Abstract: Cloud computing turns information technology into ordinary commodities and utilities by the pay-per-use pricing model. A service provider rents resources from the infrastructure providers and provides services to customers. A customer submits its request to a service provider and pays for it based on the amount and the quality of the provided service. The limited wireless spectrum becoming a bottleneck for meeting today’s fast growing demands for wireless data services. Most specifically, there is very little spectrum left that can be licensed to new wireless services and applications. A double resource renting scheme is designed firstly in which short-term renting and long-term renting are combined aiming at the existing issues. A multi-server system maintains a queue with infinite capacity. When the incoming service requests cannot be processed immediately after they arrive, they are firstly placed in the queue until they can be handled by any available server. The first-come-first-served (FCFS) queuing discipline is adopted. Finally, a series of calculations are conducted to compare the profit of our proposed scheme with that of the single renting scheme. The results show that our scheme can not only guarantee the service quality of all requests, but also obtain more profit than the latter.

Keywords: multi-server system, cloud computing, queuing discipline, services quality, wireless data server, single renting scheme,

I. INTRODUCTION

Cloud computing is able to provide the most cost-effective and energy-efficient way of computing resources management and computing services provision. Cloud computing turns information technology into ordinary commodities and utilities by using the pay-per-use pricing model. However, cloud computing will never be free and understanding the economics of cloud computing becomes critically important. One attractive cloud computing environment is a three-tier structure, which consists of infrastructure vendors, service providers, and consumers. The three parties are also called cluster nodes, cluster managers, and consumers in cluster computing systems, and resource providers, service providers, and clients in grid computing systems. An infrastructure vendor maintains basic hardware and software facilities. A service provider rents resources from the infrastructure vendors, builds appropriate multi-server systems, and provides various services to users. A consumer submits a service request to a service provider, receives the desired result from the service provider with certain service level agreement, and pays for the service based on the amount of the service and the quality of the service. A service provider can build different multi-server systems for different applications domains, such that service requests of different nature are sent to different multi-server systems. Each multi-server system contains multiple servers, and such a multi-server system can be devoted to serve one type of service requests and applications. The configuration of a multi-server system is characterized by two basic features, i.e., the size of the multi-server system (the number of servers) and the speed of the multi-server system (execution speed of the servers). The service charge to a service request is determined by two factors, i.e., the expected length of the service and the actual length of the service. The expected length of a service (i.e., the expected service time) is the execution time of an application on a standard server with a baseline or reference speed. Once the baseline speed is set, the expected length of a service is determined by a service request itself, i.e., the service requirement (amount of service) measured by the number of instructions to be executed. The longer (shorter, respectively) the expected length of a service is, the more (less, respectively) the service charge is. The actual length of a service (i.e., the actual service time) is the actual execution time of an application. The actual length of a service depends on the size of a multi-server system, the speed of the servers (which may be faster or slower than the baseline speed), and the workload of the multi-server system. Notice that the actual service time is a random variable, which is determined by the task waiting time once a multi-server system is established. There are many different service performance metrics in service level agreements. Our performance metric in this paper is the task response time (or the turnaround time), i.e., the time taken to complete a task, which includes task waiting time and task execution time.
II. LITERATURE SURVEY

1. Optimal Multi server Configuration for Profit Maximization in Cloud Computing, Junwei Cao, Tsinghua Nat. Lab. for Inf. Sci. & Technol., Tsinghua Univ., Beijing, China; Kai Hwang; Kegin Li; Zomaya, A.Y. Year: September 2008

As cloud computing becomes more and more popular, understanding the economics of cloud computing becomes critically important. To maximize the profit, a service provider should understand both service charges and business costs, and how they are determined by the characteristics of the applications and the configuration of a multi-server system. The problem of optimal multi-server configuration for profit maximization in a cloud computing environment is studied. Our pricing model takes such factors into considerations as the amount of a service, the workload of an application environment, the configuration of a multi-server system, the service-level agreement, the satisfaction of a consumer, the quality of a service, the penalty of a low-quality service, the cost of renting, the cost of energy consumption, and a service provider's margin and profit. Our approach is to treat a multi-server system as an M/M/m queuing model, such that our optimization problem can be formulated and solved analytically. Two server speed and power consumption models are considered, namely, the idle-speed model and the constant-speed model. The probability density function of the waiting time of a newly arrived service request is derived. The expected service charge to a service request is calculated. The expected net business gain in one unit of time is obtained. Numerical calculations of the optimal server size and the optimal server speed are demonstrated.

