TABLE OF CONENTS

INDEX

LIST OF FIGURES

ABSTARCT

Cloud computing is a new paradigm for delivering remote computing resources through a network. However, achieving an energy-efficiency control and simultaneously satisfying a performance guarantee have become critical issues for cloud providers. Three power-saving policies are implemented in cloud systems to mitigate server idle power. The challenges of controlling service rates and applying the N-policy to optimize operational cost within a performance guarantee are first studied. A cost function has been developed in which the costs of power consumption, system congestion and server startup are all taken into consideration. The effect of energy-efficiency controls on response times, operating modes and incurred costs are all demonstrated. Our objectives are to find the optimal service rate and mode-switching restriction, so as to minimize cost within a response time guarantee under varying arrival rates.

An efficient green control (EGC) algorithm is first proposed for solving constrained optimization problems and making costs/performances tradeoffs in systems with different power-saving policies. Simulation results show that the benefits of reducing operational costs and improving response times can be verified by applying the power-saving policies combined with the proposed algorithm as compared to a typical system under a same performance guarantee.

CHAPTER 1

CHAPTER 1
INTRODUCTION

1.1 General Overview

CLOUD computing is a new service model for sharing a pool of computing resources that can be rapidly accessed based on a converged infrastructure. In the past, an individual use or company can only use their own servers to manage application programs or store data. Nowadays, resources provided by cloud allow users to get on demand access with minimal management effort based on their needs. Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) are all existing service models.

Over recent years, significant efforts have been directed towards improving the throughput of mobile networks in order to support the massive increase in network traffic demand. However, further substantial improvements can no longer be achieved by the conventional architecture using large coverage cells. On the other hand, strategic deployment of low-power base stations (BSs), i.e., small-cells, has been shown to greatly improve indoor coverage and system throughput.

The main idea behind multi-tier cellular network or heterogeneous network (HetNet) deployment is to overlay low power and low cost devices on coverage holes or throughput demanding hotspots to supplement conventional single-tier cellular networks. Due to the overlapping topology of HetNets, where different tiers of BSs coexist in the same coverage area, tackling the cross-tier inter-cell interference (ICI) has become a major challenge and a bottleneck towards improving network performance. For example, interference coordination between large- and small-cells will significantly reduce the capacity in particular when a co-channel deployment is employed. Consequently, how to design and deploy HetNets while managing the cross-tier interference has attracted a lot of attention in the research community.

In, a transmit beam former design technique for a spectrum-sharing network with mixed quality-of service (QoS) requirement was studied. Furthermore, transmit beam

forming has also been considered as an efficient approach to interference management in HetNets. In, a downlink beam forming design is considered for minimizing the total transmits power under SINR and interference constraints on multiple-input single-output (MISO) channels in HetNets. The non-convex optimization problem is transformed into a convex semi definite programming problem, which reduces to a power control problem when all channel state information vectors are independent and identically distributed.

To satisfy uncertain workloads and to be highly available for users anywhere at any time, resource over-provisioning is a common situation in a cloud system. However, most electricity-dependent facilities will inevitably suffer from idle times or low utilization for some days or months since there usually have off-seasons caused by the nature of random arrivals. In fact, servers are only busy 10-30 percent of the time on average. As cloud computing is predicted to grow, substantial power consumption will result in not only huge operational cost but also tremendous amount of carbon dioxide (CO2) emissions. Therefore, an energy efficient control, especially in mitigating server idle power has become a critical concern in designing a modern green cloud system.

Ideally, shutting down servers when they are left idle during low-load periods is one of the most direct ways to reduce power consumption. Unfortunately, some negative effects are caused under improper system controls. First, burst arrivals may experience latency or be unable to access services. Second, there has a power consumption overhead caused by awakening servers from a power-off state too frequently. Third, the worst case is violating a service level agreement (SLA) due to the fact that shutting down servers may sacrifice quality of service (QoS). The SLA is known as an agreement in which QoS is a critical part of negotiation. A penalty is given when a cloud provider violates performance guarantees in a SLA contract.

In short, reducing power consumption in a cloud system has raised several concerns, without violating the SLA constraint or causing additional power consumption are both important. To avoid switching too often, a control approach called N policy, defined by Yadin and Naor had been extensively adopted in a variety of fields, such as computer systems, communication networks, wireless multimedia, etc. Queuing systems with the N policy will turn a server on only when items in a queue is greater than or equal to a

predetermined N threshold, instead of activating a power-off server immediately upon an item arrival. However, it may result in performance degradation when a server stays in a power-saving mode too long under a larger controlled N value.

CHAPTER 2

CHAPTER 2

LITERATURE SURVEY

Power savings in cloud systems have been extensively studied on various aspects in recent years, e.g., on the virtual machine (VM) side by migrating VMs, applying consolidation or allocation algorithms, and on the data center infrastructure side through resource allocations, energy managements, etc.

2.1 Power-Saving in Virtual Machine

In this project we tried to deal with virtual machine placement problem with a goal of minimizing the total energy consumption. A multi-dimensional space partition model and a virtual machine placement algorithm are presented[10]. When a new VM placement task arrived, their algorithm checked the posterior resource usage state for each feasible PM, and then chose the most suitable PM according to their proposed model to reduce the number of running PMs. We considered the problem of providing power budgeting support while dealing with many problems that arose when budgets virtualized systems. We managed power from a VM-centric point of view, where the goal was to be aware of global utility tradeoffs between different virtual machines (and their applications) when maintaining power constraints for the physical hardware on which they ran. Their approach to VM-aware power budgeting used multiple distributed managers integrated into the virtual power management (VPM) framework. Two issues in energy conservation algorithm are addressed. The placement of virtual machine image and the characteristics of virtual machines. Despite that the dynamic programming could find the optimal solution, its time complexity was prohibitive in practice. In the energy efficiency from the performance perspective was studied. A virtual machine based energy efficient data center architecture for cloud computing was presented. Then, potential performance overheads caused by server consolidation and lived migration of virtual machine technology was investigated. The potential performances overheads of server consolidation were evaluated[10].

2.2 Power-Saving in Computing Infrastructure

Later a basic theoretical model and used it in building managing, micro-grids, and datacenter energy management were presented. We analyzed these disparate energy management systems and defined a model for resource allocation that could be used for these and other energy management systems[12]. The Datacenter Energy Management project was focused on modeling energy consumption in data centers, with a goal to optimize electricity consumption. Their project was focused on collecting data to define basic fuel consumption curves. The problem of maximizing the revenues of cloud providers by trimming down their electricity costs was addressed. Policies were based on dynamic estimates of user demand, and system behavior models. Some approximations were used to handle the resulting models. We have demonstrated that decisions, such as how many servers were powered on can have a significant effect on the revenue earned by the provider. However, no startup power draw or performance guarantees was considered[12].

A Harmony, a Heterogeneity- Aware Resource Monitoring and management system that is capable of performing dynamic capacity provisioning(DCP) in heterogeneous data centers were studied. Using standard K-means clustering, we showed that the heterogeneous workload could be divided into multiple task classes with similar characteristics in terms of resource and performance objectives. The DCP was formulated as an optimization problem that considered machine and workload heterogeneity as well as reconfiguration costs[9].

A framework used to automatically manage computing resources of cloud infrastructures was proposed in to simultaneously achieve suitable QoS levels and to reduce the amount of energy used for providing services. We showed that via discrete-event-system (DES) simulation, their solution was able to manage physical resources of a data center in such a way to significantly reduce SLO violations with respect to a traditional approach. The energy-efficiency of the infrastructure was defined as the amount of energy used to serve a single application request[5]. A holistic resource management framework for embedding virtual data centers across geographically distributed datacenters are connected through a backbone network. The goal is to maximize the cloud provider’s revenue while ensuring that the infrastructure is as environment- friendly as possible.

CHAPTER 3

CHAPTER 3

SYSTEM ANALYSIS

More power consumption requires for cellular communication. Less Energy Efficiency Calculations of Existing system almost done based on distance calculation process. More computational complexity

3.1 Introduction to System Analysis

System

A system is an orderly group of interdependent components linked together according to a plan to achieve a specific objective. Its main characteristics are organization, interaction, interdependence, integration and a central objective.

System Analysis

System analysis and design are the application of the system approach to problem solving generally using computers. To reconstruct a system the analyst must consider its elements output and inputs, processors, controls, feedback and environment.

Analysis

Analysis is a detailed study of the various operations performed by a system and their relationships within and outside of the system. One aspect of analysis is defining the boundaries of the system and determining whether or not a candidate system should consider other related systems. During analysis data are collected on the available files decision points and transactions handled by the present system. This involves gathering information and using structured tools for analysis.

3.2 Existing System

Most existing works on resource allocation in HetNets have focused on homogeneous users with same service and rate demands. Energy consumption issue has been investigated extensively in recent years. To increase the energy efficiency (EE) of the cellular systems, traffic-aware transmission strategies have been proposed in, where under-utilized BSs are recommended to switch to sleep mode or be shut off during off-peak time of traffic loads. It is practical because the deployments of existing cellular networks are always designed for peak load traffics, leading to very inefficient usage of BSs during off-peak time.

The main difficulty is addressing the mutual interference between different kinds of BSs. Because of that reason power consumption is more. Existing system leads to no convex (Unsolvable problem with no proper solution/ More computational complexity) problem. It is generally hard to tackle.

Disadvantages of Existing System

- More power consumption requires for cellular communication
- Less Energy Efficiency
- Calculations of Existing system almost done based on distance calculation process.
- More computational complexity

3.3 Proposed System

In this project, we address the EE optimization problem for downlink two-tier HetNets comprised of a single macro cell and multiple pico-cells. Considering a heterogeneous real-time and non-real-time traffic, transmit beam forming design and power allocation policies are jointly considered in order to optimize the system energy efficiency. The EE resource allocation problem under consideration is a mixed combinatorial and no convex optimization problem, which is extremely difficult to solve. In order to reduce the computational complexity, we decompose the original problem with multiple inequality constraints into multiple optimization problems with single inequality constraint.

For the latter problem, a two-layer resource allocation algorithm is proposed based on the quasi concavity property of EE.

Green Computing is the environmentally responsible and eco-friendly use of computer and the resources. In broader terms, it is also defined as the study of designing, engineering the computing devices in a way that reduces their environmental impact.

When the resource utilization of active servers is too low, some of them can be turned off to save energy. This is handled in our green computing algorithm.

Advantages of the Proposed System

- Good Energy Efficiency.
- Calculations of proposed system done based on distance calculation with SINR, Signal Ratio along with Inference and Noise value.
- More computational complexity

3.4 Feasibility Study

Feasibility is the determination of whether or not a project is worth doing. The process followed in making this determination is called feasibility Study. This type of study if a project can and should be taken. In the conduct of the feasibility study, the analyst will usually consider seven distinct, but inter-related types of feasibility.

Technical Feasibility

This is considered with specifying equipment and software that will successful satisfy the user requirement the technical needs of the system may vary considerably but might include

- The facility to produce outputs in a given time
- Response time under certain conditions.
- Ability to process a certain column of transaction at a particular speed.

