Zhou *et al. EURASIP Journal on Wireless Communications and Networking* (2015) 2015:71
DOI 10.1186/s13638-015-0256-y

EURASIP Journal on Wireless Communications and Networking
a SpringerOpen Journal

RESEARCH Open Access

Maximizing the profits of cloud service providers via dynamic virtual resource renting approach

Ao Zhou*, Qibo Sun, Lei Sun, Jinglin Li and Fangchun Yang

Abstract

Promoted by the leading industrial companies, cloud computing has gained widespread concern recently. With an increasing number of cloud service providers (CSPs) delivering services to customers from the cloud, maximizing the profits of CSPs becomes a critical problem. Existing approaches are difficult to solve the problem because they do not make full use of temporal price differences. This paper introduces a dynamic virtual resource renting approach that attempts to dynamically adjust the virtual resource rental strategy according to price distribution and task urgency. Considering task urgency and price distribution, we design a weak equilibrium operator to calculate the acceptable price for each type of virtual resource. All types of virtual resources that are at an acceptable price are inserted into a set. Then, a price prediction algorithm is presented to predict the price of virtual resources at the next price interval. Finally, we design a novel rental decision-making algorithm to select the most profitable resource from the set. We have implemented our approach and conducted experiments on both real and synthetic datasets. The results demonstrate that our approach obtain the better profit than other five approaches.

Keywords: Cloud computing; Profit maximization; Virtual resource renting; Non-uniform mutation operator

1 Introduction

Cloud computing is becoming increasing popular recently. With the advent of cloud computing, the dream that delivers computing as the fifth utility after water, electricity, gas, and telephony has come true [1]. Because cloud computing can allocate resources transparently and instantaneously as needed, it has gained widespread concern from academia and industry [2-4]. In cloud computing, cloud service providers rent virtual resources to host their applications. The pay-as-you-go model allows cloud service providers to be charged only for the resources they use, and these costs are much lower than making investments to build their own infrastructure. Hence, cloud computing can effectively reduce overhead and increase profits.

As shown in Figure 1, generally, there are three roles in the cloud computing environment: virtual resource supplier (VRS), cloud service provider (CSP), and end user (user). The VRS is responsible for the provision of large-scale virtual resources connected by the network. They offer different types of virtual resources and profit from virtual resource renting services. The CSP purchase the virtual resources from the VRS and process requests from the end-user. The user purchases services from the CSP to meet its needs. The CSPs charge the end-users for the services they provide.

Recently, cloud computing has been promoted by the leading industrial companies and has achieved rapid development in the industry. However, many challenges [5,6] remain before cloud computing becomes a proven commercial system. One of the key challenges is whether CSPs can earn higher profits after migrating their services to the cloud. Hence, maximizing the profits of a CSP is a critical issue.

Numerous schemes have been designed to increase the profit of CSP. There are varieties of virtual resources and pricing models in cloud computing environment; therefore, it is challenging for the CSP to choose the most profitable VRS. To solve the problem, Li et al. [7] proposed a VRS selection framework called CloudCmp. CloudCmp

*Correspondence: hellozhouao@gmail.com
State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China

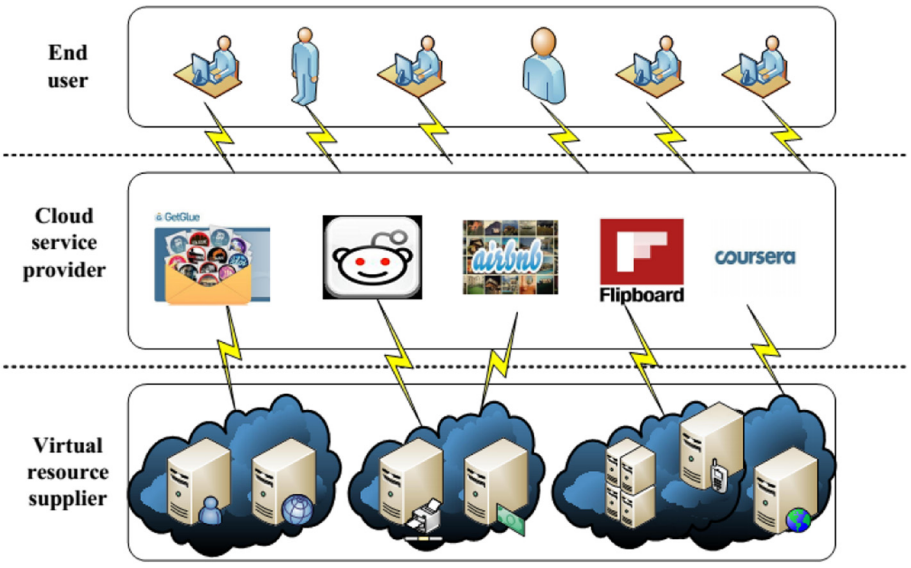


Figure 1 Business model of cloud computing.

uses a set of benchmarking tools to predict cost and performance when the same application is deployed on the virtual resources of different VRS. Deelman et al. [8] tried to study the trade-off between performance and cost by using a real-life astronomy application. The CSP can provide the right amount of computer and storage resources based on the results. Pandey et al. [9] attempted to minimize the execution cost of workflow on cloud computing environment. The work takes both computation and data transmission cost into consideration. A heuristic algorithm was proposed to solve the problem. Liu et al. [10] presented a virtual resource renting and request scheduling algorithm. The presented algorithm attempt to reduce the rental costs for CSP by improving resource utilization.

Without considering temporal price differences, aforementioned schemes are difficult to make the profit maximization in the dynamic virtual resource pricing model. To reduce the peak traffic and to achieve a higher utilization of the spare resources, academia and industry proposed the dynamic pricing model [11-15]. In the dynamic pricing model, the price of resource is not fixed, but changes dynamically and periodically based on current demand and supply. Hence, it usually determines the current prices of virtual resources through an auction.

Some existing works discussed how to increase the profit of CSP under dynamic virtual resource pricing model. Song et al. [16] investigated the optimal bidding strategy and proposed an approach to increase the profit of cloud service agents. By using the optimization theory, it selects virtual resources adaptively to maximize the average profit of the cloud service broker. Mazzucco et al. [17] presented an approach to maximize the revenue for CSPs who provide cloud web service. It supposes a virtual resource allocation and admission control policy to address the problem. Moreover, in order to achieve high performance and reduce the overhead of repeated execution, the work proposes an optimal price prediction algorithm. Zafer et al. [18] considered the profit maximization problem for parallel and serial job under dynamic virtual resource pricing model. A discrete-time stochastic dynamic programming formulation is used to format the problem. An optimization algorithm is also presented to obtain the optimal rental strategy.

