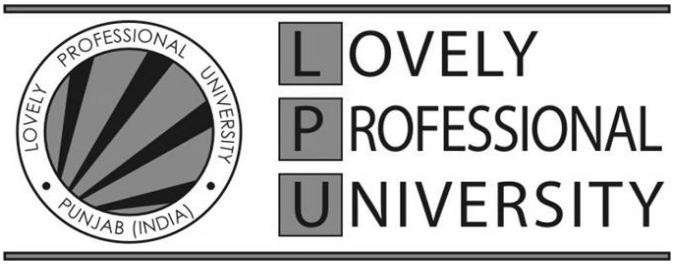

Process execution speedup under multiprocessor environment.

A Dissertation submitted

By Himat Singh
Reg No - 41100105
Section - K2112

To

Department of Computer Science and Engineering

In partial fulfillment of the requirement for the

Award of the Degree of

Master of Technology in Computer Science and Engineering

Under the guidance of

Mr. Kiran Kumar Kaki
(Assistant Professor)

May-2014

DECLARATION

I hereby declare that the dissertation entitled **— "Process Execution Speedup under Multiprocessor Environment"** submitted for the M.Tech degree is entirely my original work and all references and ideas have been duly acknowledged. It does not contain any work for the award of any other degree or diploma.

Date: 01/12/2013

Investigator: Himat Singh
Reg no: 41100105

ABSTRACT

The Cloud computing model leverages virtualization of computing resources allowing customers to provision resources on-demand on a pay-as-you-go basis. Instead of incurring high upfront costs in purchasing IT infrastructure and dealing with the maintenance and upgrades of both software and hardware, organizations can outsource their computational needs to the Cloud. Large-scale data centers containing thousands of computing nodes consuming enormous amounts of electrical energy. The virtual machine migration in the cloud computing increase the overhead which leads to the delay in the execution of processes. This dissertation focused on minimization of virtual machines migration and turn off the idle hosts in order to reduce the idle power consumption. To save energy, the highest and lowest processor utilization values are provided which can act as a threshold for the virtual machines to migrate. If utilization value of processor for the host goes below the threshold value, all virtual machines need to be migrated from current host and the host has to be turn off, for reduction of idle power consumption. If the utilization value exceed over the maximum threshold value, some virtual machines need to be migrated from the host to reduce utilization to prevent potential Service Level Agreements violation. Further for migrating least number of virtual machines this research rely on the initial selection of virtual machines to allocate the cloudlets. The initial selection of Virtual machine has great impact on the overall execution process. The research is optimizing the migration process by migrating average number of virtual machines based on the utilization of the processors. Migration of virtual machines is to be considered with lowest usage of processors and tasks are totally dependent on it. Process is helpful in minimize total potential increase of the utilization and SLA violation.

Keywords: Service Level Agreements, Quality of Service, Virtual Machine, Write once run anywhere, Java Virtual Machine

CERTIFICATE

This is to certify that **Himat Singh (41100105)** has completed M.Tech dissertation titled **Process Execution Speedup under Multiprocessor Environment** under my guidance and supervision. To the best of my knowledge, the present work is the result of his original investigation and study. No part of the dissertation has ever been submitted for any other degree or diploma.

The dissertation is fit for the submission and partial fulfilment of the conditions for the award of M.Tech. computer science and engineering.

Date:_____ Signature of Advisor:_____

Mr. Kiran Kumar Kaki

(Assistant Professor)

ACKNOWLEDGEMENT

I would like to express my deepest gratitude to Mr. Kiran Kumar Kaki (Dissertation Mentor) and Mrs. Harjeet Kaur for their valuable knowledge and expertise. It is only with their guidance that I could take up the initiative of such a good topic of thesis and complete it on time. I am also very thankful to Lovely Professional University for giving me an opportunity to propose and implement my work.

I am gratified for the successful completion of my thesis implementation. I would also like to convey thanks to all my friends who gave their full support and encouraged me for this thesis work.

Himat Singh

41100105

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

CHAPTER 1

INTRODUCTION

1.1 CLOUD COMPUTING STRUCTURE

In the last few years, we have seen the emergence of a new generation of business that operates over the Internet. The Internet has become a medium for organizations, businesses and individuals to collaborate because of technological and economic benefits. The complexity of these networks is increasing given their assets of the sub-networks that provide access to services and resources. These networks serve to strengthen businesscustomer relationships, increases profitability and customer satisfaction. Grid/Cloud computing paradigm has quickly become to realization. However, the integration of decentralized services and resources over the internet is still a challenge [1].

In early 2008 the term "cloud computing" was created. Many definitions exist in the literature about cloud computing. However, the vision of both the cloud and the Grid is the same which is to reduce the cost of computing, increase reliability, and increase flexibility by transforming computers from something that we buy and operate ourselves to something that is operated by a third party [Foster et al., 2008]. We view the "cloud" term as another marketing term hype of the Grid computing as they share the same vision, fundamental characteristics and challenges.

The Cloud systems can be classified depending on the type of usage. Similar to traditional computation model, those computation elements are the main elements in the Cloud system. However, instead of the traditional centralized node that does all the computation, the Cloud has different nodes that are distributed. The Cloud computing systems can be classified into:

- Computational: denotes a system that has a high aggregate capacity of distributed processors. It harnesses machines in "cycle-stealing" mode to have higher computational capacity than the capacity of any constituent machine in the system.
- Data: provides an infrastructure for creating information from data repositories such as data warehouses.

- Service: refers to systems that provide services that are not provided by any single local machine. An aggregate of services can compose a new service.

This thesis focuses on the Cloud systems virtual machine migration problem, minimum virtual machine migration policy is implemented.

The Cloud computing model leverages virtualization of computing resources allowing customers to provision resources on-demand on a pay-as-you-go basis [1]. Instead of incurring high upfront costs in purchasing IT infrastructure and dealing with the maintenance and upgrades of both software and hardware, organizations can outsource their computational needs to the Cloud. The proliferation of Cloud computing has resulted in the establishment of large-scale data centers containing thousands of computing nodes and consuming enormous amounts of electrical energy.

1.2 VIRTUAL MACHINES

Virtual machines are separated into two different categories based on their use and the extent up to which they correspond to physical machine. A system virtual machine provides a complete system platform which supports the execution of complete operating system.

Where as a Process virtual machine is designed to rum single program that means it supports a single process. An essential characteristic of virtual machine is that the software running inside is limited to the resources and abstractions provided by the virtual machine

Standby state of the network in virtual environment is good option when utilization of servers is less. This process for providing good concept for green computing and in this particular work green algorithm is used for this purpose. Cloud and virtualization not only accelerate the data center building but also brings the possibility of green energy.

If the workload size could allocated in different resource depend time and space, it could improve the energy efficiency and avoiding waste resources.

1.3 VIRTUAL MACHINE MIGRATION

Market-based resource allocation systems rely on consumers to set values on resources that they require. Market mechanism is to provide an allocation that is optimal. The

fundamental principle is that resources are priced based on the aggregated supply and demand. Consumers seek a quantity of resource that maximizes their utility given the current market price. Trade occurs at a clearing price that balances supply and demand as shown in Figure 1. Such allocations are economically efficient. This means no reallocation can make one better off without making another worse. Applying the economic-based framework offers an effective way to solve the issues of scheduling problems in the Grid/Cloud environment such as decentralization, autonomy, resource sharing, heterogeneity, and quality of solution.

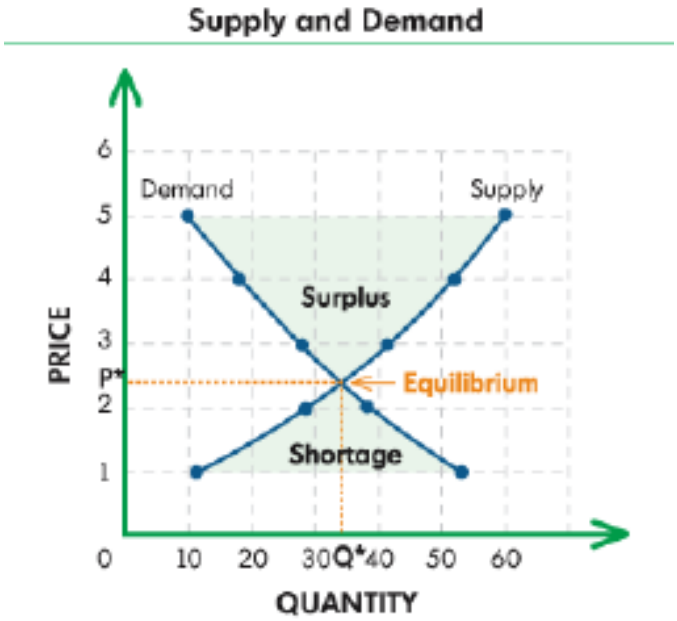

Fig. 1.1 Demonstartion of supply and demand variation in curvic form [2]

One of the ways to address the energy inefficiency is to leverage the capabilities of the virtualization technology. The virtualization technology allows Cloud providers to create multiple Virtual Machine (VMs) instances on a single physical server, thus improving the utilization of resources and increasing the Return On Investment (ROI). The reduction in energy consumption can be achieved by switching idle nodes to low-power modes (i.e., sleep, hibernation), thus eliminating the idle power consumption. Moreover, by using live migration [3] the VMs can be dynamically consolidated to the minimal number of physical nodes according to their current resource requirements. However, efficient resource management in Clouds is not trivial, as modern service applications often experience highly variable workloads causing dynamic resource usage patterns. Therefore, aggressive consolidation of VMs can lead to performance degradation when an application encounters an increasing demand resulting in an unexpected rise of the

resource usage. If the resource requirements of an application are not fulfilled, the application can face increased response times, time-outs or failures. Ensuring reliable Quality of Service (QoS) defined via Service Level Agreements (SLAs) established between Cloud providers and their customers is essential for Cloud computing environments; therefore, Cloud providers have to deal with the energy-performance trade-off – the minimization of energy consumption, while meeting the SLAs.

