ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

Cost Minimization Algorithms for Data Center Management

[1]Gotiwada Kiran Mai, M.tech scholar, Department Computer Science and Engineering, Sarada Institute of Science, Technology and Management, Srikakulam 532404
[2]Mr. S.Kalyan, M. Tech., Asst. Professor, Department Computer Science and Engineering, Sarada Institute of Science, Technology and Management, Srikakulam 532404

ABSTRACT

Due to the increasing usage of cloud computing applications, it is important to minimize energy cost consumed by a data center, and simultaneously, to improve quality of service via data center management. One promising approach is to switch some servers in a data center to the idle mode for saving energy while to keep a suitable number of servers in the active mode for providing timely service. In this paper, we design both online and offline algorithms for this problem. For the offline algorithm, we formulate data center management as a cost minimization problem by considering energy cost, delay cost (to measure service quality), and switching cost (to change servers' active/idle mode). Then, we analyse certain properties of an optimal solution which lead to a dynamic programming based algorithm. Moreover, by revising the solution procedure, we successfully eliminate the recursive procedure and achieve an optimal offline algorithm with a polynomial complexity. For the online algorithm, we design it by considering the worst case scenario for future workload. In simulation, we show this online algorithm can always provide near-optimal solutions.

Key Words—Data center management, offline algorithm, dynamic programming, and online algorithm.

1. INTRODUCTION

Recent years, many researchers devote themselves into the area of data center administration. The goals of data center management may include minimalizing energy cost and improving quality of service. Energy cost is a major part of a data centre's budget which should be minimized to decrease service provider's cost, and more importantly, to keep our Earth green. One method to minimize energy cost is to switch some servers from active mode to idle mode whenever possible. These switching conclusions are made based on environment of the servers, such as network state or storage state. For the meantime, we want to achieve good service quality, which can be measured by the average delay of serves responding time. For this purpose, there should be enough active servers in order to process tasks originated by clients in time. To achieve both goals of data center management, we should maintain a suitable number of active servers and then distribute jobs to these active servers. Research efforts on data center management can be classified as inter-center or intra-center management.

Cite this Article as: Gotiwada Kiran Mai & S.Kalyan " Cost Minimization Algorithms for Data Center Management", International Journal & Magazine of Engineering, Technology, Management and Research (IJMETMR), ISSN 2348-4845, Volume 7 Issue 6, June 2020, Page 44 -58.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

Work load among servers within a data center. In this paper, we focus on intra-center management and consider dynamic workload over a period. In the minimization problem, the cost includes energy cost, delay cost and swapping cost. Energy cost at an active server can be modelled as a function of its assigned workload, or a function of the percentage of its processing capability used to finish the assigned workload. Delay cost at an active server can be used to measure the quality of service, which is also a function of assigned workload. We call the grouping of these two costs as the operating cost. Switching cost is the cost spent when changing a server from active mode to idle mode or reversely. In the cost minimization problem, we aim to optimally adjusting servers' status and transmitting workload to active servers.

2.LITERATURE SURVEY

Commentary : Cloud computing – A security problem or solution? The move to cloud computing is the next stage of an unstoppable trend in the breakdown of the enterprise perimeter, both technically and organizationally. This new paradigm presents a number of security challenges that still need to be resolved but sufficient change in the IT environment has already happened - so that most organizations are working in a transitional state where security exploits are happening across the enterprise boundary. In this situation, the compartmentalization introduced by migrating to cloud services could result in much improved security.

Modelling Strategic Relationships for Process Reengineering Existing models for describing a process such as a business process or a software development process tend to focus on the what or the show of the process

For example a health insurance claim process would typically be described in terms of several steps for assessing and approving a claim in trying to improve or redesign a process however one also needs to understand the way for example why do physicians submit treatment plans to insurance companies before giving treatment? and why do claims managers seek medical opinions when assessing treatment plans An understanding of the motivations and interests of process participants is often crucial to the successful redesign of processes.

3 REQUIREMENT ANALYSES

A Software Requirements Specification (SRS) is a complete description of the behavior of the system to be developed. It includes a set of use cases that describe all the interactions the users will have with the software.