As we know, a VRS provides a varying type of virtual resources with different prices. The CSP should determine the most profitable renting strategy when purchasing the virtual resources from the VRS. However, none of the existing works deal with a varying number of virtual resources with different types and prices. Because the prices of different resource types fluctuate inconsistently, the most profitable resource may change at different price intervals. Chen et al. [19] took advantage of this concept and rented different types of resources for the same task at different price intervals. But for delay-tolerant service, we find that the approach ignores rental cost saving brought about by two factors: (1) it fails to take full advantage of price distribution and still rents resources when prices for all types are high and (2) the acceptable price is not calculated in view of task urgency. The highest price a CSP can accept for a special resource type is called the acceptable price, as it is well known that the acceptable price is different for a task that needs to be completed one second later or one hour later. The more urgent the task is, the higher the acceptable price should be, and a less urgent task should imply a lower acceptable price. This paper makes full use of the two factors to reduce the rental cost

as long as the finish time satisfies service level agreement (SLA).

Therefore, in contrast to existing schemas, we propose a dynamic virtual resource renting approach to maximize the profits of cloud service providers. We only take computational resource into consideration. Therefore, virtual resources and computational resources are not identical in meaning in this paper. Based on our previous work [20], our approach takes into account task urgency and price distribution when calculating the acceptable price. If the prices of all types of resources are high with respect to task urgency, it is appropriate to postpone the execution of the task. Then, we can rent resources and restart the task when the price declines. Our contributions are as follows.

First, our approach saves on execution costs by pretreating the historical price series of virtual resources using time series analysis, and then the outlier detection technique is used to filter the extreme price.

Second, to calculate the highest rental rate that a CSP can accept for a special resource type, a weak equilibrium operator is designed by considering task urgency and price distribution. As such, a price prediction algorithm is proposed to predict the price of virtual resources at the next price interval.

Third, we propose a rental decision-making algorithm to decide on the type of virtual resources to rent for each task. In the algorithm, the type of virtual machine whose current price is lower than the acceptable price is inserted into a set. If the set is not empty, it rents the most profitable resource to process the task at the next price interval. Otherwise, if the SLA allows, the task will be suspended until the price falls.

Finally, we have been implemented our approach and conducted experiments on both real and synthetic datasets. We compare our approach with five other approaches, and the results show that our approach can obtain higher profits than other approaches.

The remainder of this paper is organized as follows. To better express the proposed approach, Section 2 introduces the background and provides a formalized definition of the problem. Section 3 presents the details of our approach. Section 3.4 shows the utility of our approach through simulation-based experimental results. Finally, we conclude the paper in Section 5.

2 Preliminaries

In order to most effectively outline the proposed approach, we first introduce the three roles of cloud computing environment and define some notations that will be used throughout the paper. The formalized definition of the problem is also given in the section. The notations in Table 1 will be used throughout the paper.

In this paper, if no special specification, we use T and t denote a task and a sampling time, respectively. Request from user and task are not identical in meaning. The highest rental rate a CSP can accept for a special resource type is called the acceptable price.

Table 1 Notations

Symbol	Meaning
R	Virtual resource
VM_i	A virtual machine
T_j	A task submitted by users
pl	Price list of a virtual machine
I	Price interval of a virtual machine
p	The price of a virtual machine
Size	The time a virtual machine takes to finish the task
N	The execution state of a task
e	The urgency of the current task
AP	The acceptable price
Decision	The virtual machine rental strategy decision
Revenue	The fee a CSP can obtain after it completes a task
Rental	The accumulated cost of renting virtual machines for processing a task
Profit	The profits that a CSP gains from completing a task

2.1 Three roles in the cloud computing environment

We now introduce the background and define the notations used throughout the paper.

- *Virtual resource supplier:* The VRS charges CSPs for renting its virtual resources to deploy the service. The virtual resource is supplied in the form of a virtual machine (VM). We use VM_i to denote all virtual machines of the same configuration (such as CPU type, memory size, and thus the same price) and i is the type index. VM_i is characterized by a three-parameter tuple:

$$VM_i = (\text{name}, t_c, p_l) \tag{1}$$

  where name is used as an identification, p_c denotes the current price of, and p_l (price list)denotes the historical prices. A VRS holds a sealed-bid auction or ascending bid auction in each price interval (this paper uses I_λ instead of a price interval and I_λ as the basic time unit) to determine the price of at the next I_λ. If a CSP wins the bid, it obtains the right to use the virtual machines for which it bid. The CSP is charged by the price interval. Then, the virtual resource (R) is given in Equation 2 denoting all virtual machines a virtual source supplier provides.

$$R = (VM_1, VM_2, \ldots, VM_n) \tag{2}$$

- *Cloud service provider:* To provide services in the cloud, the CSP needs to rent virtual resources from

the VRS to process the task requested by the user. We use T_j to denote a service request from the user and j is the index. T_j is characterized by the following:

$$T_j = (\{\text{size}(\text{VM}_i)|\forall VM_i \in R\}, \text{revenue}(T_j)) \quad (3)$$

where $\text{size}(\text{VM}_i)$, which can be predicted by some measuring tools, such as the one proposed in [7], denotes the number of price interval VM_i need to complete T_j, and $\text{revenue}(T_j)$ denotes the fee the end user should pay after the CSP completes T_j.

- *End user:* The user pays for the service received from the CSP. The fee that it should pay is determined by the SLA, which can be expressed as follows [10,21]:

$$\text{revenue}(T_j) = \begin{cases} r_{\text{pay}} & t_{\text{use}} \le t_{\min} \\ r_{\text{pay}} - \varphi * (t_{\text{use}} - t_{\min}) & t_{\min} \le t_{\text{use}} \le t_{\max} \\ -r_{\text{penlty}} & t_{\text{use}} \le t_{\min} \end{cases} \quad (4)$$

where r_{pay} is the maximum fee from serving a request of an end user, t_{use} is the number of I_λ that the cloud service provider uses to complete T_j. $t_{\min}$ is the average time required to finish T_j, which is proportional to the size of T_j and the delay tolerant level. φ is the decline rate, and a larger φ indicates that the profits decrease rapidly with the increase in completion time, thus the delay tolerant level of the task is low. When t_{use} exceeds $t_{\min}$, a SLA violation occurs. Hence, revenue decreases as $t_{\min}$ increase. r_{penlty} is the bottom line of compensation. Since there are many types of virtual machines, the average time required for processing T_j can be calculated using the following:

$$t_{\min} = \phi * \left\lceil \frac{t(\text{VM}_1) + t(\text{VM}_2) + \ldots + t(\text{VM}_n)}{n} \right\rceil \quad (5)$$

where ϕ is the delay tolerance factor and its value is larger than 1.

If no special specification, the task discussed in our paper is divisible, large-scale, and delay-tolerant task, such as scientific data processing, which takes more than one price interval to complete. A task that cannot be completed in the current price interval will join the auction of the next one.