One of the ways to address the energy inefficiency is to leverage the capabilities of the virtualization technology. The virtualization technology allows Cloud providers to create multiple Virtual Machine (VMs) instances on a single physical server, thus improving the utilization of resources and increasing the Return On Investment (ROI). The reduction in energy consumption can be achieved by switching idle nodes to low-power modes (i.e., sleep, hibernation), thus eliminating the idle power consumption (Figure 1). Moreover, by using live migration [3] the VMs can be dynamically consolidated to the minimal number of physical nodes according to their current resource requirements. However, efficient resource management in Clouds is not trivial, as modern service applications often experience highly variable workloads causing dynamic resource usage patterns. Therefore, aggressive consolidation of VMs can lead to performance degradation when an application encounters an increasing demand resulting in an unexpected rise of the resource usage. If the resource requirements of an application are not fulfilled, the application can face increased response times, time-outs or failures. Ensuring reliable Quality of Service (QoS) defined via Service Level Agreements (SLAs) established between Cloud providers and their customers is essential for Cloud computing environments; therefore, Cloud providers have to deal with the energy-performance trade-off – the minimization of energy consumption, while meeting the SLAs.

We wil present a decentralized architecture of the resource management system for Cloud data centers and propose the development of the following policies for continuous optimization of VM placement: [4]

- Optimization over multiple system resources – at each time frame VMs are reallocated according to current CPU, RAM and network bandwidth utilization.
- Network optimization – optimization of virtual network topologies created by intercommunicating VMs. Network communication between VMs should be

observed and considered in reallocation decisions in order to reduce data transfer overhead and network devices load. [4]

- Thermal optimization – current temperature of physical nodes is considered in reallocation decisions. The aim is to avoid "hot spots" by reducing workload of the overheated nodes and thus decrease error-proneness and cooling system load.

The prediction algorithm used in related study [3], explains that it is very essential to find the early prediction process for different requirements of the virtual machines. In this research, we will be focusing in online available applications and this request need complete solution for statistics and requests. Modification of the virtual machines are never be fully possible. Various algorithm is used for providing details of the hot spots available while communication and thresholds have been proposed for finding cold spots and hot spots with threshold such as 80 percent to 90 percent for particular process.

Standby state of the network in virtual environment is good option when utilization of servers are less. This process for providing good concept for green computing and in this particular work green algorithm is used for this purpose.

1.4 REALLOCATION PROCESS IN CLOUD

Allocation of VMs can be divided in two: the first part is admission of new requests for VM provisioning and placement VMs on hosts, whereas the second part is optimization of current allocation of VMs. The first part can be considered as a bin packing problem with variable bin sizes and prices. To solve it we apply modification of the Best Fit Decreasing (BFD) algorithm. In our modification (MBFD) we sort all VMs in decreasing order of current utilization and allocate each VM to a host that provides the least increase of power consumption due to this allocation. This allows to leverage heterogeneity of the nodes by choosing the most powerefficient ones. The complexity of the allocation part of the algorithm is nm, where n is the number of VMs that have to be allocated and m is the number of hosts.

CHAPTER 2

LITERATURE REVIEW

2.1 INTRODUCTION

In this present chapter, a brief account has been put forth of the available literature that has been studied extensively.

2.2 LITERATURE SURVEY

Zhen Xiao in 2013 [3] explained that cloud computing allows business customers to scale up and down their resource usage based on needs. Many of the touted gains in the cloud model come from resource multiplexing through virtualization technology. In this paper, authors present a system that uses virtualization technology to allocate data center resources dynamically based on application demands and support green computing by optimizing the number of servers in use. We introduce the concept of "skewness" to measure the unevenness in the multidimensional resource utilization of a server. By minimizing skewness, we can combine different types of workloads nicely and improve the overall utilization of server resources. We develop a set of heuristics that prevent overload in the system effectively while saving energy used. Trace driven simulation and experiment results demonstrate that our algorithm achieves good performance.

Anton Beloglazov in 2010 [5] explains that Rapid growth of the demand for computational power has led to the creation of large-scale data centers. They consume enormous amounts of electrical power resulting in high operational costs and carbon dioxide emissions. Moreover, modern Cloud computing environments have to provide high Quality of Service (QoS) for their customers resulting in the necessity to deal with power-performance trade-off. We propose an efficient resource management policy for virtualized Cloud data centers. The objective is to continuously consolidate VMs leveraging live migration and switch off idle nodes to minimize power consumption, while providing required Quality of Service. We present evaluation results showing that dynamic reallocation of VMs brings substantial energy savings, thus justifying further development of the proposed policy.

Jyothi Sekhar in 2012 [6] In this paper, authors consider the problem of VM migration policies and elaborated that virtualization technologies which are heavily relied on by the cloud computing environments provide the ability to transfer virtual machines (VM) between the physical systems using the technique of live migration mainly for improving the energy efficiency. Dynamic server consolidation through live migration is an efficient way towards energy conservation in Cloud data centers. The main objective is to keep the number of power-on systems as low as possible and thus reduce the excessive power used to run idle servers. This technique of VM live migration is being used widely for various system-related issues like load balancing, online system maintenance, fault tolerance and resource distribution. Energy efficient VM migration becomes a main concern as the data centers are trying to reduce the power consumption. Aggressive consolidation may even lead to performance degradation and hence can result in Service Level Agreement (SLA) violation. Thus there is a trade-off between energy and performance. Various protocols, heuristics and architectures have been proposed for the energy aware server consolidation via live migration of VMs and are the main area for this survey.

Pablo Graubner in 2011 [8] explains an approach for improving the energy efficiency of infrastructure-as-a-service clouds is presented. The approach is based on performing live migrations of virtual machines to save energy. In contrast to related work, the energy costs of live migrations including their pre- and post-processing phases are taken into account, and the approach has been implemented in the Eucalyptus open-source cloud computing system by efficiently combining a multi-layered file system and distributed replication block devices. To evaluate the proposed approach, several short- and long-term tests based on virtual machine workloads produced with common operating system benchmarks, web-server emulations as well as different MapReduce applications have been conducted. The results indicate that energy savings of up to 16 percent can be achieved in a productive Eucalyptus environment.

Kyong Hoon Kim in 2010 [9] explains that reducing power consumption has been an essential requirement for Cloud resource providers not only to decrease operating costs,

but also to improve the system reliability. As Cloud computing becomes emergent for the Anything as a Service (XaaS) paradigm, modern real-time services also become available through Cloud computing. In this work, authors investigate power-aware provisioning of virtual machines for real-time services. This approach is (i) to model a real-time service as a real-time virtual machine request; and (ii) to provision virtual machines in Cloud data centers using Dynamic Voltage Frequency Scaling (DVFS) schemes. VM migration needs less power consumption for provide green computing environment and hence this research is very useful in discussed manner.

Corentin Dupont in 2012 [10] explained that data centres are powerful ICT facilities which constantly evolve in size, complexity, and power consumption. At the same time users' and operators' requirements become more and more complex. However, existing data centre frameworks do not typically take energy consumption into account as a key parameter of the data centre's configuration. To lower the power consumption while fulfilling performance requirements authors propose a flexible and energy-aware framework for the (re)allocation of virtual machines in a data centre. The framework, being independent from the data centre management system, computes and enacts the best possible placement of virtual machines based on constraints expressed through service level agreements. The framework's flexibility is achieved by decoupling the expressed constraints from the algorithms using the Constraint Programming (CP) paradigm and programming language, basing ourselves on a cluster management library called Entropy. Finally, the experimental and simulation results demonstrate the effectiveness of this approach in achieving the pursued energy optimization goals.

Christopher Clark in 2005 [11] explained that migrating operating system instances across distinct physical hosts is a useful tool for administrators of data centers and clusters: It allows a clean separation between hardware and software, and facilitates fault management, load balancing, and low-level system maintenance. By carrying out the majority of migration while OSes continue to run, we achieve impressive performance with minimal service downtimes; we demonstrate the migration of entire OS instances on a commodity cluster, recording service downtimes as low as 60ms. Work show that that our performance is sufficient to make live migration a practical tool even for servers

running interactive loads. In this paper authors consider the design options for migrating OSes running services with liveness constraints, focusing on data center and cluster environments.

Anju Mohan in 2013 [12] presented a survey for live migration of virtual machines techniques and explained that virtual machines refers to the software implementation of a computer that runs its own OS and applications as if it was a physical machine. Live migration of VMs allows a server administrator to move a running virtual machine or application among different physical machines without disconnecting the client or application. Total migration time and downtime are two key performance metrics that the clients of a VM service care about the most, because they are concerned about service degradation and the duration that the service is completely unavailable. When a VM is migrating, it is important that this transfer occurs in a manner that balances the requirements of minimizing both the downtime and the total migration time.

