Maximizing the Profit of Cloud Broker with Priority Aware Pricing

Xinhou Wang[1], Song Wu[1], Kezhi Wang[2], Sheng Di[3], Hai Jin[1], Kun Yang[4], Shumao Ou[5]
[1]Services Computing Technology and System Lab, Cluster and Grid Computing Lab
School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, 430074, China
[2]Northumbria University, UK [3]Argonne National Laboratory, USA [4]University of Essex, UK [5]Oxford Brookes University, UK
E-mails: [1]{xwang, wusong, hjin}@hust.edu.cn [2]kezhi.wang@northumbria.ac.uk [3]sdi1@anl.gov [4]kunyang@essex.ac.uk [5]sou@brookes.ac.uk

***Abstract*—A practical problem facing *Infrastructure-as-a-Service* (IaaS) cloud users is how to minimize their costs by choosing different pricing options based on their own demands. Recently, cloud brokerage service is introduced to tackle this problem. But due to the perishability of cloud resources, there still exists a large amount of *idle resource waste* during the reservation period of reserved instances. This *idle resource waste* problem is challenging cloud broker when buying reserved instances to accommodate users' job requests. To solve this challenge, we find that cloud users always have low priority jobs (e.g., non latency-sensitive jobs) which can be delayed to utilize these idle resources. With considering the priority of jobs, two problems need to be solved. First, how can cloud broker leverage jobs' priorities to reserve resources for profit maximization? Second, how to fairly price users' job requests with different priorities when previous studies either adopt pricing schemes from IaaS clouds or just ignore the pricing issue. To solve these problems, we first design a fair and priority aware pricing scheme, *PriorityPricing*, for the broker which charges users with different prices based on priorities. Then we propose three dynamic algorithms for the broker to make resource reservations with the objective of maximizing its profit. Experiments show that the broker's profit can be increased up to $2.5\times$ than that without considering priority for offline algorithm, and $3.7\times$ for online algorithm.**

***Index Terms*—Brokerage; Priority; Fairness; Pricing; Resource reservation;**

I. Introduction

Infrastructure-as-a-service (IaaS) cloud providers, such as Amazon EC2 [1], offer different pricing schemes to users at different commitment levels. The most popular ones could be pay-as-you-go (e.g., *on-demand instance*) and subscription (e.g., *reserved instance*). The former one allows users to pay a fixed price for instances or *virtual machines* (VMs) per billing cycle without any commitments, while the latter one requires users to pay a onetime upfront fee for a time period (e.g., one month) [1], [2]. Normally, the price of reserved instance is cheaper than that of on-demand instance in terms of billing cycle, but it is not cost-effective if users only use reserved instances for a short time. Thus, it is of great importance for users to understand their demand pattern before they purchase cloud resources, in order to save their cost or improve the efficiency [3].

Therefore, how to choose the schemes at the cost-optimal commitment level bothers cloud users and is of their best interest. In order to solve this challenge, previous studies [3], [4] have introduced a cloud brokerage service as the intermediation layer between cloud providers and users, which not only enables the broker to achieve profits, but also reduces cloud users' cost. Cloud brokerage service provides a connection between cloud users and providers by purchasing resources from cloud providers and then delivering to users with discounts. On one hand, broker needs to optimally reserve resources from providers to satisfy all users' job requests; on the other hand, it needs to design proper pricing schemes to attract cloud users.

With regard to resources reservation for the broker, existing researches [3]–[6] have designed many methods to optimally reserve resources from cloud providers, including offline methods [4], [6] that required perfect future demand information or online ones [3] vice versa. However, these methods still incur *idle resource waste* due to the reason that cloud resources are inherently nonstorable and perishable [7]. That is to say, if some of the purchased cloud resources are not sold out by the broker at any time, then there will be some wastefulness of the resource. For example, as shown in Fig. 1, if the broker has purchased three reserved instances from time slots 1-8, then time slot 1, 2, 5, 7, 8 for instance 3 and time slot 5 for instance 2 are wasted. Also, the broker can not guarantee services at time slots 4 and 6 if they do not buy more instances. This idle resource waste problem is challenging cloud broker when buying reserved instances to accommodate cloud users.

In order to solve this challenge, we find that cloud users always have latency-sensitive jobs (e.g., online game, e-commence transaction) and non latency-sensitive jobs (e.g., testing jobs, scientific computing) simultaneously. Non latency-sensitive jobs can be delayed to the future to utilize the wasted idle resources. For example, in Google cluster trace [8], a lot of low priority jobs (i.e., non latency-sensitive jobs) are requested by users. We have drawn Fig. 2 here to show the number of jobs requested over 29 days in May, 2011. One can see that there are a lot of low priority requests per hour. Therefore, as shown in Fig. 1, if non latency-sensitive jobs happen to appear in time slots 4 and 6, they probably can be

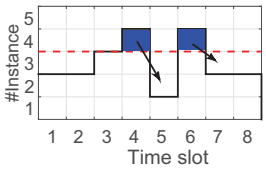


Fig. 1: *Reserved instances* reservation against time. The black line denotes the requested instance number and the red dashed line denotes the purchased reserved instances. The blue block means the unsatisfied demand.

