 7: Architecture of CloudSim

4.2 Key terminologies of CloudSim:

Some of the key terminologies of CloudSim is given below-

- **Cloudlet:** This class models the Cloud-based application services. Every application service has a pre-assigned instruction length and data transfer (both pre and post fetches) overhead that it needs to undertake during its life cycle.

- **CloudletScheduler:** This abstract class is extended by the implementation of different policies that determine the share of processing power among Cloudlets in a VM. There are two types of **cloudletscheduler:**

- ***CloudetSchedulerSpaceShared***,

- ***CloudletSchedulerTimeShared*.**

- **Datacenter:** This class models the core infrastructure-level services (hardware) that are offered by Cloud providers (Amazon, Azure, and App Engine). It encapsulates a set of compute hosts that can either be homogeneous or heterogeneous with respect to their hardware configurations. Furthermore, every Datacenter component instantiates a generalized application provisioning component that implements a set of policies for allocating bandwidth, memory, and storage devices to hosts and VMs.

- **DatacenterCharacterstics:** This class models a broker, which is responsible for mediating negotiations between SaaS and Cloud providers; and such negotiations are driven by QoS requirements. The broker acts on behalf of SaaS providers.

- **VM:** This class models a VM, which is managed and hosted by a Cloud host component. Every VM component has access to a component that stores the following characteristics

related to a VM: accessible memory, processor, storage size, and the VM's internal provisioning policy that is extended from an abstract component called theCloudletScheduler.

- **VmAllocationPolicy:** This abstract class represents a provisioning policy that a VM Monitor utilizes for allocating VMs to hosts. The chief functionality of the VmmAllocationPolicy is to select the available host in a data center that meets the memory, storage, and availability requirement for a VM deployment.

4.2 Experimental Setup

For simulation purpose we are considering the following set of data-

No. of Datacenters	1
No. of Hosts	2
No. of VMs	2
No. of Cloudlets	3
Size of Cloudlet 1	16
Size of Cloudlet 2	50
Size of Cloudlet 3	30
MIPS of VM 1	10
MIPS of VM 2	20

Table: Input Data.

4.3 Performance Evaluation

Case 1: (Min-Min):

	VM1	VM2
Task 1	2	4
Task 2	3	6
Task 3	4	10
Task 4	30	70

Step: 1

	VM1	VM2
Task 2	5	6
Task 3	6	10
Task 4	32	70

Step: 2

	VM1	VM2
Task 3	9	10
Task 4	35	70

Step: 3

	VM1	VM2
Task 4	39	70

Step: 4

Case 1: (Max-Min):

	VM1	VM2
Task 1	2	4
Task 2	3	6
Task 3	4	10
Task 4	30	70

Step: 1

	VM1	VM2
Task 1	32	4
Task 2	33	6
Task 3	34	10

Step: 2

	VM1	VM2
Task 1	32	14
Task 2	33	16

Step: 3

	VM1	VM2
Task 1	32	20

Step: 4

Case 2: (Min-min):

	VM1	VM2
Task 1	2	4
Task 2	2	4
Task 3	3	6
Task 4	3	6

Step: 1

	VM1	VM2
Task 2	4	4
Task 3	5	6
Task 4	5	6

Step: 2

	VM1	VM2
Task 3	7	6
Task 4	7	6

Step: 3

	VM1	VM2
Task 4	7	12

Step: 4

Case 2: (Max-min):

