**A New Service Mechanism for Profit Optimizations of a Cloud Provider and its Users**

**ABSTRACT**

In this paper, we try to design a service mechanism for profit optimizations of both a cloud provider and its multiple users. We consider the problem from a game theoretic perspective and characterize the relationship between the cloud provider and its multiple users as a Stackelberg game, in which the strategies of all users are subject to that of the cloud provider. The cloud provider tries to select and provision appropriate servers and configure a proper request allocation strategy to reduce energy cost while satisfying its cloud users at the same time. We approximate its servers selection space by adding a controlling parameter and configure an optimal request allocation strategy. For each user, we design a utility function which combines the net profit with time efficiency and try to maximize its value under the strategy of the cloud provider. We formulate the competitions among all users as a generalized Nash equilibrium problem (GNEP). We solve the problem by employing variational inequality (VI) theory and prove that there exists a generalized Nash equilibrium solution set for the formulated GNEP. Finally, we propose an iterative algorithm (IA), which characterizes the whole process of our proposed service mechanism. We conduct some numerical calculations to verify our theoretical analyses. The experimental results show that our IA algorithm can benefit both of a cloud provider and its multiple users by configuring proper strategies.

**LIST OF ABBREVATION**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **ABBREVATION** | **EXPANSION** |
| 1**.** | DB | Data Base |
| 2. | JVM | Java Virtual Machine |
| 3. | JSP | Java Server Page |
| 4. | CB | Collective Behavior |
| 5. | RSSS | Ramp secret sharing scheme |
| 6. | JRE | Java Runtime Environment |

**CHAPTER 1**

**INTRODUCTION**

**1.1 GENERAL**

Cloud computing is an increasingly popular paradigm of offering subscription-oriented services to enterprises and consumers [1]. Usually, the provided services refer to Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS), which are all made available to the general public in a pay-as-you-go manner [2], [3]. To support various services, more and more cloud centers are equipped with thousands of computing nodes, which results in tremendous energy cost [4]. It is reported that about 50% management budget of Amazon′s data center is used for powering and colling the physical servers [5]. There are also researchers who have studied the cost of data centers and concluded that around 40% of the amortized cost of a data center falls into power related categories [6]. Hence, it is important to reduce energy cost for improving the profit of a cloud provider. However, it can often be seen that there are many under-utilized servers in cloud centers, or on the contrary, cloud providers provide less processing capacity and thus dissatisfy their users for poor service quality. Therefore, it is important for a cloud provider to select appropriate servers to provide services, such that it reduces cost as much as possible while satisfying its users at the same time.

For a cloud provider, the income (i.e., the revenue) is the service charge to the aggregated requests from all cloud users [7]. When the per request charge is determined, servers selection and request allocation strategy are two significant factors that should be taken into account. The reason behind lies in that both of them are not just for the profit of a cloud provider, but for the appeals to more cloud users in the market to use cloud service and thus also impact the profit.

Specifically, if the provided computing capacity is large enough (i.e., many servers are under-utilized), this will result in tremendous amount of energy waste with huge cost and thus reduces the profit of the cloud provider. On the other hand, if the cloud provider provides less computing capacity or improperly configures the request allocation strategy, this will lead to low service quality (e.g, long task response time) and thus dissatisfies its cloud users or potential cloud users in the market.

A rational user will choose a strategy to use the service that maximizes his/her own net reward, i.e., the utility obtained by choosing the cloud service minus the payment [8]. In addition, the utility of a user is not only determined by the net profit of his/her requests (i.e., how much benefit the user can receive by finishing the configured tasks), but also closely related to the urgency of the tasks (i.e., how quickly they can be finished).

The same amount of tasks are able to generate more utility for a cloud user if they can be completed within a shorter period of time in the cloud center [8]. However, considering from energy saving and economic reasons, it is irrational for a cloud provider to provide enough period of time. Therefore, multiple cloud users have to configure the amount of requests in different time slots.

Since the requests from users are submitted randomly, in our paper, we approximately characterize the request arrivals as a Poisson process [9]. Since the payment and time efficiency of each of the cloud users are affected by the decisions of others, it is natural to analyze the behaviors of these users as strategic games [10].

In this paper, we try to design a new service mechanism for profit optimizations of both a cloud provider and its multiple users. We consider the problem from a game theoretic perspective and characterize the relationship between the cloud provider and its users as a Stackelberg game, in which the strategies of all users are subject to that of the cloud provider. In our mechanism, the cloud provider tries to select appropriate servers and configure a proper request allocation strategy to reduce energy cost while satisfying its users at the same time.

The main contributions of this paper are listed as follows.

* We characterize the relationship between the cloud provider and its users as a Stackelberg game, and try to optimize the profits of both a cloud provider and its users at the same time.
* We formulate the competitions among all users as a generalized Nash equilibrium problem (GNEP), and prove that there exists a generalized Nash equilibrium solution set for the formulated GNEP.
* We solve the GNEP by employing variational inequality (VI) theory and propose an iterative algorithm (IA) to characterize the whole process of our proposed service mechanism.
* Experimental results show that our IA algorithm can benefit both of the cloud provider and its multiple users by configuring proper strategies.

**1.2 SCOPE OF THE PROJECT**

Finally, we propose an iterative algorithm (IA), which characterizes the whole process of our proposed service mechanism. We conduct some numerical calculations to verify our theoretical analyses. The experimental results show that our IA algorithm can benefit both of a cloud provider and its multiple users by configuring proper strategies.

**1.3 OBJECTIVE**

In this paper, we try to design a new service mechanism for profit optimizations of both a cloud provider and its multiple users. We consider the problem from a game theoretic perspective and characterize the relationship between the cloud provider and its users as a Stackelberg game, in which the strategies of all users are subject to that of the cloud provider. In our mechanism, the cloud provider tries to select appropriate servers and configure a proper request allocation strategy to reduce energy cost while satisfying its users at the same time.

**1.4 PROBLEM STATEMENT**

The authors try to reduce energy saving under continuous dynamic voltage frequency scaling (DVFS) environment. Specifically, they try to configure appropriate speed for each server to save energy. However, as shown in Table 1, all these methods mainly consider from the perspective of the cloud provider. To our knowledge, hardly any previous works investigate multiple users′ profit optimizations, let alone optimizing the profits of a cloud provider and its users at the same time. In this work, we first try to optimize multiple users′ profits. Since multiple cloud users compete for using the resources of a cloud provider, and the utility of each user is affected by the decisions (service request strategies) of other users, it is natural to analyze the behaviors of such systems as strategic games. In our mechanism, the cloud provider tries to select appropriate servers and configure a proper request allocation strategy to reduce energy cost while satisfying its users at the same time.

* 1. **EXISTING SYSTEM**
* To our knowledge, hardly any previous works investigate multiple users*′*profit optimizations, let alone optimizing the profits of a cloud provider and its users at the same time.
* The scheduling algorithm for sporadic tasks. The authors try to reduce energy consumption by using dynamic voltage frequency scaling (DVFS) technique.
* In existing system, based on DVFS technique and the concept of slack sharing among processors, the authors also proposed two novel energy-aware scheduling algorithms.

**1.5.1 EXISTING SYSTEM DISADVANTAGES**

* Since multiple users will try to access the data application performance depends upon the user’s data requests.
* The existing system unable to avoid the server energy cost.

**1.5.2 LITERATURE SURVEY**

**Title:** Strategy configurations of multiple users competition for cloud service reservation

**Year:** 2016

**Author:** C. Liu, K. Li, C. Xu, and K. Li

**Description:** In this paper, we focus on strategy configurations of multiple users to make cloud service reservation. We consider the problem from a game theoretic perspective and formulate it into a non-cooperative game among the multiple cloud users, in which each user is informed with incomplete information of other users. For each user, we design a utility function which combines the net profit with time efficiency and try to maximize its value. We solve the problem by employing variational inequality (VI) theory and prove that there exists a Nash equilibrium solution set for the formulated game. Then, we propose an iterative proximal algorithm (IPA), which is designed to compute a Nash equilibrium solution. The convergence of the IPA algorithm is also analyzed and we find that it converges to a Nash equilibrium if several conditions are satisfied. Finally, we conduct some numerical calculations to verify our theoretical analysis. The experimental results show that our proposed IPA algorithm converges to a stable state very quickly and improves the utilities of all users to certain extent by configuring a proper request strategy.