Pradip D. Patel in 2014 [13] explained that cloud computing is a service where storage and computing resources accessed on subscription basis. Cloud computing is powered by virtualization technology. Live migration is the process of moving a running virtual machine or an application between different physical machines without disconnecting the client , memory, network connectivity and storage of the virtual machine are transferred from the original host machine to the destination. This capability is being increasingly utilized in today's enterprise environments to provide efficient online system maintenance, reconfiguration, load balancing and fault tolerance. This paper presents a detailed survey on Live Migration of Virtual machines in cloud computing.

M.Tarighi in 2010 [14] In this paper, authors show that performance of the virtualized cluster servers could be improved through intelligent decision over migration time of Virtual Machines across heterogeneous physical nodes of a cluster server. The cluster serves a variety range of services from Web Service to File Service. Some of them are CPU-Intensive while others are RAM-Intensive and so on. Virtualization has many advantages such as less hardware cost, cooling cost, more manageability.

Sheng Di in 2013 [15] explains that With virtual machine (VM) technology being increasingly mature, compute resources in cloud systems can be partitioned in fine granularity and allocated on demand. Author make three contributions in this paper: 1) It formulate a deadline-driven resource allocation problem based on the cloud environment facilitated with VM resource isolation technology, and also propose a novel solution with polynomial time, which could minimize users' payment in terms of their expected deadlines. 2) By analyzing the upper bound of task execution length based on the possibly inaccurate workload prediction, we further propose an error-tolerant method to guarantee task's completion within its deadline. 3) It validate its effectiveness over a real VM-facilitated cluster environment under different levels of competition. In our experiment, by tuning algorithmic input deadline based on our derived bound, task execution length can always be limited within its deadline in the sufficient-supply situation; the mean execution length still keeps 70 percent as high as user specified deadline under the severe competition. Under the original-deadline-based solution, about 52.5 percent of tasks are completed within 0.95-1.0 as high as their deadlines, which still conform to the deadline-guaranteed requirement. Only 20 percent of tasks violate deadlines, yet most (17.5 percent) are still finished within 1.05 times of deadlines.

Olivier Beaumont in 2013 [16] In this paper, authors consider the problem of assigning a set of clients with demands to a set of servers with capacities and degree constraints. The goal is to find an allocation such that the number of clients assigned to a server is smaller than the server's degree and their overall demand is smaller than the server's capacity, while maximizing the overall throughput. This problem has several natural applications in the context of independent tasks scheduling or virtual machines allocation. We consider both the offline (when clients are known beforehand) and the online (when clients can join and leave the system at any time) versions of the problem. It first show that the degree constraint on the maximal number of clients that a server can handle is realistic in many contexts. Then, our main contribution is to prove that even if it makes the allocation problem more difficult (NP-Complete), a very small additive resource augmentation on the servers degree is enough to find in polynomial time a solution that achieves at least the optimal throughput. After a set of theoretical results on the complexity of the offline and online versions of the problem, we propose several other greedy heuristics to solve

the online problem and we compare the performance (in terms of throughput) and the cost (in terms of disconnections and reconnections) of all proposed algorithms through a set of extensive simulation results.

Rongxing Lu in 2013 [17] explains that With the pervasiveness of smart phones and the advance of wireless body sensor networks (BSNs), mobile Healthcare (m-Healthcare), which extends the operation of Healthcare provider into a pervasive environment for better health monitoring, has attracted considerable interest recently. However, the flourish of m-Healthcare still faces many challenges including information security and privacy preservation. In this paper, we propose a secure and privacy-preserving opportunistic computing framework, called SPOC, for m-Healthcare emergency. With SPOC, smart phone resources including computing power and energy can be opportunistically gathered to process the computing-intensive personal health information (PHI) during m-Healthcare emergency with minimal privacy disclosure. In specific, to leverage the PHI privacy disclosure and the high reliability of PHI process and transmission in m-Healthcare emergency, we introduce an efficient user-centric privacy access control in SPOC framework, which is based on an attribute-based access control and a new privacy-preserving scalar product computation (PPSPC) technique, and allows a medical user to decide who can participate in the opportunistic computing to assist in processing his overwhelming PHI data. Detailed security analysis shows that the proposed SPOC framework can efficiently achieve user-centric privacy access control in m-Healthcare emergency. In addition, performance evaluations via extensive simulations demonstrate the SPOC's effectiveness in term of providing high-reliable-PHI process and transmission while minimizing the privacy disclosure during m-Healthcare emergency.

Anton Beloglazov in 2012 [17] explained that dynamic consolidation of Virtual Machines (VMs) is an effective way to improve the utilization of resources and energy efficiency in Cloud data centers. Determining when it is best to reallocate VMs from an overloaded host is an aspect of dynamic VM consolidation that directly influences the resource utilization and Quality of Service (QoS) delivered by the system. The influence on the QoS is explained by the fact that server overloads cause resource shortages and performance degradation of applications. Current solutions to the problem of host overload detection are generally heuristic-based, or rely on statistical analysis of historical

data. The limitations of these approaches are that they lead to sub-optimal results and do not allow explicit specification of a QoS goal. Authors propose a novel approach that for any known stationary workload and a given state configuration optimally solves the problem of host overload detection by maximizing the mean inter-migration time under the specified QoS goal based on a Markov chain model. Authors heuristically adapt the algorithm to handle unknown non-stationary workloads using the Multisize Sliding Window workload estimation technique. Through simulations with real-world workload traces from more than a thousand PlanetLab VMs, we show that our approach outperforms the best benchmark algorithm and provides approximately 88% of the performance of the optimal offline algorithm.

2.2 INFERENCES DRAWN OUT OF LITERATURE REVIEW

In the recent past, most of the work is done to migration of virtual machines based on fixed thresholds and these processes make it more static in nature. Some of the interference drawn from literature survey is given below.

- After reviewing many papers we found that virtualization is the solution of expanding industrial needs and cloud computing the answer for many issues like scalability and availability to fulfil.
- To fulfill the industrial and commeicial needs, huge number of resources are required and virtual machines migration process become more reliable to ful fil this process.

CHAPTER 3

PROPOSED WORK AND METHODOLOGY

3.1 PROBLEM DEFINITION

In related study, a scheme for selecting energy efficient allocation of virtual machines in cloud data center. Proposed scheme consider the maximum and minimum utilization threshold value. If the utilization of CPU for a host falls below the minimum threshold, all VMs have to be migrated from this host and the host has to be switched off in order to eliminate idle power consumption. If the utilization goes over the maximum threshold, some VMs have to be migrated from the host to reduce utilization to prevent potential Service Level Agreements violation. Further for migrating, it uses minimization of migrations to reduce migration overhead. Our research is followed similar line of implementation by using a Task scheduling algorithm that scheduled the task to the virtual machines according to CPU power. According to our concept if tasks is scheduled to the virtual machines earlier, load is managed in a better way and this resulted in less number of virtual machine migration. Further migration of virtual machines is considered with lowest usage of processor and tasks are totally dependent on it. This process is helpful to minimize total potential increase of the utilization and SLA violation. For validation of our proposed work, we have simulated Non Power Aware policy, Comparison is done with these two schemes.

3.2OBJECTIVES

In our study for various literature studies, we found that virtual machine migration is widely used and applied to current cloud infrastructure and needs healthy amount of resource usage accuracy duration the processing migration in various applications based on scheduling assigned to data center. The focused objective which can be considered for our experimentation is as follows.

To find optimal virtual machine resource management policy that provide reduction of the migration of virtual machines and which can provide virtual machine migration based on low utilization of processors. Simulation migrate the VM to other hosts that has lowest

CPU load or that has highest CPU load. The concept is to manage the energy consumption of the cloud by transferring VM to the other hosts and switching off the ideal hosts. Moreover by reducing the VM migration the speeup can be achieved.

we will target our objectives given below:

- To find the better virtual machine resource management policy that will reduce the migration of virtual machines.
- Find the solution for better scheme based on virtual machine migration based on low utilization of CPU

3.3 METHODOLOGY

This research starts with study of parallel computing management in virtual cloud environment based on cloud computing for virtualization in following steps.

1st Phase: This is the initial stage for the whole process and it contains the basic functionality along with the collection of information (virtual simulation, basic virtualization functions etc).

2nd Phase: This stage implements the basic scenario for parallel processing structure based on resources allocation scheme.

3rd Phase: This stage creates the migration instructions for the virtual machines.

4th Phase: This stage consists of the experimentation in which the resource utilization of the complete process is computed based on the threshold value of the utilization of the multiprocessor environment in parallel communication. The basic function of the selection of the threshold is done by the selection engine which is deciding the minimum threshold according to the requirement of the system. Maximum threshold value is fixed to 90% CPU Utilization. The host needs to switch off in case of migration to save the power consumption of the server which in turns provides less heating effect. In case the value crosses the upper bound then engine need to migrate the partial processes to other host server. Further for better execution time, migration of the virtual machine is on lesser side which in turns decreases the overall overhead.

5th Phase: Final stage is the comparison of the proposed work with already existing work.

CHAPTER 4

4.1 Dissertation Monthly Progress

GANTT CHART

The overall work is divided according to various tasks involved in research as well as the available time limits.