	VM1	VM2
Task 1	2	4
Task 2	2	4
Task 3	3	6
Task 4	3	6

Step: 1

	VM1	VM2
Task 1	5	4
Task 2	5	4
Task 3	6	6

Step: 2

	VM1	VM2
Task 1	8	4
Task 2	8	4

Step: 3

	VM1	VM2
Task 1	8	8

Step: 4

Case 1: Evaluation

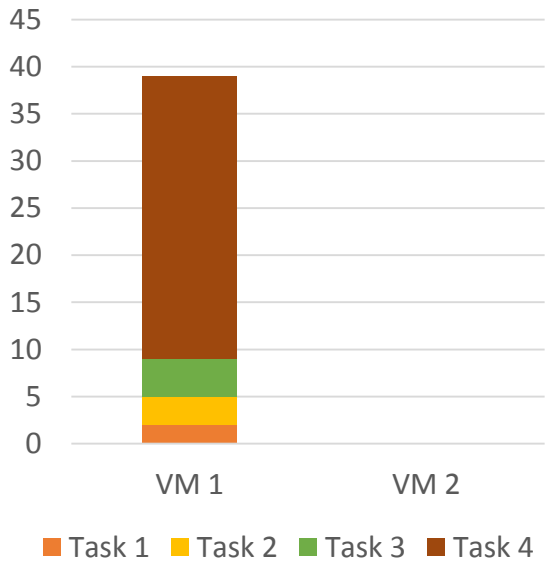


Figure 8: Case 1 min-min makespan

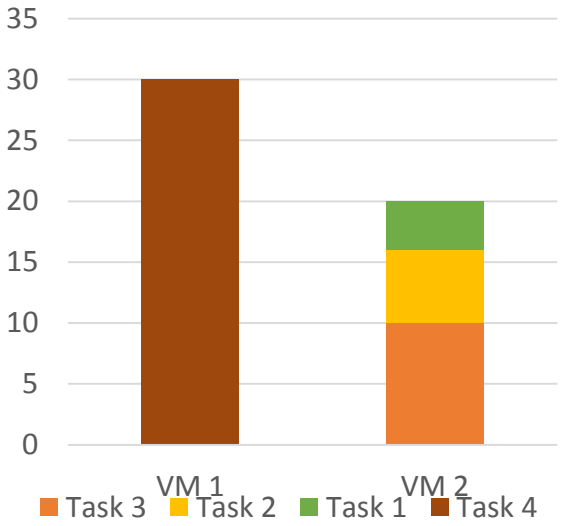


Figure 9: Case 1 max-min makespan

Case 2: Evaluation

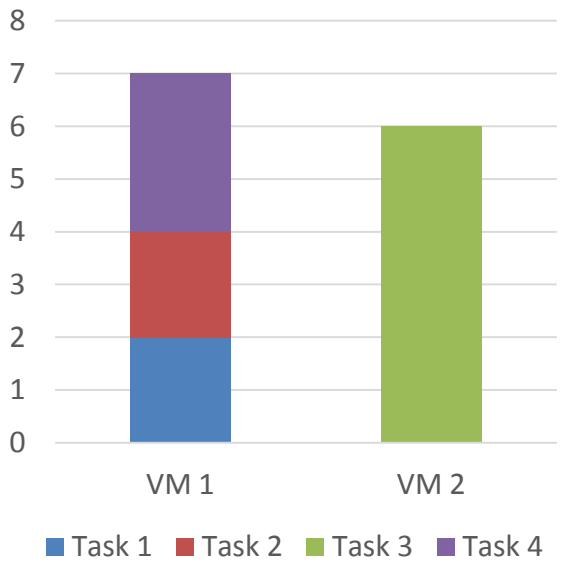


Figure 10: Case 2 min-min makespan

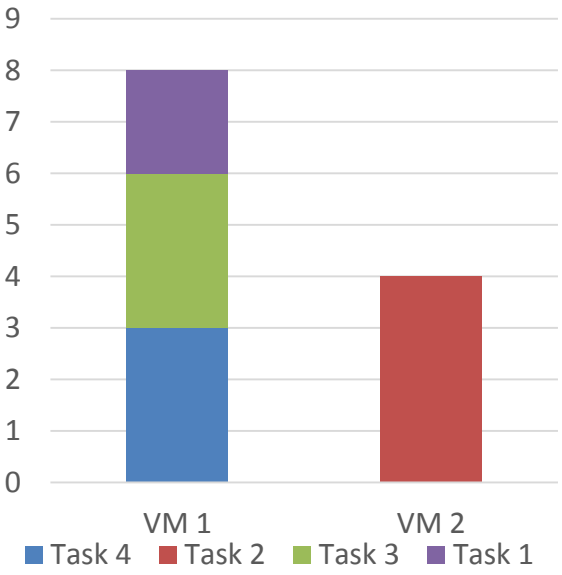


Figure 11: Case 2 max-min makespan

So, in case 1, tasks are distributed in both VMs for max-min makespan. So, max-min performs better in case 1.
Again, in case 2, tasks are more evenly distributed in both VMs and thus columspans are almost equal. Here, min-min performs better.

4.4 Comparison against FCFS

So, here is the comparison of the algorithms in a Time Sharing Mode with First Come First Serve.