Economic Feasibility

Economic analysis is the most frequently used technique for evaluating the effectiveness of a proposed system. More commonly known as cost / benefit analysis. The procedure is to

determine the benefits and savings are expected form a proposed system and a compare them with costs. It benefits outweigh costs; a decision is taken to design and implement the system will have to be made if it is to have a chance of being approved. There is an ongoing effort that improves in accuracy at each phase of the system life cycle.

Operational Feasibility

It is mainly related to human organization and political aspects. These points are considered are

- What changes will be brought with the system?
- What organizational structures are distributed?
- What new skills will be required?
- Do the existing system staff members have these skills?
- If not, can they be trained in the course of time?

CHAPTER 4

CHAPTER 4

SYSTEM DESIGN

4.1 Logical Design

Design for WebApps encompasses technical and non-technical activities. The look and feel of content is developed as part of graphic design; the aesthetic layout of the user interface is created as part of interface design; and the technical structure of the WebApp is modeled as part of architectural and navigational design.

This argues that a Web engineer must design an interface so that it answers three primary questions for the end-user:

1. Where am I*?* – The interface should provide an indication of the WebApp has been accessed and inform the user of her location in the content.
2. What can I do now*?* – The interface should always help the user understand his current options- what functions are available, what links are live, what content is relevant.
3. Where have I been; where am I going*?* – The interface must facilitate navigation. Hence it must provide a "map" of where the user has been and what paths may be taken to move elsewhere in the WebApp.

4.2 Design goals

The following are the design goals that are applicable to virtually every WebApp regardless of application domain, size, or complexity.

1. Simplicity
2. Consistency
3. Identity
4. Visual appeal
5. Compatibility.

Design leads to a model that contains the appropriate mix of aesthetics, content, and technology. The mix will vary depending upon the nature of the WebApp, and as a consequence the design activities that are emphasized will also vary.

4.3 System Architecture

Large systems are decomposed into sub-systems that provide some related set of services. The initial design process of identifying these sub-systems and establishing a framework for sub-systems control and communication is called architecture design and the output of this design process is a description of the software architecture. The architectural design process is concerned with establishing a basic structural framework for a system. It involves identifying the major components of the system and communications between these components. Fig 4.1 here represents the system architecture of green computing.

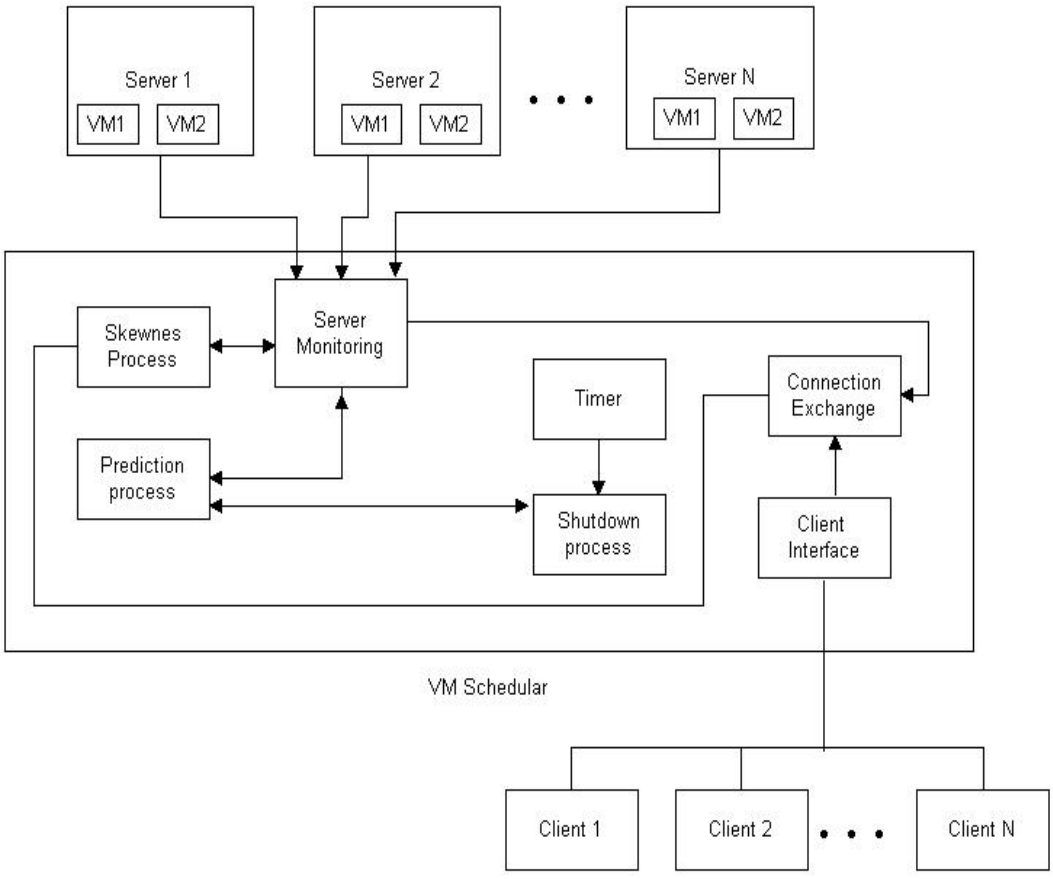


Fig. 4.1: System Architecture Overview

4.4 Flowchart

4.4.1 Flowchart for Skewness

A flowchart is common type of notation that represents an algorithm or process, this diagrammatic representation give step by step solution to a problem. In the Fig 4.2 we are representing the flowchart of skewness algorithm for computing the load.

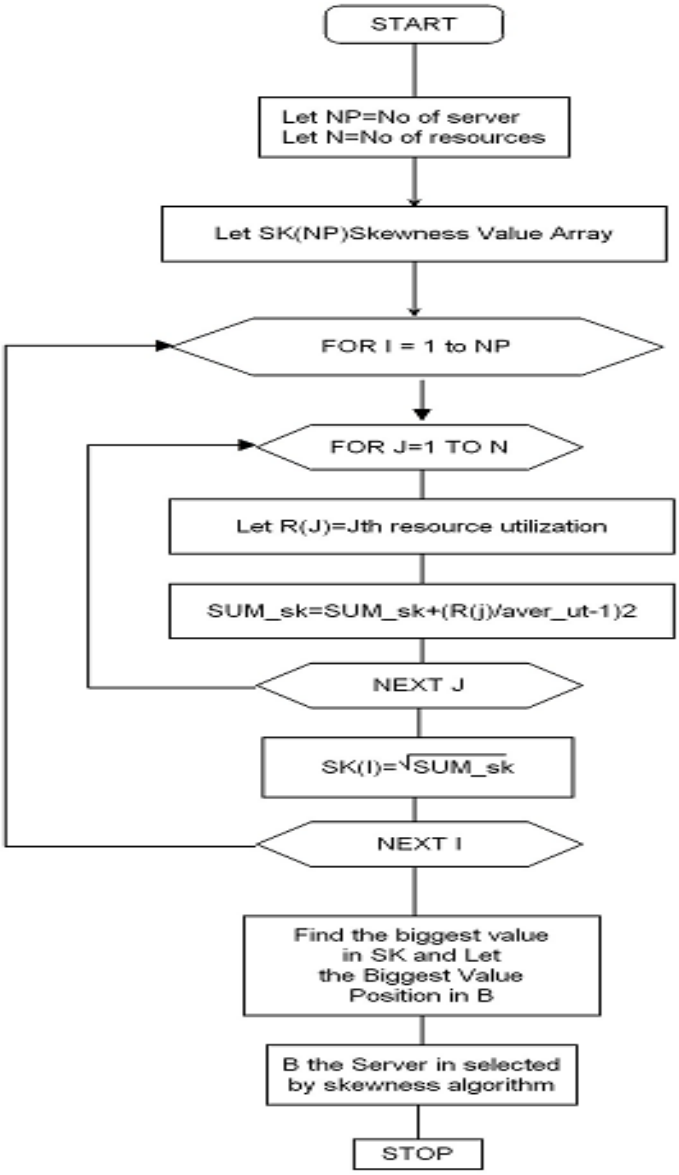


Fig. 4.2:Flowchart of Skewness Algorithm

4.4.2 Flowchart for prediction algorithm

The below Fig 4.3 shows the flowchart for prediction algorithm for shutting down the servers.

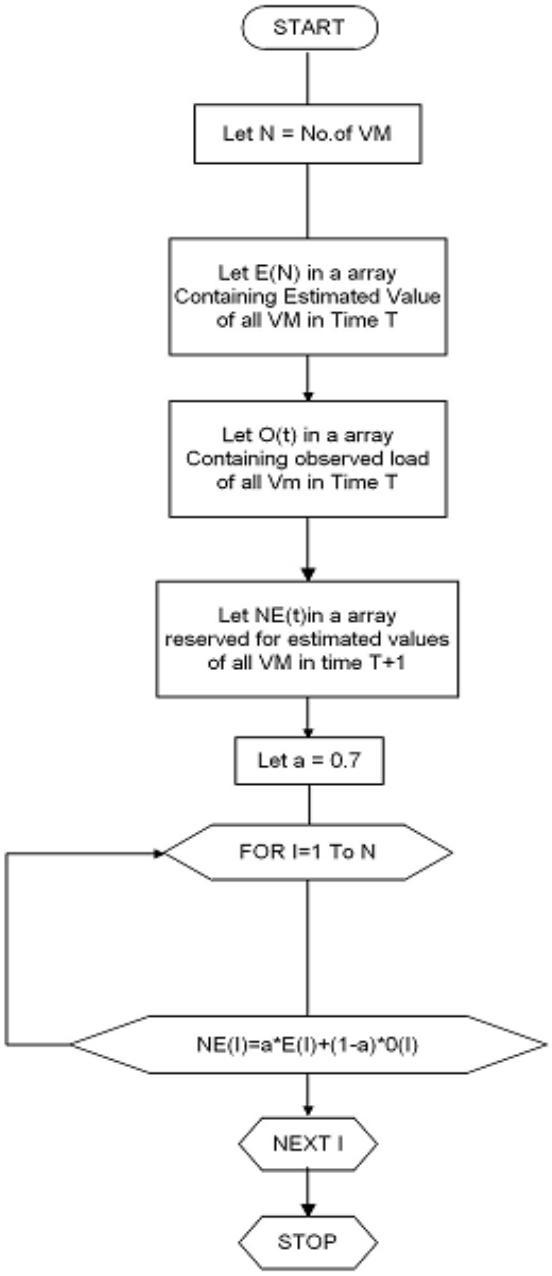


Fig 4.3:Flowchart for Prediction Algorithm

4.5 Dataflow diagram

A data flow diagram is a graphical representation of the "flow" of data through an information system. Data Flow models are used to show how data flows through a sequence of processing steps. The data is transformed at each step before moving on to the next stage.

These processing steps or transformation are program functions when Data Flow diagrams are used to document a software design. In Fig 4.4 we are representing the data flow diagram of the system.