2.2 Virtual resource rental problem

Based on this introduction, this section describes the virtual resource rental problem. Cloud service providers rent virtual resources to service users. The profits that generated from processing T_j can be calculated for CSPs using the following:

$$\text{profit}(T_j) = \text{revenue}(T_j) - \text{rental}(T_j) \quad (6)$$

where $\text{profit}(T_j)$ denotes the profits that the CSP gains from processing T_j and $\text{rental}(T_j)$ denotes the accumulated cost of renting virtual machines for processing T_j. An SLA violation can result in a loss of revenue. Hence, to maximize Equation 6, we need to minimize the rental cost but still comply with the SLA. Because the large-scale task cannot be completed within one I_λ, we should decide on the type of VM to rent for each incomplete task in every auction. The rental decision made for T_j in the current auction time t can be denoted by the following:

$$\text{Decision}(T_j) = (\text{VM}_k, p) \quad (7)$$

where VM_k denotes the type of VM that we rent to process T_j at the next I_λ and p' denotes the price of VM_k at the next I_λ . Because the prices of different types of VMs fluctuate inconsistently, the most profitable type may change at different price intervals. Hence, we may rent different types of VMs for the same task, which is similar to reference [19]. Because the VMs can share the same file system, moving the remaining workload of a divisible task to another VM is feasible. The startup latency for a cloud application is less than 100 s [22], much smaller than the task size; therefore, we ignore it. Based on this description, $\text{rental}(T_j)$ can be derived from:

$$\text{rental}(T_j) = \sum_{t=1}^{n} (\text{Decision}(T_j, t).p) \quad (8)$$

where n denotes the number of price intervals that the approach takes to finish T_j. Different from other approaches that still rent resources even if all types of VM are at the peak price, we believe that the right Decision $\text{Decision}(T_j, t)$ should suspend the task and restart it when the price falls, as follows:

$$\text{Decision}(T_j, t) = \begin{cases} (\text{VM}_{\text{none}}, 0) & \text{when the prices of all types} \\ & \text{of VMS are high} \\ (\text{VM}_{\text{k}}, 0) & \text{otherwise} \end{cases} \quad (9)$$

where $(\text{VM}_{\text{none}}, 0)$ denotes that we suspend renting resources to process T_j.

This introduction shows that we attempt to avoid the 'peak price' and attempt to rent resources when doing so is relatively inexpensive. When to suspend the execution of T_j and how to choose the most profitable resources are addressed in the next section.

3 Virtual resource renting approach

To rent the most profitable resource, this paper presents a dynamic virtual resource renting approach called DRAP. Taking full advantage of price distribution and task urgency, this approach attempts to rent virtual machines to process tasks when the rental rate is relatively inexpensive. As shown in Figure 2, DRAP contains the following

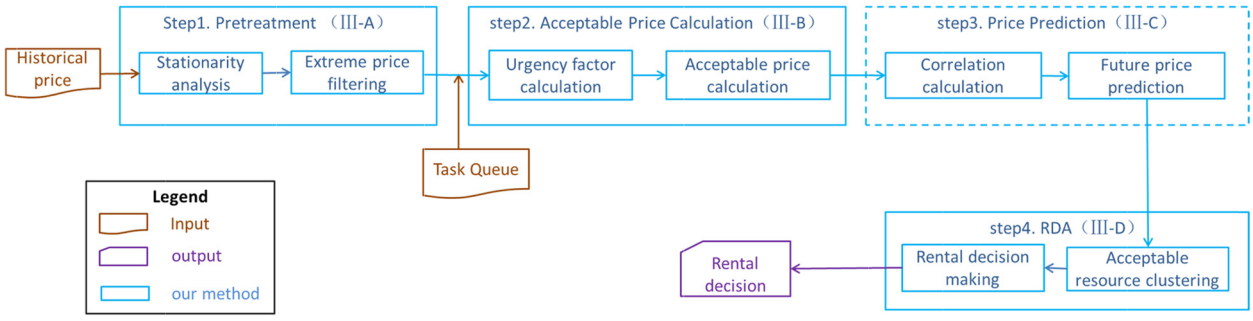


Figure 2 DRAP work procedures.

four phases: in phase 1 (Section 3.1), to save on execution costs for our approach, we test the stationarity of the price series using time series analysis. Moreover, we adopt an outlier detection technique to filter the extreme price from the series. In phase 2 (Section 3.2), taking into account task urgency and price distribution, we design a weak equilibrium operator to calculate the acceptable price. In phase 3 (Section 3.3), we predict the future price of each type of VM. Phase 4 (Section 3.4) presents our rental decision-making algorithm (RDA) to select the most profitable VM. In our approach, the first two steps are semi-offline, and the last two steps are online.

3.1 Pretreatment

We first pretreat the historical prices before executing the follow-up steps. The series of observed historical prices of a special VM type VM_i is called the price series ($\mathrm{PS}(\mathrm{VM}_i)$).

In time series analysis, if the mean, variance, and autocorrelation of the series do not change over time, the series is stationary [23]. If the price series is stationary, the price series processing result in the latter step is available for multiple auctions. We use this result to lower costs and save time.

However, practically, a strictly stationary sequence does not exist. We simply focus on the 'daily weakly stationary' price series, which implies that the mean and standard deviation of prices do not change much on different days. Therefore, in this paper, as long as the price series during a period satisfies (10), we recognize that the price series is daily weakly stationary:

$$\frac{\mathrm{std}(E(1,n))}{E(1,n)} \le 10^{-2} \wedge \frac{\mathrm{std}(\mathrm{std}(1,n))}{\mathrm{std}(1,n)} \le 10^{-2} \tag{10}$$

$$E(E(1,n)) = \frac{1}{n/k}\sum_{i=0}^{n/k} E(i*k,(i+1)*k) \tag{11}$$

$$\mathrm{std}(E(1,n))^2 = \frac{1}{n/k}\sum_{i=0}^{n/k}(E(i*k,(i+1)*k) - E(E(1,n)))^2 \tag{12}$$

$$E(\mathrm{std}(1,n)) = \frac{1}{n/k}\sum_{i=0}^{n/k} \mathrm{std}(i*k,(i+1)*k) \tag{13}$$

$$\mathrm{std}(\mathrm{std}(1,n))^2 = \frac{1}{n/k}\sum_{i=0}^{n/k}(\mathrm{std}(i*k,(i+1)*k) - E(\mathrm{std}(1,n)))^2 \tag{14}$$

where k is the number of price intervals in a day and E and std denote the mean and standard deviation, respectively.