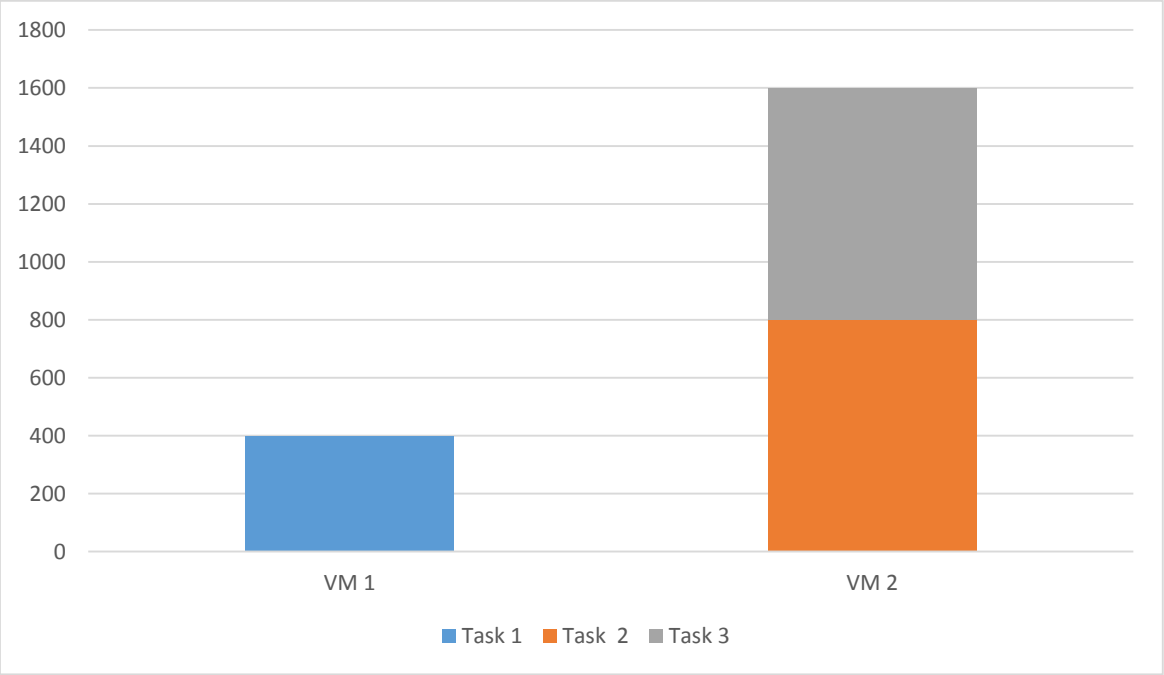


Figure: Column Span using FCFS during Time sharing

Now from our Selective algorithm we get the following data and columnspan.

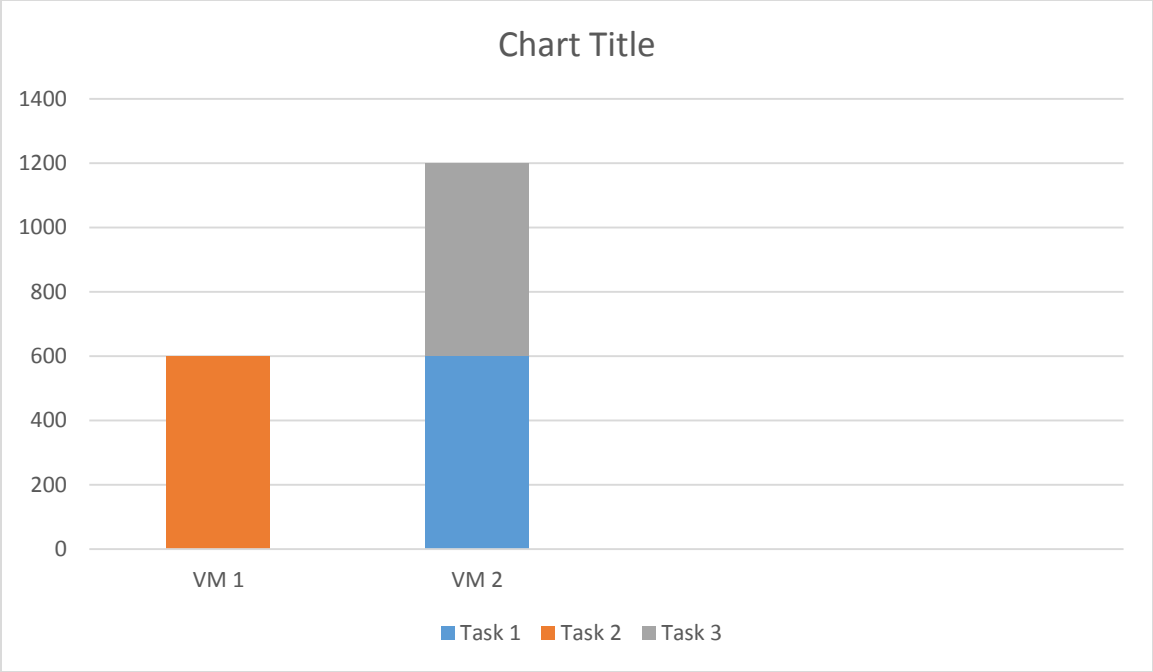


Figure: Columnspan during MAX-MIN Sharing

So, using a selective approach is way better than the existing algorithms. For the first case the columnspan is less so the execution time is less than the first one. So, it will use the less executional power than the previous one. So, the power consumption is another important thing here. So, the load balancing is done by this.

So, the algorithm provides a better perfection when the task have close range or the distance between two consecutive tasks is either very high or low. If they are scattered in this scenario algorithm works on a worst case scenario. So, the complexity of our algorithm during MAX-MIN and MIN-MIN O (C^2V).