Functional Requirements:

In software engineering, a functional requirement defines a function of a software system or its component. A function is described as a set of inputs, the behavior, and outputs. Functional requirements may be calculations, technical details, data manipulation and processing and other specific functionality that define what a system is supposed to accomplish. Behavioral requirements describing all the cases where the system uses the functional requirements are captured in use cases. Functional requirements are supported by non-functional requirements (also known as quality requirements), which impose constraints on

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

the design or implementation (such as performance requirements, security, or reliability)

Non-Functional Requirements

Non-functional requirements are often called qualities of a system. Other terms for non-functional requirements are "constraints", "quality attributes", "quality goals" and "quality of service requirements, “and "non-behavioral requirements. Qualities, that is, nonfunctional requirements, can be divided into two main categories: 1. Execution qualities, such as security and usability, which are observable at run time. 2. Evolution qualities, such as testability, maintainability, extensibility, and scalability, which are embodied in the static structure of the software system.

Hard ware and software Requirements

The following sub-sections discuss the various aspects of hardware requirements. Hardware Requirements for Present Project:

- System : Pentium IV 2.4 GHz.
- Hard Disk : 500 GB.
- Ram : 4 GB

Software Requirements for Present Project:

Operating system : Windows XP / 7

Coding Language : Java (Jdk 1.7)

Web Technology : Servlet, JSP

Web Server : Tomcat 6.0

IDE : Eclipse Galileo

Database : My-SQL 5.0

UGI for DB : SQLyog

JDBC Connection : Type 4 Driver

4 SYSTEM DESIGN

SYSTEM MODEL

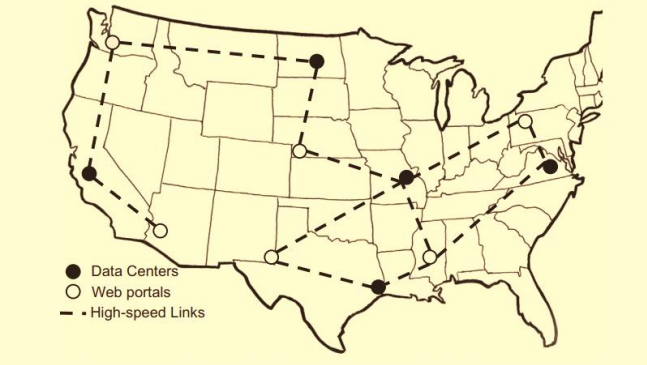

Figure 1 SYSTEM MODEL

An example of the distributed cloud is shown in Fig. 1. Each user sends its requests to its nearby web portal, and the requests at each web portal are then allocated to different data centers. Associated with each request, there is an average tolerant delay requirement which is the negotiated SLA between the user and the cloud service provider. Such a delay usually consists of the network latency incurred on the links between web-portals and data centers and the average delay waiting for being processed.

ELECTRICITY COST MODEL

The electricity cost of a data center DC during each time slot t is determined by the amount of power it consumed and the local electricity price during that time period. Let be the unit-energy electricity price at the location of DC and the total energy consumption of DC in the time duration of time slot t, if the load among

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

the servers at DC is well balanced. Thus, is proportional to the amount of energy consumption per request in data center DC. Let be the amount of average energy consumed per request with the average delay requirement DC, which is a constant.

DESIGN GOALS

In order to achieve practicality, both security and efficiency are considered in the proposed scheme. To be more specific, design goals of the proposed scheme are described as follows:

• Efficiency: Computational costs should be as low as possible at both the database owner side and the user side. To gain high efficiency, most biometric identification operations should be executed in the cloud.

• Security: During the identification process, the privacy of biometric data should be protected.

5. LANGUAGE SPECIFICATIONS

Java Technology

Java Architecture:

Java's architecture arises out of four distinct but interrelated technologies:

The Java programming language

The Java class file format

The Java Application Programming Interface

The Java virtual machine

When you write and run a Java program, you are tapping the power of these four technologies.