**Title:** A framework of price bidding configurations for resource usage in cloud computing

**Year:** 2016

**Author:** K. Li, C. Liu, K. Li, and A. Y. Zomaya.

**Description:** In this paper, we focus on price bidding strategies of multiple users competition for resource usage in cloud computing. We consider the problem from a game theoretic perspective and formulate it into a non-cooperative game among the multiple cloud users, in which each cloud user is informed with incomplete information of other users. For each user, we design a utility function which combines the net profit with time efficiency and try to maximize its value. We design a mechanism for the multiple users to evaluate their utilities and decide whether to use the cloud service. Furthermore, we propose a framework for each cloud user to compute an appropriate bidding price. At the beginning, by relaxing the condition that the allocated number of servers can be fractional, we prove the existence of Nash equilibrium solution set for the formulated game. Then, we propose an iterative algorithm (IA), which is designed to compute a Nash equilibrium solution. The convergency of the proposed algorithm is also analyzed and we find that it converges to a Nash equilibrium if several conditions are satisfied. Finally, we revise the obtained solution and propose a near-equilibrium price bidding algorithm (NPBA) to characterize the whole process of our proposed framework. The experimental results show that the obtained near-equilibrium solution is close to the equilibrium one.

**Title:** Optimizing cloud data center energy efficiency via dynamic prediction of cpu idle intervals

**Year:** 2015

**Author:** L. Duan, D. Zhan, and J. Hohnerlein

**Description:** The energy consumption of cloud data centers has been growing drastically in recent years. In particular, CPUs are the most power hungry components in the data center. On the one hand, CPUs are not energy proportional with respect to their utilization levels because a cloud server's energy efficiency is much lower with limited CPU utilizations. On the other hand, current cloud computing applications usually exhibit significant CPU idle time composed of idle intervals of variable lengths. The power consumption in these idle intervals is significant due to the prominent leakage current in recent technology nodes. There are a few existing schemes that transition a CPU into various low-power and sleep states to reduce its idle power. But none of them is optimal due to the fact that entering a sleep state may result in negative power savings if its wake-up latency is longer than the current idle interval. Therefore, intelligent sleep state entry is a key challenge in improving data centers' CPU energy efficiency. In this work, we propose a dynamic idle interval prediction scheme that can estimate future CPU idle interval lengths and thereby choose the most cost-effective sleep state to minimize power consumption at runtime. Experiments show that our proposed approach can significantly outperform other schemes, achieving 10% - 50% power savings compared to DVFS for a variety of CPU idle patterns. Of short and variable idle intervals. The power consumption in these idle intervals is significant due to leakage power being prominent in recent technologies. Therefore, we study a number of schemes that transition the CPU into various low power and sleep states to reduce the CPU idle power. Entering a sleep state may result in negative power savings if its wakeup latency is longer than the current idle interval. Therefore, intelligent sleep state entry is a key challenge in improving data center CPU energy usage. In this work, we propose a dynamic idle interval prediction scheme.

**Title:** Cost and energy aware scheduling algorithm for scientific workflows with deadline constraint in clouds

**Year:** 2015

**Author:** Z. Li, J. Ge, H. Hu, W. Song, H. Hu, and B. Luo

**Description:** Cloud computing is a suitable platform to execute the deadline-constrained scientific workflows which are typical big data applications and often require many hours to finish. Moreover, the problem of energy consumption has become one of the major concerns in clouds. In this paper, we present a cost and energy aware scheduling (CEAS) algorithm for cloud scheduler to minimize the execution cost of workflow and reduce the energy consumption while meeting the deadline constraint. The CEAS algorithm consists of five sub-algorithms. First, we use the VM selection algorithm which applies the concept of cost utility to map tasks to their optimal virtual machine (VM) types by the sub-makespan constraint. Then, two tasks merging methods are employed to reduce execution cost and energy consumption of workflow. Further, In order to reuse the idle VM instances which have been leased, the VM reuse policy is also proposed. Finally, the scheme of slack time reclamation is utilized to save energy of leased VM instances. According to the time complexity analysis, we conclude that the time complexity of each sub-algorithm is polynomial. The CEAS algorithm is evaluated using Cloudsim and four real-world scientific workflow applications, which demonstrates that it outperforms the related well-known approaches.

**Title:** Power-aware optimization for heterogeneous multi-tier clusters

**Year:** 2015

**Author:** P. Wang, Y. Qi, and X. Liu

**Description:**

Complex web applications are usually served by multi-tier web clusters. With the growing cost of energy, the importance of reducing power consumption in server systems is now well-known and has become a major research topic. However, most existing research focused solely on homogeneous clusters. This paper addresses the challenge of power management in Heterogeneous Multi-tier Web Clusters. We apply Generalized Benders Decomposition (GBD) to decompose the global optimization problem into small sub-problems. This algorithm achieves the optimal solution in an iterative fashion. The evaluation results show that our algorithm achieves more energy conservation than the previous work.

**Title:** A mechanism design approach to resource procurement in cloud computing

**Year:** 2014

**Author:** A. Prasad and S. Rao

**Description:** We present a cloud resource procurement approach which not only automates the selection of an appropriate cloud vendor but also implements dynamic pricing. Three possible mechanisms are suggested for cloud resource procurement: cloud-dominant strategy incentive compatible (C-DSIC), cloud-Bayesian incentive compatible (C-BIC), and cloud optimal (C-OPT). C-DSIC is dominant strategy incentive compatible, based on the VCG mechanism, and is a low-bid Vickrey auction. C-BIC is Bayesian incentive compatible, which achieves budget balance. C-BIC does not satisfy individual rationality. In C-DSIC and C-BIC, the cloud vendor who charges the lowest cost per unit QoS is declared the winner. In C-OPT, the cloud vendor with the least virtual cost is declared the winner. C-OPT overcomes the limitations of both C-DSIC and C-BIC. C-OPT is not only Bayesian incentive compatible, but also individually rational. Our experiments indicate that the resource procurement cost decreases with increase in number of cloud vendors irrespective of the mechanisms. We also propose a procurement module for a cloud broker which can implement C-DSIC, C-BIC, or C--OPT to perform resource procurement in a cloud computing context. A cloud broker with such a procurement module enables users to automate the choice of a cloud vendor among many with diverse offerings, and is also an essential first step toward implementing dynamic pricing in the cloud.

The text-based password is still believed to remain a

dominating and irreplaceable authentication method in the

foreseeable future. Although researchers have proposed dif-

ferent authentication mechanisms, no alternative can bring all

the beneﬁts of passwords without introducing extra burdens

to users [3]. However, passwords have long been criticized

as being one of the weakest links in authentication. Due

to the human memorability limitation, user passwords are

usually far from truly random [2], [28], [33], [42], [46]. For

instance, “secret” is more likely a human-chosen password

than “zjorqpe. In other words, human users are prone to choose

weak passwords simply because they are easier to remember.

As a result, most passwords are chosen within a small portion

of the entire password space, leaving them vulnerable to brute-

force or dictionary attacks.

The text-based password is still believed to remain a

dominating and irreplaceable authentication method in the

foreseeable future. Although researchers have proposed dif-

ferent authentication mechanisms, no alternative can bring all

the beneﬁts of passwords without introducing extra burdens

to users [3]. However, passwords have long been criticized

as being one of the weakest links in authentication. Due

to the human memorability limitation, user passwords are

usually far from truly random [2], [28], [33], [42], [46]. For

instance, “secret” is more likely a human-chosen password

than “zjorqpe. In other words, human users are prone to choose

weak passwords simply because they are easier to remember.

As a result, most passwords are chosen within a small portion

of the entire password space, leaving them vulnerable to brute-

force or dictionary attacks.

The text-based password is still believed to remain a

dominating and irreplaceable authentication method in the

foreseeable future. Although researchers have proposed dif-

ferent authentication mechanisms, no alternative can bring all

the beneﬁts of passwords without introducing extra burdens

to users [3]. However, passwords have long been criticized

as being one of the weakest links in authentication. Due

to the human memorability limitation, user passwords are

usually far from truly random [2], [28], [33], [42], [46]. For

instance, “secret” is more likely a human-chosen password

than “zjorqpe. In other words, human users are prone to choose

weak passwords simply because they are easier to remember.