Chapter 5 Future Work

Cloud computing has recently emerged as a compelling paradigm for managing and delivering services over the Internet. The rise of cloud computing is rapidly changing the landscape of information technology, and ultimately turning the long-held promise of utility computing into a reality.

As cloud computing is an evolving business so does research challenges are coming quite new here. For the task allocation methods we declare energy consumption but still we didn't simulate this in our work. So, we can easily make this possible by adding some sort of estimation workloads from the expectation and result by representing graph. Now we just simulate the algorithms only.

In future we need to work on several key aspects like VM migration, VM allocation and more on load balancing algorithm. So, that the power consumption and QoS awareness will raise by this time during allocating VMs in normal.

Cloud Computing involves investment of huge capital to install tremendous resources at a single Datacenter. Therefore, to maximize the use of tremendous capabilities offered by Cloud these resources need to be provisioned efficiently. The algorithm given in the paper ensures that all resources are efficiently utilized and jobs given by the user are executed with smaller delays. The provisioning technique given in the paper attempts to improve the throughput by minimizing Makespan. In future, dependencies between tasks can be modeled to further minimize the Makespan and maximize the throughput using Workflows.

Appendix:

CloudSim Simulation Code for Proposed Method

MAX-MIN:

```
package pkgfinal;

import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.LinkedList;
import java.util.List;
import org.cloudbus.cloudsim.*;
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;

public class Final {

        private static List<Cloudlet> cloudletList;

        private static List<Vm> vmlist;

        public static void main(String[] args) {
```

```
Log.printLine("Allocating cloudlets according to MAX-MIN algorithm");

try {

        int num_user = 1;   // number of cloud users
        Calendar calendar = Calendar.getInstance();
        boolean trace_flag = false;  // mean trace events

        // Initialize the CloudSim library
        CloudSim.init(num_user, calendar, trace_flag);

        // Second step: Create Datacenters
        //Datacenters are the resource providers in CloudSim. We need at list one of them to
run a CloudSim simulation
        Datacenter datacenter0 = createDatacenter("Datacenter");

        //Third step: Create Broker
        DatacenterBroker broker = createBroker();
        int brokerId = broker.getId();

        //Fourth step: Create one virtual machine
        vmlist = new ArrayList<Vm>();

        //VM description
        int vmid = 0;
        int mips = 10;
        long size = 10000; //image size (MB)
        int ram = 512; //vm memory (MB)
        long bw = 1000;
        int pesNumber = 1; //number of cpus
        String vmm = "Xen"; //VMM name

        //create two VMs
        Vm vm1 = new Vm(vmid, brokerId, mips, pesNumber, ram, bw, size, vmm, new
CloudletSchedulerTimeShared());

        //the second VM will have twice the priority of VM1 and so will receive twice CPU time
        vmid++;
        Vm vm2 = new Vm(vmid, brokerId, mips * 2, pesNumber, ram, bw, size, vmm, new
CloudletSchedulerTimeShared());

        //add the VMs to the vmList
        vmlist.add(vm1);
        vmlist.add(vm2);

        //submit vm list to the broker
        broker.submitVmList(vmlist);


        //Fifth step: Create two Cloudlets
        cloudletList = new ArrayList<Cloudlet>();
```

```
                        //Cloudlet properties
                        int id = 0;
                        long length = 4000;
                        long fileSize = 100;
                        long outputSize = 100;
                        UtilizationModel utilizationModel = new UtilizationModelFull();

                        Cloudlet cloudlet1 = new Cloudlet(id, length, pesNumber, fileSize, outputSize,
utilizationModel, utilizationModel, utilizationModel);
                        cloudlet1.setUserId(brokerId);

            long fileSize2 = 500;
                        id++;
                        Cloudlet cloudlet2 = new Cloudlet(id, length, pesNumber, fileSize2, outputSize,
utilizationModel, utilizationModel, utilizationModel);
                        cloudlet2.setUserId(brokerId);

            long fileSize3 = 300;
            id++;
            Cloudlet cloudlet3 = new
Cloudlet(id,length,pesNumber,fileSize3,outputSize,utilizationModel,utilizationModel,utilizationModel);
            cloudlet3.setUserId(brokerId);


                        //add the cloudlets to the list
                        cloudletList.add(cloudlet1);
                        cloudletList.add(cloudlet2);
            cloudletList.add(cloudlet3);