Task	Start	End	Duration	2014				
				Jan	Feb	Mar	Apr	May
1. Identification of initial load distribution policies	07/01/14	02/02/14	25					
2. Identification of migration techniques	26/01/14	18/02/14	22					
3. Design layout for migration	17/02/14	25/03/14	35					
4. Implementation using cloudsim	24/03/14	15/04/14	20					
5. Comparison	15/04/14	05/05/14	20					

Table 4.1 Gantt Chart

CHAPTER 5

SIMULATION RESULTS AND DISCUSSIONS

5.1 Creating Virtual Machines

In this work scheduling of the jobs on the multiple resources in a gird-system is considered. We have used java platform for the development and implementation of the system. We have use cloudsim to simulate the cloud environment which consist of 100 physical nodes which are heterogenous in nature with 2000 MIPS and 6 GB of RAM with limited storage. Simulation is configured to show our environment. To create our environment we have created 5 resources with different configuration.

```
public static List<Vm> createVmList(int brokerId, int vmsNumber) {
    List<Vm> vms = new ArrayList<Vm>();
    for (int i = 0; i < 10; i++) {
        int vmType = i / (int) Math.ceil((double) 10 / Constants.VM_TYPES);
        vms.add(new PowerVm(
                i,
                brokerId,
                Constants.VM_MIPS[vmType],
                Constants.VM_PES[vmType],
                Constants.VM_RAM[vmType],
                Constants.VM_BW,
                Constants.VM_SIZE,
                1,
                "Xen",
                new CloudletSchedulerDynamicWorkload(Constants.VM_MIPS[vmType], Constants.VM_PES[vmT
                Constants.SCHEDULING_INTERVAL));
    }
    return vms;
}

public static List<PowerHost> createHostList(int hostsNumber) {
    List<PowerHost> hostList = new ArrayList<PowerHost>();
    for (int i = 0; i < 5; i++) {
        int hostType = i % Constants.HOST_TYPES;

        List<Pe> peList = new ArrayList<Pe>();
        for (int j = 0; j < Constants.HOST_PES[hostType]; j++) {
            peList.add(new Pe(j, new PeProvisionerSimple(Constants.HOST_MIPS[hostType])));
```

```
Datacenter is starting...
Entities started.
0.0: Broker: Cloud Resource List received with 1 resource(s)
0.0: Broker: Trying to Create VM #0 in Datacenter
0.0: Broker: Trying to Create VM #1 in Datacenter
0.0: Broker: Trying to Create VM #2 in Datacenter
0.0: Broker: Trying to Create VM #3 in Datacenter
0.0: Broker: Trying to Create VM #4 in Datacenter
0.0: Broker: Trying to Create VM #5 in Datacenter
0.0: Broker: Trying to Create VM #6 in Datacenter
0.0: Broker: Trying to Create VM #7 in Datacenter
0.0: Broker: Trying to Create VM #8 in Datacenter
0.0: Broker: Trying to Create VM #9 in Datacenter
0.00: VM #0 has been allocated to the host #0
0.00: VM #1 has been allocated to the host #2
0.00: VM #2 has been allocated to the host #4
0.00: VM #3 has been allocated to the host #0
0.00: VM #4 has been allocated to the host #2
0.00: VM #5 has been allocated to the host #4
0.00: VM #6 has been allocated to the host #1
0.00: VM #7 has been allocated to the host #3
0.00: VM #8 has been allocated to the host #1
0.00: VM #9 has been allocated to the host #3
```

Fig. 5.1 Virtual Machine creation and allocation process

Each resource had 1 cpu. In cloudsim environment cpu power is measured by mips(million instructions per second). Mips is a measure that how many instruction cpu can process in one second. Higher the mips, higher will be the cpu power.

To simulate tasks we have created 11 cloudlets in the simulation. Each task had different length. Cloudlet length is the measure of the size of task, length means number of instructions in the task.

```
Cloudlet cloudlet = new Cloudlet(id, 4000000, pesNumber, fileSize, outputSize, utilizationModel, u
cloudlet.setUserId(brokerId);
cloudletList.add(cloudlet);

Cloudlet cloudlet1 = new Cloudlet(id+1, 6000000, pesNumber, fileSize, outputSize, utilizationModel
cloudlet1.setUserId(brokerId);
cloudletList.add(cloudlet1);

Cloudlet cloudlet2 = new Cloudlet(id+2, 200000, pesNumber, fileSize, outputSize, utilizationModel,
cloudlet2.setUserId(brokerId);
cloudletList.add(cloudlet2);

Cloudlet cloudlet3 = new Cloudlet(id+3, 1100000, pesNumber, fileSize, outputSize, utilizationModel
cloudlet3.setUserId(brokerId);
cloudletList.add(cloudlet3);
Cloudlet cloudlet4 = new Cloudlet(id+4, 150000, pesNumber, fileSize, outputSize, utilizationModel,
cloudlet4.setUserId(brokerId);
cloudletList.add(cloudlet4);

Cloudlet cloudlet5 = new Cloudlet(id+5, 700000, pesNumber, fileSize, outputSize, utilizationModel,
cloudlet5.setUserId(brokerId);
cloudletList.add(cloudlet5);

Cloudlet cloudlet6 = new Cloudlet(id+6,2000000, pesNumber, fileSize, outputSize, utilizationModel,
cloudlet6.setUserId(brokerId);
cloudletList.add(cloudlet6);
```

Fig. 5.2 Task creation process

5.2 Distributing cloudlet to virtual machines at initial stage

The basic scheduling approach we follow is a kind of greedy alogithm. In this algoritm we just sort the list of available vm list and cloudlet list. The vm list is sorted on the basis of mips and for the cloudletsthe basis is length. After sorting process the new list are created. Then the actual process of allocation takes place in a serial fashion. In below Figure, basic scheduling is shown.

```
0.00: VM #0 has been allocated to the host #0
0.00: VM #1 has been allocated to the host #2
0.00: VM #2 has been allocated to the host #4
0.00: VM #3 has been allocated to the host #0
0.00: VM #4 has been allocated to the host #2
0.00: VM #5 has been allocated to the host #4
0.00: VM #6 has been allocated to the host #1
0.00: VM #7 has been allocated to the host #3
0.00: VM #8 has been allocated to the host #1
0.00: VM #9 has been allocated to the host #3
0.1: Broker: VM #0 has been created in Datacenter #3, Host #0
0.1: Broker: VM #1 has been created in Datacenter #3, Host #2
0.1: Broker: VM #2 has been created in Datacenter #3, Host #4
0.1: Broker: VM #3 has been created in Datacenter #3, Host #0
0.1: Broker: VM #4 has been created in Datacenter #3, Host #2
0.1: Broker: VM #5 has been created in Datacenter #3, Host #4
0.1: Broker: VM #6 has been created in Datacenter #3, Host #1
0.1: Broker: VM #7 has been created in Datacenter #3, Host #3
0.1: Broker: VM #8 has been created in Datacenter #3, Host #1
0.1: Broker: VM #9 has been created in Datacenter #3, Host #3
0.1: Broker: Sending cloudlet 0 to VM #9
0.1: Broker: Sending cloudlet 1 to VM #9
0.1: Broker: Sending cloudlet 2 to VM #9
0.1: Broker: Sending cloudlet 3 to VM #6
0.1: Broker: Sending cloudlet 4 to VM #6
0.1: Broker: Sending cloudlet 5 to VM #6
0.1: Broker: Sending cloudlet 6 to VM #7
```

Fig. 5.3 Basic implementation of the CPU Scheduling

In below Figure, basic process of virtual machine scheduling is explained. CPU scheduling time and Cloudlet which are used is explained.

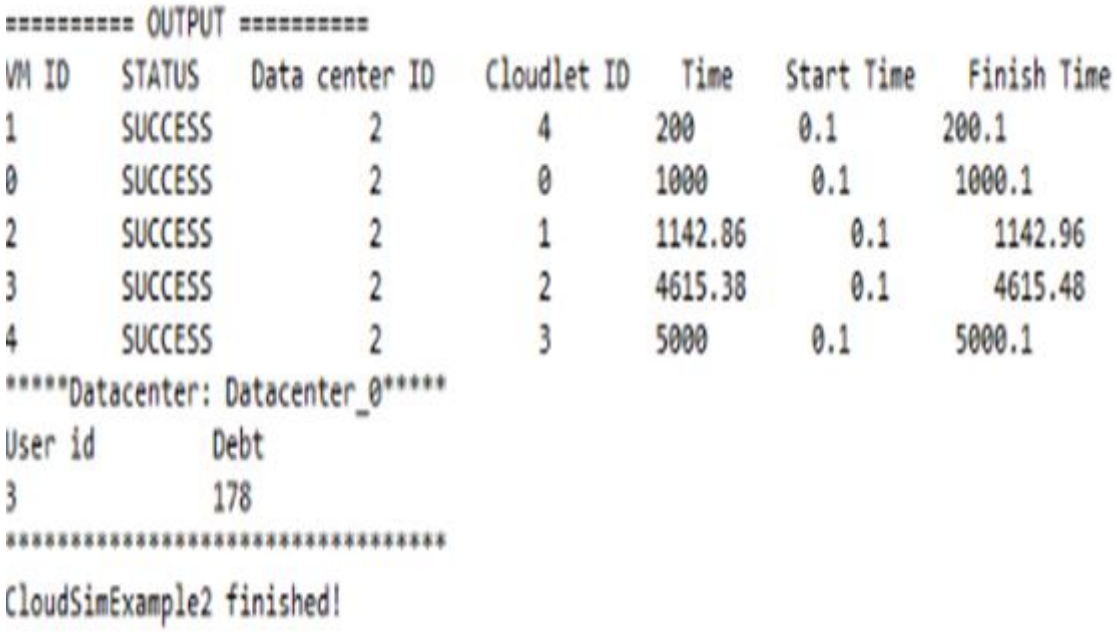

Fig. 5.4 VM Scheduling process

VmAlgo

VmAglo(List<Vm> vmlist, List<Cloudlet> clist)