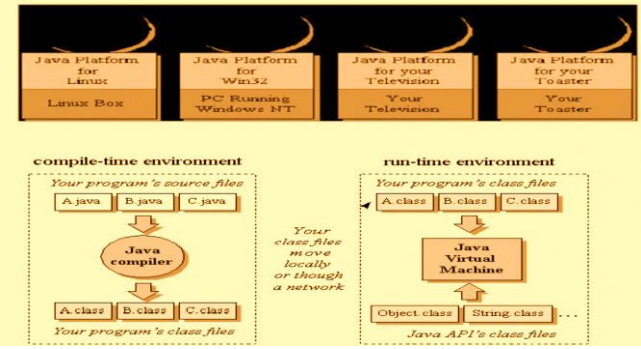

Figure 2 Java Architecture

Java Programming Environment

Together, the Java virtual machine and Java API form a "platform" for which all Java programs are compiled. In addition to being called the Java runtime system, the combination of the Java virtual machine and Java API is called the Java Platform (or, starting with version 1.2, the Java 2 Platform). Java programs can run on many kinds of computers because the Java Platform can itself be implemented in software. As you can see in Figure 1- 2, a Java program can run anywhere the Java Platform is present.

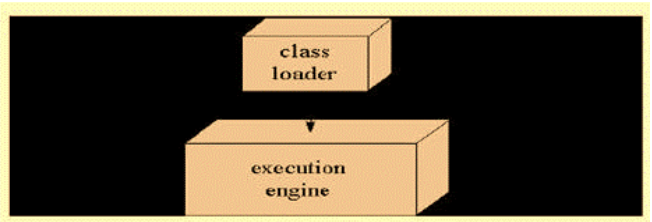

Figure 3 The Java Virtual Machine

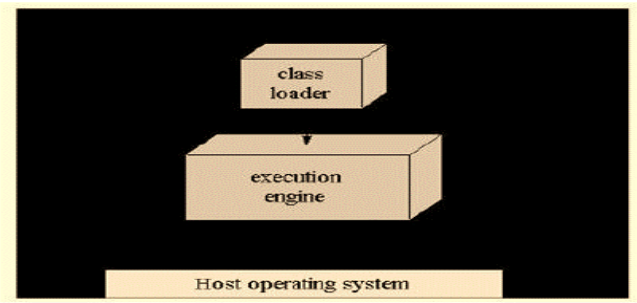

Figure 4 The Java Virtual Machine 2

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

Features of Java:

Platform Independence

The Write-Once-Run-Anywhere ideal has not been achieved platforms usually required, but closer than with other languages.

Object-Oriented: Object-oriented throughout - no coding outside of class definitions, including main ().

An extensive class library is available in the core language packages.

Compiler/Interpreter Combo:

Code is compiled to byte codes that are interpreted by Java virtual machines (JVM).

This provides portability to any machine for which a virtual machine has been written.

The two steps of compilation and interpretation allow for extensive code checking

and improved security.

Robust :

Exception handling built-in, strong type checking (that is, all data must be declared an explicit type), local variables must be initialized.

Security

No memory pointers

A program runs inside the virtual machine sandbox. Array index limit checking

Automatic Memory Management:

Automatic garbage collection

Memory management handled by JVM.

Limitations of C & C++ eliminated:

No memory pointers No preprocessor

Array index limit checking

6. SYSTEM IMPLEMENTATION

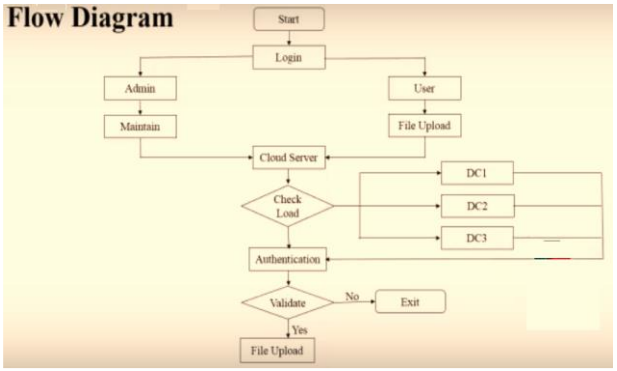

Figure 5 Flow Diagram

```
function AssignmentAlgorithm
    while VM arrives
        for each remote data center DC_i
            Request/retrieve values of C_i, E_i, P_i
        end for
        for each DC_i :
            if DC_i is not full, that is, U_i < U_Ti
                f^i_assign = (C_i − E_i) P_i
            else
                f^i_assign = ∞
            end if
        end for
        DC_target = DC_j such that f^j_assign = min{f^i_assign | i = 1 ··· N_DC}
        Assign VM to DC_target
    end while
end function
```

Figure 6 Cloud assignment algorithms

Java Code

```
/**
* A simple example showing how to create a datacenter with one host and run one
* cloudlet on it.
*/
public class pso_example2 {
static FileOutputStream out;
static PrintStream ps;
static ArrayList<Vm> vmlist;
/**
* Creates main() to run this example.
*
* @param args
* the args
* @throws IOException
*/
```