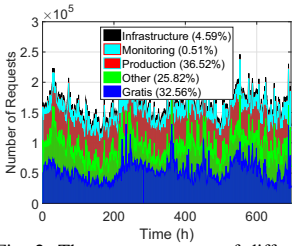


Fig. 2: The request curves of different priorities in Google cluster trace. Different prioritized request groups occupy different proportions. Job requests in *Infrastructure* group have the highest priority while *Gratis* vice versa.

delayed to execute in the future time, i.e., time slots 5 and 7 in instance 3. By delay scheduling the low priority requests, the purchased instances can be fully utilized and broker's profit can be increased considering the priority of requests. Hence, in this paper, we will take advantage of jobs' priority to design new algorithms to maximize the profit of cloud broker.

After the broker has reserved resources from cloud providers, the next step is to price the resources to users. Pricing is one of the most critical component of cloud computing since it can directly influence the revenue of cloud providers/brokers and the budget of customers [9]–[11]. However, most previous researches about brokerage service mainly focus on designing resource reservation approaches from providers, while seldom consider the pricing schemes for users [3], [4]. In this paper, we notice that priority is of huge benefit for broker's resources reservation, because low priority jobs can be delayed to the future. Accordingly, we need to design a priority aware pricing for the broker.

Since low priority jobs may be delayed to utilize the idle resource by the broker, it is unfair to charge low priority jobs just the same as high priority ones. While for the same low priority job request, the pricing needs to charge less if the job has been delayed longer. To guarantee the pricing fairness, we design a fair and priority aware pricing scheme for the broker, called *PriorityPricing*, which charges users with different prices based on priorities.

The main contributions of this paper are as follows,

- By analyzing the widely used Google cluster trace [8], we find that cloud users' jobs always have different priorities and low priority jobs can be preempted by high priority ones. This priority characteristic can be applied into the resource reservation of cloud broker to reduce the cost.
- We propose the first priority aware pricing scheme, PriorityPricing, for the broker which has been ignored by previous work. Meanwhile, the proposed pricing attracts cloud users to trade with cloud broker by fairly charging job requests based on priorities.
- We design resource reservation algorithms with considering the priority of users' requests to solve the idle resource waste problem for the broker. The algorithms significantly reduce the reservation cost to satisfy all users' requests thus increase the profit of broker.
- We evaluate the effectiveness of proposed algorithms by conducting simulations with a 1-month Google trace. The results have shown that broker's profit can be increased up to 2.5× than that without considering priority for offline algorithm, and 3.7× for online algorithm.

The organization of this paper is as follows. We first briefly review the classic IaaS pricing schemes and brokerage services, then analyze the priority of Google trace in Sec. II. We design our PriorityPricing scheme for the broker in Sec. III and formulate the broker's profit problem in Sec. IV. In Sec. V, we describe our algorithms and complexity. In Sec. VI, we evaluate our solution. The related work is discussed in Sec. VII. Finally, we conclude the paper and future work in Sec. VIII.

II. Background and Motivation

In this section, we first analyze classic cloud pricing schemes and explain the break-even point between on-demand and reserved instances. Then we introduce the brokerage service which can provide cloud users more flexible resources. After that, we take an analysis about the Google cluster trace to show the importance of priority.

A. Existing Classic Pricing Schemes

IaaS clouds typically provide multiple pricing schemes to cloud users, including *on-demand*, *reserved*, and other instances [1], [2]. On-demand instances require users to pay a fixed price per billing cycle without any commitments. In Amazon EC2 [1], for example, users pay $0.026 per hour for a t2.small instance in US East (N. Virginia). Formally, we denote the hourly price as P for the on-demand instance, which costs Ph when running for h hours.

In another pricing scheme, reserved instances require users to pay a onetime upfront fee for one time period (e.g., one month or one year). During the reservation period, the usage is either charged with a discount rate or free. In common, the cost of reserved instances is fixed [2]. Formally, we denote the upfront fee as γ and the reservation period as τ, for a certain type of reserved instance[1].

Individual users cannot utilize the discount of reserved instances sufficiently. Actually, there exists a *break-even* point [3], [12] at which the cost is identical with running an on-demand and a reserved instance. We can image that an on-demand instance costs Ph for running the workload with h hours while for a reserved instance, the cost will be γ. Intuitively, there is no difference to the user when $Ph = \gamma$ thus we have the break-even point $h_b = \gamma/P$. It is cost-effective to reserve an instance if and only if its usage during the reservation period is beyond the break-even point h_b.

[1]In this paper, we target at broker's profit maximization with on-demand and reserved instances. Increasing broker's profit by using spot instance is included in our future work.