Sometimes, there is a long tail in the price distribution graph that is affected by unexpected factors. In other words, the low probability exists for an extremely low or high price to occur. The acceptable price rises as task urgency increases. If the starting value of the acceptable price is the lowest price, the chance that the current price is lower than the acceptable price is too small for the long tail. Therefore, the task cannot be processed until the acceptable price increases to fall within the normal range. To avoid the 'slow start symptom', we need to first confirm the existence of the extreme price. The Pauta criterion [24] identifies the abnormal values of the sample data and is suitable to our problem. Therefore, if the highest price $p_{\max}$ or the lowest price $p_{\min}$ meets condition (15), we can recognize the existence of the extreme price. Otherwise, all prices are absolutely not extreme prices.

$$\left(\frac{p_{\min} - E(1,n)}{\mathrm{std}(1,n)} < -3\right) \cup \left(\frac{p_{\max} - E(1,n)}{\mathrm{std}(1,n)} > 3\right) \tag{15}$$

To filter the extreme price, we calculate the lower cut-off $l_{\min}(\mathrm{VM}_i)$ and the upper cut-off $l_{\max}(\mathrm{VM}_i)$. These prices are lower than $l_{\min}(\mathrm{VM}_i)$ or higher than $l_{\max}(\mathrm{VM}_i)$ would be treated separately in our approach. The outlier detection [23] can effectively eliminate exceptions in the actual data; hence, we adopt outlier detection to calculate the value of $l_{\min}(\mathrm{VM}_i)$ and $l_{\max}(\mathrm{VM}_i)$. For each type of VM, we first sort the price series in ascending order. Suppose

the result is $(p(1), p(2) \ldots p(\max))$; we can use the following equations to calculate $l_{\min}(\mathrm{VM}_i)$ and $l_{\max}(\mathrm{VM}_i)$:

$$l_{\min} = \begin{cases} p(\mathrm{floor}(m)+1) - 1.5 * Q & m \notin Z \\ (p(m) + p(m+1))/2 - 1.5 * Q & \text{other} \end{cases} \quad (16)$$

$$l_{\max} = \begin{cases} p(\mathrm{floor}(n)+1) + 1.5 * Q & n \notin Z \\ (p(n) + p(n+1))/2 + 1.5 * Q & \text{other} \end{cases} \quad (17)$$

where $m = 0.25$*length(PS), $n = 0.75$*length(PS), length(PS) returns the length of PS, floor(m) denotes the maximum integer that is less than m, and Q denotes the interquartile range [23]. If no price meets the condition of (15), $l_{\min}(\mathrm{VM}_i)$ and $l_{\max}(\mathrm{VM}_i)$ are set as the lowest and the highest prices of PS(VM_i) , respectively.

3.2 Acceptable price calculation

In this section, we calculate the acceptable price for T_j based on its urgency. The goal of this section is expressed by the following:

$$f : e(T_j, \mathrm{VM}_i) \rightarrow [\, l_{\min}(\mathrm{VM}_i), l_{\max}(\mathrm{VM}_i)] \,, \mathrm{VM}_i \in R \quad (18)$$

As shown in Equation 18, for each type of resource VM_i and task T_j, we want to map the task urgency factor (e) of T_j to a price p that belongs to the interval [$l_{\min}(\mathrm{VM}_i)$, $l_{\max}(\mathrm{VM}_i)$]. The price represents the highest rental rate of VM_i that the CSP can afford to pay based on the urgency of the current task. If the current price of VM_i is lower than p, we can consider renting VM_i to process T_j.

We first give the definition of the current status of T_j before introducing e. The current status of T_j , denoted by $N(T_i)$, is given as follows:

$$N\left(T_j\right) = (t(\mathrm{VM}_i), t_{\mathrm{deadline}}) \quad (19)$$

where $t(\mathrm{VM}_i)$ denotes the time required if the rest workload of T_j is processed by VM_i and t_{deadline} denotes the remaining time to deadline.

Suppose the current status of T_j is $N(T_j) = (t(\mathrm{VM}_i), t_{\mathrm{deadline}}$, the current task urgency factor is obtained as:

$$e(T_j, \mathrm{VM}_i) = ((N(T_j).t_{\mathrm{VM}_i})/(N(T_j).t_{\mathrm{deadline}}) \quad (20)$$

As e approaches 0, the task becomes more urgent; as e is further from 0, the task becomes less urgent. If e equals 1, the remaining time to the deadline equals the required time if T_j is processed by VM_i.

We can learn from the definition that the acceptable price is in proportion to e. We design a weak equilibrium operator to fit the mapping f in (18), which uses an exponential function and a non-uniform mutation operator [25]. The exponential function can control the overall shape of its curve, and the non-uniform mutation operator is used to locally fine-tune the value of the operator. The weak equilibrium operator is expressed as follows:

$$\Delta = \exp\left(-\frac{K-e}{\delta}\right) + \left(1 - \mathrm{rand}(0,1)^{(1-\frac{K-\varepsilon}{k})*\delta}\right) \quad (21)$$

where K is the max value of e and rand$(0,1)$ is a random value between 0 and 1. Figure 3 shows the curve of the weak equilibrium operator under different values of δ and ε. We observe that a greater δ results in a higher curve. Moreover, as ε increases, the adjustment increases.

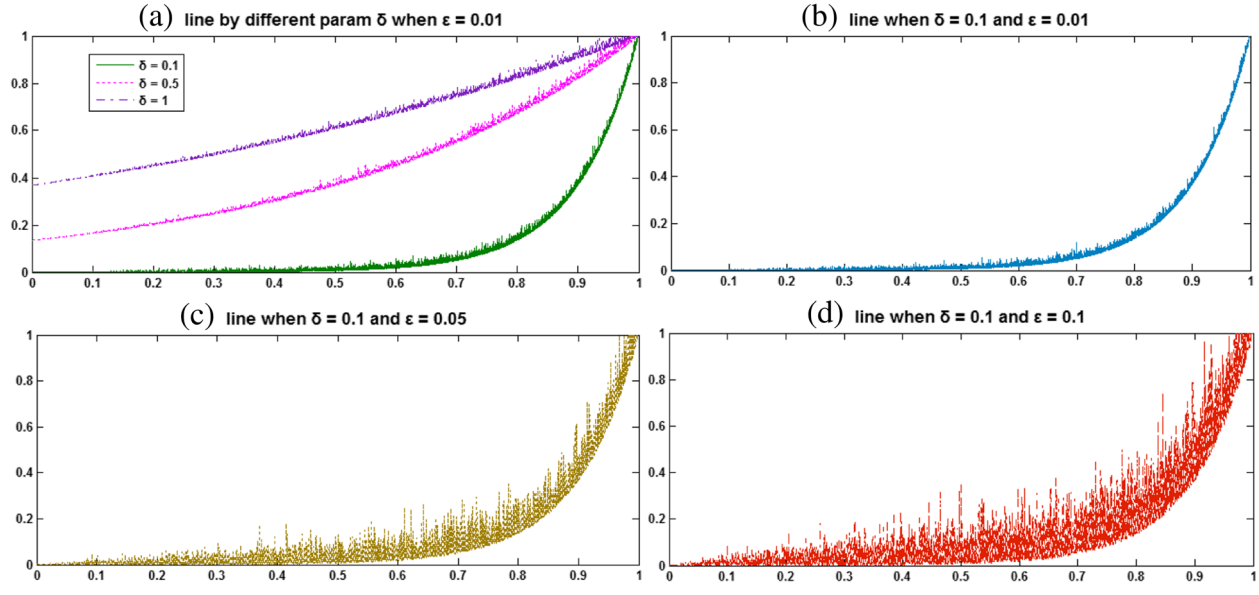


Figure 3 Graph of non-uniform mutation operator under different parameter values. (a) Line by different param δ when $\varepsilon = 0.01$. **(b)** Line when $\delta = 0.1$ and $\varepsilon = 0.01$. **(c)** Line when $\delta = 0.1$ and $\varepsilon = 0.05$. **(d)** Line when $\delta = 0.1$ and $\varepsilon = 0.1$.