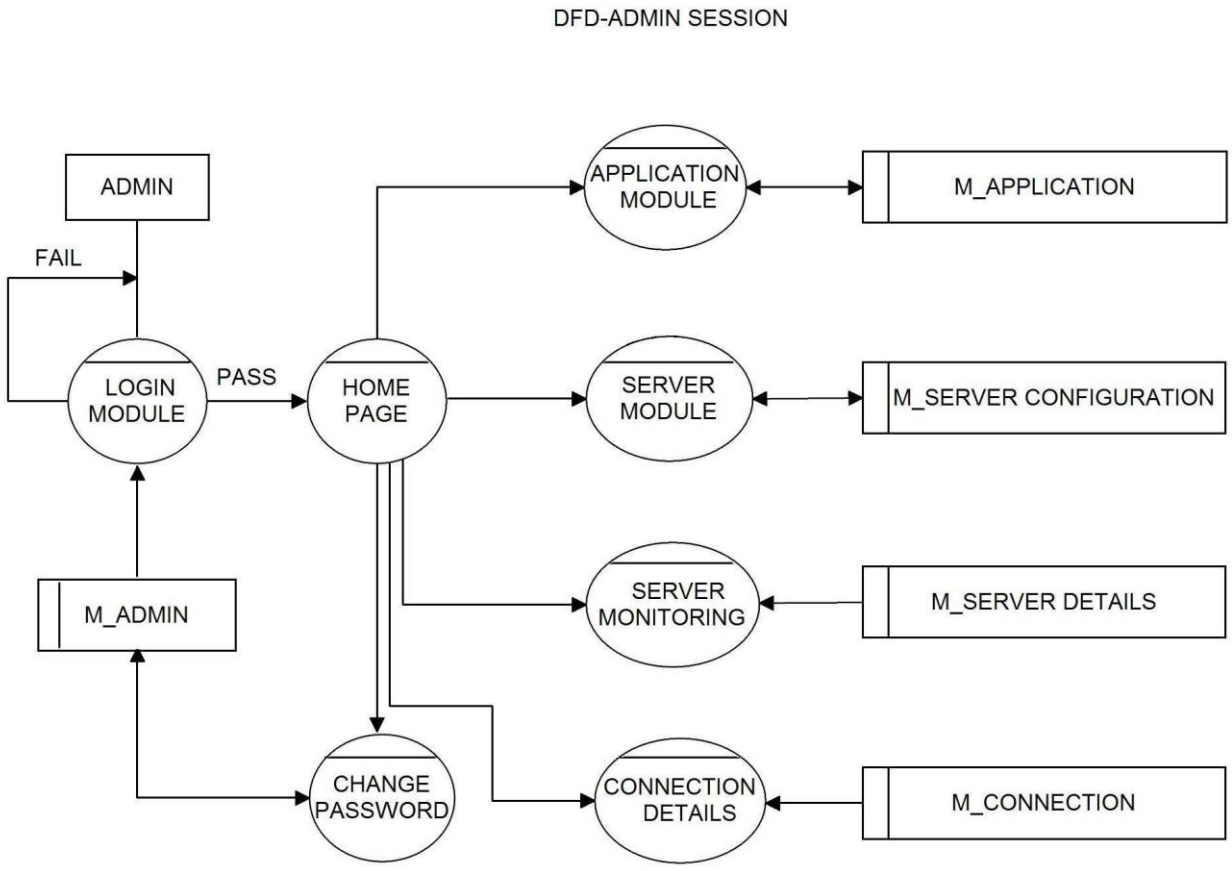


Fig. 4.4:Dataflow diagram of the system

4.5 Detailed Architecture

4.5.1 Skewness Algorithm

We introduce the concept of skewness to quantify the unevenness in the utilization of multiple resources on a server.

Let *n* be the number of resources we consider and *ri* be the utilization of the *i*-th resource. We define the resource skewness of a server *p* as

$$skewness(p) = \sqrt{\sum_{i=1}^{n}(\frac{r_i}{\overline{r}} - 1)^2}$$

where *r* is the average utilization of all resources for server *p*. In practice, not all types of resources are performance critical and hence we only need to consider bottleneck resources in the above calculation. By minimizing the skewness, we can combine different types of workloads nicely and improve the overall utilization of server resources.

4.5.2 Hot and Cold Spots

Our algorithm executes periodically to evaluate the resource allocation status based on the predicted future resource demands of VMs.

We define a server as a hot spot if the utilization of any of its resources is above a hot threshold. This indicates that the server is overloaded and hence some VMs running on it should be migrated away. We define the temperature of a hot spot p as the square sum of its resource utilization beyond the hot threshold. We define a server as a cold spot if the utilizations of all its resources are below a cold threshold.

This indicates that the server is mostly idle and a potential candidate to turn off to save energy. However, we do so only when the average resource utilization of all actively used servers (i.e., APMs) in the system is below a green computing threshold. A server is actively used if it has at least one VM running. Otherwise, it is inactive. Finally, we define the warm threshold to be a level of resource utilization that is sufficiently high to justify having the server running but not so high as to risk becoming a hot spot in the face of temporary fluctuation of application resource demands. Different types of resources can have different thresholds.

For example, we can define the hot thresholds for CPU and memory resources to be 90% and 80%, respectively. Thus a server is a hot spot if either its CPU usage is above 90% or its memory usage is above 80%.

4.5.3 Hot spot mitigation

We sort the list of hot spots in the system in descending temperature. Our goal is to eliminate all hot spots if possible. Otherwise, keep their temperature as low as possible. For each server, we first decide which of its VMs should be migrated away. We sort its list of VMs based on the resulting temperature of the server if that VM is migrated away. We aim to migrate away the VM that can reduce the server's temperature the most. In case of ties, we select the VM whose removal can reduce the skewness of the server the most. For each VM in the list, we see if we can find a destination server to accommodate it. The server must not become a hot spot after accepting this VM. Among all such servers, we select one whose skewness can be reduced the most by accepting this VM. Note that this reduction can be negative which means we select the server whose skewness increases the least.

If a destination server is found, we record the migration of the VM to that server and update the predicted load of related servers. Otherwise, we move on to the next VM in the list and try to find a destination server for it.

As long as we can find a destination server for any of its VMs, we consider this run of the algorithm a success and then move on to the next hot spot. Note that each run of the algorithm migrates away at most one VM from the overloaded server.

This does not necessarily eliminate the hot spot, but at least reduces its temperature. If it remains a hot spot in the next decision run, the algorithm will repeat this process. It is possible to design the algorithm so that it can migrate away multiple VMs during each run. But this can add more load on the related servers during a period when they are already overloaded. We decide to use this more conservative approach and leave the system some time to react before initiating additional migrations.

4.5.4 Green computing

When the resource utilization of active servers is too low, some of them can be turned off to save energy. This is handled in our green computing algorithm.

The challenge here is to reduce the number of active servers during low load without sacrificing performance either now or in the future. We need to avoid oscillation in the system. Our green computing algorithm is invoked when the average utilizations of all resources on active servers are below the green computing threshold. We sort the list of cold spots in the system based on the ascending order of their memory size.

Since we need to migrate away all its VMs before we can shut down an under-utilized server, we define the memory size of a cold spot as the aggregate memory size of all VMs running on it. Recall that our model assumes all VMs connect to a shared back-end storage. Hence, the cost of a VM live migration is determined mostly by its memory footprint.

We try to eliminate the cold spot with the lowest cost first. For a cold spot *p*, we check if we can migrate all its VMs somewhere else. For each VM on *p*, we try to find a destination server to accommodate it. The resource utilizations of the server after accepting the VM must be below the warm threshold. While we can save energy by consolidating under-utilized servers, overdoing it may create hot spots in the future. The warm threshold is designed to prevent that. If multiple servers satisfy the above criterion, we prefer one that is not a current cold spot. This is because increasing load on a cold spot reduces the likelihood that it can be eliminated.

However, we will accept a cold spot as the destination server if necessary. All things being equal, we select a destination server whose skewness can be reduced the most by accepting this VM. If we can find destination servers for all VMs on a cold spot, we record the sequence of migrations and update the predicted load of related servers. Otherwise, we do not migrate any of its VMs. The list of cold spots is also updated because some of them may no longer be cold due to the proposed VM migrations in the above process.

The above consolidation adds extra load onto the related servers. This is not as serious a problem as in the hot spot mitigation case because green computing is initiated only when the load in the system is low. Nevertheless, we want to bind the extra load due to server consolidation. We restrict the number of cold spots that can be eliminated in each run of the

algorithm to be no more than a certain percentage of active servers in the system. This is called the consolidation limit.

Note that we eliminate cold spots in the system only when the average load of all active servers (APMs) is below the green computing threshold. Otherwise, we leave those cold spots there as potential destination machines for future offloading. This is consistent with our philosophy that green computing should be conducted conservatively.

4.6 Sequence Diagram

Sequence diagram emphasizes on time sequences of messages and collaboration diagram emphasizes on the structural organization of the objects that send and receive messages.

Purpose: The purpose of interaction diagrams is to visualize the interactive behaviour of the system. Now visualizing interaction is a difficult task. So the solution is to use the different types of models to capture the different aspects of the interaction.

In Fig 4.6.1 shows the sequence diagram for the interaction between the components.