As a result, most passwords are chosen within a small portion

of the entire password space, leaving them vulnerable to brute-

force or dictionary attacks

The text-based password is still believed to remain a

dominating and irreplaceable authentication method in the

foreseeable future. Although researchers have proposed dif-

ferent authentication mechanisms, no alternative can bring all

the beneﬁts of passwords without introducing extra burdens

to users [3]. However, passwords have long been criticized

as being one of the weakest links in authentication. Due

to the human memorability limitation, user passwords are

usually far from truly random [2], [28], [33], [42], [46]. For

instance, “secret” is more likely a human-chosen password

than “zjorqpe. In other words, human users are prone to choose

weak passwords simply because they are easier to remember.

As a result, most passwords are chosen within a small portion

of the entire password space, leaving them vulnerable to brute-

force or dictionary attacks

* 1. **PROPOSED SYSTEM**
* In this paper, we try to design a new service mechanism for profit optimizations of both a cloud provider and its multiple users.
* We consider the problem from a game theoretic perspective and characterize the relationship between the cloud provider and its users as a Stackelberg game, in which the strategies of all users are subject to that of the cloud provider.
* In our mechanism, the cloud provider tries to select appropriate servers and configure a proper request allocation strategy to reduce energy cost while satisfying its users at the same time.

**1.6.1 PROPOSED SYSTEM ADVANTAGES**

* Cost effectiveness will be provided.
* Application performance will be improved.
* In this work, we first try to optimize multiple users′ profits

**CHAPTER 2**

**PROJECT DESCRIPTION**

**2.1 GENERAL**

However, according to, even an energy efficient server still consumes about half of its full power when doing no work. Therefore, powering off idle servers when possible is regarded as an effective way to reduce energy cost, especially during off-peak traffic hours for a relative long period of time. In this work, we try to power off some idle servers (i.e., select appropriate servers to provide services) to reduce energy cost for the cloud provider. In addition, we configure server selection strategy for multiple time slots, i.e., for a relative long period of time.

**2.2 METHODOLOGIES**

**2.2.1** **MODULES NAME:**

**This project having the following 5 modules:**

**1. User Interface.**

**2. Admin.**

**3. Data Center.**

**4. Customer (User).**

**5. Data Allocation and Resource Reservation (DAR)**

**2.2.2** **MODULES EXPLANATION AND DIAGRAM**

* **User Interface Design**

In this module we design the windows for the project. In this module mainly we are focusing the login design page with the Partial knowledge information. Application Users need to view the application they need to login through the User Interface GUI is the media to connect User and Media Database and login screen where user can input his/her user name, password and password will check in database, if that will be a valid username and password then he/she can access the database.

User Login

User Registration

Login verification

User Page

Error Page

User

Cloud Database

* **Admin**

In this project the admin was done handling the data centers.

Admin was having following operations.

1. Login
2. Add data center regarding different data service providers.
3. Respond (Accept/decline) to customer data center request.
4. View data centers details.
5. Logout.

Validation

Admin

Admin Login Page

Admin Login

Cloud Database

Add datacenter regarding different data service providers

View datacenters details

Respond (Accept/decline) to customer data center request.

* **Data Center.**

In this project data centers will store the store (Hold) the customer’s data and each data center will have different costs as well as capacity for storage, transfer, get requests and put requests.

Display Result

Get Request

Put Request

Data Center

Different Costs & Storage Capacity

Cloud Database

Hold the customer’s data

Storage Data & Transfer Data

* **Customer (User)**

In this project the customers or users will store the data into the cloud across multiple cloud providers.

A customer was having following operations.

1. Registration.
2. Login.
3. Register for data center.
4. Send data center request to admin.
5. Store the data into data center.
6. View the stored data.
7. Get the data from data center.
8. View the prices (Cost for service).
9. Logout.

Validation

User

User Login Page

Register/Login

Cloud Database

View the stored admin data

User Send datacenter request to admin

Store the data into data center & Get the data from data center

View the prices (Cost for Services)

* **Data Allocation and Resource reservation (DAR)**

DAR (Data Allocation and Resource reservation) is a mechanism in our project to minimize the cost of cloud service across multiple cloud providers.

1. Getting the customer requirements when they storing the data.
2. Estimate the dominant cost of unit data.
3. Finding the minimum cost cloud data center.
4. Store data in that data center which have a minimum cost.

User

Data Allocation and Resource Reservation (DAR)

Cloud Database

Getting the customer requirements & uploading data

Estimate the dominant cost of unit data.

Finding the minimum cost cloud data center

Display Result

Get Request

Put Request

Store data in that data center which have a minimum cost

**2.2.3 GIVEN INPUT EXPECTED OUTPUT:**

* **User Interface**

**Input:** Enter login name and password.

**Output:** If valid user means directly open the home page otherwise show the error message and redirect to the registration page.

* **Admin**

**Input:** Admin name and Password.

**Output:** Admin Login after that Add dataenters, View datacenters details & customer data center request

* **Data Center**

**Input:** Checking all data

**Output:** the store the customer’s data and each data center will have different costs as well as capacity for storage, transfer, get requests and put requests.

* **Customer**

**Input:** Customer Register & Login.

**Output:** The customers will store the data into the cloud across multiple cloud providers.

* **Data Allocation and Resource reservation (DAR)**

**Input:** DAR is minimize the cost of cloud service across multiple cloud providers.

**Output:** DAR is a mechanism finding the minimum cost cloud data center & Store data in that data center which have a minimum cost.

* 1. **TECHNIQUE USED OR ALGORITHM USED**
* **Iterative Algorithm**

In this section, we describe the whole process of our proposed service mechanism, which is formalized in Algorithm.

We describe operational process of the proposed iterative algorithm. At the beginning, the cloud provider approximates it’s sever selection space (QL) and obtains the approximated one (Q(ε)L). For each servers subset (S͂) in (Q(ε)L), it initializes the allocation strategy (PS͂) in different time slot h (h ∈ H). Under this servers subset and allocation strategy, all of the users calculate the proper request strategies. The cloud provider reconfigures the allocation strategy such that the average response time over all users is minimized. Each of the user in the current set (Sc) calculates its utility, if the value is less than its reserved value (vi), then he/she refuses to use the cloud service. This process is terminated when all of the users who choose the cloud service and their corresponding request strategies are kept unchanged. The algorithm terminates until it selects the optimal servers subset from the approximated subset solution space (Q(ε)L).

**Algorithm:** Iterative Algorithm (IA)

**Input:** ε,µ, a, b, r, τ,M

**Output:** S, PS.

**1: Initialization:** The cloud provider approximates its solution space, i.e., Q(ε)L ← Calculate\_Q(ε)L (ε, c,µ,E,M). **Set** πS ← 0.

2: for (each server subset S͂ ∈ Q(ε)L) do

3: Set Sc ← N, and Sl ← ∅.

4: for (each time slot h ∈ H) do

5: for (each server j ∈ S͂) do

6: Set phj= μj/ (Σ|j|∈S͂ μj).

7: end for

8: end for

9: while (Sc ̸= Sl) do

10: Set Sl ← Sc, and ← Calculate ƛ(ε, S, PS, τ ).

11: for (each time slot h ∈ H) do

12: Set ph S͂ ← Calculate PS͂h (ε, λh, S).

13: end for

14: for (each user i ∈ Sc) do

15: if (Ui(ƛ(k)i , ƛ(k)i)< vi) then

16: Set ƛi ← 0, and Sc ← Sc − {i}.

17: end if

18: end for

19: end while

20: Set π S͂ ← c Σ i∈N Σ h∈H λhi – ET (S͂).

21: if (π ~ S > πS) then

22: Set πs ← π S͂, S ← S͂, and PS ← PS͂.

23: end if

24: end for

25: return S, PS.

* **DATA ALLOCATION AND RESOURCE RESERVATION**

DAR has two steps. First, its dominant-cost based data allocation algorithm conducts storage and request allocation scheduling that leads to the lowest total payment only in the pay-as-you-go manner. Second, its optimal resource reservation algorithm makes a reservation in each used storage datacenter to maximally reduce the total payment.

* Dominant-cost based data allocation algorithm. To reduce the total payment in the pay-as-you-go manner as much as possible, DAR tries to reduce the payment for each data item. Specifically, it finds the dominant cost (Storage, Get or Put) of each data item and allocates it to the datacenter with the minimum unit price of this dominant cost.
* Optimal resource reservation algorithm. It is a challenge to maximize the saved payment cost by reservation from the pay-as-you-go payment while avoiding over reservation. To handle this challenge, through theoretical analysis, we find the optimal reservation amount, which avoids both over reservation and under reservation as much as possible.