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

```
public static void main(String[] args) throws IOException {
try {
out = new FileOutputStream("Simulation Files/simulation_output.txt");
} catch (FileNotFoundException e) { // TODO Auto-generated catch block
e.printStackTrace();
}
ps = new PrintStream(out);
32
for (int i = 0; i < 100; i++) {
vmlist = new ArrayList<Vm>();
initSimulation();
System.out.println(i+1);
}
ps.close();
}
private static void initSimulation() {
Log.printLine("Starting CloudSim simulation Example using PSO...");
try {
// <<< [1]: Initialize the CloudSim package. It should be called
// before creating any entities. >>>
int num_user = 1; // number of cloud users
Calendar calendar = Calendar.getInstance(); // get calendar using
// current time zone
boolean trace_flag = false; // mean trace events
CloudSim.init(num_user, calendar, trace_flag); // Initialize the
// CloudSim library
// <<< [2]: Create Datacenters >>>
Datacenter datacenter1 = createDatacenter("Datacenter_1");
33
// <<< [3]: Create Cloud Broker and name it Broker1 >>>
DatacenterBroker broker = createBroker(1);
int brokerId = broker.getId(); // gets the id of the created broker
// <<< [4]: Create 5 virtual machines that uses time shared
// scheduling >>>
addVMs(4, brokerId, true, 6000);//0-3
addVMs(6, brokerId, true, 12000);//4-9
// <<< [5]: submit vm list to the broker >>>
broker.submitVmList(vmlist);
// <<< [6]: Read the workload file and create Cloudlets from it
List<Cloudlet> cloudletList = createCloudLets();
// <<< [7]: assign specific VMs to run specific cloudlets
// ----------------------------------------
broker.UsePSO();
for (Cloudlet cloudlet : cloudletList) {
// set all cloudlets to be managed by one broker.
cloudlet.setUserId(brokerId);
}
// ----------------------------------------------------------
----------
----------
// <<< [8]: submit cloudlet list to the broker >>>
broker.submitCloudletList(cloudletList);
34
// <<< [9]: Starts the simulation >>>
// start the simulation
CloudSim.startSimulation();
// stop the simulation
CloudSim.stopSimulation();
// <<< [10]: Print results when simulation is over
```

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

```
// retrieve all recieved cloudlet list
List<Cloudlet> newList = broker.getCloudletReceivedList();
// print the list
printCloudletList(newList);
// Print the debt of each user to each datacenter
// datacenter1.printDebts();
Log.printLine("CloudSimExample finished!");
} catch (Exception e) {
e.printStackTrace();
Log.printLine("Unwanted errors happen");
}
}
/**
* Creates the datacenter.
*
* desc: Datacenters are the resource providers in CloudSim. We need at
35
* least one of them to run a CloudSim simulation
*
* @param name
* the name of the data center
*
* @return the datacenter
*/
private static Datacenter createDatacenter(String name) {
// Here are the steps needed to create a PowerDatacenter:
// 1. We need to create a list to store our machine
List<Host> hostList = new ArrayList<Host>();
// 2. create hosts, where Every Machine contains one or more PEs or
// CPUs/Cores
// ((( Host 1 )))----------------------------
List<Pe> Host_1_peList = new ArrayList<Pe>();
// get the mips value of the selected processor
int Host_1_mips = Processors.Intel.Core_2_Extreme_X6800.mips;
// get processor's number of cores
int Host_1_cores = Processors.Intel.Core_2_Extreme_X6800.cores;
// 3. Create PEs and add these into a list.
for (int i = 0; i < Host_1_cores; i++) {
// mips/cores => MIPS value is cumulative for all cores so we divide
// the MIPS value among all of the cores
Host_1_peList.add(new Pe(i, new PeProvisionerSimple(Host_1_mips
36
/ Host_1_cores))); // need to store Pe id and MIPS
Rating
}
// 4. Create Host with its id and list of PEs and add them to the list
// of machines
int host_1_ID = 1;
int host_1_ram = 4096; // host memory (MB)
long host_1_storage = 1048576; // host storage in MBs
int host_1_bw = 10240; // bandwidth in MB/s
hostList.add(new Host(host_1_ID, new RamProvisionerSimple(host_1_ram),
new BwProvisionerSimple(host_1_bw), host_1_storage,
//Host_1_peList, new VmSchedulerTimeShared(Host_1_peList)));
Host_1_peList, new VmSchedulerTimeSharedOverSubscription(Host_1_peList)));
```