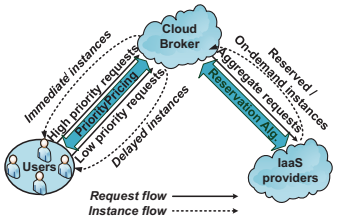


Fig. 3: The overview of cloud service broker model

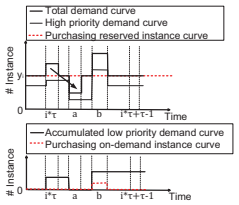


Fig. 4: Illustration of Alg. 1

B. Cloud Brokerage Service

In order to fully utilize the benefits of different pricing schemes while go beyond the limitations, cloud brokerage service has been introduced recently [3], [4]. Cloud broker connects both cloud users and providers, which means that it not only optimally purchases resources from multiple providers, but also resells the resources to users, just as shown in Fig. 3. Previous works mainly focus on when and how many resources the broker should purchase from providers. But none of them notice that cloud users' jobs always have different priorities and low priority jobs can be delayed to the future. Meanwhile, they either adopt the pricing from IaaS clouds or just ignore it. *This motivates us to design a priority aware pricing scheme between cloud users and the broker.*

C. Analysis of Google Cluster Trace

In cloud computing environment, jobs from different users always have different priorities and jobs from the same user also may have different priorities [13]. We take the widely used trace from Google [8] as an example. The jobs in google have a broad range of priorities, numbered from 0 to 11, which can be classified into five groups, i.e., *gratis* (0-1), *other* (2-8), *production* (9), *monitoring* (10), and *infrastructure* (11). High values mean high priority jobs, while low priority jobs can be delayed by high priority jobs [8], [13]. We plot the number and percentage of requests with different priorities in Fig. 2.

It can be seen from the the figure that different priority requests occupy different proportions of the total requests. The *infrastructure* and *monitoring* groups have very few requests with 4.59% and 0.51% of the total requests respectively. The *production* and *other* groups (*other* inferred as batch jobs [13]) occupy the majority of requests with 36.52% and 25.82% respectively. Surprisingly, the *gratis* group, i.e., the lowest priority, occupies a large proportion of the total requests with 32.56% of the total requests. Interestingly, the trace providers indicate that the resources used by job requests in the *gratis* group are generally not charged [13], so that they can be delayed by any other high priority requests. *This motivates us to design priority aware resource reservation algorithms to reduce cost for the broker.*

Note that in this paper, we only consider two types of priorities, i.e., the *low priority* (the *gratis* group and *other* group) and the *high priority* (the rest groups) and design priority aware cost-effective algorithms for the resource reservations in the following parts. Actually, our model can be easily extended for more fine-grained types of priorities in the future work.

III. Priority Aware Pricing Scheme: PriorityPricing

In this section, we design a fair and priority aware pricing, called *PriorityPricing*, between cloud users and the broker. After that, we model the cost for each cloud user under our *PriorityPricing*.

A. PriorityPricing

Suppose U cloud users $\mathcal{U} \triangleq \{1, 2, \cdots, U\}$ would submit their job requests to the broker for virtual machine resources. In this paper, a discrete time slotted system has been applied [14], [15], in which the length of a time slot can be set as the same as a billing cycle (e.g., one hour in Amazon EC2 [1]). At each time slot t, user i requests $d_i(t)$ instances[1]. Also, the user will set the priority for their requests by paying different prices. As shown in Fig. 3, the broker will allocate *immediate instances* to the user for high priority requests, and charge instances with a high price p^h. However, for low priority requests, the broker will allocate instances to the user when idle reserved instances are existed, i.e., *delayed instances*. In this case, the low priority requests may be executed after delaying x time slots by the brokerage. As a result, the broker will charge the *delayed instances* with a low price p^l.

With regard to the delay time of different prioritized requests, the PriorityPricing is formulated as follows:

$$p(x) = -\frac{P}{d_{max}}x + P \tag{1}$$

where x denotes the time slots delayed for the low priority request and d_{max} denotes the maximum delay tolerated by low priority requests. As mentioned in Sec. II-A, we use the on-demand price P for the price of high priority requests[2]. For example, the broker allocates immediate instances for users' high priority requests, then the delayed time equals to zero. That is, $x = 0$ and $p^h = p(x) = p(0) = P$. While for a specific low priority request Q_0, if the broker executes it after time delay x_0, then the price for this request is $p^l = p(x_0) = -\frac{P}{d_{max}}x_0 + P$. Specifically, d_{max} guarantees no requests will be delayed for an indefinite amount of time. Meanwhile, the broker will allocate an on-demand instances once an request has been delayed time d_{max}. That is, $p^l = p(d_{max}) = -\frac{P}{d_{max}}d_{max} + P = 0$ when x equals to d_{max}. In this way, no job requests would be starving in the proposed broker model. Furthermore, the parameter d_{max} allows us to reply to different types of jobs. Since different types of jobs may have different sensitivities for the delay, we can change the value of d_{max} accordingly.