1. If vmaxcount of vmlist =0 //NO vm is available

Show wait

2. If cmaxcount of cist =0 //NO cloudlet is available

Show wait

3. Repeat steps 4 and 5 for i=1 to vmaxcount-1

4. Repeat step 5 for j= 1 to vmaxcount-i

5. if vmlist<j>.getmips()>vmlist<j+1>

then temp= vmlist<j+1>, vmlist<j+1>=vmlist<j>, vmlist<j>=temp

6. Repeat steps 7 and 8 for i=1 to cmaxcount-1

7. Repeat step 8 for j= 1 to cmaxcount-i

8. if clist<j>.getcloudletLength() > clist<j+1>.getcloudletLength()

then temp= clist<j+1>, clist<j+1>=clist<j>, clist<j>=temp

9. Assign cloudlets to vm by selecting each from the list

```
public class VmAlgo1 {
    public void startAlgo(List<Vm> vmlist,List<Cloudlet> clist){

        Collections.sort(clist,new CompareCloudlet());
         Collections.sort(vmlist,new CompareVm());
         int i=0;
         int j=0;

         for(Cloudlet c: clist){
             if(i==3){
                 i=0;
                 j++;
             }
             c.setVmId(vmlist.get(j).getId());
             i++;
         }
    }
    private class CompareCloudlet implements Comparator<Cloudlet>{

        @Override
        public int compare(Cloudlet c1, Cloudlet c2) {
            int value=0;
            if(c1.getCloudletLength()<c2.getCloudletLength()){
                value=-1;
            }
            else if(c1.getCloudletLength()==c2.getCloudletLength()){
                value=0;
            }
            else if(c1.getCloudletLength()>c2.getCloudletLength()){
                value=1;
            }
            return value;
        }

    }
```

Fig.5.5 VmAlgo

First simulation has to be initialized, than all the resources and tasks is created by the simulation. After that tasks are assigned according to scheduling policy, after assigning the task to the resources, simulation start processing. In processing mode simulation actually perform all the processing and calculate results. After processing simulation stops and results are produced to the user.

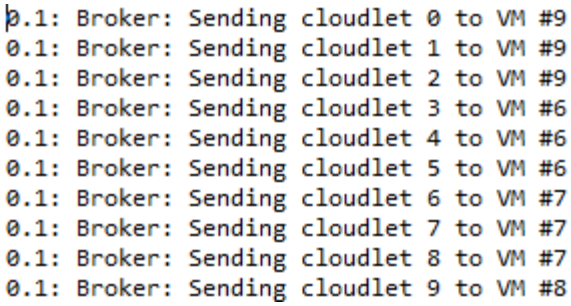

Fig. 5.6 Cloudlets Distribution

The above figure shows the process assignment of cloudlets to vms. Once the VMs and cloudlets are created, they are arranged as per the initial greedy algorithm. The cloudlets are arranged in the ascending order of their lengths. The length of individual cloudlet is find out with the help of getlength() function. The Vms are also arranged in ascending order of their mips(million instruction per second). Mips is the mesure to rate the virtual machine. After arranging the cloudlets and Vm list the assignment process is initiated. The cloudlets are assigned in a serial order.

```
// create cloudlets.
List<Cloudlet> cloudletList= PlanetLabHelper.createCloudletLi

// create Vms.
List<Vm> vmList= Helper.createVmList(brokerId, cloudletList.s

// create Hosts.
List<PowerHost> hostList= Helper.createHostList(PlanetLabCons

// create Datacenter.
PowerDatacenter datacenter= (PowerDatacenter) Helper.createDc
        new PowerVmAllocationPolicyMigrationStaticThreshold(h

// enable migration.
datacenter.setDisableMigrations(false);

// submit list of cloudlets and Vms to the broker.
broker.submitVmList(vmList);
broker.submitCloudletList(cloudletList);

CloudSim.terminateSimulation(Constants.SIMULATION_LIMIT);
double lastClock = CloudSim.startSimulation();

List<Cloudlet> newList = broker.getCloudletReceivedList();
Log.printLine("Received " + newList.size() + " cloudlets");

Helper.printResults(
        datacenter,
        vmList,
        lastClock,
        experimentName,
        Constants.OUTPUT_CSV,
        outputFolder);


CloudSim.stopSimulation();
```

Fig. 5.7 Simulation Start and Stop Process

Cloudsim is the simulation that is developed in java and it is used to implement our proposed work. First simulation is initialized that includes all the configuration, after simulation is ready to run it is started that is the phase where simulation performs all the executions. After that simulation is stopped.

5.2 Result Discussions

Results obtained after running simulation for different scheduling algorithm are discussed and compared here in the following sections.

5.2.1 Result Of Running Simulation For Fcfs

FCFS is also a well-known cpu scheduling technique. This technique is very simple and according to it tasks should be given priority on the basis of their arrival that means task that came first should be selected first. But our results shows this criteria of selecting results does not assure to give good results every time. As we are selection resources on the basis of their arrival order in this case we have executed simulation 10 time to simulate different arrival order for the task. Sometime we got very good results and sometime our result was poor. So this technique of task selection can-not be considered a much reliable technique.

```
========== OUTPUT ==========
Cloudlet ID    STATUS    Data center ID    VM ID    Time    Start Time    Finish Time
    4        SUCCESS        2             0        150.1       86.72        236.82
    2        SUCCESS        2             3        150.1       45.53        195.63
    8        SUCCESS        2             5        200.1       26.28        226.38
    7        SUCCESS        2             2        333.43       55.99        389.42
    5        SUCCESS        2             3        400.1       55.6        455.7
    3        SUCCESS        2             4        440.1       40.27        480.37
    10        SUCCESS        2             1        450.1       75.52        525.62
    9        SUCCESS        2             2        600.1       82.16        682.26
    6        SUCCESS        2             3        725.1       64.34        789.44
    0        SUCCESS        2             4        1020.1       89.63        1109.73
    1        SUCCESS        2             5        1100.1       56.02        1156.12
CloudSimExample1 finished!
```

Fig. 5.8 Results for First Come First Serve

This is a snapshot from running the simulation one time for FCFS. Here we can see we are getting an overall make-span of 1156.12 ms. That is better than SJF, but still poor as compare to our proposed priority based technique.

We have mentioned above that in FCFS jobs arrival order decide how job will be picked and this may affect the schedule and may give different results for different run of simulation. Therefore we cannot consider it a very reliable technique.

```
========== OUTPUT ==========
Cloudlet ID    STATUS    Data center ID    VM ID    Time    Start Time    Finish Time
    2         SUCCESS         2             2         133.43         45.86         179.29
    8         SUCCESS         2             5         200.1         12.89         212.99
    6         SUCCESS         2             4         400.1         27.34         427.44
    4         SUCCESS         2             0         450.1         74.27         524.37
    9         SUCCESS         2             2         500.1         73.22         573.32
    5         SUCCESS         2             1         700.1         109.38         809.48
    3         SUCCESS         2             1         900.1         73.42         973.52
    0         SUCCESS         2             3         1000.1         30.78         1030.88
    1         SUCCESS         2             5         1100.1         102.77         1202.87
    7         SUCCESS         2             0         1150.1         94.14         1244.24
    10         SUCCESS         2             0         1550.1         73.48         1623.57
CloudSimExample1 finished!
```

Fig. 5.9 Results for First Come First Serve with Second Iteration

To prove our point here is another snapshot, these are the results that we have got on the second run of simulation. Here we can see that make-span is 1623.57 ms which is much higher than both SJF and our priority based technique.

5.2.2 Results Of Simulation For Greedy Algorithm

Greedy is an old resource scheduling technique which is based on the idea that assign any resource to the task that may perform our work. Greedy do not consider any factor before assigning the task to the resource, the only thing is that if resource execute the task then assign it to the resource.

```
========== OUTPUT ==========
Cloudlet ID    STATUS    Data center ID    VM ID    Time    Start Time    Finish Time
    5         SUCCESS         2             0         140         0.1         140.1
    4         SUCCESS         2             1         150         0.1         150.1
    2         SUCCESS         2             1         175         0.1         175.1
    9         SUCCESS         2             0         200         0.1         200.1
    8         SUCCESS         2             2         200         0.1         200.1
    7         SUCCESS         2             3         250         0.1         250.1
    10         SUCCESS         2             4         360         0.1         360.1
    3         SUCCESS         2             3         400         0.1         400.1
    6         SUCCESS         2             4         580         0.1         580.1
    0         SUCCESS         2             5         1333.33         0.1         1333.43
    1         SUCCESS         2             5         1666.67         0.1         1666.77
CloudSimExample1 finished!
```

Fig. 5.10 Results for Greedy Algorithm

Greedy is also an unreliable technique which may produce some good results sometime. But most of the time results are not good as greedy pick resources randomly without using a proper criteria. Here we can see that overall make-span on running simulation for greedy give 1666.77 ms.

Comparison of energy efficient cloud with other clouds. In our program we are using a task scheduling algorithm in the beginning for better load management. The concept is that if the resources will be managed in the beginning that means load will be evenly distributed over the Vms. Then number of over utilized and under-utilized hosts will be less. This will result in less number of Vms migration from one host to another host. Moreover we are using Minimum CPU utilization selection policy that means when a VM has to be selected for migration simulation will select the VM which has minimum utilization. This approach will further help in reducing the number of VM migration. Because we are transferring some load from over-utilized host to another host this will bring the utilization of that host below threshold limit and as we are moving minimum utilization Vm to another host, therefore other host will also not get over-utilized by migrated VM.