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

```
// ((( /Host 1 )))----------------------------
// ((( Host 2 )))----------------------------
List<Pe> Host_2_peList = new ArrayList<Pe>();
// get the mips value of the selected processor
int Host_2_mips = Processors.Intel.Core_2_Extreme_QX6700.mips;
// get processor's number of cores
int Host_2_cores = Processors.Intel.Core_2_Extreme_QX6700.cores;
// 3. Create PEs and add these into a list.
for (int i = 0; i < Host_2_cores; i++) {
37
// mips/cores => MIPS value is cumulative for all cores so we divide
// the MIPS value among all of the cores
Host_2_peList.add(new Pe(i, new PeProvisionerSimple(Host_2_mips
/ Host_2_cores))); // need to store Pe id and MIPS
Rating
}
// 4. Create Host with its id and list of PEs and add them to the list
// of machines
int host_2_id = 2;
int host_2_ram = 4096; // host memory (MB)
long host_2_storage = 1048576; // host storage in MBs
int host_2_bw = 10240; // bandwidth in MB/s
hostList.add(new Host(host_2_id, new RamProvisionerSimple(host_2_ram),
new BwProvisionerSimple(host_2_bw), host_2_storage,
//Host_2_peList, new VmSchedulerTimeShared(Host_2_peList)));
Host_2_peList, new VmSchedulerTimeSharedOverSubscription(Host_2_peList)));
// ((( /Host 2 )))----------------------------
// 5. Create a DatacenterCharacteristics object that stores the
// properties of a data center: architecture, OS, list of
// Machines, allocation policy: time- or space-shared, time zone
// and its price (G$/Pe time unit).
String arch = "x86"; // system architecture
String os = "Linux"; // operating system
String vmm = "Xen";
double time_zone = 10.0; // time zone this resource located
38
double cost = 3.0; // the cost of using processing in this resource
double costPerMem = 0.05; // the cost of using memory in this resource
double costPerStorage = 0.001; // the cost of using storage in this
// resource
double costPerBw = 0.0; // the cost of using bw in this resource
// we are not adding SAN devices by now
LinkedList<Storage> storageList = new LinkedList<Storage>();
DatacenterCharacteristics characteristics = new DatacenterCharacteristics(
arch, os, vmm, hostList, time_zone, cost, costPerMem,
costPerStorage, costPerBw);
// 6. Finally, we need to create a PowerDatacenter object.
Datacenter datacenter = null;
try {
datacenter = new Datacenter(name, characteristics,
```

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

```
new VmAllocationPolicySimple(hostList), storageList,
0);
} catch (Exception e) {
e.printStackTrace();
}
return datacenter;
}
/**
* Creates the broker.
*
* @param id
* : the broker id
39
*
* @return the datacenter broker
*/
private static Data center Broker create Broker(int id) {
DatacenterBroker broker = null;
try {
broker = new DatacenterBroker("Broker" + id);
} catch (Exception e) {
e.printStackTrace();
return null;
}
return broker;
}
/**
* Creates the virtual machines.
*
* @param VMNr
* : the number of virtual machines to create brokerId: the id of
* the broker created timeSharedScheduling: to choose between the
* time shared or space shared shceduling algorithms
*
* @return list of virtual machines
*
*/
private static void addVMs(int VMNr, int brokerId, boolean timeSharedScheduling,
int mips) {
// VM description
//int mips = Processors.Intel.Pentium_4_Extreme_Edition.mips;
//int mips = Processors.AMD.Athlon_FX_57.mips;
40
long size = 10240; // image size (MB)
int ram = 512; // vm memory (MB)
long bw = 1024; // MB/s
int pesNumber = 1; // number of cpus
String vmm = "Xen"; // VMM name
for (int i = 0; i < VMNr; i++) {
Vm vm;
int VM_ID = vmlist.size();
if (timeSharedScheduling) {
// create VM that uses time shared scheduling to schedule
Cloudlets
vm = new Vm(VM_ID, brokerId, mips, pesNumber, ram, bw,
size, vmm,
new CloudletSchedulerTimeShared());
}
else {
// create VM that uses space shared scheduling to schedule
Cloudlets
vm = new Vm(VM_ID, brokerId, mips, pesNumber, ram, bw,
size, vmm,
new CloudletSchedulerSpaceShared());
```