                        //submit cloudlet list to the broker
                        broker.submitCloudletList(cloudletList);

            Selection s1 = new Selection();

            int x1 = s1.executionTime(cloudlet1.getCloudletFileSize(),vm1.getMips());
            int x2 = s1.executionTime(cloudlet1.getCloudletFileSize(),vm2.getMips());
            int x3 = s1.executionTime(cloudlet2.getCloudletFileSize(), vm1.getMips());
            int x4 = s1.executionTime(cloudlet2.getCloudletFileSize(), vm2.getMips());
            int x5 = s1.executionTime(cloudlet3.getCloudletFileSize(), vm1.getMips());
            int x6 = s1.executionTime(cloudlet3.getCloudletFileSize(), vm2.getMips());

            int c1 = s1.completionTime(x1, 0);
            int c2 = s1.completionTime(x2, 0);
            int c3 = s1.completionTime(x3, x1);
            int c4 = s1.completionTime(x4, x2);
            int c5 = s1.completionTime(x5,x1,x3);
            int c6 = s1.completionTime(x6,x2,x4);

            double mean;

            mean = s1.mean(c1, c2, c3, c4, c5, c6, 3);
```

```
            double SD;

            SD = s1.standardDeviation(c1, c2, c3, c4, c5, c6, mean, 3);

            if(SD >= mean){
              Time[][] metaTask = new Time[][] {
        {new Time(cloudlet1.getCloudletFileSize()/vm1.getMips()),new
Time(cloudlet1.getCloudletFileSize()/vm2.getMips()) },
        {new Time(cloudlet2.getCloudletFileSize()/vm1.getMips()),new
Time(cloudlet2.getCloudletFileSize()/vm2.getMips())},
        {new Time(cloudlet3.getCloudletFileSize()/vm1.getMips()),new
Time(cloudlet3.getCloudletFileSize()/vm2.getMips())},

    };


    for(int i = 0; i < cloudletList.size(); i++){
      for(int j = 0; j < vmlist.size(); j++){
        metaTask[i][j].setCurrentValue(metaTask[i][j].getBaseValue());
      }
    }

    double tempMin = 123456;
    int tempMinRow = 0;
    int tempMinCol = 0;

    //inserted here
    int[] tCol= new int[cloudletList.size()];
    double[] minArray = new double[cloudletList.size()];

    for(int i = 0; i < cloudletList.size(); i++){
      for(int j = 0; j < vmlist.size(); j++){
        if (metaTask[i][j].getCurrentValue() < tempMin){
          tempMin = metaTask[i][j].getBaseValue();
          //tempMinRow = i;
          tempMinCol = j;
        }
      }
      minArray[i] = tempMin;
      tempMin = 123456;
      tCol[i] = tempMinCol;
    }
    double tempMax=0.0;

    for(int i=0;i<cloudletList.size();i++){
       if( minArray[i] > tempMax){
            tempMax = minArray[i];
            tempMinRow = i;
            tempMinCol = tCol[i];
       }
    }
```

```
        broker.bindCloudletToVm(cloudletList.get(tempMinRow).getCloudletId(),vmlist.get(tempMinCol).getId());

       for(int i = 0; i < cloudletList.size(); i++){
          if(i != tempMinRow)
             metaTask[i][tempMinCol].setCurrentValue(metaTask[i][tempMinCol].getCurrentValue() +
metaTask[tempMinRow][tempMinCol].getBaseValue());
        }

        double tempMin2 = 123456;
        int tempMinRow2 = 0;
        int tempMinCol2 = 0;

        //inserted here
        int[] tCol2= new int[cloudletList.size()];
        double[] minArray2 = new double[cloudletList.size()];

        for(int i = 0; i < cloudletList.size(); i++){
           for(int j = 0; j < vmlist.size(); j++){
                if (metaTask[i][j].getCurrentValue() < tempMin2) {
                   tempMin2 = metaTask[i][j].getCurrentValue();
                   //tempMinRow2 = i;
                   tempMinCol2 = j;
                }

           }
           minArray2[i] = tempMin2;
           tempMin2 = 123456;
           tCol2[i] = tempMinCol2;
        }

        double tempMax2=0.0;

        for(int i=0;i<cloudletList.size();i++){
           if( minArray2[i] > tempMax2 && minArray2[i] != tempMinRow){
              tempMax2 = minArray2[i];
              tempMinRow2 = i;
              tempMinCol2 = tCol2[i];
           }
        }


broker.bindCloudletToVm(cloudletList.get(tempMinRow2).getCloudletId(),vmlist.get(tempMinCol2).getId());

        for(int i = 0; i < cloudletList.size() && i != tempMinRow2; i++){
           metaTask[i][tempMinCol2].setCurrentValue(metaTask[i][tempMinCol2].getCurrentValue() +
metaTask[tempMinRow2][tempMinCol2].getBaseValue());
        }

        double tempMin3 = 123456;
        int tempMinRow3 = 0;
        int tempMinCol3 = 0;

        //inserted here
```

```
        int[] tCol3 = new int[cloudletList.size()];
        double[] minArray3 = new double[cloudletList.size()];

                for (int i = 0; i < cloudletList.size(); i++){
                for(int j = 0; j < vmlist.size(); j++){

                   if (metaTask[i][j].getCurrentValue() < tempMin3){
                      tempMin3 = metaTask[i][j].getCurrentValue();
                      //tempMinRow3 = i;
                      tempMinCol3 = j;
                      }

                   }