Remarks: An intuition of the PriorityPricing is that we set a fixed price P for job requests from all users. Once a low priority job has been delayed for time x, we will give

[1]In this paper, we only consider one type of instance. Our model can easily extend to the multiple types (e.g., j types) of instances by extending the $d_i(t)$ to $d_{ij}(t)$.

[2]For high priority jobs, it is reasonable for users to use on-demand instances with guaranteed availability [16].

some compensation $\frac{P}{d_{max}}x$ to the user accordingly. Since cloud computing is an economically-oriented computing paradigm, pricing fairness, including personal fairness and social fairness, needs to be considered when designing a pricing scheme [5], [17]. Personal fairness means that the price is reasonable or low enough for customers while social fairness means that the price charges the same financial cost for the same service among all users. PriorityPricing satisfies both kinds of fairnesses. First, PriorityPricing charges all users with prices which are below or equal to on-demand price. It is subjective and reasonable to cloud users. Hence, PriorityPricing satisfies the personal fairness. Second, PriorityPricing will charge the same price for the same service (P for high priority requests with no delay and $p(x)$ for low priority requests with the same delay x). While for requests with different delay, requests with longer delay need to pay smaller price. Hence, PriorityPricing also satisfies the social fairness.

B. Cloud Users' Cost Under PriorityPricing

Under the PriorityPricing, suppose cloud user i requires instance $d_i(t)$ including $d_i^h(t)$ instances for user's high priority jobs and $d_i^l(t)$ for low priority jobs at each time slot $t = 0, 1, \cdots$. For each user i, the cost needed under the PriorityPricing can be given as follows,

$$C_i = \sum_{t=0}^{T} \sum_{j=1}^{d_i(t)} p(x(j)) \tag{2}$$

where T is the total time slots and $x(j)$ means the delay time incurred by the broker for the j-th job request from user i at time slot t, $0 \leq x(j) \leq d_{max}$. Hence, $\sum_{j=1}^{d_i(t)} p(x(j))$ denotes the cost for all job requests for user i at time t. Then for cloud broker, the revenue gained from all cloud users can be given as

$$C_{Revenue} = \sum_{i=1}^{U} C_i \tag{3}$$

Obviously, broker's profit can be computed by subtracting the cost for purchasing proper instances from the revenue $C_{Revenue}$ gained from users. In the next subsection, we will model the resource reservation problem and then try to maximize the profit of the broker.

IV. Resource Reservation Model

After that, we model the resource reservation problem between cloud providers and the broker with considering the priority. Our main objective is to maximize the broker's profit while satisfying users' demands.

A. The Reservation Problem between Broker and Provider

At each time $t = 0, 1, \cdots$, users send the requests to the broker and the broker needs to purchase reserved instances $r(t)$ and on-demand instances $o(t)$ to accommodate the high priority demands $d^h(t)$ which cannot be deferred to the future. On the other hand, the unsatisfied low priority demands $d^l(t)$ can be delayed to the future when reserved instances are idle. We denote $n(t)$ as the active reserved instances at time t. Then one can have,

$$n(t) = \sum_{i=t-\tau+1}^{t} r(i) \tag{4}$$

where τ denotes the reservation period of reserved instances and the reserved instances purchased after time $t-\tau+1$ still remain active at time t. Apparently, more on-demand instances $(d^h(t) - n(t))^+$ are needed at time t for the broker, where $x^+ = max\{0, x\}$. Obviously, the broker needs to satisfy the high priority job requests $d^h(t)$ at every time t:

$$o(t) + n(t) \geq d^h(t) \tag{5}$$

After making reservation decision, we have $W(t)$ low priority demands accumulated at time t as follows:

$$W(t) = ((W(t-1) + d^l(t) - (o(t) + n(t) - d^h(t)))^+ \tag{6}$$

where $W(t-1)$ is the low priority requests remained before time t and $d^l(t)$ is the low priority requests arrived at time t. Thus $W(t-1) + d^l(t)$ denotes the low priority requests needed to be processed at time t. $o(t) + n(t) - d^h(t)$ denotes the low priority demands completed at time t. For simplicity, we set $W(t) = 0$ for $t \leq 0$.

Then, the cost for purchasing instances from the cloud provider can be given as $r(t)\gamma + o(t)P$ while the total revenue gained from cloud users is $C_{Revenue}$. Thus the total profit of the broker can be given as

$$C_{Profit} = C_{Revenue} - \sum_{i=1}^{T} (r(t)\gamma + o(t)P) \tag{7}$$

The broker will try to make dynamic reservation decisions to maximize its profit as follows:

$$\mathcal{P}: \quad \max_{r(t), o(t)} \quad C_{Profit} \tag{8}$$
$$s.t. \quad (5)$$

Problem ($\mathcal{P}$) is an integer programming problem and may be solved by exhaustion research. Previous work [3], [18] derived the optimal solution for a simpler problem without considering the priority feature via dynamic programming, but it still needs to reduce the curse of dimensionality [19]. Hence, the dynamic programming algorithms to our problem suffer from more serious problem [19]. In this paper, we design the approximate algorithms to work out Problem ($\mathcal{P}$).