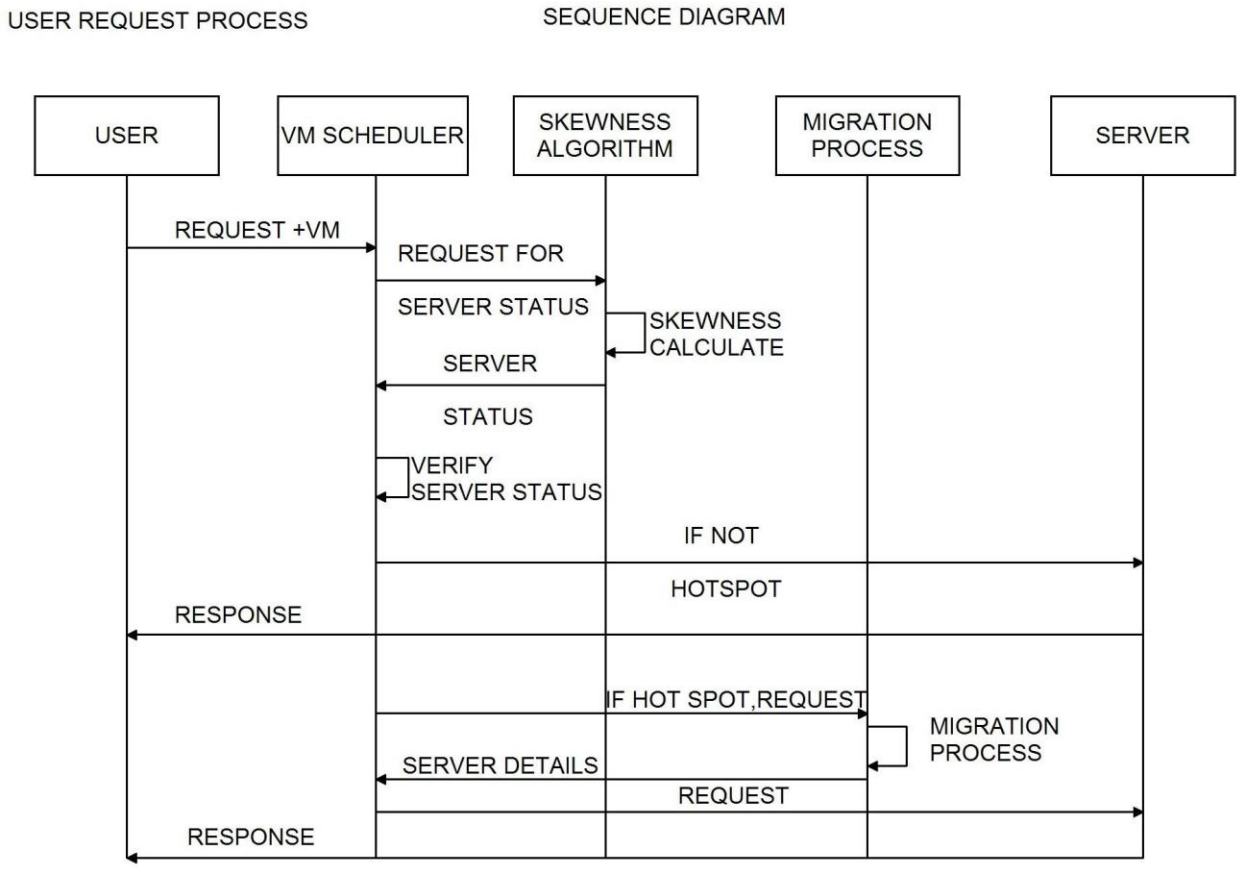


Fig 4.6.1:Sequence Diagram-1

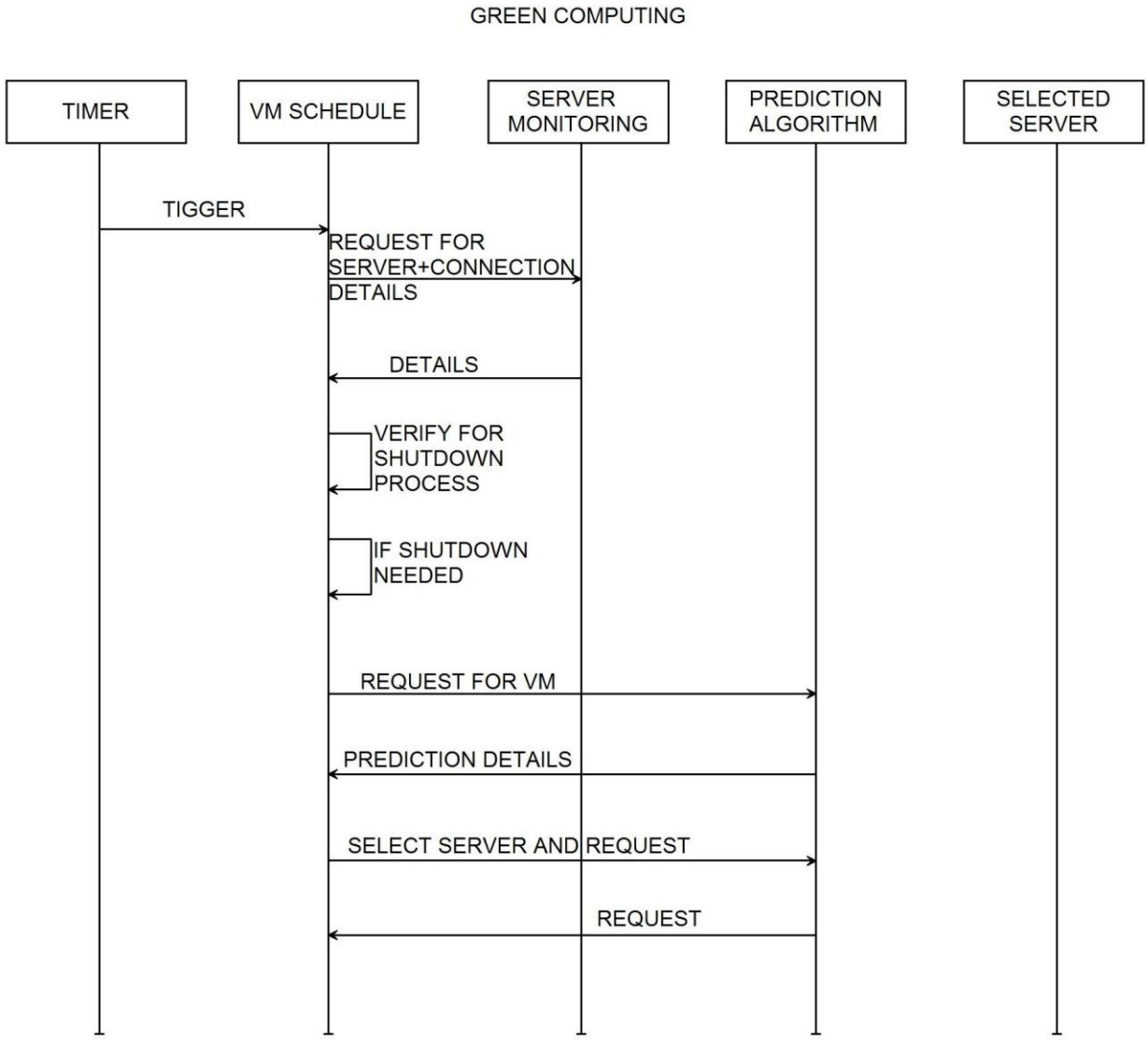


Fig 4.6.2: Sequence Diagram-2

The above Fig 4.6.2 shows the sequence diagram of green computing.

4.7 Use case diagram

These internal and external agents are known as actors. So use case diagrams consists of actors, use cases and their relationships. The diagram is used to model the system/subsystem of an application. A single use case diagram captures a particular functionality of a system. So to model to entire system numbers of use case diagrams are used. The purpose of use case diagram is to capture the dynamic aspect of a system because other four diagrams (activity, sequence, collaboration and state chart) are also having the same purpose. So look into specific purpose which will distinguish it from other four diagrams. Use case diagrams are used the gather the requirement of a system including internal and external influences, these requirements. So when a system is analyzed to gather its functionalities use cases are prepared and actors are identified. Below Fig 4.7.1 shows Use case diagram.

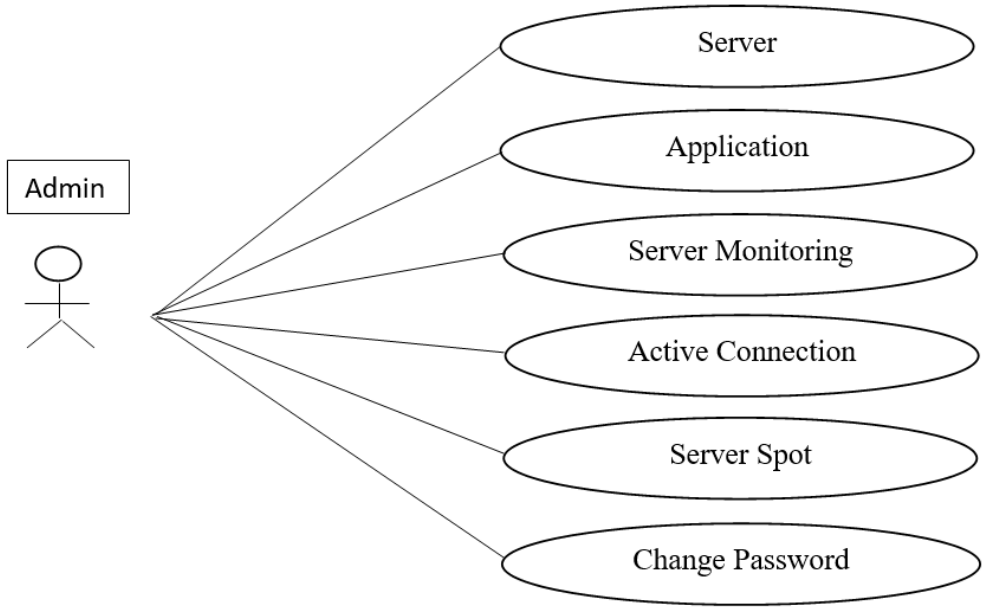


Fig 4.7.1 Use case diagram

4.8 Class Diagram

Below Fig 4.8 represents the class diagram for green computing.

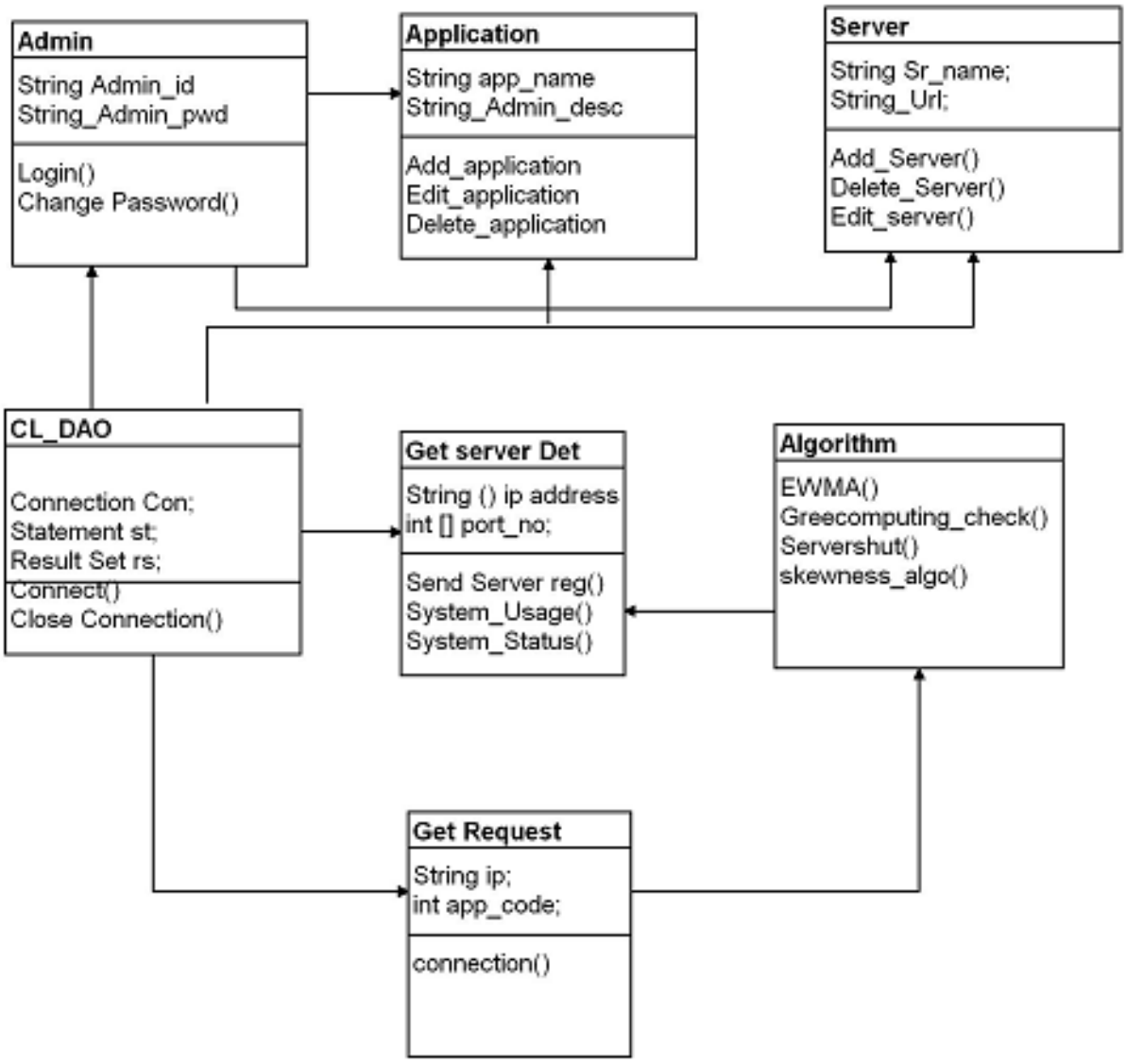


Fig 4.8: Class Diagram

4.9 E-R Diagram

An entity–relationship model (ER model) describes inter-related things of interest in a specific domain of knowledge. An ER model is composed of entity types (which classify the things of interest) and specifies relationships that can exist between instances of those entity types. Below Fig 4.9 represents the entity-relationship model for managing the connection.