Based on the weak equilibrium operator, we calculate the acceptable price of VM_i by using the following:

$$\mathrm{AP}(\mathrm{VM}_i) = l_{\min} + (l_{\max} - l_{\min}) * \left[\exp\left(-\frac{K-e}{\delta}\right) + \left(1 - \mathrm{rand}(0,1)^{\left(1-\frac{K-\varepsilon}{k}\right)*\delta}\right)\right] \tag{22}$$

where $l_{\min}(\mathrm{VM}_i)$ is the upper cut-off and $l_{\max}(\mathrm{VM}s_i)$ is the lower cut-off. The values of δ and ε can be trained using historical prices.

3.3 Future price prediction algorithm

In this section, we predict the future price of each type of VM. This step is needed only if the VRS holds a sealed bid auction [15] to determine the price of VMs, e.g., Amazon EC2. If the VRS adopts an ascending bid auction [14], this step can be skipped. We use the prices the VRS declares to determine our rental strategy. Many researchers have exploited properties of the Amazon EC2 historical price, and we just adopt the proposed prediction algorithm in this step [17].

In order to predict the future price, we first calculate the autocorrelation coefficient [23] of historical prices. The value denotes the correlation of historical prices with itself at different time point. The value of autocorrelation function (ACF) lies between -1 and 1, with -1 indicating anti-correlation, 0 indicating no correlation, and 1 indicating perfect correlation. If the autocorrelation coefficient is higher than 0.4, then the future prices are predicted by using linear regression [23]; if the autocorrelation coefficient is lower than 0.4, the future prices are predicted by using the inverse cumulative distribution function of the normal distribution. The inverse cumulative distribution function return the value of x such that $P(\text{price} < x)$ with the availability target p. We set p as 0.99 to obtain a high level of availability.

We will introduce our rental decision-making algorithm in the next section.

3.4 Rental decision-making algorithm

Based on these steps, we propose a rental decision-making algorithm called RDA (Algorithm 1). Before the auction, all incomplete tasks are inserted into a queue. RDA makes decisions on the type of VM to rent for each task in the queue. In the algorithm, the type of virtual machine whose current price is lower than the acceptable price is inserted into a set. If the set is not empty, it rents the most profitable resource to process the task at the next price interval. Otherwise, while allowed by the SLA, the task is suspended until the price declines. Apart from the penalty caused by SLA violations, a task that fails to be completed in time affects the quality of service (QoS). Therefore, we would like to complete every task within $t_{\min}$.

Algorithm 1 describes our rental decision-making algorithm. RDA traverses the task queue and makes rental decisions for each task using the following steps:

Step 1. RDA first initializes the rental decision list
Step 2. If the task queue is empty, we go to step 10. Otherwise, we fetch the head task T_j in the task queue, remove it from the queue.
Step 3. We calculate the weighted urgency factor e_{ave} of T_j. t_{ave} in lines 4 denotes the average time required to finish T_j l.
Step 4. If the weighted urgency factor is equal to 1, the task is very urgent. To complete T_j in time, all types of VMs that can finish T_j in time are added to the price-acceptable resource set (lines 5 to 6).
Step 5. If the weighted urgency factor is less than 1, RDA traverses the resource set R.
Step 6. If all VMs in R have been visited, we go to step 9. Otherwise, we assign the next VM in the set to VM_i .
Step 7. If VM_i can finish T_j in time and the current price of VM_i is less than $l_{\min}(\mathrm{VM}_i)$, VM_i is added to the price-acceptable resource set. We now go back to step 6 (lines 11 to 12).
Step 8. If VM_i can finish T_j in time and its current price is higher than $l_{\min}(\mathrm{VM}_i)$, RDA calculates the acceptable price of using (22). If the current price of VM_i is lower than the acceptable price, VM_i is added to the price-acceptable resource set. We now go back to step 6 (lines 14 to 18).
Step 9. If the acceptable-price resource set is not empty, RDA will sort the set by rental rate per unit computing power. The head VM is chosen to rent in the following price interval. We now go back to step 2 (lines 22 to 26).
Step 10. If the set is empty, we suspend renting VM for T_j. Therefore, $(\mathrm{VM}_{\text{none}}, 0)$ is added to the DL (line 28). We now go back to step 2.
Step 11. All rental decisions in DL are combined as the final rental strategy (line 31).

Therefore, based on the preceding steps, we obtain a virtual resource renting approach that can maximize the profits of a cloud service provider and satisfy the SLA.

The time complexity of RDA is linear to the number of requests in the queue. To reduce the execution time of our algorithm, we divide it into sub-queues and schedule them to the distributed computing node.

Algorithm 1: Rental decision-making algorithm

Input: current price $\{p_1,p_2,\ldots,p_n\}$ of each VM type$\{\mathrm{VM}_1, \mathrm{VM}_2 \ldots \mathrm{VM}_n\}$,state of each task T_j in the task queue

Output: Decision list DL that stores the rental decision of each T_j

```
Initialize rental decision list DL;
for each T_j in task queue do
    Initialize the acceptable VM list AL;
    Calculate the weighted urgency factor e_ave for T_i
    by using: t_ave = (N(T_j)→t(VM_1)+...+N(T_j)→t(VM_n))/n
    e_ave = N(T_j).t_deadline / t_ave ;
    if e_ave ==1 then
        For any VM_i in R, add VM_i to AL if
        (N(T_j).t(VM_i))<t_min ;
    end
    else
        for each VM type VM_i do
            Calculate l_min(VM_i),l_max(VM_i);
            if p_j < l_min(VM_i) and
            (N(T_j).t(VM_i))<t_min then
                Add VM_j to AL ;
            end
            if p_j > l_min(VM_i) and p_j < l_max(VM_i)
            and (N(T_j).t(VM_i)) < t_min then
                PMAX=acceptPrice(N(T_j),VM_i);
                if p_j < PMAX then
                    Add VM_j to AL ;
                end
            end
        end
    end
    if length(AL)≠0 then
        Rank AL according to p_i/(T_j.size(VM_i)) by
        desc;
        Add the head of AL named VM_top as target
        rental type;
        Add the decision(VM_top, p_ctop) to DL
    end
    else
        Rent none VM for T_j ,and add (VM_none, 0) to
        DL;
    end
end
Return DL
```

4 Performance evaluation

The performance of our approach is evaluated using our three-layer simulator developed in java. The following sections first describe the experimental setting. Then, we compare five other approaches in terms of profit, rental cost, and execution time. Finally, we study the parameters of our approach.