2. Energy and performance management of green data centers a profit maximization approach Ghamkhari, M. ; EE Dept., Univ. of California at Riverside, Riverside, CA, USA ; Mohsenian-Rad, H. Year: September 2008

While a large body of work has recently focused on reducing data center's energy expenses, there exists no prior work on investigating the trade-off between minimizing data center's energy expenditure and maximizing their revenue for various Internet and cloud computing services that they may offer. In this paper, we seek to tackle this shortcoming by proposing a systematic approach to maximize green data center's profit, i.e., revenue minus cost. In this regard, we explicitly take into account practical service-level agreements (SLAs) that currently exist between data centers and their customers. Our model also incorporates various other factors such as availability of local renewable power generation at data centers and the stochastic nature of data centers' workload. Furthermore, we propose a novel optimization-based profit maximization strategy for data centers for two different cases, without and with behind-the-meter renewable generators. We show that the formulated optimization problems in both cases are convex programs; therefore, they are tractable and appropriate for practical implementation. Using various experimental data and via computer simulations, we assess the performance of the proposed optimization-based profit maximization strategy and show that it significantly outperforms two comparable energy and performance management algorithms that are recently proposed in the literature.

3. Dynamic cloud pricing for revenue Maximization Hong Xu ; Baochun Li Year: 10-15 June 2012

In this note, we discuss issues pertaining to end-to-end quality- of-service management of commodity Internet applications and associated pricing incentive mechanisms. The issue of service differentiation is first studied using a simple two-level model including delay and throughput sensitive traffic. We show that by introducing service differentiation one can make more efficient use of resources; however this depends on the differences in required QoS as well as the typical capacities of the systems involved. As such, service differentiation may be more beneficial in lower capacity access networks than in high capacity core networks. We then focus on delay-sensitive and study flat-rate versus usage-based pricing under overload conditions. Our results suggest that in overload scenarios usage-based pricing is advantageous both from the system perspective, i.e., reduces the degree of overload, and individual users’ perspective, increases their perceived utilization.


We study revenue-maximizing pricing by a service provider in a communication network and compare revenues from simple pricing rules to the maximum revenues that are feasible. In particular, we focus on flat entry fees as the simplest pricing rule. We provide a lower bound for the ratio between the revenue from this pricing rule and maximum revenue, which we refer to as the price of simplicity. We characterize what types of environments lead to a low price of simplicity and show that in a range of environments, the loss of revenue from using simple entry fees is small. We then study the price of simplicity for a simple non-linear pricing (price discrimination) scheme based on the Paris Metro Pricing. The service provider creates different service classes and charges differential entry fees for these classes. We show that the gain from this type of price discrimination is small, particularly in environments in which the simple entry fee pricing leads to a low price of simplicity.

5. A Genetic Model for Pricing in Cloud Computing Markets Authors: Mario Macas and Jordi Guitart Year: September 2010

Cloud Computing markets arise as an efficient way to allocate resources for the execution of tasks and services within a set of geographically dispersed providers from different organizations. Client applications and service providers meet in a market and negotiate for the sales of services by means of the signature of a Service Level Agreement. Depending on the status of the demand, the provider is able to offer higher or lower prices for maximizing its profit. It is difficult to establish a profitable pricing function in competitive markets, because there are several factors that can influence in the prices. This paper deals with the problem of offering competitive prices in the negotiation of services in Cloud Computing markets. A Genetic Algorithms approach is proposed, in which a naive pricing function evolves to a pricing function that offers suitable prices in function of the
system status. Its results are compared with other pricing strategies, demonstrating its validity.

III. EXISTING SYSTEM

In this system, Profit of service providers is related with many factors such as the price, the market demand, the system configuration, the customer satisfaction and so forth. Service providers naturally wish to set a higher price to get a higher profit margin; but doing so would decrease the customer satisfaction, which leads to a risk of discouraging demand in the future. Hence, selecting a reasonable pricing strategy is important for service providers.

The pricing strategies are divided into two categories, i.e., static pricing and dynamic pricing. Static pricing means that the price of a service request is fixed and known in advance, and it does not change with the conditions. With dynamic pricing, a service provider delays the pricing decision until after the customer demand is revealed, so that the service provider can adjust prices accordingly. The factor affecting the profit of service providers is customer satisfaction which is determined by the quality of service and the charge. In order to improve the customer satisfaction level, there is a service-level agreement (SLA) between a service provider and the customers. The SLA adopts a price compensation mechanism for the customers with low service quality. The mechanism is to guarantee the service quality and the customer satisfaction so that more customers are attracted since profit is an important concern to cloud service providers; many works have been done on how to boost their profit.