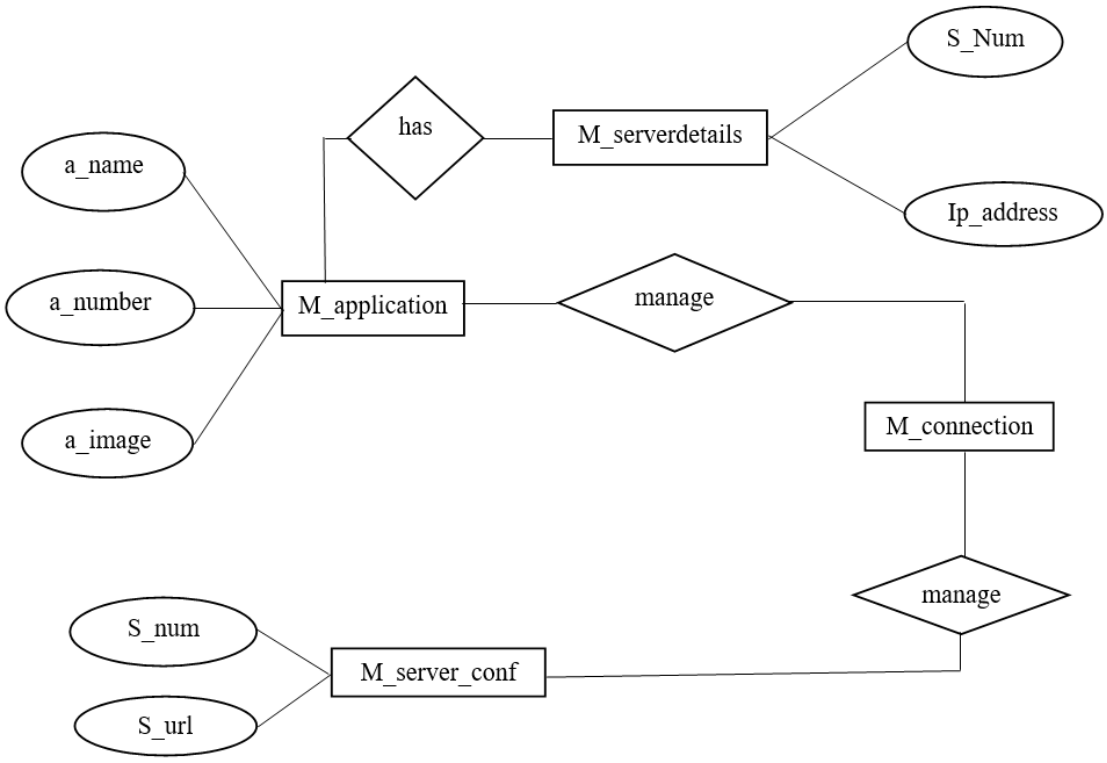


Fig. 4.9: E-R Diagram

CHAPTER 5

CHAPTER 5

SYSTEM REQUIREMENTS

5.1 Hardware Requirements

To be used efficiently, all computer software needs certain hardware components. The most common set of requirements defined by any operating system is the physical computer resources, also known as hardware. A hardware requirements list is often accompanied by a hardware compatibility list.

The following list discusses the various aspects of hardware requirements used for this project.

- Processor : Dual Core Processor Or above
- RAM : 2GB RAM or above
- Hard Disk : 40GB

5.2 Software Requirements

Software requirements deal with defining software resource requirements and prerequisites that need to be installed on a computer to provide optimal functioning of an application. These requirements or prerequisites are generally not included in the software installation package and need to be installed separately before the software is installed.

The following list discusses the various aspects of software requirements used for this project.

5.2.1 Front end:

HTML5, CSS3, JavaScript

5.2.2 Back end:

Java, J2EE (Servlet, JDBC, JSP)

5.2.3 Database:

MySQL / Oracle

5.2.4 Tools:

- Eclipse IDE, Web Browser

CHAPTER 6

CHAPTER 6

IMPLEMENTATION

6.1 Introduction

Implementation is the realization of an application, or execution of a plan, idea, model, design, specification, standard, algorithm, or policy. In other words, an implementation is a realization of a technical specification or algorithm as a program, software component, or other computer system through programming and deployment. Many implementations may exist for a given specification or standard.

Implementation is one of the most important phases of the Software Development Life Cycle (SDLC). It encompasses all the processes involved in getting new software or hardware operating properly in its environment, including installation, configuration, running, testing, and making necessary changes. Specifically, it involves coding the system using a particular programming language and transferring the design into an actual working system.

This phase of the system is conducted with the idea that whatever is designed should be implemented, keeping in mind that it fulfills user requirements, objective and scope of the system. The implementation phase produces the solution to the user problem.

6.2 Overview of System Implementation

This project is implemented considering the following aspects:

1. Usability Aspect.
2. Technical Aspect.

6.2.1 Usability Aspect

The usability aspect of implementation of the project is realized using two principles:

a.) The project is implemented as a Java application

There could be many ways of implementing this project. We have chosen JAVA to come up with the required reader. The reason being many:

Firstly, Java provides a wonderful libraries which simplifies the implementation part of it. Secondly, JAVA is platform independent, meaning the project can run on literally any platform which has JVM installed within it. Thirdly, Oracle Corporation claims more than 70

billion devices run on JAVA which makes the end users used to it. Lastly, it can be readily portable to any devices like mobile phones, iPad, PDA, and any hand held devices that are capable of running JAVA.

6.2.2 Technical Aspect

The technical aspect of implementation of the project is realized as explained below:

6.2.2.1 Servers

Apache Tomcat to develop the product

Apache Tomcat (or simply Tomcat, formerly also Jakarta Tomcat) is an open source web server and servlet container developed by the Apache Software Foundation (ASF). Tomcat implements the Java Servlet and the Java Server Pages (JSP) specifications from Sun Microsystems, and provides a "pure Java" HTTP web server environment for Java code to run.

Apache Tomcat includes tools for configuration and management, but can also be configured by editing XML configuration files

6.2.2.1 Database

MySQL officially, the world's most widely used open source relational database management system (RDBMS) that runs as a server providing multi-user access to a number of databases. The SQL phrase stands for Structured Query Language. The MySQL development project has made its source code available under the terms of the GNU General Public License, as well as under a variety of proprietary agreements. MySQL was owned and sponsored by a single for-profit firm, the Swedish company MySQL AB, now owned by Oracle Corporation.

MySQL is a popular choice of database for use in web applications, and is a central component of the widely used LAMP open source web application software stack (and other 'AMP' stacks). LAMP is an acronym for "Linux, Apache, MySQL, Perl/PHP/Python." Free-software-open source projects that require a full-featured database management system often use MySQL.

6.3 Project Module Description

This project is implemented using an admin, user and another system running as server with resource allocation for green computing.

1. **Admin**

- Login
- Server Details (Add, Edit, Delete)
- Applications Details (Add, Edit, Delete)
- Information System
    - Server Monitoring
    - Active Connections
    - Server spot
- Change Password

2. **User Session**

- Applications access

3. **Resource Allocation with Green Computing**

6.3.1 Modules

1. Connection Handling
    - Application access based on Resource allocation system
2. Overload avoidance
3. Green computing
    - Skewness Calculation
    - Load Prediction Management

6.3.2 Modules Description

1. **Application access based on Resource allocation system**

    If the user access the application, based on Load Prediction and Server status server will be selected.

2. **Overload avoidance**

The capacity of a PM should be sufficient to satisfy the resource needs of all VMs running on it. Otherwise, the PM is overloaded and can lead to degraded performance of its VMs. We introduce the concept of "skewness" to measure the uneven utilization of a server. By minimizing skewness, we can improve the overall utilization of servers in the face of multi-dimensional resource constraints. To perform skewness algorithm we should need server status spot. Types of server status spot:

a. **Hot spot**-: We define a server as a hot spot if the utilization of any of its resources is above a hot threshold. This indicates that the server is overloaded and hence some VMs running on it should be migrated away.
b. **Cold spot**-: We define a server as a cold spot if the utilizations of all its resources are below a cold threshold. This indicates that the server is mostly idle and a potential candidate to turn off to save energy.
c. **Warm spot**-: We define a server as a warm spot if the utilizations of all its resources are below a hot threshold and below cold threshold. This indicates that the server is ready to run VMs.

3. **Green computing**

The number of PMs used should be minimized as long as they can still satisfy the needs of all VMs. Idle PMs can be turned off to save energy. When the server becomes cold spot, we should not give further more connections to that server and we will turn off the server. This will do by using Green computing.

GREEN COMPUTE CHECK

Step 1: Input: Let C represent the computer usage and M be the memory usage, threshold denoted by Ω.

Step 2: The threshold values is calculated using the skewness algorithm S.

Step 3: Server accepts the input from the client C,M.

Step 4: Server checks the Total load T(C,M).

Step 5: Run EGC algorithm.

Step 6: if T>Ω then migrate request to the next server

else

Step 7: The request is handled by the current server.

SKEWNESS ALGORITHM(S)

Step 1: Let NP be the number of servers, N be the number of resources.

Step 2: for i=1 to NP

Step 3: for j=1 to N

Step 4: Let R(J) = Jth resource utilization

Step 5: SUM_sk = SUM_sk + (R(j)/average_utilization-1)*2

Step 6: J= J+1

Step 7: SK(i) = $\sqrt{SUM\_sk}$

Step 8: i = i+1

Step 9: Find the largest value in SK and L would position to the specific server.

Step 10: Let L now represent the current server.

Step 11: end for i.

Step 12: end for j.

ALGORITHM FOR SERVER SHUTDOWN

Step 1: EGC algorithm calculates the Total Load T.

Step 2: When the total load reaches the min(Ω),here Ω denotes the threshold.

Step 3: The server would migrate back in return switching the mode to idle.

Step 4: If the load on the current server drops way below the threshold Ω.

Step 5: Switch to sleep mode.