V. Priority Aware Cost-Effective Reservation Algorithm

In this section, we first design two offline algorithms with considering priority, i.e, *period decision algorithm* (PDA) and *greedy decision algorithm* (GDA) to solve Problem ($\mathcal{P}$) with the help of predicted demand information. For the offline algorithms, we assume that the future demand information can be achieved or accurately predicted. After that, we will relax this assumption and design an *online decision algorithm* (ODA) to solve Problem ($\mathcal{P}$) without any future information as a *prior*.

A. Periodic Decision Algorithm (PDA)

In this part, we first design a heuristic algorithm that purchases reserved instances periodically. The whole time period

T can be divided into M time intervals $\{I_i, 0 \leq i \leq T/\tau\}$ with the same length as the reservation period τ. We will make a reservation decision at the beginning of each interval. For each interval, we first check the usage of the high priority demand $d^h(t)$ and the low priority demand $d^l(t)$, then we decide how many reserved instances the broker needs to purchase for the next time period from $i * \tau$ to $i * \tau + \tau - 1$. We start by purchasing one reserved instance for the broker, then check the relations between the utilization and break-even point for the reserved instance. Recall the break-even point in Sec. II-A, the purchased reserved instance has been utilized effectively if and only if its utilization during the reservation period is beyond the break-even point h_b. If the reserved instance has been used effectively, we repeat to purchase a new reserved instance. Once the utilization of the reserved instance is below the break-even point, we stop the reserved instance purchasing. The pseudo-code of PDA has been presented in Alg. 1.

Algorithm 1 *Periodic Decision Algorithm (PDA)*

Input: The high priority demand $d^h(t)$ and low priority demand $d^l(t)$ for time t in time interval i, $i = 0, 1, \cdots, T/\tau$.
Output: Reserved instance $r(t)$ at the beginning of the reservation interval i, $t = i * \tau$.

1: Let $y(i)$ be the number of reserved instances at the beginning of this interval I_i and u_l be the utilization of level $y(i)$ in interval i, initially, $y(i) = 0$, $u_l = \tau$
2: **while** $u_l > h_b$ **do**
3: $y(i) = y(i) + 1$ /*Purchasing a new reserved instance.*/
4: $u_l = \tau$
5: **for** $t = i * \tau$ to $i * \tau + \tau - 1$ **do**
6: **if** $d(t) + W(t) < y(i)$ **then**
7: $u_l = u_l - 1$, $o(t) = 0$ and $W(t) = 0$
8: /*Using all previous low priority demands to fill the idle reserved instances, idle reserved instances still exist at time t.*/
9: **else**
10: $o(t) = (d^h(t) - x(i))^+$
11: /*On-demand instances $o(t)$ are needed when high priority demands are unsatisfied .*/
12: $W(t) = (W(t) - (x(t) + o(t) - d(t)))^+$
13: /*Using partial previous low priority demands to fill the idle reserved instances, no idle reserved instance exists at time t.*/
14: **end if**
15: **end for**
16: **end while**
17: $r(t) = y(i)$ /*Reserve $y(i)$ reserved instances at the beginning of interval i, i.e., $t = i * \tau$*/

We also use Fig. 4 to illustrate Alg. 1. The second figure in Fig. 4 shows the amounts of accumulated low priority demands and purchased on-demand instances.

Algorithm Complexity: In Alg. 1, for each interval, we need to check the entire period for every purchasing reserved instance. The time complexity for each interval is $O(D\tau)$, where D denotes the maximum demand during the interval. Therefore, the complexity for the whole algorithm is $O(DT)$.

B. Greedy Decision Algorithm (GDA)

The PDA only makes the reservation decisions for each intervals once. In this part, we propose a greedy algorithm (GDA) which can make reservation decisions at any time slot. It is obviously that GDA has better performance than PDA but incurs more time complexity. We summarize the GDA as Alg. 2.

Algorithm 2 *Greedy Decision Algorithm (GDA)*

Input: The high priority demand $d^h(t)$ and low priority demand $d^l(t)$ at each time t
Output: Reserved instances $r(t)$ and on-demand instances $o(t)$ at each time t

1: **for** $t = 0$ to T **do**
2: **for** $i = t$ to $t + \tau - 1$ **do**
3: $n(i) = \sum_{j=i-\tau+1}^{i} r(j)$
4: **end for**
5: Let $g(i) = (d^h(i) - n(i))^+$ for all $i = t, t+1, \cdots, t+\tau-1$
6: Take $g(t), \cdots, g(t+\tau-1)$ as the input and run Alg. 1. Let $r(t)$ equals the output of Alg. 1
7: $n(t) = n(t-1) + r(t)$
8: $o(t) = (d^h(t) - n(t))^+$
9: **end for**

In GDA, at each time t, we first calculate the active reserved instances in the following reservation period τ, i.e., $n(i), i = t, t+1, \cdots, t+\tau-1$ (line 2-4). Then we get the gap between the high priority demands $d^h(t)$ and the remain active reserved instances $n(i)$, i.e., $g(i) = (d^h(i) - n(i))^+, i = t, t+1, \cdots, t-\tau+1$ (line 5). At last, we run Alg. 1 with $g(i)$ as input to get the number of reserved instances $r(t)$. After that, we can get the number of on-demand instances $o(t)$ (line 8). Clearly, the time complexity of Alg. 2 at each time t is the same as Alg. 1.