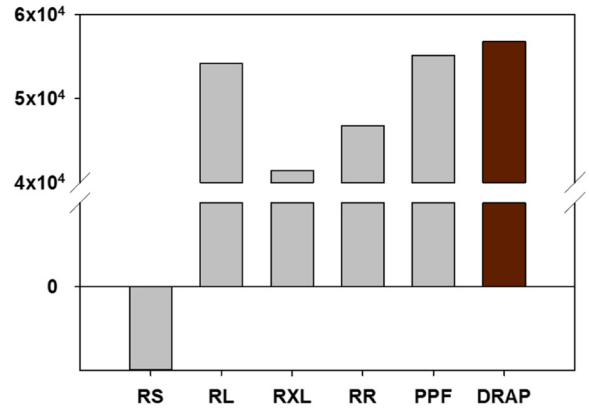
4.1 Experiment setup

Comparisons are carried out under the same experimental environment. The CPU is the Intel Core Duo T2250, memory size is 1.99 GB, and the operating system is Windows 7.

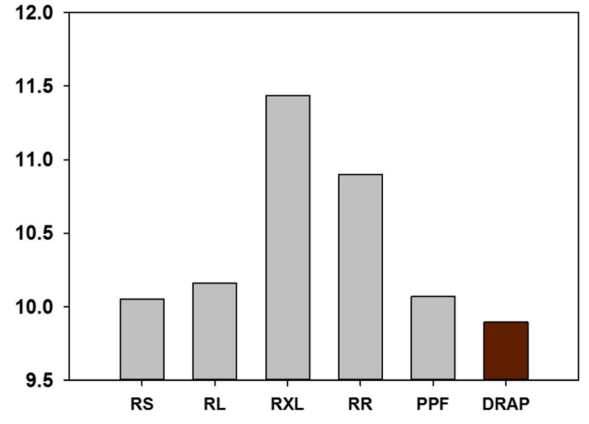
To effectively evaluate the performance of DRAP, we first build a three-layer simulator that contains a virtual resource-supplying layer, a cloud service providing layer, and a service-requesting layer. Table 2 shows that the configuration of virtual machines is the same as the Amazon EC2 small instance, large instance, and extra large instance [15]. We experiment with two types of datasets: a generated dataset with normally distributed prices generated by Matlab and the Amazon EC2 spot price dataset. We set the price of the next I_λ for each VM type according to the dataset. For all experiments on Amazon spot prices, the prices from May 20, 2011 to May 31, 2011 are used to train the values of δ and ε. We also generate 10 days of training prices by

Table 2 Experimental configuration

Type	Parameter	Value
Service requesting layer	Number	10,000
	Arrive rate	Poisson task at the rate of 80 per I_λ Measured by small instance, the size is uniformly distributed
	Task size	between 100 I_λ to 300 I_λ, that is 60 I_λ to 160 I_λ when measured by extra large instance
Service providing layer	Rental approach	RS, RL, RXL, RR, FFP, DRAP
Virtual resource supplying layer	Virtual machine	Small instance, large instance, xlarge instance with the same configuration as Amazon small, large, xlarge spot instance
	Price	Dataset 1: spot prices of Amazon small, large, xlarge instance from June 1, 2011 to June 30, 2011 in eastern United States.
		The OS is Windows
		Dataset 2: normally distributed prices. The mean and variance is equal to Dataset 1
SLA	r_{pay}	\$ 0.078 per unit size
	φ	0.8
	ϕ	1.1
	r_{penlty}	0
Parameter of weak equilibrium operator	δ	0.9 and 0.8, respectively, for dataset1 and dataset2
	ε	0.01 and 0.05, respectively, for dataset1 and dataset2



(a) Profit($)



(b) Rental($)

Figure 4 Results under Amazon EC2 historical prices. (a) Profit ($). **(b)** Rental ($).

Matlab for all experiments on normally distributed prices. The service fee is set as the average rental cost plus 50%. $t_{\min}$ is set as average executing time plus 10%, and the revenue decline rate is 0.8. All of the tasks arrive in a Poisson process at the rate of 80 per price interval.

We compare our approach with five other approaches. The first one always rents the small instance in the auction (RS). The second one always rents the large instance in the auction (RL). The third one always rents the extra large instance in the auction (RXL). The fourth randomly chooses a type to rent from the three VM types. Finally, we compare our approach with the 'First Fit Profit' (FFP) [19]. FFP rents the current cheapest virtual machine in the auction.

Table 3 Result under normally distributed price

Approach type	Rental ($)	Profit ($)
RS	10.05	-100,500
RL	10.1279	54,541
RXL	11.4382	41,438
RR	10.8864	46,956
FFP	10.0473	55,347
DRAP	9.8901	56,908

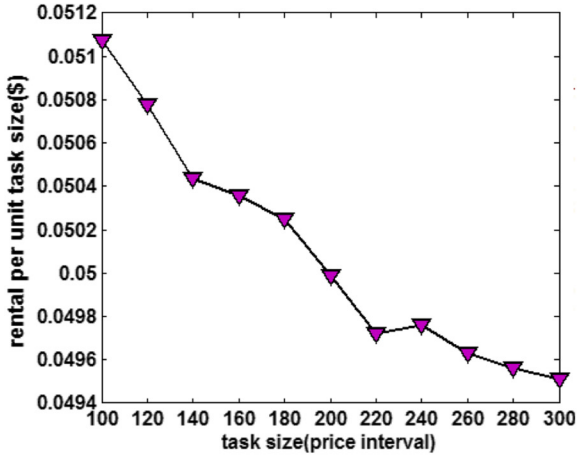
We evaluate our approach using the following performance metrics:

- *Average unit rental cost per task (rental):* The average rental rate paid by CSP for the processing of task Ti. Rental can be calculated as below:

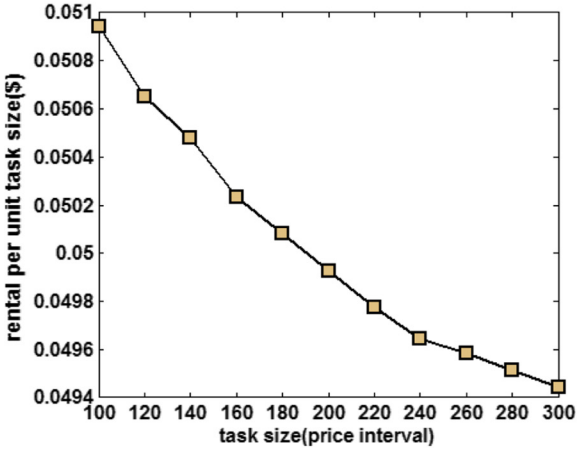
$$\text{Rental} = \frac{\sum_{i=1}^{n}[\,\text{rental}(T_i)]}{n} \tag{23}$$

- *Total profit (profit):* Total profit gained by CSP after completes T_i. Profit can be obtained as follows:

$$\text{Profit} = \sum_{i=1}^{n}\text{revenue}(T_i) - \sum_{i=1}^{n}\text{rental}(T_i) \tag{24}$$



(a) Result on Amazon EC2 historical prices



(b) Result on normally distributed prices

Figure 5 Performance of DRAP under different task sizes. (a) Result on Amazon EC2 spot prices. **(b)** Result on normally distributed prices.