**CHAPTER 3**

**REQUIREMENTS ENGINEERING**

**3.1 GENERAL**

To our knowledge, hardly any previous works investigate multiple users′ profit optimizations, let alone optimizing the profits of a cloud provider and its users at the same time. In this work, we first try to optimize multiple users′ profits. Since multiple cloud users compete for using the resources of a cloud provider, and the utility of each user is affected by the decisions (service request strategies) of other users, it is natural to analyze the behaviors of such systems as strategic games.

**3.2 HARDWARE REQUIREMENTS**

The hardware requirements may serve as the basis for a contract for the implementation of the system and should therefore be a complete and consistent specification of the whole system. They are used by software engineers as the starting point for the system design. It shoulds what the system do and not how it should be implemented.

**HARDWARE**

* PROCESSOR : PENTIUM IV 2.6 GHz, Intel Core 2 Duo.
* RAM : 512 MB DD RAM
* MONITOR : 15” COLOR
* HARD DISK : 40 GB

**3.3 SOFTWARE REQUIREMENTS**

The software requirements document is the specification of the system. It should include both a definition and a specification of requirements. It is a set of what the system should do rather than how it should do it. The software requirements provide a basis for creating the software requirements specification. It is useful in estimating cost, planning team activities, performing tasks and tracking the teams and tracking the team’s progress throughout the development activity.

* + Front End : J2EE (JSP, SERVLET)
  + Back End : MY SQL 5.5
  + Operating System : Windows 7
  + IDE : Eclipse

**3.4 FUNCTIONAL REQUIREMENTS**

A functional requirement defines a function of a software-system or its component. A function is described as a set of inputs, the behaviour, and outputs. The outsourced computation is data is more secured.

* **User**
* Register
* Send datacenter
* Store the data
* Security
* **Admin**
* Login
* Add datacenter regarding different data service providers.
* Respond (Accept/decline) to customer data center request.
* View datacenters details.
* Logout.

**3.5 NON-FUNCTIONAL REQUIREMENTS**

The major non-functional Requirements of the system are as follows

* **Usability**

The system is designed with completely automated process hence there is no or less user intervention.

* **Reliability**

The system is more reliable because of the qualities that are inherited from the chosen platform java. The code built by using java is more reliable.

* **Performance**

This system is developing in the high-level languages and using the advanced front-end and back-end technologies it will give response to the end user on client system with in very less time.

* **Supportability**

The system is designed to be the cross platform supportable. The system is supported on a wide range of hardware and any software platform, which is having JVM, built into the system.

* **Implementation**

The system is implemented in web environment using struts framework. The apache tomcat is used as the web server and windows xp professional is used as the platform. Interface the user interface is based on Struts provides HTML Tag

**CHAPTER 4**

**DESIGN ENGINEERING**

**4.1 GENERAL**

Design Engineering deals with the various UML [Unified Modelling language] diagrams for the implementation of project. Design is a meaningful engineering representation of a thing that is to be built. Software design is a process through which the requirements are translated into representation of the software. Design is the place where quality is rendered in software engineering. Design is the means to accurately translate customer requirements into finished product.

**4.1.1 Use Case Diagram**



**EXPLANATION:**

Use cases are used during requirements elicitation and analysis to represent the functionality of the system. Use case focus on the behaviour of the system from an external point of view. The identification of actors and use cases results in the definition of the boundary of the system, which is, in differentiating the tasks accomplished by the system and the tasks accomplished by its environment. The actors are outside the boundary of the system, where as the use cases are inside the boundary of the system.

**4.1.2 Class Diagram**

****

**EXPLANATION**

In this class diagram represents how the classes with attributes and methods are linked together to perform the verification.

**4.1.3 Object Diagram**



**EXPLANATION:**

In the above digram tells about the flow of objects between the classes. It is a diagram that shows a complete or partial view of the structure of a modeled system. In this object diagram represents how the classes with attributes and methods are linked together to perform the verification with security.

**4.1.4 State Chart Diagram**



**EXPLANATION:**

State diagram are a loosely defined diagram to show workflows of stepwise activities and actions, with support for choice, iteration and concurrency. State diagrams require that the system described is composed of a finite number of states; sometimes, this is indeed the case, while at other times this is a reasonable abstraction. Many forms of state diagrams exist, which differ slightly and have different semantics.

**4.1.5 Sequence Diagram**



**EXPLANATION:**

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario.

**4.1.6 Collaboration Diagram**

****

**EXPLANATION:**

A collaboration diagram, also called a communication diagram or interaction diagram, is an illustration of the relationships and interactions among software objects in the Unified Modeling Language (UML). The concept is more than a decade old although it has been refined as modeling paradigms have evolved.

**4.1.7 Activity Diagram**



**EXPLANATION:**

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

**4.1.8 Component Diagram**



**EXPLANATION:**

In the Unified Modeling Language, a component diagram depicts how components are wired together to form larger components and or software systems. They are used to illustrate the structure of arbitrarily complex systems. User gives main query and it converted into sub queries and sends through data dissemination to data aggregators. Results are to be showed to user by data aggregators. All boxes are components and arrow indicates dependencies.

**4.1.9 E-R Diagram:**

Admin

Verify Req

Verify Rsp

Data Center & DAR

Customer

Cloud Database

**EXPLANATION:**

Entity-Relationship Model (ERM) is an abstract and conceptual representation of data. Entity-relationship modeling is a database modeling method, used to produce a type of conceptual schema or semantic data model of a system, often a relational database.

**4.1.10 Data Flow Diagram:**

Validation

Admin

Admin Login Page

Admin Login

Cloud Database

Add datacenter regarding different data service providers

View datacenters details

Respond (Accept/decline) to customer data center request.

**Level 0:**

**Level 1:**

User

Data Allocation and Resource Reservation (DAR)

Cloud Database

Getting the customer requirements & uploading data

Estimate the dominant cost of unit data.

Finding the minimum cost cloud data center

Display Result

Get Request

Put Request

Store data in that data center which have a minimum cost

**EXPLANATION:**

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system, modeling its process aspects. Often they are a preliminary step used to create an overview of the system which can later be elaborated. DFDs can also be used for the visualization of data processing (structured design).

A DFD shows what kinds of data will be input to and output from the system, where the data will come from and go to, and where the data will be stored. It does not show information about the timing of processes, or information about whether processes will operate in sequence or in parallel.

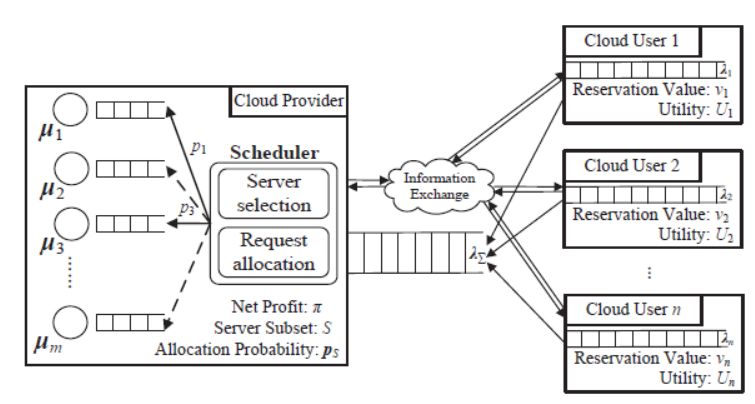
**4.1.11 Deployment Diagram:**



**EXPLANATION:**

In the Unified Modeling Language, a component diagram depicts how components are wired together to form larger deployment and or software systems. They are used to illustrate the structure of arbitrarily complex systems. User gives main query and it converted into sub queries and sends through data dissemination. Results are to be showed to user by data aggregators. All boxes are arrow indicates dependencies.

**4.2 System Architecture**



**System Architecture Model**

**CHAPTER 5**

**DEVELOPMENT TOOLS**

**5.1 GENERAL**

This chapter is about the software language and the tools used in the development of the project. The platform used here is JAVA. The Primary languages are JAVA, J2EE and J2ME. In this project J2EE is chosen for implementation.