EGC ALGORITHM

Input:

1. An arrival rate μ.

2. Upper bound of the server rate and the waiting buffer, denoted by mu and Nb.

4. A response time guarantee x.

5. System parameters {memory m and cpu c}

Step 1: Calculate the system utilization.

If

Step 2:The current test parameters satisfy the constraint of

Step 3: Calculate the response time;

Else

Return to step 1

End

Step 4. If

The current test parameters satisfy the constraint then allocate resources to client request

Else

Return to step 1

END

4. Load Prediction Management

We need to predict the future resource needs of Virtual machine (VMs). As said earlier, our focus is on Internet applications. One solution is to look inside a VM for application level statistics, e.g., by parsing logs of pending requests. Doing so requires modification of the VM which may not always be possible. Instead, we make our prediction based on the past external behaviors of VMs. In this project, the exponentially weighted moving average prediction algorithm plays an important role in improving the stability and performance of our resource allocation decisions. Based on Physical machines (PMs) Usage we will select server using algorithm.

PREDICTION ALGORITHM

Step 1: Let N denotes the No of virtual machines.

Step 2: Let E(N) in a array containing estimated value of a VM in Time T.

Step 3: Let O(t) in a array containing observed load if all VM in Time(T).

Step 4: Let NE(t) in a array reserved for estimated values of all VM in time T+1

Step 5: So by taking a=0.7

Step 6: for i=1 to N

Step 7: NE(i) = a*E(i) + (1-a) * O(i)

Step 8: i = i+1

Step 9: end for i.

6.4 MySQL database created

m_admin

Fields

Field	Type	Null	Key	Extra
admin_id	char(100)	NO	PRI	
admin_pwd	char(20)	NO		
admin_ipaddress	varchar(50)	YES		
admin_port	int(20)	YES		

m_application

Fields

Field	Type	Null	Key	Extra
a_no	int(10)	NO	PRI	auto_increment
a_name	varchar(100)	NO		
s_no	int(50)	YES	MUL	
a_image	varchar(100)	YES		
a_status	varchar(50)	YES		

m_connections

Fields

Field	Type	Null	Key	Extra
con_no	int(5)	NO	PRI	auto_increment
con_date	varchar(25)	YES		
con_time_begin	varchar(25)	YES		
con_time_end	varchar(25)	YES		
con_ip	varchar(50)	YES		
s_no	int(10)	YES	MUL	
a_no	int(10)	YES	MUL	
da_status	varchar(15)	YES		

m_server_configuration

Fields

Field	Type	Null	Key	Extra
s_no	int(10)	NO	PRI	auto_increment
s_name	varchar(30)	NO		
s_url	varchar(100)	YES		
s_current_connection	int(25)	YES		
s_status	varchar(10)	YES		
s_ipaddress	varchar(50)	NO	PRI	
s_port	int(50)	NO	PRI	

m_serverdetails

Fields

Field	Type	Null	Key	Extra
s_no	int(11)	NO	PRI	auto_increment
ip_address	varchar(100)	NO	PRI	
port_number	int(20)	NO	PRI	
vendor	varchar(50)	YES		
processor	varchar(50)	YES		
mhz	int(50)	YES		
number_of_cpu	int(10)	YES		
cpu_usage	varchar(100)	YES		
system_memory	varchar(50)	YES		
used_memory	varchar(50)	YES		
free_memory	varchar(50)	YES		
no_access	int(50)	YES		
status	varchar(100)	YES		
usage_percent	Float	YES		
server_status	varchar(100)	YES		

6.5 Implementation Support

6.5.1 Installation of Eclipse

The following steps should be followed to install eclipse:

- *Installation of JVM*: Regardless of the operating system, some Java virtual machine (JVM) has to be installed. A Java Runtime Environment (JRE), or a Java Development Kit (JDK) can be installed depending on what is to be done with Eclipse. If Eclipse is intended for Java development, then a JDK (the JDK includes--among other useful things--the source code for the standard Java libraries)has to be installed.
- Download Eclipse from the Eclipse Downloads Page.
- The download will be delivered as a compressed (i.e. a ".zip", or ".tar.gz") file. Decompress this file into the directory of your choice (e.g. "c:\Program Files\Eclipse Indigo" on Windows). You can optionally create a shortcut of the executable file ("eclipse.exe" on Windows, or "eclipse" on Linux).

6.5.2 Installation of Apache Tomcat Server

The following steps should be followed to install Apache Tomcat in Eclipse:

- If Apache Tomcat is not present on the machine, it has to be downloaded and unzipped.
- Start the Eclipse WTP workbench.
- Open Window -> Preferences -> Server -> Installed Runtimesto create a Tomcat installed runtime.
- Click on Add to open the New Server Runtime dialog, then select your runtime under Apache (Apache Tomcat v7.0 in this project).
- Ensure the selected JRE is a full JDK and is of a version that will satisfy Apache Tomcat (this scenario was written using SUN JDK 1.6.029). If necessary, you can click on Installed JREs to add JDKs to Eclipse.
- Click Finish. Click Next, and fill in your Tomcat installation directory.

6.5.3 Installation of MySQL Database

- Log into the computer as an administrator
- Download the free MySQL Server Community Edition
- Double-click the downloaded file
- Double-click Setup.exe
- Click Next.
- Click Custom > Next
- Click Install
- Click Skip Sign-Up and then Next
- Configure MySQL.
- Click Next
- Check Standard Configuration and then click Next
- Make sure Install As Windows Service and Launch the MySQL Server Automatically are checked, then click Next.
- Create a root password. Type in what you want your root password to be and make sure Enable root access from remote machines is checked. Make sure you choose a difficult to guess password and 'write it down so you don't forget it. Click Next.
- Click Execute. This will start the MySQL server. After MySQL has done its thing, click Finish.
- From the Windows task bar, go to Start > All Programs > MySQL > MySQL Server 4.x > MySQL Command line client. This will open a command window asking you for a password.
- Enter your root password and hit Enter. This should initiate the program.

CHAPTER 7

CHAPTER 7

TESTING

7.1 Definition

Unit testing is a development procedure where programmers create tests as they develop software. The tests are simple short tests that test functionally of a particular unit or module of their code, such as a class or function.

Using open source libraries like cunit, cppunit and nunit (for C, C++ and C#) these tests can be automatically run and any problems found quickly. As the tests are developed in parallel with the source unit test demonstrates its correctness.

7.2 Validation and System Testing

Validation testing overlaps with integration testing. Ensuring that the application fulfils its specification is a major criterion for the construction of an integration test. Validation testing also overlaps to a large extent with ***System Testing***, where the application is tested with respect to its typical working environment. Consequently for many processes no clear division between validation and system testing can be made. Specific tests which can be performed in either or both stages include the following.

- **Regression Testing:** Where this version of the software is tested with the automated test harness used with previous versions to ensure that the required features of the previous version are still working in the new version.

- **Recovery Testing:** Where the software is deliberately interrupted in a number of ways, to ensure that the appropriate techniques for restoring any lost data will function.

- **Security Testing:** Where unauthorized attempts to operate the software, or parts of it, are attempted. It might also include attempts to obtain access to the data, or harm the software installation or even the system software.

- **Stress Testing:** Where abnormal demands are made upon the software by increasing the rate at which it is asked to accept, or the rate at which it is asked to produce information. More complex tests may attempt to create very large data sets or cause the software to make excessive demands on the operating system.

- **Performance testing:** Where the performance requirements, if any, are checked. These may include the size of the software when installed, type amount of main memory and/or secondary storage it requires and the demands made of the operating when running with normal limits or the response time.

- **Usability Testing:**Even if usability prototypes have been tested whilst the application was constructed, a validation test of the finished product will always be required.

- **Alpha and beta testing:** This is where the software is released to the actual end users. An initial release, the alpha release, might be made to selected users who be expected to report bugs and other detailed observations back to the production team. Once the application changes necessitated by the alpha phase can be made to larger more representative set of users, before the final release is made to all users.

  The final process should be a **Software audit** where the complete software project is checked to ensure that it meets production management requirements. This ensures that all required documentation has been produced, is in the correct format and is of acceptable quality. The purpose of this review is: firstly to assure the quality of the production process and by implication construction phase commences. A formal hand over from the development team at the end of the audit will mark the transition between the two phases.

7.2.1 Integration Testing:

Integration Testing can proceed in a number of different ways, which can be broadly characterized as top down or bottom up.

In top down integration testing the high level control routines are tested first, possibly with the middle level control structures present only as stubs. Top down testing can proceed in a **depth-first** or a **breadth-first** manner. For depth-first integration each module is tested in increasing detail, replacing more and more levels of detail with actual code rather than stubs. Alternatively breadth-first would processed by refining all the modules at the same level of control throughout the application ,in practice a combination of the two techniques would be used. At the initial stages all the modules might be only partly functional, possibly being implemented only to deal with non-erroneous data. These would be tested in breadth-first manner, but over a period of time each would be replaced with successive refinements which were closer to the full functionality. This allows depth-first testing of a module to be performed simultaneously with breadth-first testing of all the modules.

The other major category of integration testing is **Bottom Up Integration Testing** where an individual module is tested form a test harness. Once a set of individual module have been tested they are then combined into a collection of modules ,known as builds, which are then tested by a second test harness. This process can continue until the build consists of the entire application.

In practice a combination of top down and bottom-up testing would be used. In a large software project being developed by a number of sub-teams, or a smaller project where different modules were built by individuals. The sub teams or individuals would conduct bottom-up testing of the modules which they were constructing before releasing them to an integration team which would assemble them together for top-down testing.

7.2.2 Unit Testing:

Unit testing deals with testing a unit as a whole. This would test the interaction of many functions but confine the test within one unit. The exact scope of a unit is left to interpretation. Supporting test code, sometimes called ***Scaffolding***, may be necessary to support an individual test. This type of testing is driven by the architecture and implementation teams. This focus is also called black-box testing because only the details of the interface are visible to the test. Limits that are global to a unit are tested here.

In the construction industry, scaffolding is a temporary, easy to assemble and disassemble, frame placed around a building to facilitate the construction of the building. The construction workers first build the scaffolding and then the building. Later the scaffolding is removed, exposing the completed building. Similarly, in software testing, one particular test may need some supporting software. This software establishes can a correct evaluation of the test take place. The scaffolding software may establish state and values for data structures as well as providing dummy external functions for the test. Different scaffolding software may be needed form one test to another test. Scaffolding software rarely is considered part of the system.

Sometimes the scaffolding software becomes larger than the system software being tested. Usually the scaffolding software is not of the same quality as the system software and frequently is quite fragile. A small change in test may lead to much larger changes in the scaffolding.

Internal and unit testing can be automated with the help of coverage tools. Analyzes the source code and generated a test that will execute every alternative thread of execution. Typically, the coverage tool is used in a slightly different way. First the coverage tool is used to augment the source by placing information prints after each line of code. Then the testing suite is executed generating an audit trail. This audit trail is analyzed and reports the percent of the total system code executed during the test suite. If the coverage is high and the untested source lines are of low impact to the system's overall quality, then no more additional tests are required.