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

```
}
// add the VM to the vmList
vmlist.add(vm);
}
41
}
/**
* generate cloudlets from the workload file
*
* @return list of cloudlets
*
*/
private static List<Cloudlet>
createCloudLets()
throws FileNotFoundException {
/** The cloudlet list. */
List<Cloudlet> cloudletList;
// Read Cloudlets from workload file in the
swf format
WorkloadFileReader workloadFileReader =
new WorkloadFileReader(
"Simulation Files/HPC2N-2002-2.1-cln.swf",
1);
// generate cloudlets from workload file
cloudletList =
workloadFileReader.generateWorkload();
return cloudletList;
}
/**
* get all user ids in case we want to consider
the user id as a parameter
* and create a broker for every user
*
* @param cloudletList
*
* @return list of userIDs
42
*
*/
@SuppressWarnings("unused")
private static ArrayList<Integer>
getUsersIDs(List<Cloudlet> cloudletList) {
ArrayList<Integer> usersIDs = new
ArrayList<Integer>();
ArrayList<Integer> usersLists = new
ArrayList<Integer>();
for (Cloudlet cloudlet : cloudletList) {
usersLists.add(cloudlet.getUserId());
}
HashSet<Integer> uniqueValues = new
HashSet<Integer>(usersLists);
for (int value : uniqueValues) {
usersIDs.add(value);
}
return usersIDs;
}
/**
* Prints the Cloudlet objects.
*
* @param list
* list of Cloudlets
* @throws IOException
*/
private static void
printCloudletList(List<Cloudlet> list) throws
IOException {
int size = list.size();
Cloudlet cloudlet;
String indent = "\t";
Log.printLine();
Log.printLine("========== OUTPUT
==========");
Log.printLine("Cloudlet_ID" + indent +
"STATUS" + indent

+ "DataCenter_ID" + indent + "VM_ID" +
indent + "Time" +

indent
```

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

```
+ "Start_Time" + indent + "Finish_Time");

DecimalFormat dft = new DecimalFormat("###.##");

for (int i = 0; i < size; i++) {

cloudlet = list.get(i);

Log.print(cloudlet.getCloudletId() + indent);

if (cloudlet.getCloudletStatus() == Cloudlet.SUCCESS) {

Log.print("SUCCESS" + indent);

Log.printLine(cloudlet.getResourceId() + indent

+ cloudlet.getVmId() + indent

+ dft.format(cloudlet.getActualCPUTime()) + indent

+ dft.format(cloudlet.getExecStartTime()) + indent

+ dft.format(cloudlet.getFinishTime()));

}

}

ps.println(list.get(size-1).getFinishTime());

}

}
```

7 CHAPTER TESTING

Software Test Life Cycle

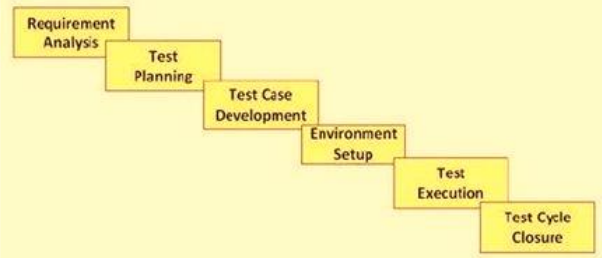

Figure 7 Testing Lifecycle

Test Environment Setup:

Test environment decides the software and hardware conditions under which a work product is tested. Test environment set-up is one of the critical aspects of testing process and can be done in parallel with Test Case Development Stage. Test team may not be involved in this activity if the customer team provides the test environment in which case the test team is required to do a readiness check (smoke testing) of the given environment.

Activities:

- Understand the required architecture, environment set-up and hardware and software requirement list for the Test Environment.
- Setup test Environment and test data
- Perform smoke test on the build

Test Execution

During this phase test team will carry out the testing based on the test plans and the test cases prepared. Bugs will be reported back to the development team for correction and retesting will be performed.