**5.2 FEATURES OF JAVA**

**5.2.1 THE JAVA FRAMEWORK**

**Java** is a programming language originally developed by James Gosling at Microsystems and released in 1995 as a core component of Sun Microsystems' Java platform. The language derives much of its syntax from C and C++ but has a simpler object model and fewer low-level facilities. Java applications are typically compiled to byte code that can run on any Java Virtual Machine (JVM) regardless of computer architecture. Java is general-purpose, concurrent, class-based, and object-oriented, and is specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere".

Java is considered by many as one of the most influential programming languages of the 20th century, and is widely used from application software to web applications the java framework is a new platform independent that simplifies application development internet. Java technology's versatility, efficiency, platform portability, and security make it the ideal technology for network computing. From laptops to datacenters, game consoles to scientific supercomputers, cell phones to the Internet, Java is everywhere!

**5.2.2 OBJECTIVES OF JAVA**

To see places of Java in Action in our daily life, explore java.com.

## Why Software Developers Choose Java

Java has been tested, refined, extended, and proven by a dedicated community. And numbering more than 6.5 million developers, it's the largest and most active on the planet. With its versatility, efficiency, and portability, Java has become invaluable to developers by enabling them to:

* Write software on one platform and run it on virtually any other platform
* Create programs to run within a Web browser and Web services
* Develop server-side applications for online forums, stores, polls, HTML forms processing, and more
* Combine applications or services using the Java language to create highly customized applications or services
* Write powerful and efficient applications for mobile phones, remote processors, low-cost consumer products, and practically any other device with a digital heartbeat

## Some Ways Software Developers Learn Java

Today, many colleges and universities offer courses in programming for the Java platform. In addition, developers can also enhance their Java programming skills by reading Sun's java.sun.com Web site, subscribing to Java technology-focused newsletters, using the Java Tutorial and the New to Java Programming Center, and signing up for Web, virtual, or instructor-led courses.

**ObjectOriented** To be an Object Oriented language, any language must follow at least the four characteristics.

**1. Inheritance:** It is the process of creating the new classes and using the behavior of the existing classes by extending them just to reuse the existing code and adding addition a feature as needed.

**2. Encapsulation:** It is the mechanism of combining the information and providing the abstraction.

**3. Polymorphism:** As the name suggest one name multiple form, Polymorphism is the way of providing the different functionality by the functions having the same name based on the signatures of the methods.

**4. Dynamic binding:** Sometimes we don't have the knowledge of objects about their specific types while writing our code. It is the way of providing the maximum functionality to a program about the specific type at runtime.

**5.2.3 JAVA SWING OVERVIEW**

**Abstract Window Toolkit (AWT) is cross-platform**

Swing provides many controls and widgets to build user interfaces with. Swing class names typically begin with a J such as JButton, JList, JFrame. This is mainly to differentiate them from their AWT counterparts and in general is one-to-one replacements. Swing is built on the concept of Lightweight components vs AWT and SWT's concept of Heavyweight components. The difference between the two is that the Lightweight components are rendered (drawn) using purely Java code, such as drawLine and drawImage, whereas Heavyweight components use the native operating system to render the components.

Some components in Swing are actually heavyweight components. The top-level classes and any derived from them are heavyweight as they extend the AWT versions. This is needed because at the root of the UI, the parent windows need to be provided by the OS. These top-level classes include JWindow, JFrame, JDialog and JApplet. All Swing components to be rendered to the screen must be able to trace their way to a root window of one of those classes.

**Note**: It generally it is not a good idea to mix heavyweight components with lightweight components (other than as previously mentioned) as you will encounter layering issues, e.g., a lightweight component that should appear "on top" ends up being obscured by a heavyweight component. The few exceptions to this include using heavyweight components as the root pane and for popup windows. Generally speaking, heavyweight components will render on top of lightweight components and will not be consistent with the look and feel being used in Swing. There are exceptions, but that is an advanced topic. The truly adventurous may want to consider reading this article from Sun on mixing heavyweight and lightweight components.

**5.2.4 EVOLUTION OF COLLECTION FRAMEWORK:**

Almost all collections in Java are derived from the [**java.util.Collection**](http://download.oracle.com/javase/7/docs/api/java/util/Collection.html) interface. Collection defines the basic parts of all collections. The interface states the add() and remove() methods for adding to and removing from a collection respectively. Also required is the toArray() method, which converts the collection into a simple array of all the elements in the collection. Finally, the contains() method checks if a specified element is in the collection. The Collection interface is a sub interface of [**java.util.Iterable**](http://download.oracle.com/javase/7/docs/api/java/util/Iterable.html), so the iterator() method is also provided. All collections have an iterator that goes through all of the elements in the collection. Additionally, Collection is a generic. Any collection can be written to store any class. For example, Collection<String> can hold strings, and the elements from the collection can be used as strings without any casting required.

There are three main types of collections:

* Lists: always ordered, may contain duplicates and can be handled the same way as usual arrays
* Sets: cannot contain duplicates and provide random access to their elements
* Maps: connect unique keys with values, provide random access to its keys and may host duplicate values

**LIST:**

Lists are implemented in the JCF via the java.util.List interface. It defines a list as essentially a more flexible version of an array. Elements have a specific order, and duplicate elements are allowed. Elements can be placed in a specific position. They can also be searched for within the list. Two concrete classes implement List. The first is java.util.ArrayList, which implements the list as an array. Whenever functions specific to a list are required, the class moves the elements around within the array in order to do it. The other implementation is java.util.LinkedList. This class stores the elements in nodes that each have a pointer to the previous and next nodes in the list. The list can be traversed by following the pointers, and elements can be added or removed simply by changing the pointers around to place the node in its proper place.

**SET:**

Java's [java.util.Set](http://download.oracle.com/javase/7/docs/api/java/util/Set.html) interface defines the set. A set can't have any duplicate elements in it. Additionally, the set has no set order. As such, elements can't be found by index. Set is implemented by java.util.HashSet, java.util.LinkedHashSet, and java.util.TreeSet. HashSet uses a hash table. More specifically, it uses a [java.util.HashMap](http://download.oracle.com/javase/7/docs/api/java/util/HashMap.html) to store the hashes and elements and to prevent duplicates. Java.util.LinkedHashSet extends this by creating a doubly linked list that links all of the elements by their insertion order. This ensures that the iteration order over the set is predictable. [java.util.TreeSet](http://download.oracle.com/javase/7/docs/api/java/util/TreeSet.html) uses a red-black tree implemented by a [java.util.TreeMap](http://download.oracle.com/javase/7/docs/api/java/util/TreeMap.html). The red-black tree makes sure that there are no duplicates. Additionally, it allows Tree Set to implement java.util.SortedSet.

The [java.util.Set](http://download.oracle.com/javase/7/docs/api/java/util/Set.html) interface is extended by the java.util.SortedSet interface. Unlike a regular set, the elements in a sorted set are sorted, either by the element's compareTo() method, or a method provided to the constructor of the sorted set. The first and last elements of the sorted set can be retrieved, and subsets can be created via minimum and maximum values, as well as beginning or ending at the beginning or ending of the sorted set. The SortedSet interface is implemented by java.util.TreeSet

[java.util.SortedSet](http://download.oracle.com/javase/7/docs/api/java/util/SortedSet.html) is extended further via the java.util.NavigableSet interface. It's similar to SortedSet, but there are a few additional methods. The floor(), ceiling(), lower(), and higher() methods find an element in the set that's close to the parameter. Additionally, a descending iterator over the items in the set is provided. As with SortedSet, java.util.TreeSet implements NavigableSet.

**MAP:**

Maps are defined by the java.util.Map interface in Java. Maps are simple data structures that associate a key with a value. The element is the value. This lets the map be very flexible. If the key is the hash code of the element, the map is essentially a set. If it's just an increasing number, it becomes a list. Maps are implemented by java.util.HashMap, java.util.LinkedHashMap, and java.util.TreeMap. HashMap uses a hash table. The hashes of the keys are used to find the values in various buckets. LinkedHashMap extends this by creating a doubly linked list between the elements. This allows the elements to be accessed in the order in which they were inserted into the map. TreeMap, in contrast to HashMap and LinkedHashMap, uses a red-black tree. The keys are used as the values for the nodes in the tree, and the nodes point to the values in the map

**THREAD:**

Simply put, a threadis a program's path of execution. Most programs written today run as a single thread, causing problems when multiple events or actions need to occur at the same time. Let's say, for example, a program is not capable of drawing pictures while reading keystrokes. The program must give its full attention to the keyboard input lacking the ability to handle more than one event at a time. The ideal solution to this problem is the seamless execution of two or more sections of a program at the same time.