C. Online Reservation Algorithm (ODA)

The above PDA and GDA can only apply to the situation where the future information can be obtained or predicted. However in reality, the future information is always not acquirable. Also, it is not easy to predict the demand in cloud data centers, as workload in cloud has higher variances [20] compared with that in traditional grids and high performance computing (HPC) systems. Therefore, we design another ODA algorithm to solve the problem ($\mathcal{P}$) that future information is not available. In this case, ODA decides the number of reserved intances $r(t)$ only based on historical information $g(t-\tau+1), \cdots, g(t)$ in the past reservation period, where $g(i) = (d^h(i) - n(i))^+$ for all $i = t-\tau+1, t-\tau+2, \cdots, t$. This is similar to the online deterministic algorithm of the ski-rental problem [21]. Also, our instance reservation problem can be reduced to the ski-rental problem with the assumption that there is only one instance at a time [3].

The historical information $g(t-\tau+1), \cdots, g(t)$ denotes the reservation gap between high priority demand $d^h(i)$ and the number of reservation $n(i)$. One can run Alg. 1 to get $r(t)$ with input from these $g(i)$ (line 3). Meanwhile, we need to update the remain active reserved instance for both past reservation period and future reservation period, i.e., $n(i) = n(i) + r(t)$ for all $i = t-\tau+1, \cdots, t+\tau-1$ (line 4). After that, at time i, the unsatisfied high priority demand will be filled by purchasing on-demand instances (line 5). We summarize the details of ODA in Alg. 3 and it has the same time complexity as Alg. 2

VI. Performance Evaluation

In this section, we conduct simulations based on a real-world Google trace [8] to first evaluate the efficiency of our

Algorithm 3 *Online Decision Algorithm (ODA)*

Input: The history of high priority demand $d^h(i)$ and low priority demand $d^l(i)$ before time t, $i = 0, 1, \cdots, t$
Output: Reserved instances $r(t)$ and on-demand instances $o(t)$ at each time t

1: **for** $t = 0$ to T **do**
2: Let $g(i) = (d^h(i) - n(i))^+$ for all $i = t - \tau + 1, \cdots, t$
3: Take $g(t - \tau + 1), \cdots, g(t)$ as the input and run Alg. 1. Let $r(t)$ equals the output of Alg. 1
4: Update $n(i) = n(i) + r(t)$ for all $i = t - \tau + 1, \cdots, t + \tau - 1$
5: $o(t) = (d^h(t) - n(t))^+$
6: **end for**

PriorityPricing in Sec. VI-B, and then evaluate the performance of the PDA, GDA, and ODA algorithms in Sec. VI-C.

A. Dataset Preprocessing and Experimental Setting

Since no public IaaS clouds have released their workload traces so far [10], public cloud traces are often confidential. We leverage the widely used Google cluster trace [8] in the experiment. Google cluster trace consists about 370,000 jobs owned by 933 users running across over 12,000 hosts. More than 4,000 applications such as MapReduce and machine learning programs exist in the datacenter. Moreover, every job may have variable tasks that have different priorities ranged from 0 to 11, and also there are over 40 million tasks in the trace.

In the simulation, with regard to the resource reservation pricing from cloud providers for the broker, we apply the pricing from Amazon EC2 with hourly price for on-demand instance and the All Upfront option for reserved instance [1]. Since Google trace only spans about one month, we set the reservation period for reserved instance as one week, i.e., $\tau = 168$ hours, and a 50% discount compared with on-demand instance, which is a common pricing scheme in most IaaS clouds [1], [2]. Specifically, we set the $P = \$0.026$ as the hourly price of t2.small on-demand instance in EC2 [1] and the upfront fee for a reserved instance is $\gamma = P * \tau * 50\% = \$2.184$ per week. For the priority aware pricing (PriorityPricing) between cloud users and the broker, we set the maximum delay $d_{max} = \tau$.

To evaluate the performance of the proposed algorithms, we first introduce some *benchmark algorithms* as follows. The first benchmark algorithm is *All-On-Demand*, in which users never purchase reserved instances but run all their jobs with on-demand instances. It is simple and common for latency-sensitive workloads users [22]. The second algorithm is *All-Reserved* algorithm, where users only purchase reserved instances. Obviously, the cost for *All-On-Demand* with broker is the same to one without broker. Also, the cost is the same no matter we have considered the priority or not. Hence, we can set the cost for *All-On-Demand* as the baseline and normalize the cost incurred by other algorithms to the cost for *All-On-Demand*, i.e., the cost for *All-On-Demand* is 1 in the following experiments. Therefore, we omit the results of *All-On-Demand* algorithm.