As a proof of our concept we are showing simulation result snapshots and a comparison table that is showing our results are much better than base paper based power aware cloud, and non power aware cloud.

Snapshots and comparison tables are below.

5.2.3 ENERGY EFFICIENT CLOUD.

```
Experiment name: EnergyEfficientCloud
Number of hosts: 5
Number of VMs: 10
Total simulation time: 1440.00 sec
Energy consumption: 0.03 kWh
Number of VM migrations: 2
SLA: 0.00200%
SLA perf degradation due to migration: 0.07%
SLA time per active host: 3.00%
Overall SLA violation: 0.07%
Average SLA violation: 10.00%
Number of host shutdowns: 4
Mean time before a host shutdown: 307.06 sec
StDev time before a host shutdown: 13.92 sec
Mean time before a VM migration: 18.82 sec
StDev time before a VM migration: 12.75 sec
Execution time - VM selection mean: 0.00000 sec
Execution time - VM selection stDev: 0.00000 sec
Execution time - host selection mean: 0.00000 sec
Execution time - host selection stDev: 0.00000 sec
Execution time - VM reallocation mean: 0.00000 sec
Execution time - VM reallocation stDev: 0.00000 sec
Execution time - total mean: 0.00075 sec
Execution time - total stDev: 0.00096 sec
```

Fig. 5.11 Results of Energy efficienct cloud

5.2.4 POWER AWARE CLOUD

```
Experiment name: PowerAwareCloud
Number of hosts: 5
Number of VMs: 10
Total simulation time: 1440.00 sec
Energy consumption: 0.05 kWh
Number of VM migrations: 13
SLA: 0.01127%
SLA perf degradation due to migration: 0.16%
SLA time per active host: 7.07%
Overall SLA violation: 0.16%
Average SLA violation: 10.00%
Number of host shutdowns: 4
Mean time before a host shutdown: 552.94 sec
StDev time before a host shutdown: 287.23 sec
Mean time before a VM migration: 24.31 sec
StDev time before a VM migration: 6.78 sec
Execution time - VM selection mean: 0.00000 sec
Execution time - VM selection stDev: 0.00000 sec
Execution time - host selection mean: 0.00000 sec
Execution time - host selection stDev: 0.00000 sec
Execution time - VM reallocation mean: 0.00250 sec
Execution time - VM reallocation stDev: 0.00500 sec
Execution time - total mean: 0.00325 sec
Execution time - total stDev: 0.00457 sec
```

Fig. 5.9 Results Power Aware Cloud

5.2.5 NON POWER AWARE CLOUD

```
Experiment name: NonPowerAwareCloud
Number of hosts: 5
Number of VMs: 10
Total simulation time: 1440.00 sec
Energy consumption: 0.17 kWh
Number of VM migrations: 13
SLA: 0.26180%
SLA perf degradation due to migration: 0.85%
SLA time per active host: 30.68%
Overall SLA violation: 0.85%
Average SLA violation: 10.00%
Number of host shutdowns: 4
Mean time before a host shutdown: 674.30 sec
StDev time before a host shutdown: 287.23 sec
Mean time before a VM migration: 131.56 sec
StDev time before a VM migration: 33.88 sec
Execution time - VM selection mean: 0.00000 sec
Execution time - VM selection stDev: 0.00000 sec
Execution time - host selection mean: 0.00000 sec
Execution time - host selection stDev: 0.00000 sec
Execution time - VM reallocation mean: 0.00000 sec
Execution time - VM reallocation stDev: 0.00000 sec
Execution time - total mean: 0.00000 sec
Execution time - total stDev: 0.00000 sec
```

Fig. 5.10 Results of Non Power Aware Cloud

5.2.6 COMPARISON OF RESULTS.

Parametrs	Efficient Cloud	PowerAware Cloud	NonPowerAware Cloud
Number of Hosts	5	5	5
Total Simulation Time (in sec)	10	10	10
Energy Consumption (in kwh)	1440	1440	1440
Number of Vm Migrations	2	13	13
SLA (in percentage)	0.002	0.01127	0.2618
SLA degradation due to migration (in percentage)	0.07	0.16	0.85
SLA Time per active Host	3	7.07	30.68
Overall SLA violation (in percentage)	0.07	0.16	0.85
Number of Hosts shutdown(in percentage)	4	4	4

Table. 5.1 Comparisons of results

5.3 SUMMARY

The overall work is to find a scheduling technique that may schedule virtual machine migration over multiple resources by picking the best allocation scheme for the job in such a way that overall performance of executing the jobs will be higher and less number of migration are done. This work is useful in the cloud computing environments or similar environments where we have to choose a resource from a number of available resources. Proposed work assign threshold values to the jobs on a hybrid technique that assign resources initially and process the migration of jobs in term of machines according to the requirement of the cloud structure accuracy and robustness. Our approach uses a combination of small and big task to run on the machines. This approach distribute load on the machines in better way. After assigning tasks are scheduled to the resources by choosing best performing task for the resource. And in the proposed technique each resource maintain a list that store the information for all the tasks assigned to the resource along with number of virtual machines which are migrated.

CHAPTER 6

CONCLUSIONS AND FUTURE SCOPE

6.1. CONCLUSION

In this research finding a best allocation scheme for decreasing number of virtual machine migrations is considered by applying efficient resources judgement method and finding the threshold boundries for assigning the resources and migration of virtual machines for better performance of cloud compuitng structure.

Our research proposed a solution that will provide a less number of virtual machine migration under suitable applications. Proposed scheme consider the maximum and minimum utilization threshold value. If the utilization of CPU for a host falls below the minimum threshold, all VMs have to be migrated from this host and the host has to be switched off in order to eliminate idle power consumption. If the utilization goes over the maximum threshold, some VMs have to be migrated from the host to reduce utilization to prevent potential Service Level Agreements violation.

In our proposed work we have worked on two important problems that is job selection and resource scheduling along with virtual machine migration slection policy. Our research proves that selection of jobs also effects the overall performance of cloud network. For validation of our proposed work, we have used power aware cloud and non power aware clouds so that we can show the difference.

So in our proposed work has proved that virtual machine migration selection affect the overall performance of cloud and better execution of the jobs. Picking resources on the basis of defined thresholds while considering the better resource allocation could be a good solution.

In nutshell the proposed work provide a good scheduling technique for virtual machine migration in cloud computing structure.

6.2 FUTURE SCOPE

In future, this research can be enhanced by implementing on the real cloud systems on live environment. This proposed work is implemented on a simulation platform that was developed and implemented in java. Running this work on the real environment and comparing them with simulation results may help us to further enhance this proposed work.

Further research can be done to implement this work on the grid environment. Cloud and grid environment has lots of similarities and with little modification proposed work can be implemented to the grid compuitng. It will be interesting to run this proposed work for grids and analyzing its performance for grid that is technology in demand now a days for research purposes.

CHAPTER7

REFERENCES

[1] Czajkowski K., Fitzgerald S., Foster I., and Kesselman C, "Grid Information Services for Distributed Resource Sharing", in Proceeding the 10th IEEE International Symposium on High- Performance Distributed Computing (HPDC-10), pp. 181-194, San Francisco, California, August 2001, USA.

[2] Economic blog, http://enthusiasm.cozy.org/.

[3] Zhen Xiao, ''Dynamic Resource Allocation Using Virtual Machines for Cloud Computing Environment'', IEEE Transactions on Parallel and Distributed Systems, Vol. 24, No. 6, June 2013.

[4] Siegele, "Let It Rise Around: A Special Report on Corporate IT," Magazine of Economist Time, Vol.389, pp.13-16, October, 2008.

[5] Anton Beloglazov and Rajkumar Buyya, "Energy Efficient Allocation of Virtual Machines in Cloud Data Centers", IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, pp.129-134, June 2010.

[6] Jyothi Sekhar, Getzi Jeba, S. Durga, "A Survey on Energy Efficient Server Consolidation Through VM Live Migration," International Journal of Advances in Engineering & Technology, pp. 342- 362, Nov. 2012.

[7] Amzon Process, "Amazon elastic compute cloud " http://aws.amazon.com/ec2/, last accessed on Decmebr 6, 2013.

[8] Pablo Graubner, Matthias Schmidt, Bernd Freisleben, "Energy-efficient Management of Virtual Machines in Eucalyptus", IEEE Conference on Parallel Computing, June 2011.

[9] Kyong Hoon Kim, Anton Beloglazov, "Power-Aware Provisioning of Virtual Machines for Real-Time Cloud Services", CLOUDS Lab, The University of Melbourne, Parkville, Victoria 3010, Australia, John Wiley & Sons, Ltd., 30 December 2010.

[10] Corentin Dupont," An Energy Aware Framework for Virtual Machine Placement in Cloud Federated Data Centres", ACM Publications, Madrid, Spain, May 9-11 2012.

[11] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, "Live Migration of Virtual Machines," NSDI, 2nd Symposium on Networked Systems Design & Implementation, USENIX Association, pp. 22- 31, August 2005.

[12] Anju Mohan, Shine S,"Survey on Live VM Migration Techniques", International Journal of Advanced Research in Computer Engineering & Technology, Vol. 2, Issue. 1, January 2013.