- *Average time difference (Time-Diff):* The average difference between deadline and task completion time. Time − Diff is given by:

$$\text{Time} - \text{Diff} = \frac{\sum_{i=1}^{n}[\,t_{\text{deadline}}(T_i) - t_{\text{finish}}(T_i)]}{n} \tag{25}$$

4.2 Experiment results based on the Amazon EC2 dataset

We first study the performance of the various approaches on Amazon EC2 spot prices. The request arrival rate follows a Poisson distribution, as shown in Table 1. The results are shown in Figure 4.

As Figure 4a shows, our approach achieves higher profits than other approaches. Figure 4b also shows that our approach experiences a significant gain in rental cost reduction. We need to note that a lower rental cost does not mean a higher profit. Figure 4b shows that the rental cost of RS is lower than RL, RXL, RR, FFP, and DRAP. However, Figure 4a illustrates that the average profit of RS is much lower than other approaches except DRAP. That is because the computing power of the small instance is weaker than other instances. Therefore, RS takes a much longer time to complete the same task, which may result in numerous of SLA violations. DRAP pays the lowest rental rate but satisfies SLA. Hence, it can obtain higher profits than other approaches.

Because considering the price distribution and the task urgency, we appropriately postponed the execution of the task when complying with the SLA; therefore, the task completion time is closer to the deadline. Overall, our approach can escape from the peak price and enable renting of VMs when they are inexpensive, thus reducing the rental cost and increasing the profits of a CSP.

4.3 Experiment results based on generated dataset

The experiment focuses on comparing the performance of all approaches on normally distributed prices, and the configuration is the same as in Section 4.2. Table 3 shows the result.

The first column shows that the average rental cost ($9.8901) of our approach is much lower than that of

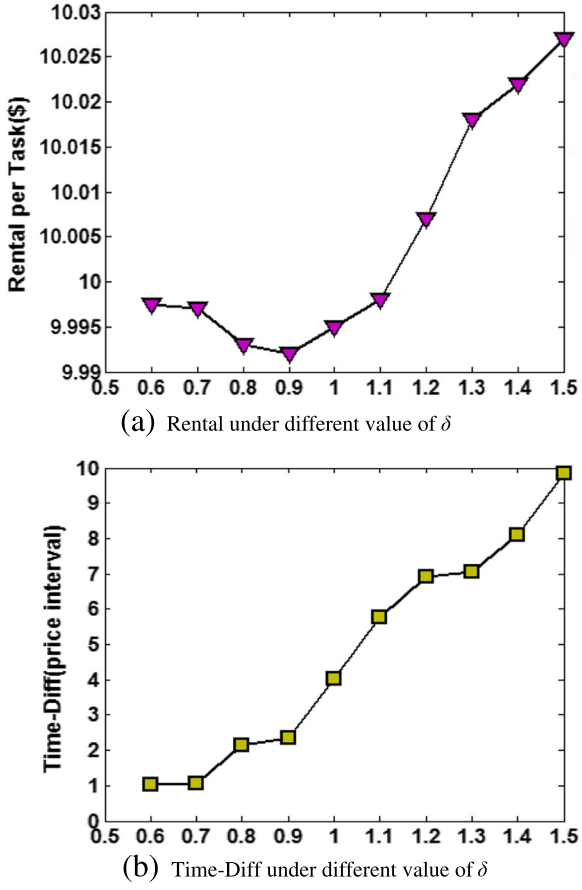


Figure 6 DRAP performance under different δ values. (a) Rental under different value of δ. **(b)** Time-Diff under different value of δ.

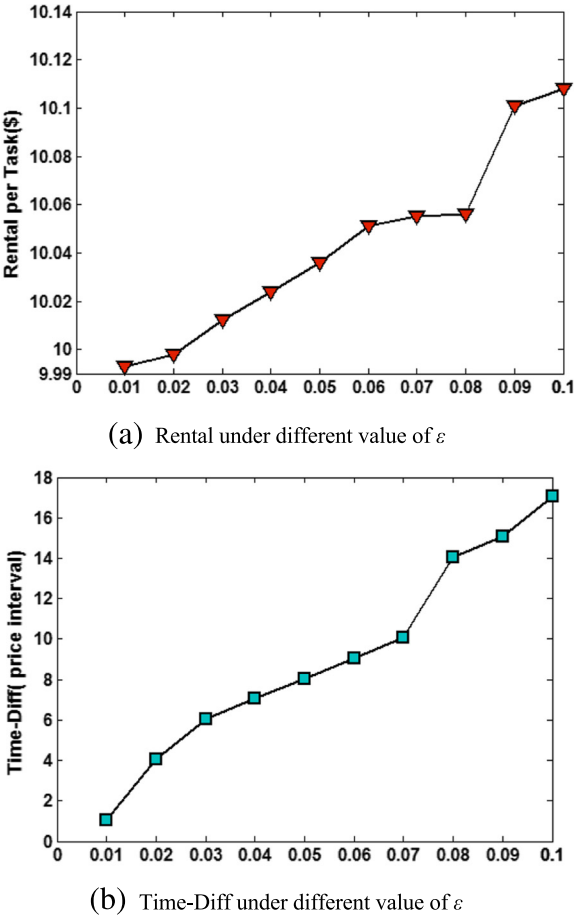


Figure 7 DRAP performance under different ε values. (a) Rental under different value of ε. **(b)** Time-Diff under different value of ε.

other approaches. The second column shows that our average profit ($5.6908) is the highest. In other words, DRAP can make full use of the delay tolerance to avoid the price and reduce the rental cost, which can improve profits.

4.4 Study the paramenters

This section studies the parameters of our approach. First, we investigate how task size affects our approach. Next, we attempt to determine how the parameters δ and ε of the non-uniform mutation operator affect DRAP.

4.4.1 Effect of parameter task size

We now investigate how task size affects our approach. Task size is the price interval that the small instance needs to finish the task. Figure 5a displays the result using the Amazon spot prices. Figure 5b shows the result using the normally distributed prices. The Y-axis represents rental cost per unit task size, which can be calculated from dividing the total rental cost by the task size.

Figure 5 shows that the rental cost per unit task size is affected when task size is varied. The rental cost falls with an increase in task size. That is because this increase provides a greater opportunity for us to avoid a high price, enabling the task to be processed when the rental rate is relatively low. Hence, the rental cost per unit task size decreases and we earn higher profits from processing the task.