                minArray3[i]=tempMin3;
                tempMin3 = 123456;
                tCol3[i] = tempMinCol3;
                }

                 //Inserted here
                 double tempMax3 = 0.0;

                for(int i=0;i<cloudletList.size();i++){
                if (minArray3[i] != tempMinRow && minArray3[i] != tempMinRow2){
                   if( minArray3[i] > tempMax3 ){
                         tempMax3 = minArray3[i];
                         tempMinRow3 = i;
                         tempMinCol3 = tCol2[i];
                    }
                   }
                  }

broker.bindCloudletToVm(cloudletList.get(tempMinRow3).getCloudletId(),vmlist.get(tempMinCol3).getId());

                }

                //MIN MIN ALGORITHM

                else if(SD < mean){
                   Time[][] metaTask = new Time[][] {
           {new Time(cloudlet1.getCloudletFileSize()/vm1.getMips()),new
Time(cloudlet1.getCloudletFileSize()/vm2.getMips()) },
           {new Time(cloudlet2.getCloudletFileSize()/vm1.getMips()),new
Time(cloudlet2.getCloudletFileSize()/vm2.getMips())},
           {new Time(cloudlet3.getCloudletFileSize()/vm1.getMips()),new
Time(cloudlet3.getCloudletFileSize()/vm2.getMips())},

        };


        for(int i = 0; i < 3; i++){
           for(int j = 0; j < 2; j++){
              metaTask[i][j].setCurrentValue(metaTask[i][j].getBaseValue());
           }
```

```
        }

        double tempMin = 123456;
        int tempMinRow = 0;
        int tempMinCol = 0;

        for(int i = 0; i < 3; i++){
           for(int j = 0; j < 2; j++){
              if (metaTask[i][j].getBaseValue() < tempMin){
                 tempMin = metaTask[i][j].getBaseValue();
                 tempMinRow = i;
                 tempMinCol = j;
              }
           }
        }

        broker.bindCloudletToVm(cloudletList.get(tempMinRow).getCloudletId(),vmlist.get(tempMinCol).getId());

        for(int i = 0; i < 3; i++){
           if(i != tempMinRow)
              metaTask[i][tempMinCol].setCurrentValue(metaTask[i][tempMinCol].getCurrentValue() +
metaTask[tempMinRow][tempMinCol].getBaseValue());
        }

        double tempMin2 = 123456;
        int tempMinRow2 = 0;
        int tempMinCol2 = 0;

        for(int i = 0; i < 3; i++){
           for(int j = 0; j < 2; j++){
              if(i != tempMinRow){
                 if (metaTask[i][j].getCurrentValue() < tempMin2) {
                    tempMin2 = metaTask[i][j].getCurrentValue();
                    tempMinRow2 = i;
                    tempMinCol2 = j;
                 }
              }
           }
        }


broker.bindCloudletToVm(cloudletList.get(tempMinRow2).getCloudletId(),vmlist.get(tempMinCol2).getId());

        for(int i = 0; i < 3 && i != tempMinRow2; i++){
           metaTask[i][tempMinCol2].setCurrentValue(metaTask[i][tempMinCol2].getCurrentValue() +
metaTask[tempMinRow2][tempMinCol2].getBaseValue());
        }

        double tempMin3 = 123456;
        int tempMinRow3 = 0;
        int tempMinCol3 = 0;

        for (int i = 0; i < 3; i++){
```

```
            for(int j = 0; j < 2; j++){
               if(i != tempMinRow && i != tempMinRow2){
                  if (metaTask[i][j].getCurrentValue() < tempMin3){
                     tempMin3 = metaTask[i][j].getCurrentValue();
                     tempMinRow3 = i;
                     tempMinCol3 = j;
                     }
                  }
               }
            }


broker.bindCloudletToVm(cloudletList.get(tempMinRow3).getCloudletId(),vmlist.get(tempMinCol3).getId());


                  }

                  //bind the cloudlets to the vms. This way, the broker
                              // will submit the bound cloudlets only to the specific VM
                              // Sixth step: Starts the simulation
                              CloudSim.startSimulation();


                              // Final step: Print results when simulation is over
                              List<Cloudlet> newList = broker.getCloudletReceivedList();

                              CloudSim.stopSimulation();

                  printCloudletList(newList);

                              //Print the debt of each user to each datacenter
                              datacenter0.printDebts();

                              Log.printLine("Simulation finished!");
                        }
                        catch (Exception e) {
                              e.printStackTrace();
                              Log.printLine("The simulation has been terminated due to an unexpected error");
                        }
            }

            private static Datacenter createDatacenter(String name){

                        // Here are the steps needed to create a PowerDatacenter:
                        // 1. We need to create a list to store
                        //    our machine