**CREATING THREADS:**

Java's creators have graciously designed two ways of creating threads: implementing an interface and extending a class. Extending a class is the way Java inherits methods and variables from a parent class. In this case, one can only extend or inherit from a single parent class. This limitation within Java can be overcome by implementing interfaces, which is the most common way to create threads. (Note that the act of inheriting merely allows the class to be run as a thread. It is up to the class to start() execution, etc.)

Interfaces provide a way for programmers to lay the groundwork of a class. They are used to design the requirements for a set of classes to implement. The interface sets everything up, and the class or classes that implement the interface do all the work. The different set of classes that implement the interface have to follow the same rules.

**5.3 CONCLUSION**

Swing's high level of flexibility is reflected in its inherent ability to override the native host [operating system](http://en.wikipedia.org/wiki/Operating_system) (OS)'s GUI controls for displaying itself. Swing "paints" its controls using the Java 2D APIs, rather than calling a native user interface toolkit. The Java thread scheduler is very simple. All threads have a priority value which can be changed dynamically by calls to the threads setPriority() method . Implementing the above concepts in our project to do the efficient work among the Server.

**CHAPTER 6**

**IMPLEMENTATION**

**6.1 GENERAL**

The Implementation is nothing but sores code of project.

**6.2 IMPLEMENTATION**

**Coding:**

**Reg.java**

package com.servlets;

import java.io.IOException;

import java.sql.SQLException;

import javax.servlet.RequestDispatcher;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

import com.beans.UserBean;

import com.contriller.DBConnect;

@WebServlet("/Reg")

public class Reg extends HttpServlet {

private static final long serialVersionUID = 1L;

public Reg() {

super();

} protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

// TODO Auto-generated method stub

String uname=request.getParameter("uname");

String email=request.getParameter("email");

String pass=request.getParameter("pass");

String city=request.getParameter("city");

UserBean u=new UserBean();

u.setCname(uname);

u.setEmail(email);

u.setCity(city);

u.setPass(pass);

try { int i=DBConnect.Register(u);

if(i>0) {

RequestDispatcher rd=request.getRequestDispatcher("success.jsp?msg=registered successfully &&to=login.jsp");

rd.forward(request, response);

}else { RequestDispatcher rd=request.getRequestDispatcher("success.jsp?msg='registration failed'&&to=reg.jsp");

rd.forward(request, response); }

} catch (SQLException e) {

e.printStackTrace();

} }

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

// TODO Auto-generated method stub

}}

**login.jsp**

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"

pageEncoding="ISO-8859-1"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html><head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

<title>Insert title here</title>

<link rel="stylesheet" href="w3.css">

</head>

<body class="w3-cyan">

<h1 STYLE="font-size: 70px" class="w3-center w3-blue">A NEW SERVICE MECHANISM FOR PROFIT OPTIMIZATIONS OF A CLOUD PROVIDER AND ITS USERS</h1>

<div class="w3-bar w3-center w3-orange">

<a href=index.html class="w3-button w3-blue" style="width:300px;">Home</a>

<a href=reg.jsp class="w3-button w3-blue"style="width:300px;">Register</a>

</div><center>

<div class="w3-card w3-orange " style="width: 400px;">

<h1 CLASS="w3-center w3-blue-grey">LOGIN HERE</h1>

<form action="Login" method="get">

<table class="w3-table">

<tr><td></td><td><input type=radio name=utype value=user> Customer</td></tr>

<tr><td></td><td><input type=radio name=utype value=admin> Admin</td></tr>

<tr><td>Enter UserID </td><td><input type=text name=uname></td></tr>

<tr><td>Enter Password</td><td><input Type=password name=pass></td></tr>

<tr><td><input Type=reset value=clear class="w3-button w3-blue" style="width:100%"></td><td><input type=submit value=Login class="w3-button w3-blue" style="width:100%"></td></tr>

</table></form>

</div></center>

</body>

</html>

**Login.java**

package com.servlets;

import java.io.IOException;

import java.sql.ResultSet;

import java.sql.SQLException;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

import com.beans.UserBean;

import com.contriller.DBConnect;

@WebServlet("/Login")

public class Login extends HttpServlet {

private static final long serialVersionUID = 1L;

public Login() {

super();

// TODO Auto-generated constructor stub

}

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

// TODO Auto-generated method stub

String utype=request.getParameter("utype");

String uname=request.getParameter("uname");

String pass=request.getParameter("pass");

if(utype.equals("admin")) {

if(uname.equals("admin")&&pass.equals("admin"))

{

response.sendRedirect("adminHome.jsp");

} }else if(utype.equals("user")){

UserBean u=new UserBean();

u.setEmail(uname);

u.setPass(pass);

try {

if(DBConnect.checkLog(u))

{

HttpSession h=request.getSession();

h.setAttribute("uname", uname);

response.sendRedirect("userhome.jsp");

} } catch (SQLException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} }}

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

// TODO Auto-generated method stub

}}

**AddingDCs.java**

package com.servlets;

import java.io.IOException;

import java.sql.SQLException;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import com.beans.DcBean;

import com.contriller.DBConnect;

@WebServlet("/AddingDCs")

public class AddingDCs extends HttpServlet {

private static final long serialVersionUID = 1L;

public AddingDCs() {

super();

}

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

// TODO Auto-generated method stub

String dcname=request.getParameter("dcname");

String cspname=request.getParameter("cspname");

int scap=Integer.parseInt(request.getParameter("scap"));

int pos=Integer.parseInt(request.getParameter("pos"));

int tcap=Integer.parseInt(request.getParameter("tcap"));

int pot=Integer.parseInt(request.getParameter("pot"));

int gcap=Integer.parseInt(request.getParameter("gcap"));

int pog=Integer.parseInt(request.getParameter("pog"));

int pcap=Integer.parseInt(request.getParameter("pcap"));

int pop=Integer.parseInt(request.getParameter("pop"));

int por=Integer.parseInt(request.getParameter("por"));

int nom=Integer.parseInt(request.getParameter("nom"));

DcBean d=new DcBean();

d.setCspname(cspname);

d.setDcname(dcname);

d.setScap(scap);

d.setPos(pos);

d.setTcap(tcap);

d.setPot(pot);

d.setGcap(gcap);

d.setPog(pog);

d.setPop(pop);

d.setPcap(pcap);

d.setPor(por);

d.setNom(nom);

try {

int i=DBConnect.addDataCenter(d);

if(i>0)

{

response .sendRedirect("success.jsp?msg=DataCenter Added Successfully&&to=adminHome.jsp");

}else{

response .sendRedirect("success.jsp?msg=Fail to Add DataCenter&&to=adminHome.jsp");

}

} catch (SQLException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

// TODO Auto-generated method stub

}

}

**adminhome.jsp**

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"

pageEncoding="ISO-8859-1"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

<title>Insert title here</title>

<link rel="stylesheet" href="w3.css">

<style>

.mySlides {display:none;}

</style></head>

<body class="w3-cyan">

<h1 STYLE="font-size: 70px" class="w3-center w3-blue"> A New Service Mechanism for Profit Optimizations of a Cloud Provider and Its Users</h1>

<div class="w3-bar w3-center w3-blue">

<a href=addDCs.jsp class="w3-button w3-orange">Add DataCenters</a>

<a href=displayDCsAtAdmin.jsp class="w3-button w3-orange">DataCenters List</a>

<a href=customerRequests.jsp class="w3-button w3-orange">Customer Requests</a>

<a href=login.jsp class="w3-button w3-orange">Logout</a></div>

<div class="w3-content w3-section" style="max-width:90%;max-height: 400px;">

<img class="mySlides" src="img1.jpg" style="width:100%;height: 400px;">

<img class="mySlides" src="img2.jpg" style="width:100%;height: 400px;">

<img class="mySlides" src="img3.jpeg" style="width:100%;height: 400px;">

</div><script>

var myIndex = 0;

carousel();

function carousel() {

var i;

var x = document.getElementsByClassName("mySlides");

for (i = 0; i < x.length; i++) {

x[i].style.display = "none";