[13] Pradip D. Patel, "Live Virtual Machine Migration Techniques in Cloud Computing: A Survey", International Journal of Computer Applications,Vol. 86, No. 16, January 2014.

[14] M.Tarighi, S.A.Motamedi, S.Sharifian, "A new model for virtual machine migration in virtualized cluster server based on Fuzzy Decision Making", Journal of Telecommunications, Vol. 1, Issue. 1, February 2010..

[15] Sheng Di, "Error-Tolerant Resource Allocation and Payment Minimization for Cloud System", IEEE Transactions on Parallel and Distributed Systems, Vol. 24, No. 6, June 2013.

[16] Olivier Beaumont, Lionel Eyraud-Dubois, Christopher Thraves Caro, and Hejer Rejeb, "Heterogeneous Resource Allocation under Degree Constraints", IEEE Transactions on Parallel and Distributed Systems, Vol. 24, No. 5, May 2013.

[17] Anton Beloglazov and Rajkumar Buyya," Managing Overloaded Hosts for Dynamic Consolidation of Virtual Machines in Cloud Data Centers Under Quality of Service Constraints", IEEE Transactions on Parallel and Distributed Systems , Vol. 23, No. 4, pp.19-23, 2012.

CHAPTER 8

APPENDIX

8.1 ECLIPSE AND JAVA PLATFORM

In computer programming, Eclipse is a multi-language software development environment comprising a base workspace and an extensible plug-in system for customizing the environment. It is written mostly in Java. It can be used to develop applications in Java and, by means of various plugins, other programming languages including Ada, C, C++, COBOL, Fortran, Haskell, JavaScript, Perl, PHP, Python, R, Ruby (including Ruby on Rails framework), Scala, Clojure, Groovy, Scheme, and Erlang. It can also be used to develop packages for the software Mathematica. Development environments include the Eclipse Java development tools (JDT) for Java and Scala, Eclipse CDT for C/C++ and Eclipse PDT for PHP, among others.

The initial codebase originated from IBM VisualAge. The Eclipse software development kit (SDK), which includes the Java development tools, is meant for Java developers. Users can extend its abilities by installing plug-ins written for the Eclipse Platform, such as development toolkits for other programming languages, and can write and contribute their own plug-in modules. Released under the terms of the Eclipse Public License, Eclipse SDK is free and open source software (although it is incompatible with the GNU General Public License). It was one of the first IDEs to run under GNU Classpath and it runs without problems under IcedTea. Following are the ways to create the project in this IDE.

You will want to create a new project for each assignment in CS108. This will let you separate the files for each HW into different folders. Before you start, make sure you're using the Java 1.5 environment. Go to Window->Preferences. On the left pane, click Java->Editor->Installed JREs. Make sure that the 1.5 JRE is added and checked.

1. Create a new project from starter files. Bring up the New Project Dialog by selecting File->New->Project.... Click Java and over on the right you should see Java Project appear. Click on Java Project.

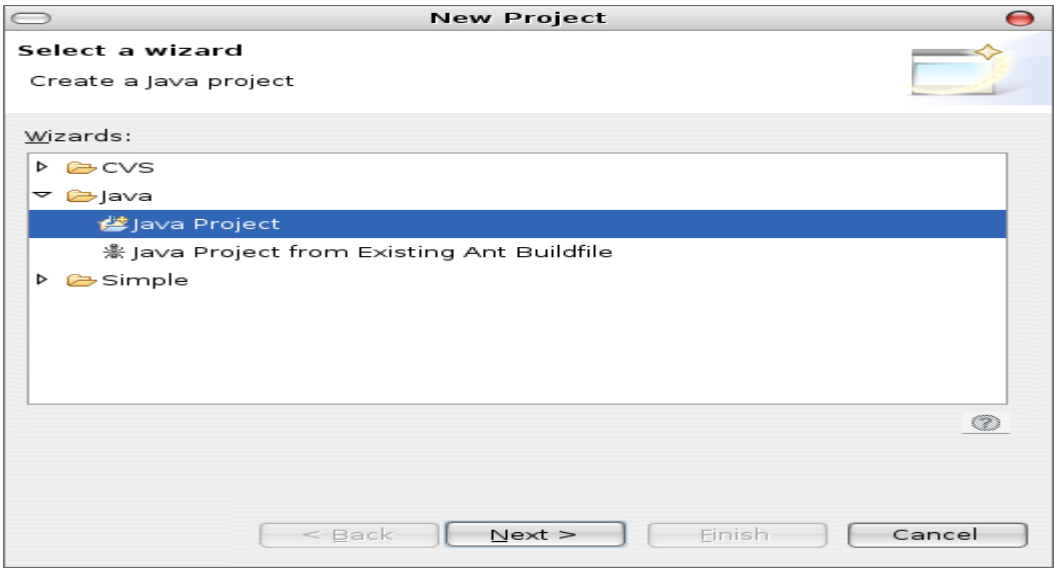

Fig. 8.1 Eclipse environment for project creation

2. Then click Next. Now, give your project a name. Click "Create project from existing source" and add then add the directory in which your project lives. We want to use Java 5 (also known as Java 1.5), so you can either specifically select it, or set 5.0 to as the default. Also, click on "Use project folder as root for sources and class files."

3. Now, just click Finish and you're done creating the project! If it asks you to switch to the Java Perspective, say yes. Another thing you want to do is make sure the project is using the correct JRE. Right click on your project in the Package Explorer on the left and click on Properties. Select Java Build Path and Choose the Libraries Tab. If the JRE is 1.5, you're fine.

If this isn't the case, you need to set a different JRE for the project. Click on "JRE System Library", and then click Edit. Select Alternative JRE and select the JRE you want from the drop down box.

4. Adding existing files to the project. If you had a Hello World directory with all your source files, an easy way to add these files to your Hello World is probably to just Import the Hello World directory. After adding the Hello World project,

you should now see it in the main window. Expand the 'Hello World' node and you should see a folder for your sources, 'src'. You will also see a node for the Java libraries, but you can ignore that. Right click on the 'src' directory and click Import.... Select File System as the input source and then click Next. Enter the CS108 HelloWorld directory as the directory of files to import. You should see the directory and its contents in the tree view. Click on the files you wish to import, or click Select All... to import all the files from that directory. Make sure that you also select Create Selected Folders Only. This way it will put all the files in the 'src' directory. Otherwise it would make a '/usr/class/cs108/HelloWorld/' directory in your 'src' directory, and everything would be in there, and it wouldn't be pretty. And finally, click Finish and you're done! Alternatively, you can just create a project from existing source, as mentioned above.

5. Adding new classes/files to the project. To add a new class to the project click File->New->Class. This will bring up the New Class Dialog. In this dialog, you can set a bunch of properties for the class. In my example, I've created a HelloWorld class in the default package, and I clicked on the option to create a main() stub, so I wouldn't have to type it out myself. Once you are done filling in the info for your class, hit Finish and you're done! The Java Perspective should look something like this:

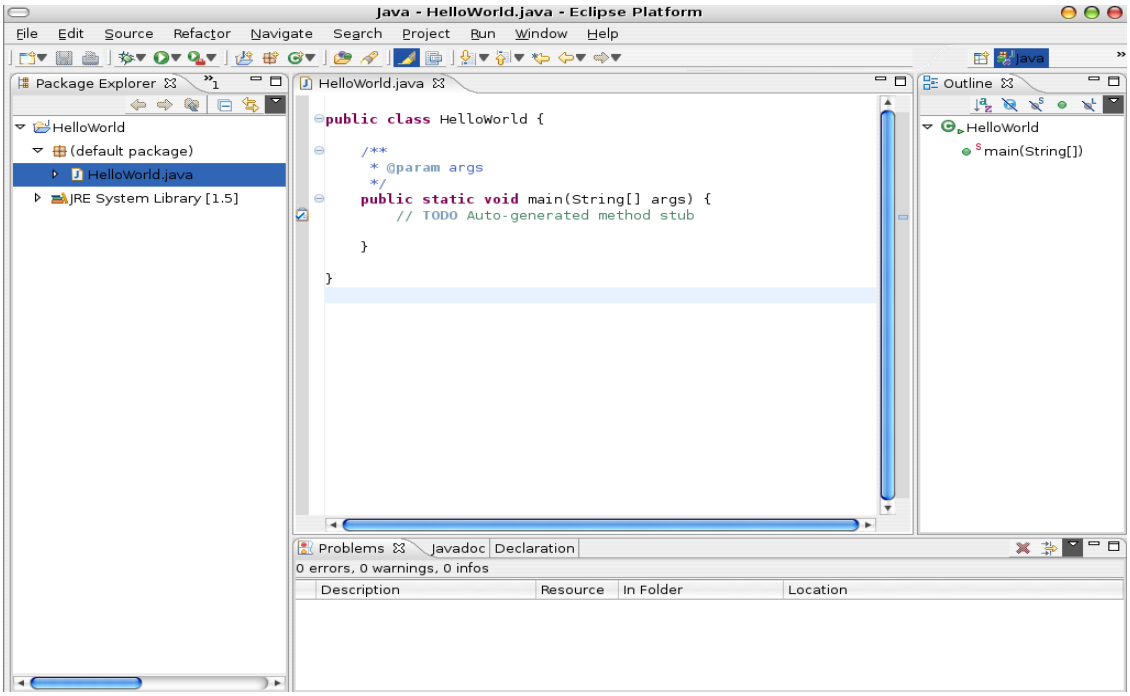

Fig. 8.2 Running process of eclipse

6. Building the project. To compile the .java files into .class files select Project->Build All. (from then on you can do Build Project which will only compile those .java files that haven't been compiled yet).