                        List<Host> hostList = new ArrayList<Host>();

                        // 2. A Machine contains one or more PEs or CPUs/Cores.
                        // In this example, it will have only one core.
                        List<Pe> peList = new ArrayList<Pe>();

                        int mips = 1000;
```

```
// 3. Create PEs and add these into a list.
peList.add(new Pe(0, new PeProvisionerSimple(mips))); // need to store Pe id and MIPS Rating

//4. Create Hosts with its id and list of PEs and add them to the list of machines
int hostId=0;
int ram = 2048; //host memory (MB)
long storage = 1000000; //host storage
int bw = 10000;

hostList.add(
        new Host(
                hostId,
                new RamProvisionerSimple(ram),
                new BwProvisionerSimple(bw),
                storage,
                peList,
                new VmSchedulerSpaceShared(peList)
        )
); // This is our first machine

//create another machine in the Data center
List<Pe> peList2 = new ArrayList<Pe>();

peList2.add(new Pe(0, new PeProvisionerSimple(mips)));

hostId++;

hostList.add(
        new Host(
                hostId,
                new RamProvisionerSimple(ram),
                new BwProvisionerSimple(bw),
                storage,
                peList2,
                new VmSchedulerSpaceShared(peList2)
        )
); // This is our second machine



// 5. Create a DatacenterCharacteristics object that stores the
//    properties of a data center: architecture, OS, list of
//    Machines, allocation policy: time- or space-shared, time zone
//    and its price (G$/Pe time unit).
String arch = "x86";     // system architecture
String os = "Linux";        // operating system
String vmm = "Xen";
double time_zone = 10.0;       // time zone this resource located
double cost = 3.0;           // the cost of using processing in this resource
double costPerMem = 0.05;                    // the cost of using memory in this resource
double costPerStorage = 0.001;      // the cost of using storage in this resource
double costPerBw = 0.0;                      // the cost of using bw in this resource
```

```
                LinkedList<Storage> storageList = new LinkedList<Storage>();  //we are not adding SAN devices
by now

        DatacenterCharacteristics characteristics = new DatacenterCharacteristics(
                arch, os, vmm, hostList, time_zone, cost, costPerMem, costPerStorage, costPerBw);

                // 6. Finally, we need to create a PowerDatacenter object.
                Datacenter datacenter = null;
                try {
                        datacenter = new Datacenter(name, characteristics, new
VmAllocationPolicySimple(hostList), storageList, 0);
                } catch (Exception e) {
                        e.printStackTrace();
                }

                return datacenter;
        }

        //We strongly encourage users to develop their own broker policies, to submit vms and cloudlets
according
        //to the specific rules of the simulated scenario
        private static DatacenterBroker createBroker(){

                DatacenterBroker broker = null;
                try {
                        broker = new DatacenterBroker("Broker");
                } catch (Exception e) {
                        e.printStackTrace();
                        return null;
                }
                return broker;
        }

        /**
         * Prints the Cloudlet objects
         * @param list  list of Cloudlets
         */
        private static void printCloudletList(List<Cloudlet> list) {
                int size = list.size();
                Cloudlet cloudlet;

                String indent = "    ";
                Log.printLine();
                Log.printLine("========== OUTPUT ==========");
                Log.printLine("Cloudlet ID" + indent + "STATUS" + indent +
                                "Data center ID" + indent + "VM ID" + indent + "Time" + indent + "Start Time" +
indent + "Finish Time");

                DecimalFormat dft = new DecimalFormat("###.##");
                for (int i = 0; i < size; i++) {
                        cloudlet = list.get(i);
                        Log.print(indent + cloudlet.getCloudletId() + indent + indent);
```

```
                        if (cloudlet.getCloudletStatus() == Cloudlet.SUCCESS){
                                Log.print("SUCCESS");

                                Log.printLine( indent + indent + cloudlet.getResourceId() + indent + indent +
indent + cloudlet.getVmId() +
                                                indent + indent + dft.format(cloudlet.getActualCPUTime()) +
indent + indent + dft.format(cloudlet.getExecStartTime())+
                                                indent + indent + dft.format(cloudlet.getFinishTime()));
                        }
                }

        }
}
```