CHAPTER 8

CHAPTER 8

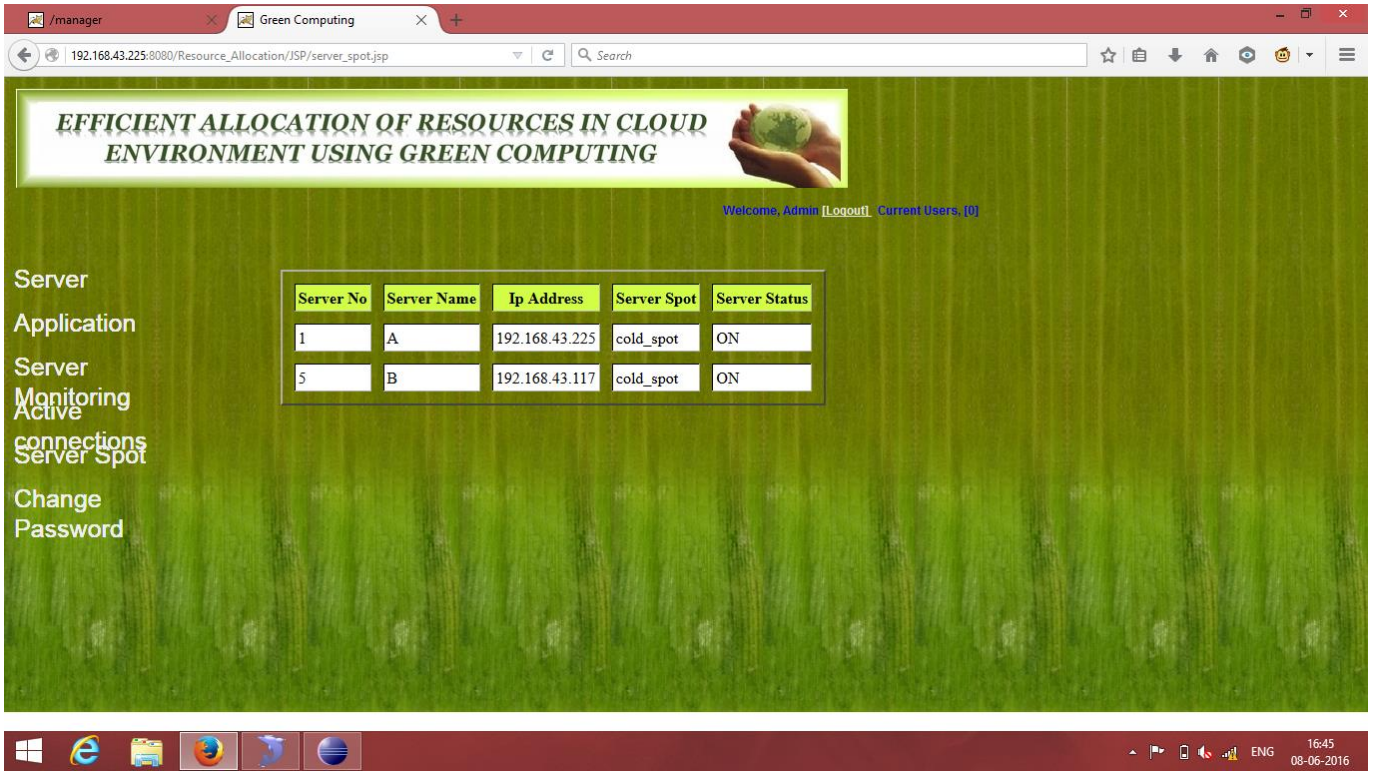
RESULTS AND ANALYSIS

The results show:

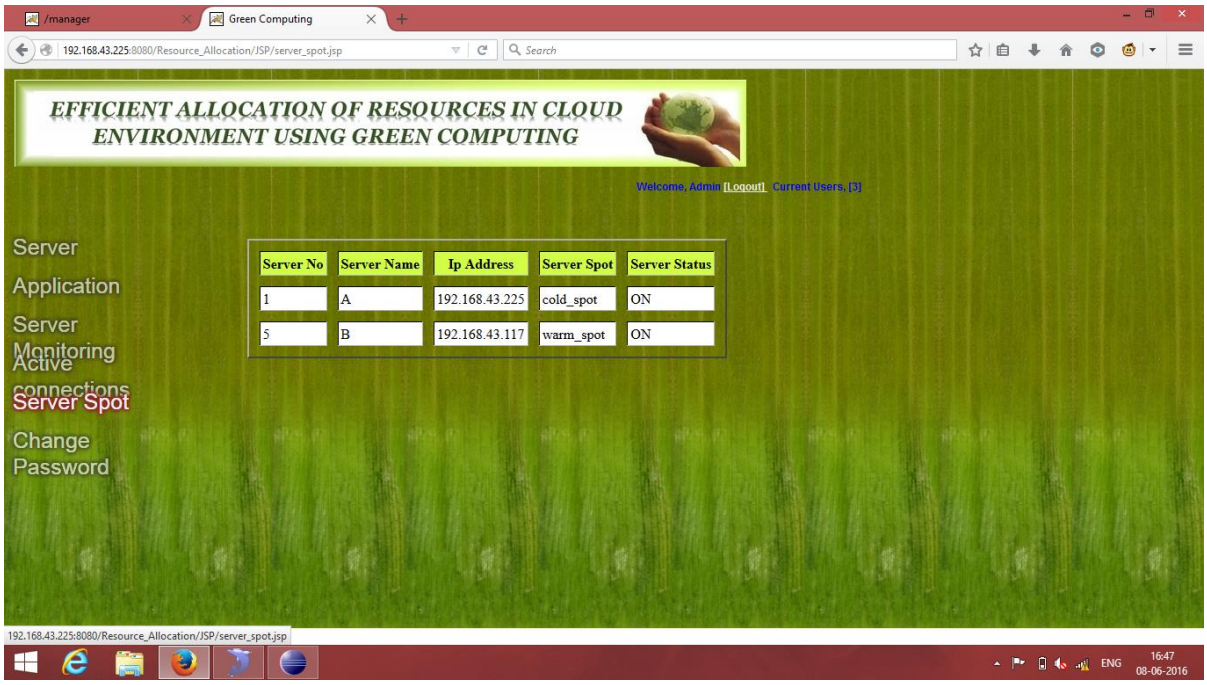
- The applications running on each server.
- The number of connections to each server.
- The states of each server.
- The migration process.

SCREENSHOTS

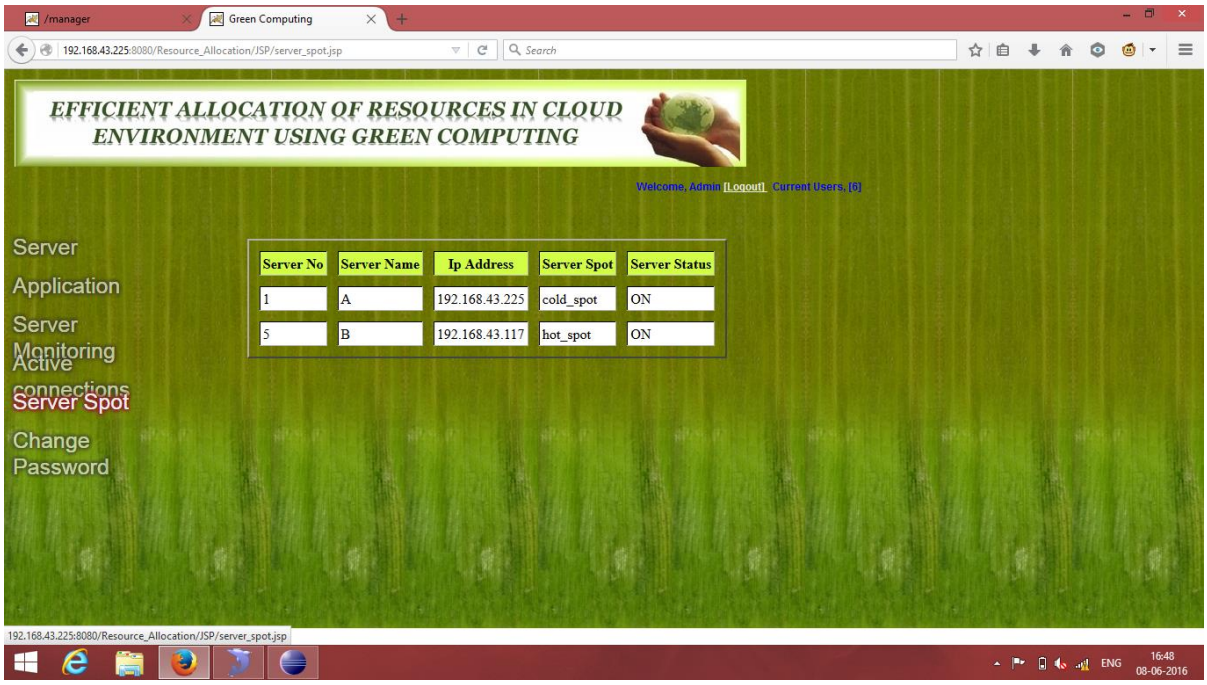
When both servers are on and no active connections