IV. PROPOSED SYSTEM

In this system, a double renting scheme is designed to configure a cloud service platform, which can guarantee the service quality of all requests and reduce the resource waste greatly. A novel double renting scheme is proposed for service providers. It combines long-term renting with short-term renting, which can not only satisfy quality-of-service requirements under the varying system workload, but also reduce the resource waste greatly in term renting and short-term renting. In general, the rental price of long-term renting is much cheaper than that of short-term renting.

A customer submits a service request to a service provider which delivers services on demand. The customer receives the desired result from the service provider with certain service-level agreement, and pays for the service based on the amount of the service and the service quality. Service providers pay infrastructure providers for renting their physical resources, and charge customers for processing their service requests, which generates cost and revenue, respectively. The profit is generated from the gap between the revenue and the cost. Moreover, a profit maximization problem is formulated and solved to get the optimal multi-server configuration which can product more profit than the optimal configuration.

V. ARCHITECTURE DIAGRAM

The architecture diagram explains how the clients request is processed using a multi-server system in a cloud environment. The data owner stores the resource in a cloud server for which priority is given by using set priority. The priority is given as per the cost paid to the cloud server. The details about the data owner and their recourses along with the priority is stored in the priority table. Once a client request is given, the resource with higher priority can be accessed faster. Thus profit maximization is achieved.

VI. MODULE DESCRIPTION

Data Owner Interface:

The data owner interface is an interface between the data owner and the cloud server. The data owner is a person who stores the data source that are requested by the client into the cloud server. This is used to establish client and server connection using “http” protocol for client ease to interface with server. All Data owner stores data in cloud database with priority. Once data stored successfully in cloud, it responds with “URI” for respective data owner.

Set Priority:

Data owners can set priority to data source, all client requests are handled based on this priority. Data owners must be pay to set priority based on priority level. The data owner to pays the respected cost for the first priority is given the first priority. The cost for setting the priority is set by the cloud server. The data owner who chooses the first priority should pay the respected amount for setting the priority to the data source.

User Query:

For user Interface we develop a Cloud server that is responsible to handle all client requests. When user request to particular URI, first that request is processed in priority table. High priority URI’s are processed first then followed by other URI’s. URI’s are nothing but User Resource Identifier. User queries are used to connect to the database.
Check priority in priority table:
Priority table stores all data owner URI and related Priority in Tree format. This Table is very important to process all client requests based on priority level. The admin of the cloud server can only access the priority table and set the priority values to the data source based on the amount paid to the cloud server. The data owner stores the data source and chooses the priority to be set to the data source. The admin of the cloud server sets the priority to the data owner’s data source. The priority table can be viewed and edited only by the admin of the cloud server. The priority values can be reset or changed by the admin of the cloud server.

DQG renting scheme:
The Double-Quality-Guaranteed (DQG) resource renting scheme which combines long-term renting with short-term renting. The main computing capacity is provided by the long-term rented servers due to their low price. The short-term rented servers provide the extra capacity in peak period. The proposed DQG scheme adopts the traditional FCFS queuing discipline. For each service request entering the system, the system records its waiting time. The requests are assigned and executed on the long-term rented servers in the order of arrival times. Once the waiting time of a request reaches D, a temporary server is rented from infrastructure.

Algorithm 1 Double-Quality-Guaranteed (DQG) Scheme

1: A multi-server system with m servers is running and waiting for the events as follows
2: A queue Q is initialized as empty
3: Event – A service request arrives
4: Search if any server is available
5: if true then
6: Assign the service request to one available server
7: else
8: Put it at the end of queue Q and record its waiting time
9: end if
10: End Event
11: Event – A server becomes idle
12: Search if the queue Q is empty
13: if true then
14: Wait for a new service request
15: else
16: Take the first service request from queue Q and assign it to the idle server
17: end if
18: End Event
19: Event – The deadline of a request is achieved
20: Rent a temporary server to execute the request and release the temporary server when the request is completed
21: End Event

Rank Aggregation Algorithm:
The rank aggregation algorithm is used to process the user request and set request priority to them based on the priority set by the cloud server. Based on the priority of the request the request is processed. The request with first priority is processed first then the requests with next priorities are processed later.

Algorithm 2: Rank aggregation algorithm

1. Input: user-request;
2. Input: request-priority;
3. Output: request process;
4. for each (request: priority)
5. if
6. (request==1)
7. process first;
8. else if
9. (request==2)
10. process next;
11. else if
12. (request==0)
13. process last;
14. end if;
15. end foreach
VII. CONCLUSION

Swing's high level of flexibility is reflected in its inherent ability to override the native host 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 set Priority() method. Implementing the above concepts in our project to do the efficient work among the Server.

VIII. REFERENCE