Activities:

- Execute tests as per plan
- Document test results, and log defects for failed cases Map defects to test cases in RTM
- Retest the defect fixes
- Track the defects to closure

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

White Box Testing:

White box testing is when the tester has access to the internal data structures and algorithms including the code that implement these.

 The following types of white box testing exist:

 API testing (application programming interface) - testing of the application using public and private APIs

 Code coverage - creating tests to satisfy some criteria of code coverage (e.g., the test designer can create tests to cause all statements in the program to be executed at least once)

 Fault injection methods - improving the coverage of a test by introducing faults to test code paths

 Mutation testing methods

 Static testing - White box testing includes all static testing

Black box testing:

Black box testing treats the software as a "black box"—without any knowledge of internal implementation. Black box testing methods include: equivalence partitioning, boundary value analysis, all-pairs testing, fuzz testing, model-based testing, traceability matrix, exploratory testing and specification-based testing.

Grey box testing:

Grey box testing (American spelling: gray box testing) involves having knowledge of internal data structures and algorithms for purposes of designing the test cases, but testing at the user, or black-box level. Manipulating input data and formatting output do not qualify as greybox, because the input and output are clearly outside of the "black-box" that we are calling the system under test. This distinction is particularly important when conducting integration testing between two modules of code written by two different developers, where only the interfaces are exposed for test. However, modifying a data repository does qualify as grey box, as the user would not normally be able to change the data outside of the system under test. Grey box testing may also include reverse engineering to determine, for instance, boundary values or error.

8. RESULTS

Login Portal and adding User and adding files to DATA CENTER:

Steps to adding user to Data Center:

Step:1

Figure 8 Adding user to Data Center Step 1

Step:2

Figure 9 Adding user to Data Center Step 2

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

Step:3

Figure 10 Adding user to Data Center Step 3

Steps to adding files to DATA CENTER

Step:1

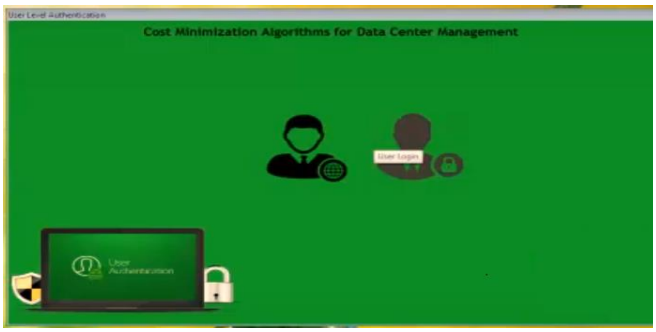

Figure 11 Adding files to DATA CENTER Step 1

Step:2

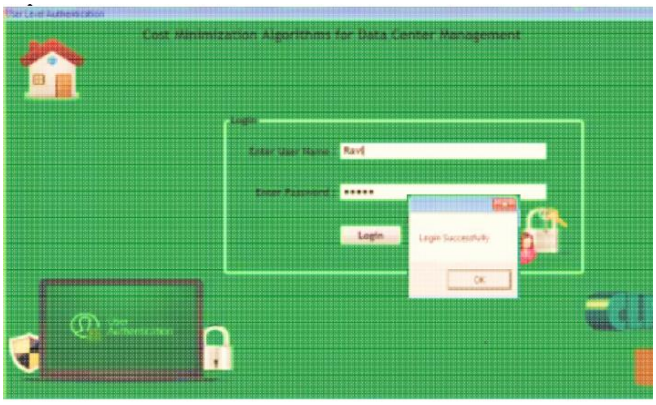

Figure 12:Adding files to DATA CENTER Step 2

Step:3

Figure 13 Adding files to DATA CENTER Step 3

Step:4

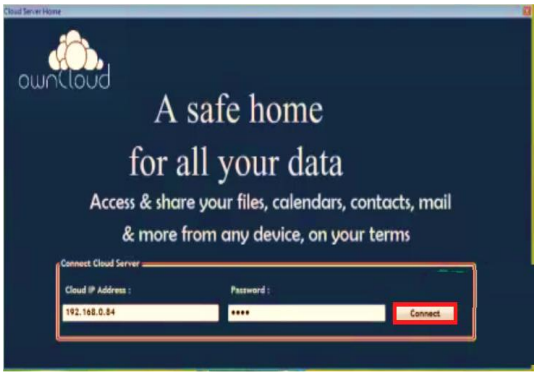

Figure 14 Adding files to DATA CENTER Step 4

Step:5

Figure 15 Adding files to DATA CENTER Step 5

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

Step:6

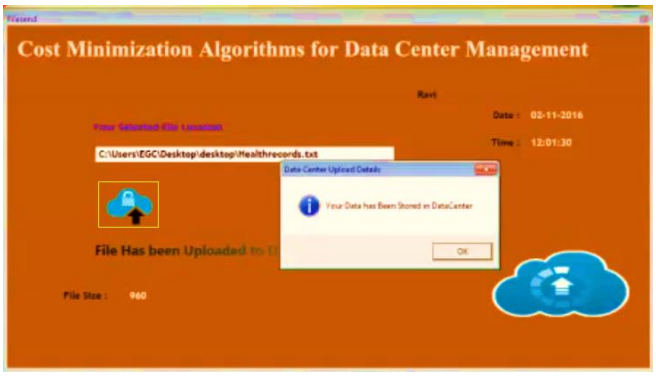

Figure 16 Adding files to DATA CENTER Step 6

Step:7

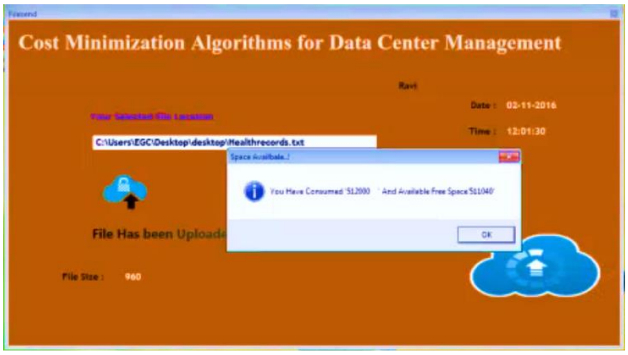

Figure 17 Adding files to DATA CENTER Step 7

9. CONCLUSION

In this paper, we studied the operational cost minimization problem in a distributed cloud computing environment that not only provides fair rate allocations among web portals but also meets multi-level user SLA requirements, by exploiting time varying electricity prices and user request rates, for which we first proposed an adaptive operational cost optimization framework. We then devised fast, scalable approximation algorithm with a provable approximation ratio for the problem. We finally conducted extensive experiments by simulations to evaluate the performance of the proposed algorithm, using real-life electricity price data traces. Experimental results demonstrate that the proposed algorithm is very promising, and the solution obtained is fractional of the optimum.

10. References

1. Beloglazov, R. Buyya, Y. C. Lee, A. Zomaya, "A taxonomy and survey of energy-efficient data centers and cloud computing systems", Elsevier Adv. Comput., vol. 82, pp. 47-111, 2011.
2. Gandhi, V. Gupta, M. Harchol-Balter, A. Kozuch, "Optimality analysis of energy-performance trade-off for server farm management", Elsevier Perform. Eval., vol. 67, pp. 1155-1171, Nov. 2010.
3. L. A. Barroso, U. Hölzle, "The case for energy-proportional computing", IEEE Comput., vol. 40, no. 12, pp. 33-37, Dec. 2007.
4. G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, F. Zhao, "Energy-aware server provisioning and load dispatching for connection-intensive internet services", Proc. USENIX NSDI, vol. 8, pp. 337-350, Apr. 2008.
5. H. Amur, J. Cipar, V. Gupta, G. R. Ganger, M. A. Kozuch, K. Schwan, "Robust and Flexible power-proportional storage", Proc. 1st ACM Symp. Cloud Comput., pp. 217-228, 2010. Access at ACM
6. M. R. Garey, D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-completeness, New York, NY, USA:Freeman, 1979. 56
7. B. Guenter, N. Jain, C. Williams, "Managing cost performance and

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

reliability tradeoffs for energy-aware server provisioning", Proc. IEEE INFOCOM, pp. 1332-1340, Apr. 2011.

8. Y. Guo, Y. Fang, "Electricity cost saving strategy in data centers by using energy storage", IEEE Trans. Parallel Distrib. Syst., vol. 24, no. 6, pp. 1149-1160.
9. J. Li, K. Shuang, S. Su, Q. Huang, P. Xu, X. Cheng, J. Wang, "Reducing operational costs through consolidation with resource prediction in the cloud", Proc. IEEE/ACM CCGrid, pp. 793-798, May 2012.