4.4.2 Effect of parameter δ and ε

To study how parameters δ and ε affect DRAP, we conduct the experiment on the Amazon spot prices. In the experiment, the task size is set to 200 I_λ. Figure 6 depicts the performance of our approach when ε is 0.01. When δ is less than 0.8, the rental cost is observed to decline with an increase in δ. However, when δ exceeds 0.8, rental cost increases with an increase in δ. Time-Diff is also on an upward trend.

We discern the reason from Figure 6b. With an increase in δ, our approach is more afraid that the task cannot be completed on time. In other words, when the task urgency stays unchanged, the acceptable price increases as δ increases. Therefore, a higher chance exists that the current price is lower than the acceptable price, leading to an earlier completion time. An earlier completion time results in losing the chance to avoid the peak price, and the rental cost increases. However, this increase does not mean that smaller value of δ is better. When the value of δ is very small, it is unwilling to rent resources unless the price is very low. Therefore, other things being equal, the acceptable price declines while δ decreases. Because the chance is too small that the current price is lower than the acceptable price, the task is always suspended. As time goes by, to complete the task on time, the acceptable price increases as task urgency dramatically increases. Therefore, when δ is 0.6 or 0.7, the rental cost is higher than when δ is 0.8, as in Figure 6a.

Figure 7 depicts the performance of our approach when δ is 0.9. δ is used to control the overall shape of the weak equilibrium operator, and ε is used to locally fine-tune the value of the operator. The two have an effect similar to our approach. As Figure 7 shows, the rental cost and Time-Diff also show an upward trend as ε increases.

5 Conclusions

This paper studied the profit maximization problem for cloud service providers using a dynamic pricing model. Taking full advantage of the temporal price differences and the delay tolerance of the task, this paper presents a dynamic virtual resource renting approach. We have experimented on Amazon EC2 spot prices as well as normally distributed prices. The experimental result shows that the proposed approach can reduce rental costs and maximize profits of cloud service providers.

Competing interests

The authors declare that they have no competing interests.

Acknowledgements

The work presented in this study is supported by NSFC (61272521), SRFDP (20110005130001), the Fundamental Research Funds for the Central Universities (2014RC1101), and Beijing Natural Science Foundation (4132048).

Received: 24 September 2014 Accepted: 7 January 2015
Published online: 13 March 2015

References

1. R Buyya, CS Yeo, S Venugopal, J Broberg, I Brandic, Cloud computing and emerging it platforms: vision, hype, and reality for delivering computing as the 5th utility. Future Generation Comput. Syst. **25**(6), 599–616 (2009)
2. M Armbrust, A Fox, R Griffith, AD Joseph, R Katz, A Konwinski, G Lee, D Patterson, A Rabkin, I Stoica, A view of cloud computing. Commun. ACM. **53**(4), 50–58 (2010)
3. S Marston, Z Li, S Bandyopadhyay, J Zhang, A Ghalsasi, Cloud computing: the business perspective. Decision Support Syst. **51**(1), 176–189 (2011)
4. S Wang, Q Sun, H Zou, F Yang, Towards an accurate evaluation of quality of cloud service in service-oriented cloud computing. Springer J. Intell. Manuf. **25**(2), 283–291 (2014)
5. Q Zhang, L Cheng, R Boutaba, Cloud computing: state of the art and research challenges. J. Internet Services Appl. **1**(1), 7–18 (2010)
6. A Khajeh-Hosseini, I Sommerville, I Sriram, Research challenges for enterprise cloud computing. arXiv preprint arXiv:1001.3257 (2010)
7. A Li, X Yang, S Kandula, M Zhang, in *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*. Cloudcmp: shopping for a cloud made easy (USENIX Association USENIX Association, Seattle, 2010), pp. 5–5
8. E Deelman, G Singh, M Livny, B Berriman, J Good, in *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*. The cost of doing science on the cloud: the montage example (IEEE Press, Piscataway, 2008), p. 50
9. S Pandey, L Wu, SM Guru, R Buyya, in *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference On*. A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments (IEEE, Piscataway, 2010), pp. 400–407
10. Z Liu, S Wang, Q Sun, H Zou, F Yang, Cost-aware cloud service request scheduling for SaaS providers. Comput. J. **57**(2), 291–301 (2014)
11. D Niu, C Feng, B Li, in *INFOCOM, 2012 Proceedings IEEE*. A theory of cloud bandwidth pricing for video-on-demand providers (IEEE, Piscataway, 2012), pp. 711–719

12. Q Wang, K Ren, X Meng, in *INFOCOM, 2012 Proceedings IEEE*. When cloud meets eBay: towards effective pricing for cloud computing (IEEE, Piscataway, 2012), pp. 936–944
13. M Stokely, J Winget, E Keyes, C Grimes, B Yolken, in *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium On*. Using a market economy to provision compute resources across planet-wide clusters (IEEE, Piscataway, 2009), pp. 1–8
14. CloudBay. http://www.s3lab.ece.ufl.edu/projects/
15. Amazon Spot Instance. http://aws.amazon.com/ec2/
16. Y Song, M Zafer, K-W Lee, in *INFOCOM, 2012 Proceedings IEEE*. Optimal bidding in spot instance market (IEEE, Piscataway, 2012), pp. 190–198
17. M Mazzucco, M Dumas, in *High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference On*. Achieving performance and availability guarantees with spot instances (IEEE, Piscataway, 2011), pp. 296–303
18. M Zafer, Y Song, K-W Lee, in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference On*. Optimal bids for spot VMs in a cloud for deadline constrained jobs (IEEE, Piscataway, 2012), pp. 75–82
19. J Chen, C Wang, BB Zhou, L Sun, YC Lee, AY Zomaya, in *Proceedings of the 20th International Symposium on High Performance Distributed Computing*. Tradeoffs between profit and customer satisfaction for service provisioning in the cloud (ACM, New York, 2011), pp. 229–238
20. A Zhou, S Wang, Q Sun, H Zou, F Yang, in *Parallel and Distributed Systems (ICPADS), 2013 International Conference On*. Dynamic virtual resource renting method for maximizing the profits of a cloud service provider in a dynamic pricing model (IEEE, Piscataway, 2013), pp. 118–125
21. BN Chun, DE Culler, in *Cluster Computing and the Grid, 2002. 2nd IEEE/ACM International Symposium On*. User-centric performance analysis of market-based cluster batch schedulers (IEEE, Piscataway, 2002), pp. 30–30
22. J Zhu, Z Jiang, Z Xiao, in *INFOCOM, 2011 Proceedings IEEE*. Twinkle: a fast resource provisioning mechanism for internet services (IEEE, Piscataway, 2011), pp. 802–810
23. GE Box, GM Jenkins, GC Reinsel, *Time Series Analysis: Forecasting and Control*. (John Wiley & Sons, Hoboken, 2013)
24. S Bernstein, R Bernstein, *Schaum's Outline of Elements of Statistics I: Descriptive Statistics and Probability*. (McGraw Hill Professional, New York, 1999)
25. Z Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*. (Springer, Berlin, 1996)