Change of state of server from cold spot to warm spot

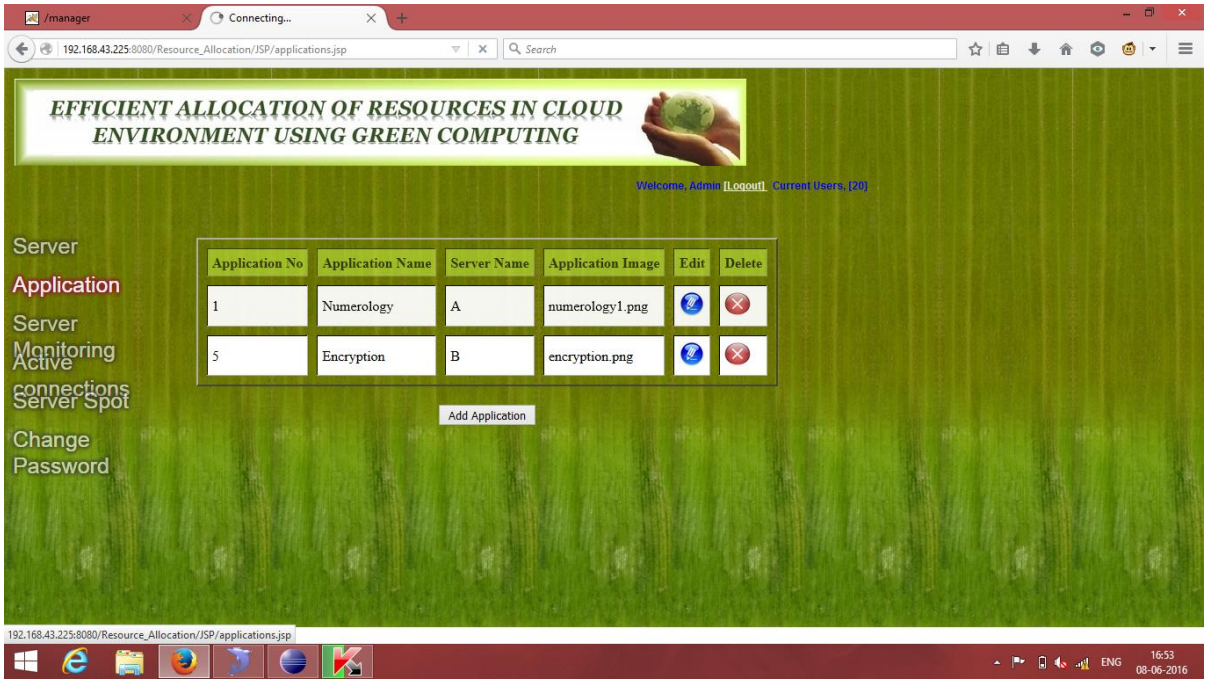


Change of state of server from warm spot to hotspot

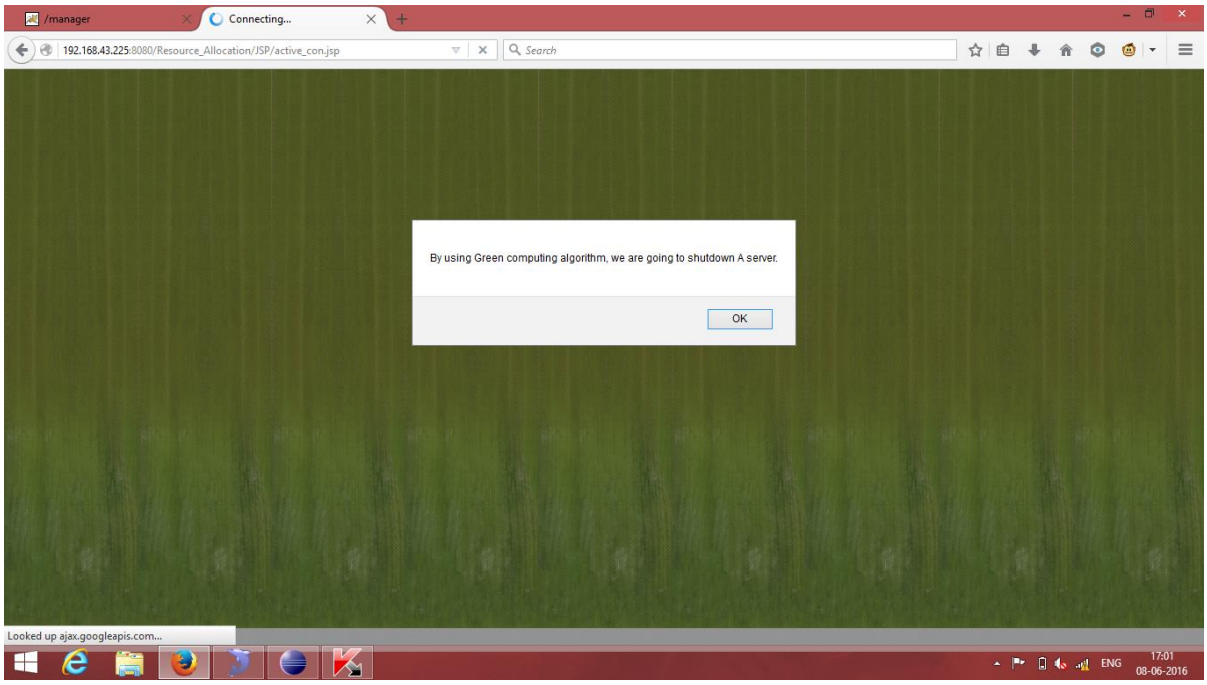


Server Monitoring

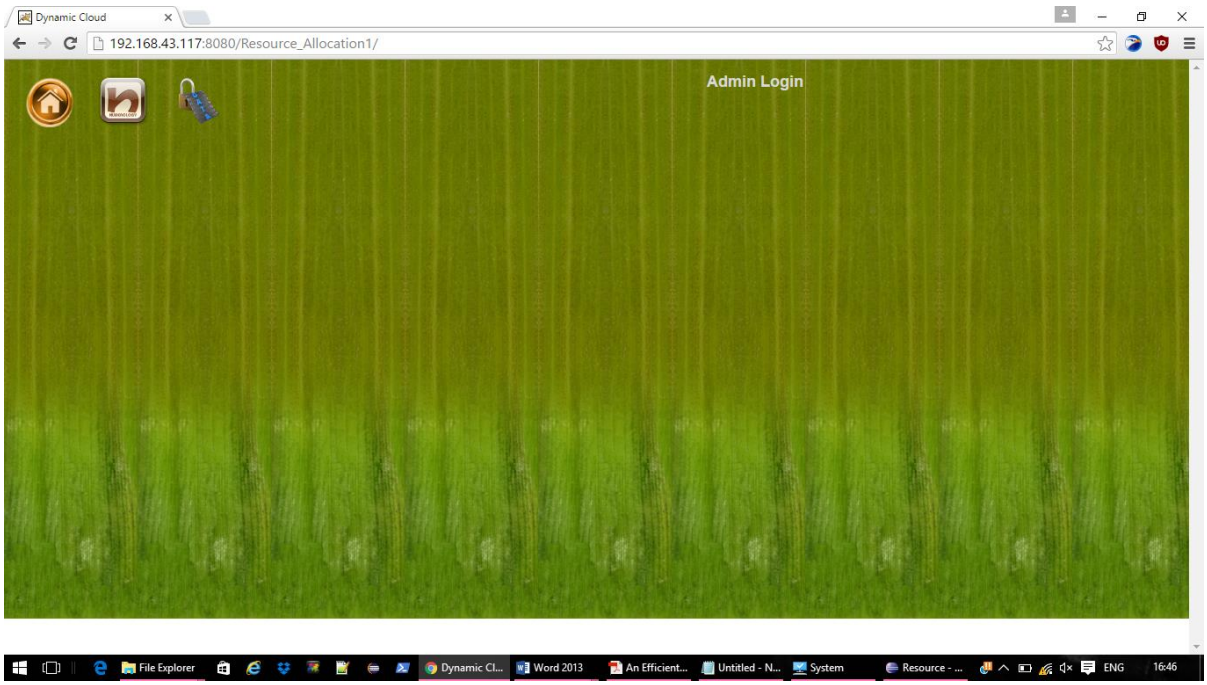
Application running on servers



Shutting of server A when there is no load



Client application



CHAPTER 9

CHAPTER 9

CONCLUSION

Over the past decade, green computing has proven itself to be the most concerning issue for the businesses and governments around the globe as computing becomes increasingly pervasive.

In simple words, green computing is the practice of reducing environmental footprints of technology by efficiently using the resources. Broadly, green computing includes:

· **Green Use –** Using resources in such a way that reduces the usage of hazardous materials.

· **Green Design –** Designing objects and services that comply with the environment.

· **Green Disposal –** Recycling e-waste with no or little impact on the environment.

· **Green Manufacturing –** The discovery and development of new products that reduces or eliminates the use or generation of hazardous substances in manufacturing.

Green cloud computing is a trend which has become popular with the emergence of internet driven services in every field of life. It refers to the prospective environmental advantages that computer based internet services can guarantee to the environment, by processing huge amount of data and information from collective resources pool. The cloud computing is emerged as an effective substitute to a traditional office based processing of services.

Final Words

Cloud computing is emerging as a critical information communication technology to heavily impact our daily life in the future. As computing becomes increasingly pervasive, the energy consumption attributable to computing is climbing that marked the foundation of Green Computing.

Green Cloud is an Internet Data Center architecture which aims to reduce data center power consumption, and at the same time guarantee the performance from users' perspective, leveraging live virtual machine migration technology.

Saving energy or reduction of carbon footprints is one of the aspects of Green Computing. Green Cloud Architecture enables comprehensive online-monitoring, live virtual machine migration and VM placement optimization.

A Green Cloud System responds to peak utilization periods and adjusts availability of resources based on them expanding or shrinking the cloud as needed. The aim of this paper is a literature study of the challenges and various issues in field of Green Computing and discusses the future scope of Green Clouds.

We have presented the design, implementation, and evaluation of a resource management system for cloud computing services. Our system multiplexes virtual to physical resources adaptively based on the changing demand. We use the skewness metric to combine VMs with different resource characteristics appropriately so that the capacities of servers are well utilized. Our algorithm achieves both overload avoidance and green computing for systems with multi-resource constraints.

CHAPTER 10

CHAPTER 10

FUTURE ENHANCEMENTS

As the prevalence of Cloud computing continues to rise, the need for power saving mechanisms within the Cloud also increases. This project presents a Green Cloud framework for improving system efficiency in a data center. To demonstrate the potential of framework, presented new energy efficient scheduling.

Through this project, we have found new ways to save vast amounts of energy while minimally impacting performance. Not only do the components in this project complement each other, they leave space for future work.

Future opportunities could explore a scheduling system that is both power-aware and thermal-aware to maximize energy savings both from physical servers and the cooling systems used. Such a scheduler would also drive the need for better data center designs, both in server placements within racks and closed-loop cooling systems integrated into each rack.

CHAPTER 11

CHAPTER 11

BIBLIOGRAPHY

9.1 Review through Books

[1] Cloud Computing: Concepts, Technology & Architecture (The Prentice Hall Service Technology Series from Thomas Erl) Hardcover – Import, 10 May 2013

[2] Diffie, W., and Hellman, M.E., New Directions in Cryptography, IEEE Transactions on Information Theory, vol. 22, no. 6, November 1976, pp.

[3] Garret, Paul. Making, Breaking Codes: An Introduction to Cryptology. Upper Saddle River, NJ: Prentice-Hall, 2001

[4] Hoffstein, Jeffery, Pipher, Jill and Silverman, Joseph H. NTRU: A Public Key Crypto http://grouper.ieee.org/groups/1363/lattPK/submissions.html#NTRU1

[5] Kurose, James F., Ross, Keith W., Computer Networking: A top Down Approach Featuring the Internet. 2nd edition. Addison Wesley 2002.

[6] Architecting the Cloud: Design Decisions for Cloud Computing Service Models (SaaS, PaaS, and IaaS) 1st Edition by Michael J Kavis (Author)

[7] Cloud Computing Protected: Security Assessment Handbook John Rhoton , 2013

[8] G. Wang and T. E. Ng, "The impact of virtualization on network performance of amazon ec2 data center," in Proc. IEEE Proc. INFOCOM, 2010, pp. 1–9.

[9] R. Ranjan, L. Zhao, X. Wu, A. Liu, A. Quiroz, and M. Parashar, "Peer-to-peer cloud provisioning: Service discovery and load-balancing," in Cloud Computing. London, U.K.: Springer, 2010, pp. 195–217.

[10] R. N. Calheiros, R. Ranjan, and R. Buyya, "Virtual machine provisioning based on analytical performance and QoS in cloud computing environments," in Proc. Int. Conf. Parallel Process., 2011, pp. 295–304.

[11] Server virtualization has stalled, despite the hype [Online]. Available: http://www.infoworld.com/print/146901, 2010.

[12] Y. C. Lee and A. Y. Zomaya, "Energy efficient utilization of resources in cloud computing systems," J. Supercomputers., vol. 60, no. 2, pp. 268–280, 2012.

APPENDIX

BASE PAPER