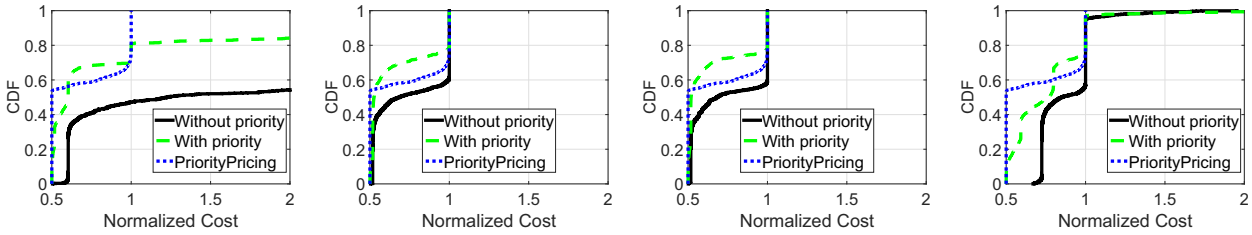
B. Efficiency of PriorityPricing

Cloud users can either purchase cloud resources from cloud providers directly or trade with cloud brokers for some mutual benefits [3]. In Sec. III, we have designed a PriorityPricing for our brokerage service, which charges users' high priority requests with a high price and a low price for the low priority requests.

In order to evaluate the efficiency of the proposed pricing, we compare cloud users' cost under our brokerage service to the cost purchased from cloud provider directly. When trading with the proposed brokerage service, cloud users will be charged by the proposed PriorityPricing. While trading with the cloud provider directly, we apply our designed algorithms for each individual users, namely, PDA (Alg. 1), GDA (Alg. 2), and ODA (Alg. 3), together with the benchmark algorithms, i.e., *All-On-Demand* and *All-Reserved*. As mentioned in Sec. VI-A, we normalized the cost incurred by other algorithms to cost for All-On-Demand. Hence, the cost for All-On-Demand is 1 and omitted. For each algorithm, we obtain the cost for individual users with and without considering the priority, respectively. We draw the cost CDF of individual users in Fig. 5.

We observe several findings in Fig. 5. First, when trading with the provider directly for all users, the cost considering priority is no more than the cost without considering priority. That means the *idle resource waste* problem can be alleviated by considering the priority feature no matter what reservation approaches have used. Second, more than 70% of users can achieve cost saving under our PriorityPricing. The reason is that these users have both high and low priority job requests while low priority requests are charged with a low price in our brokerage service. Third, with considering the priority of users' job requests, both offline algorithms (PDA and GDA) and online algorithm (ODA) can hugely reduce users' cost. At last, for ODA, the majority of users need to pay more even with considering the priority compared with our PriorityPricing. Hence, cloud users would prefer to trade with our broker rather than trade with resources from cloud providers directly. All these findings validate the attractiveness of the proposed PriorityPricing scheme.

In order to further illustrate the cost saving for each individual user, we plot the saving percentage CDF of users with priority consideration for different algorithms in Fig. 6. We see that more than 40% of users can achieve cost saving due to priority for both PDA and GDA algorithms. While for ODA algorithm, there exist 50% users can gain cost saving due to priority. The reason is that ODA algorithm makes reservation decision without future information, which leads to large idle resources waste. Hence, by considering priority of users' job requests, more users under ODA can gain cost saving than that under PDA and GDA algorithms. However, even though the future knowledge has been given as a *prior* for offline algorithms (i.e., PDA and GDA), cost saving still can be achieved by considering priortiy. This phenomenon proves the effectiveness of priority consideration.



(a) All-Reserved (b) PDA (c) GDA (d) ODA

Fig. 5: The cost CDF of users with and without considering the priority under different algorithms. PriorityPricing means users' cost are charged by our priority aware pricing.

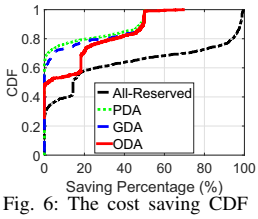


Fig. 6: The cost saving CDF of users due to priority

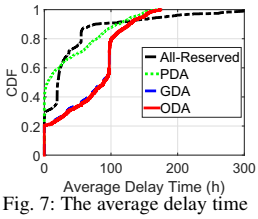


Fig. 7: The average delay time CDF of users due to priority consideration

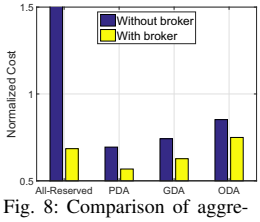


Fig. 8: Comparison of aggregate cost for all users without broker and the cost for broker under different algorithms