} myIndex++;

if (myIndex > x.length) {myIndex = 1}

x[myIndex-1].style.display = "block";

setTimeout(carousel, 2000); // Change image every 2 seconds

}</script>

</body>

</html>

**RequestToAdmin.java**

package com.servlets;

import java.io.IOException;

import java.sql.SQLException;

import javax.servlet.RequestDispatcher;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import com.beans.DcBean;

import com.contriller.DBConnect;

@WebServlet("/RequestToAdmin")

public class RequestToAdmin extends HttpServlet {

private static final long serialVersionUID = 1L;

public RequestToAdmin() {

super();

}

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

String email=request.getParameter("email");

String dcname=request.getParameter("dcname");

String dcid=request.getParameter("dcid");

String scap=request.getParameter("scap");

String gcap=request.getParameter("gcap");

String pcap=request.getParameter("pcap");

String por=request.getParameter("por");

String price=request.getParameter("fee");

DcBean d=new DcBean();

d.setCspname(email);

d.setDcname(dcname);

d.setId(Integer.parseInt(dcid));

d.setScap(Integer.parseInt(scap));

d.setGcap(Integer.parseInt(gcap));

d.setPcap(Integer.parseInt(pcap));

d.setPor(Integer.parseInt(price));

d.setNom(Integer.parseInt(por));

try { int i=DBConnect.storeToAdmin(d);

if(i>0)

{ RequestDispatcher rd=request.getRequestDispatcher("success.jsp?msg='request successfully sended to admin'&&to=userhome.jsp");

rd.forward(request, response);

}else{

RequestDispatcher rd=request.getRequestDispatcher("success.jsp?msg='request sending failed'&&to=displayDCs.jsp");

rd.forward(request, response);}

} catch (SQLException e) {

e.printStackTrace();

} }

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

}

}

**CHAPTER 7**

**SNAPSHOTS**

**7.1 GENERAL**

This project is implements like web application using COREJAVA and the Server process is maintained using the SOCKET & SERVERSOCKET and the Design part is played by Cascading Style Sheet.

**7.2 VARIOUS SNAPSHOTS**

**CHAPTER 8**

**SOFTWARE TESTING**

**8.1 GENERAL**

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of tests. Each test type addresses a specific testing requirement.

**8.2 DEVELOPING METHODOLOGIES**

The test process is initiated by developing a comprehensive plan to test the general functionality and special features on a variety of platform combinations. Strict quality control procedures are used. The process verifies that the application meets the requirements specified in the system requirements document and is bug free. The following are the considerations used to develop the framework from developing the testing methodologies.

**8.3 Types of Tests**

**8.3.1 Unit testing**

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program input produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

**8.3.2 Functional test**

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures : interfacing systems or procedures must be invoked.

**8.3.3 System Test**

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration-oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

**8.3.4 Performance Test**

The Performance test ensures that the output be produced within the time limits, and the time taken by the system for compiling, giving response to the users and request being send to the system for to retrieve the results.

**8.3.5 Integration Testing**

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

**8.3.6 Acceptance Testing**

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

**Acceptance testing for Data Synchronization:**

* The Acknowledgements will be received by the Sender Node after the Packets are received by the Destination Node
* The Route add operation is done only when there is a Route request in need
* The Status of Nodes information is done automatically in the Cache Updation process

**8.3.7 Build the test plan**

Any project can be divided into units that can be further performed for detailed processing. Then a testing strategy for each of this unit is carried out. Unit testing helps to identity the possible bugs in the individual component, so the component that has bugs can be identified and can be rectified from errors.

**TEST CASES:**

Test cases can be divided in to two types. First one is Positive test cases and second one is negative test cases. In positive test cases are conducted by the developer intention is to get the output. In negative test cases are conducted by the developer intention is to don’t get the output.

**TEST PLAN**

The test procedure is started by building up a thorough arrangement to test the general usefulness and extraordinary highlights on an assortment of stage mixes. Exacting quality control methods are utilized. The procedure checks that the application meets the necessities indicated in the framework prerequisites report and is sans bug.

Any project can be separated into units that can be further performed for detailed processing. Then a testing strategy for each of this unit is carried out. Unit testing serves to character the potential bugs in the individual segment, so the segment that has bugs can be recognized and can be redressed from mistakes.

**TEST CASES**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sl.No** | **Test Scenario** | **Action** | **Expected result** | **Actual Result** | **Remarks** |
| 1. | Registration | Enter all the details asked for registration. | Registration Successful. | Successful | Pass |
| 2. | User Login. | Enter Correct Username & Password. | User Login Successful. | Login Successful | Pass |
| 3. | Send information to the Admin. | Upload the data related for user. | Data Uploaded to the Cloud. | Upload Successful | Pass |
| 6. | Check for User Information. | Check for all the registered user’s visibility. | List of all the registered users should be visible. | List of all the registered Users. | Pass |
| 9. | Admin Login | Entre the Correct Name & Password of the Admin Server. | Admin Server Login Successful. | Login Successful. | Pass |
| 10. | Admin view all the files stored in cloud. | Check for all the files Stored in cloud through data. | The list of the user data is visible. | All the Stored files are visible. | Pass. |

**CHAPTER 9**

**APPLICATION**

**9.1 GENERAL**

Many previous works focus on finding the minimum amount of resources to support the application workload to reduce cloud storage cost in a single CSP. However, there have been only a few works that studied cloud storage cost minimization for a storage service across multiple CSPs with different prices. Within our knowledge, SPAN Store is the only work that handles this problem. It aims to minimize the cloud storage cost while satisfying the latency and failure requirement across multiple CSPs. However, it neglects both the resource reservation pricing model and the datacenter capacity limits for serving Get/Put requests. A datacenter’s Get/Put capacity is represented by the Get/Put rate (i.e., the number of Gets/Puts in a unit time period) it can handle. Reserving resources in advance can save significant payment cost for customers and capacity limit is critical for guaranteeing SLOs.

**9.2 FUTURE ENHANCEMENT**

Finally, we propose an Iterative Algorithm (IA), which characterizes the whole process of our proposed service mechanism. We conduct some numerical calculations to verify our theoretical analyses. The experimental results show that our IA algorithm can reduce energy cost and improve users utilities to certain extent by configuring proper strategies. As part of future work, we will study the cloud center choice among multiple different cloud providers or determine a proper mixed choice strategy. Another direction is the opposite, we consider problem from cloud providers and study the competitions among multiple cloud providers, which may incorporate charge price, service quality, and so on.

**CHAPTER 10**

**CONCLUSION & REFERENCE**

**10.1 CONCLUSION**

With the popularization of cloud computing and its many advantages such as cost-effectiveness, elasticity, and scalability, more and more applications are moved from local computing environment to cloud center. In this work, we try to design a new service mechanism for profit optimizations of both a cloud provider and its multiple users.

We consider the problem from a game theoretic perspective and characterize the relationship between the cloud provider and its multiple users as a Stackelberg game, in which the strategies of all users are subject to that of the cloud provider. The cloud provider tries to select appropriate servers and configure a proper request allocation strategy to reduce energy cost while satisfying its cloud users at the same time. We approximate its server selection space by adding a controlling parameter and configure an optimal request allocation strategy. For each user, we design a utility function which combines the net profit with time efficiency and try to maximize its value under the strategy of the cloud provider. We formulate the competitions among all users as a generalized Nash equilibrium problem (GNEP). We solve the problem by employing varational inequality (VI) theory and prove that there exists a generalized Nash equilibrium solution set for the formulated GNEP.

**10.2 REFERENCE**

[1] A. Prasad and S. Rao, “A mechanism design approach to resource procurement in cloud computing,” Computers, IEEE Transactions on, vol. 63, no. 1, pp. 17–30, Jan 2014.

[2] R. Pal and P. Hui, “Economic models for cloud service markets: Pricing and capacity planning,” Theoretical Computer Science, vol. 496, no. 0, pp. 113 – 124, 2013.

[3] P. D. Kaur and I. Chana, “A resource elasticity framework for QoS-aware execution of cloud applications,” Future Generation Computer Systems, vol. 37, no. 0, pp. 14 – 25, 2014.

[4] L. Duan, D. Zhan, and J. Hohnerlein, “Optimizing cloud data center energy efficiency via dynamic prediction of CPU idle intervals,” in 2015 IEEE 8th International Conference on Cloud Computing. IEEE, 2015, pp. 985–988.