Hhhkh

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package pkgfinal;

/**
 *
 * @author Mitul
 */
public class Selection {
    public Selection(){
    }

    public int executionTime(long fileSize,double mips){
        int executionTime;

        executionTime = (int) (fileSize / mips);

        return executionTime;
    }

    public int completionTime(int e,int w){
        int completionTime;

        completionTime = e + w;

        return completionTime;
    }

    public int completionTime (int e,int w1,int w2){
        int completionTime;

        completionTime = e + w1 + w2;

        return completionTime;
    }
```

```
    public double mean(int c1,int c2,int c3,int c4,int c5,int c6,int n){

        double mean_comp;

        mean_comp = (c1 + c2 + c3 + c4 + c5 + c6)/n;

        return mean_comp;


    }

    public double standardDeviation(int c1,int c2,int c3,int c4,int c5,int c6,double mean,int n){
        double sd;

        int e1 = (int) Math.pow((c1-mean), 2);
        int e2 = (int) Math.pow((c2-mean), 2);
        int e3 = (int) Math.pow((c3-mean), 2);
        int e4 = (int) Math.pow((c4-mean), 2);
        int e5 = (int) Math.pow((c5-mean), 2);
        int e6 = (int) Math.pow((c6-mean), 2);

            sd = Math.sqrt(e1+e2+e3+e4+e5+e6)/Math.sqrt(n);

        return sd;

    }
}
```

Hjg

```
package pkgfinal;

/**
 * Created with IntelliJ IDEA.
 * User: mitul
 * Date: 9/16/14
 * Time: 10:49 PM
 * To change this template use File | Settings | File Templates.
 */
public class Time {
    private double baseValue = 0;
    private double currentValue = 0;

    Time(double baseValue){
        this.baseValue = baseValue;
    }

    public double getBaseValue() {
        return baseValue;
    }

    public void setBaseValue(double baseValue) {
        this.baseValue = baseValue;
```

```
    }

    public double getCurrentValue() {
        return currentValue;
    }

    public void setCurrentValue(double currentValue) {
        this.currentValue = currentValue;
    }
}
```

Bibliography

[1] Cloud computing: state-of-the-art and research challenges byQi Zhang·Lu Cheng·Raouf Boutaba in IJSA 2010 vol 1: 7-18.

[2] A game-theoretic method of fair resource allocation for cloud computing services by Guiyi Wei·Athanasios V. Vasilakos·Yao Zheng Naixue Xiong.in IEEE 2012.

[3] CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms by Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, C´esar A. F. De Rose and Rajkumar Buyya

[4] Optimal Online Deterministic Algorithms and Adaptive Heuristics for Energy and Performance Efficient Dynamic Consolidation of Virtual Machines in Cloud Data Centers Anton Beloglazov and Rajkumar Buyya in online Willey Online.

[5] Research and Simulation of Task Scheduling Algorithm in Cloud Computing Hong Sun, Shi-ping Chen, Chen Jin, Kai Guo in TELKOMNIKA, Vol.11, No.11, November 2013.

[6] Survey on Resource Allocation Policy and Job Scheduling Algorithms of Cloud Computing by Lu Huang and Hai-shan Chen in JOURNAL OF SOFTWARE, VOL. 8, NO. 2, FEBRUARY 2013.

[7] A Particle Swarm Optimization-based Heuristic for Scheduling Workflow Applications in Cloud Computing Environments by Suraj Pandey, Linlin Wu, Siddeswara Mayura Guru, Rajkumar Buyya.

[8] S. Pandey, W. Voorsluys, M. Rahman, R. Buyya, J. Dobson, and K. Chiu. A grid workflow environment for brain imaging analysis on distributed systems. Concurrency and Computation: Practice & Experience, 21(16):2118–2139,November 2009.

[9] R. Buyya, S. Pandey, and C. Vecchiola. Cloudbus toolkit for market-oriented cloud computing. In CloudCom ’09: Proceedings of the 1st International Conference on Cloud Computing, volume 5931 of LNCS, pages 24–44. Springer, Germany, December 2009.

[10] Energy Efficient Resource Management in Virtualized Cloud Data Centers Anton Beloglazov* and Rajkumar Buyya in 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing.

[11] Resource Management and Scheduling in Cloud Environment Vignesh V, Sendhil Kumar KS, Jaisankar N in International Journal of Scientific and Research Publications, Volume 3, Issue 6, June 2013,ISSN 2250-3153.

[12] SLA-based Resource Allocation for Software as a Service Provider (SaaS) in Cloud Computing Environments Linlin Wu, Saurabh Kumar Garg and Rajkumar Buyya in 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing.

[13] Energy efficient utilization of resources in cloud computing systems Young Choon Lee·Albert Y. Zomaya in SPRINGER online 2010.

[14] Efficient Resource Management for Cloud Computing Environments Andrew J. Younge, Gregor von Laszewski, Lizhe Wang, Pervasive Technology Institute, Indiana University, Bloomington, IN USA and Sonia Lopez-Alarcon, Warren Carithers , Rochester Institute of Technology, Rochester, NY USA.

[15] Resource Allocation and Scheduling in the Cloud Ms. Shubhangi D. Patil, Dr. S. C. Mehrotra in International Journal of Emerging Trends & Technology in Computer Science (IJETTCS) 2012.