7. Running your program. To run your program once it has compiled without errors, either click Run->Run... or click the button that looks like 'Play'. Select Java Application and then click New. Then you will have a chance to select which class's main() you want to use as a starting point. Eclipse will automatically filter out classes that don't have a main(). You also get to specify any arguments you want to pass to main().

Once you're ready, hit the Run button, and you're off and running!

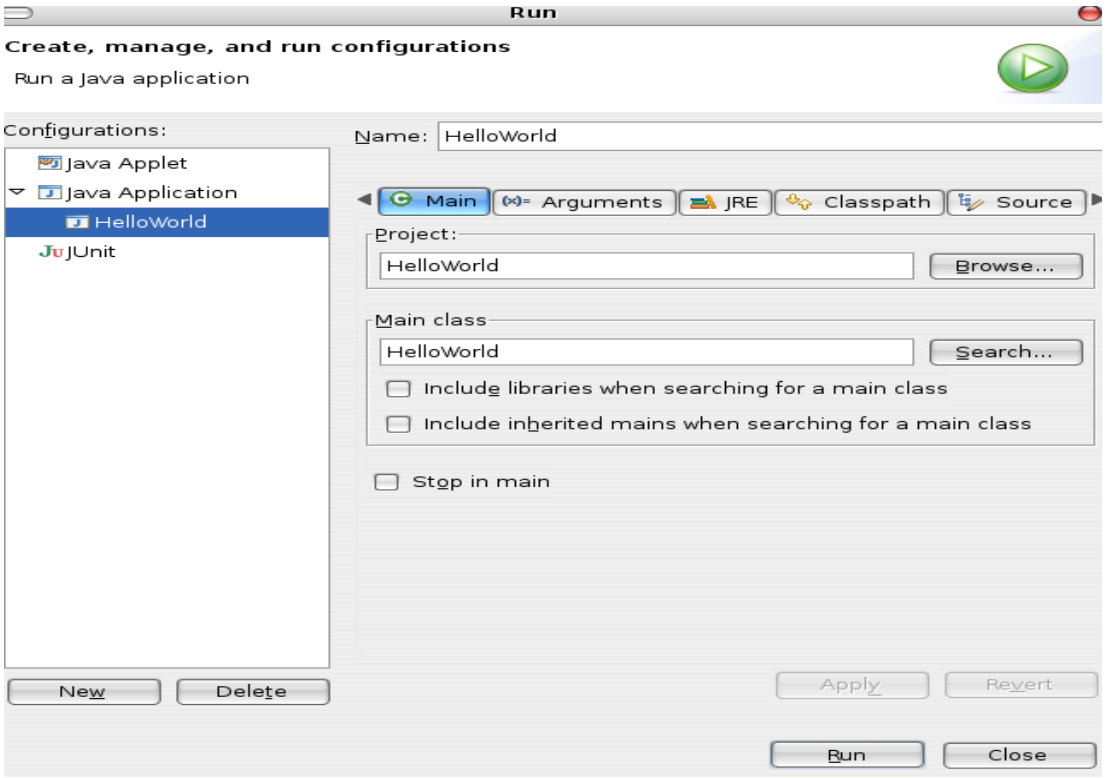

Fig 8.3 Building of project in eclipse

8. Debugging your program. To debug your program click Run->Debug.... This will pop up a dialog similar to the one used to Run the program. Click Debug (or the icon of the bug) and two things will happen. First, this will change your perspective to the Debug perspective. Secondly it will start running your program.

JAVA: Java is a general-purpose, class-based, object Oriented computer programming language that is specifically designed to have as few implementation dependencies as possible. It provides us the magical byte code. It is intended to let application developers "write once, run anywhere" (WORA), meaning that code that runs on one platform does not need to be recompiled to run on another. Java applications are typically compiled to byte code (class file) that can run on any Java virtual machine (JVM) regardless of computer architecture . Java is, as of 2012, one of the most popular programming languages in use, particularly for client-server web applications, with a reported 10 million users.[10][11] Java was originally developed by James Gosling at Sun Microsystems (which has since merged into Oracle Corporation) and released in 1995 as a core component of Sun Microsystems' Java platform. The language derives much of its syntax from C and C++, but it has fewer low-level facilities than either of them. While java was designed there was five principal in the mind of the James Gosling.

Principles

There were five primary goals in the creation of the Java language

1. It should be "simple, object-oriented and familiar"
2. It should be "robust and secure"
3. It should be "architecture-neutral and portable"
4. It should execute with "high performance"
5. It should be "interpreted, threaded, and dynamic

One characteristic of Java is portability, which means that computer programs written in the Java language must run similarly on any hardware/operating-system platform. This is achieved by compiling the Java language code to an intermediate representation called Java byte code, instead of directly to platform-specific machine code. Java byte code instructions are analogous to machine code, but they are intended to be interpreted by a virtual machine (VM) written specifically for the host hardware. End-users commonly use

a Java Runtime Environment (JRE) installed on their own machine for standalone Java applications, or in a Web browser for Java applets.

8.2 CLOUDSIM

Features & Advantages

- Discrete Time Event-Driven
- Support modeling and simulation of large scale Cloud computing environments, including data centers
- Support simulation of network connections among simulated elements

Advantages

- Time effectiveness
- Flexibility and applicability
- Test policies in repeatable and controllable environment
- Tune system bottlenecks before deploying on real clouds

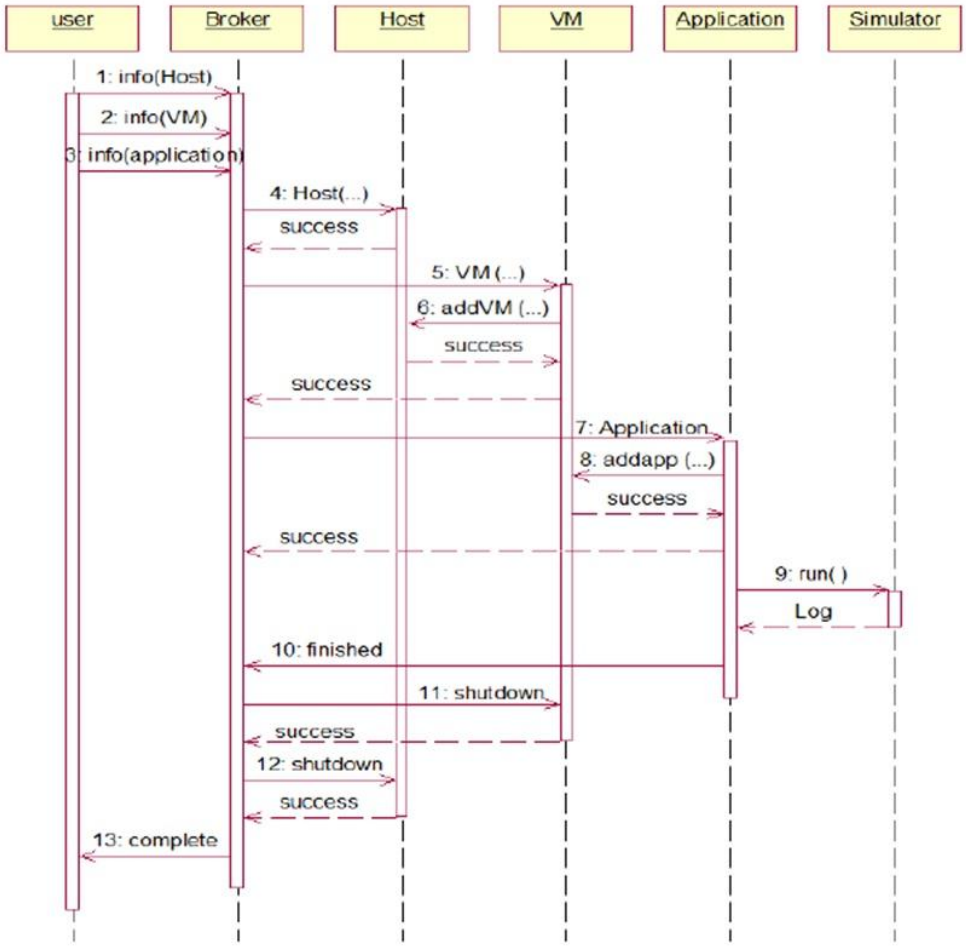

Fig. 8.4 Process flow of Cloudsim

8.3 Glossary

A

Augmentation

Autonomy

C

Cloudlet

Constraint programming

Consumption

CPU intensive

Cycle-stealing

D

Datacenter

Decentralization

E

Eclipse

Entropy

Eucalyptus environment

G

General Public license

Granularity

Green computing

L

Leverage

Load balancing

M

Migration

Multi-language

Multi-layered

Multiplexing

N

NP complete

O

Optimization

P

Pay-as-you-go

Pervasiveness

R

Reallocation

Return on investment

S

Service level agreement

Simulation

T

Threshold

Trade-off

V

Virtual machine

Virtualization

8.4 Abbrivations

SLA- Service Level Agreements

QoS- Quality of Service

VM- Virtual Machine

WORA- Write once, run anywhere

JVM- Java Virtual Machine

ROI- Return On Investment

BFDA- Best Fit Decreasing Algorithm

CP- Constraint Programming

Mips- million instruction per Second

SDK- Software Development Kit