[5] Z. Li, J. Ge, H. Hu, W. Song, H. Hu, and B. Luo, “Cost and energy aware scheduling algorithm for scientific workflows with deadline constraint in clouds,” IEEE Transactions on Services Computing, 2015, doi: 10.1109/TSC.2015.2466545.

[6] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, “The cost of a cloud: research problems in data center networks,” ACM SIGCOMM computer communication review, vol. 39, no. 1, pp. 68–73, 2008.

[7] J. Cao, K. Hwang, K. Li, and A. Zomaya, “Optimal multi server configuration for profit maximization in cloud computing,” Parallel and Distributed Systems, IEEE Transactions on, vol. 24, no. 6, pp. 1087–1096, June 2013.

[8] Y. Feng, B. Li, and B. Li, “Price competition in an oligopoly market with multiple iaas cloud providers,” Computers, IEEE Transactions on, vol. 63, no. 1, pp. 59–73, Jan 2014.

[9] J. Cao, K. Li, and I. Stojmenovic, “Optimal power allocation and load distribution for multiple heterogeneous multicore server processors across clouds and data centers,” Computers, IEEE Transactions on, vol. 63, no. 1, pp. 45–58, Jan 2014.

[10] S. Jrgensen and G. Zaccour, “A survey of game-theoretic models of cooperative advertising,” European Journal of Operational Research, vol. 237, no. 1, pp. 1 – 14, 2014.

[11] S. Liu, S. Ren, G. Quan, M. Zhao, and S. Ren, “Profit aware load balancing for distributed cloud data centers,” in Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on, May 2013, pp. 611–622.

[12] U. Lampe, M. Siebenhaar, A. Papageorgiou, D. Schuller, and R. Steinmetz, “Maximizing cloud provider profit from equilibrium price auctions,” in Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on, June 2012, pp. 83–90.

[13] H. Goudarzi and M. Pedram, “Maximizing profit in cloud computing system via resource allocation,” in Proceedings of the 2011 31st International Conference on Distributed Computing Systems Workshops, ser. ICDCSW ’11. IEEE Computer Society, 2011, pp.1–6.

[14] E. K¨orpeo˘ glu, A. S¸en, and K. G¨ uler, “Non-cooperative joint replenishment under asymmetric information,” European Journal of Operational Research, vol. 227, no. 3, pp. 434–443, 2013.

[15] C. Liu, K. Li, C. Xu, and K. Li, “Strategy configurations of multiple users competition for cloud service reservation,” IEEE Transactions on Parallel and Distributed Systems, vol. 27, no. 2, pp. 508–520, 2016.

[16] M. J. Osborne and A. Rubinstein, “A course in game theory,” MIT press, 1994.

[17] S. S. Aote and M. U. Kharat, “A game-theoretic model for dynamic load balancing in distributed systems,” in Proceedings of the International Conference on Advances in Computing, Communication and Control, ser. ICAC3 ’09. ACM, 2009, pp. 235–238.

[18] N. Li and J. Marden, “Designing games for distributed optimization,” Selected Topics in Signal Processing, IEEE Journal of, vol. 7, no. 2, pp. 230–242, April 2013.

[19] S. Penmatsa and A. T. Chronopoulos, “Game-theoretic static load balancing for distributed systems,” Journal of Parallel and Distributed Computing, vol. 71, no. 4, pp. 537 – 555, 2011.

[20] G. Scutari and J.-S. Pang, “Joint sensing and power allocation in nonconvex cognitive radio games: Nash equilibria and distributed algorithms,” Information Theory, IEEE Transactions on, vol. 59, no. 7, pp. 4626–4661, July 2013.

[21] Z. Wang, A. Szolnoki, and M. Perc, “Rewarding evolutionary fitness with links between populations promotes cooperation,” Journal of Theoretical Biology, vol. 349, no. 0, pp. 50 – 56, 2014.

[22] G. Scutari, D. Palomar, F. Facchinei, and J.-S. Pang, “Convex optimization, game theory, and variational inequality theory,” Signal Processing Magazine, IEEE, vol. 27, no. 3, pp. 35–49, May 2010.

[23] K. Li, C. Liu, and K. Li, “An approximation algorithm based on game theory for scheduling simple linear deteriorating jobs,” Theoretical Computer Science, vol. 543, no. 0, pp. 46 – 51, 2014.

[24] C. A. Ioannou and J. Romero, “A generalized approach to belief learning in repeated games,” Games and Economic Behavior, vol. 87, no. 0, pp. 178 – 203, 2014.

[25] K. Li, C. Liu, K. Li, and A. Y. Zomaya, “A framework of price bidding configurations for resource usage in cloud computing,” IEEE Transactions on Parallel and Distributed Systems, vol. 27, no. 8, pp. 2168–2181, 2016.

[26] P. Wang, Y. Qi, and X. Liu, “Power-aware optimization for heterogeneous multi-tier clusters,” Journal of Parallel and Distributed Computing, vol. 74, no. 1, pp. 2005 – 2015, 2014.

[27] Y. Gao, H. Guan, Z. Qi, T. Song, F. Huan, and L. Liu, “Service level agreement-based energy-efficient resource management in cloud data centers,” Computers & Electrical Engineering, vol. 40, no. 5, pp. 1621 – 1633, 2014.

[28] J. Mei, K. Li, J. Hu, S. Yin, and E. H.-M. Sha, “Energy-aware pre-emptive scheduling algorithm for sporadic tasks on fDVSg platform,” Microprocessors and Microsystems, vol. 37, no. 1, pp. 99 – 112, 2013.

[29] D. Zhu, R. Melhem, and B. Childers, “Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems,” Parallel and Distributed Systems, IEEE Transactions on, vol. 14, no. 7, pp. 686–700, July 2003.

[30] A. Beloglazov, J. Abawajy, and R. Buyya, “Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing,” Future Generation Computer Systems, vol. 28, no. 5, pp. 755 – 768, 2012, special Section: Energy efficiency in large-scale distributed systems.

[31] R. N. Calheiros, A. N. Toosi, C. Vecchiola, and R. Buyya, “A coordinator for scaling elastic applications across multiple clouds,” Future Generation Computer Systems, vol. 28, no. 8, pp. 1350 – 1362, 2012, including Special sections SS: Trusting Software Behavior and SS: Economics of Computing Services.

[32] M. Mezmaz, N. Melab, Y. Kessaci, Y. Lee, E.-G. Talbi, A. Zomaya, and D. Tuyttens, “A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems,” Journal of Parallel and Distributed Computing, vol. 71, no. 11, pp. 1497 – 1508, 2011.

[33] B. Zhai, D. Blaauw, D. Sylvester, and K. Flautner, “Theoretical and practical limits of dynamic voltage scaling,” Design Automation Conference, vol. 0, pp. 868–873, 2004.

[34] I. Atzeni, L. Ordonez, G. Scutari, D. Palomar, and J. Fonollosa, “Noncooperative day-ahead bidding strategies for demand-side expected cost minimization with real-time adjustments: A gnep approach,” Signal Processing, IEEE Transactions on, vol. 62, no. 9, pp. 2397–2412, 2014.

[35] H. Chen, Y. Li, R. Louie, and B. Vucetic, “Autonomous demand side management based on energy consumption scheduling and instantaneous load billing: An aggregative game approach,” Smart Grid, IEEE Transactions on, vol. 5, no. 4, pp. 1744–1754, 2014.

[36] H. Soliman and A. Leon-Garcia, “Game-theoretic demand-side management with storage devices for the future smart grid,” Smart Grid, IEEE Transactions on, vol. 5, no. 3, pp. 1475–1485, 2014.

[37] G. Du, R. J. Jiao, and M. Chen, “Joint optimization of product family configuration and scaling design by stackelberg game,” European Journal of Operational Research, vol. 232, no. 2, pp. 330 – 341, 2014.

[38] J. Wang, M. Peng, S. Jin, and C. Zhao, “A generalized nash equilibrium approach for robust cognitive radio networks via generalized variational inequalities,” Wireless Communications, IEEE Transactions on, vol. 13, no. 7, pp. 3701–3714, 2014.

[39] G. Scutari, D. Palomar, F. Facchinei, and J.-S. Pang, “Monotone games for cognitive radio systems,” in Distributed Decision Making and Control, ser. Lecture Notes in Control and Information Sciences. Springer London, 2012, vol. 417, pp. 83–112.