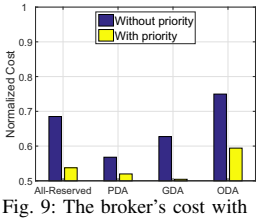


Fig. 9: The broker's cost with and without considering the priority under different algorithms

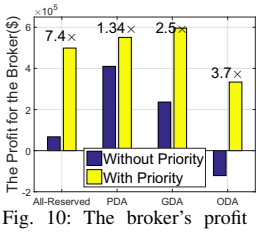


Fig. 10: The broker's profit with and without considering the priority under different algorithms

C. Evaluations of Priority Aware Resource Reservation Approaches

In this part, we first evaluate the cost saving simply brought by cloud broker for cloud users, without considering the priority of job requests. By doing so, we find that the cost with brokerage service still incurs high cost for both offline and online algorithms. Then, we will evaluate the cost of the broker considering the priority. Finally, we will evaluate the broker's profit under our PriorityPricing.

In order to evaluate the cost saving simply brought by cloud broker. We compare the aggregate costs for all users without broker and the cost for broker. For the former one, we first evaluate the cost to purchase resources from provider directly for each individual user and then sum up those costs. The comparison result is plotted in Fig. 8.

As can be seen from Fig. 8, without brokerage service, the *All-Reserved* algorithm will incur about $2.5\times$ cost compared with the *All-On-Demand* strategy. This is because the *All-Reserved* algorithm will purchase reserved instance for any arrival job requests and cause large amount of idle resource waste. But the cost can be largely reduced by using the brokerage service. Also, under both offline and online algorithms, the broker will reduce the total cost. But broker still needs a very high cost (about 75% of the cost with *All-On-Demand*) under online algorithm (ODA). That is because ODA makes reservation decision without any future knowledge. One can alleviate this problem by considering job requests' priority.

From the result in Fig. 8 we can see, there still exist many *idle resource waste* for the cloud broker which proves the existence of the problem presented in Fig. 1. As expected, the broker can fully utilize these idle resource waste by considering the priority. We will evaluate the broker's cost under two different scenarios, i.e., with and without considering priority, respectively.

Fig. 9 plots the total cost with and without considering priority for the broker under different algorithms respectively. From Fig. 9, we see that the total cost for all algorithms considering priority are less than 60%, which means more than 40% cost saving gained compared with *All-On-Demand* algorithm. Especially, for GDA algorithm, the incurred cost considering priority is only 51.05% compared to *All-On-Demand*, which almost reaches the maximum discount (i.e., 50%) of reserved instance in our evaluation.

With implementing our PriorityPricing between cloud broker and users, the broker can gain profit considering the priority. In Fig. 10, we plot the broker's profit with and without considering the priority respectively under different algorithms. From Fig. 10, we can see that the broker cannot achieve any profit if ODA is used without considering priority. The reason is that our PriorityPricing charges low priority requests with a low price while ODA algorithm without considering the priority cannot utilize the idle resource by delaying the low priority job requests. Hence, it would incur more cost to purchase resource from the provider than that gained from cloud users. By considering the priority of users' requests, the broker's profit can be increased up to $1.34\times$ ($2.5\times$) than that without considering priority for PDA (GDA) algorithm, and $3.7\times$ for ODA algorithm.

VII. Related Work

In this paper, we design a priority aware pricing and three dynamic resource reservation algorithms for the broker

considering the priority. So our related work includes cloud brokerage and cloud pricing schemes.

A. Cloud Brokerage Service

Brokerage service has been introduced to the cloud computing in both industry and academia in recent years [3], [4], [23]. In industry, a SaaS-based cloud management company, called RightScale, manages cloud resource from other IaaS clouds [22] and receives several millions investments in both 2008 and 2010.

In academia, some studies [3], [4], [23] provide a connection between multiple cloud providers and cloud users. A similar work to the resource reservation algorithms in our work is proposed in [3], in which a brokerage service is used to serve cloud users by aggregating reserved and on-demand instances. The main difference between these work and our work is two-fold. First, we leverage the priority of job requests to solve the *idle resource waste* problem for the broker, thus profit achieved by the broker. Second, we design a fair and priority pricing between the broker and cloud users, which has been ignored in previous work.

B. Cloud Pricing Schemes

Though very few researches focus on the pricing for cloud brokerage service, the design of pricing for cloud computing is a very hot topic [7], [10], [11], [15]. For example, we design a fine-grained pricing for IaaS cloud to solve the partial usage waste problem caused by existing coarse-grained pricing in cloud markets [10]. The proposed priority aware pricing in this paper is complementary to these studies in cloud pricing and we mainly focus on the pricing for cloud brokerage which has been ignored before. A similar work to the priority aware pricing for brokerage service in our work is proposed in [5]. Aazam et al [5] propose a relinquish probability based pricing which charges a high price if a user would give up the resources in next time period. While in this paper, we notice that priority is of huge benefit for broker's resources reservation. Accordingly, we design a fair and priority aware pricing for the broker.

VIII. Conclusion

In this paper, we propose cloud brokerage services considering both pricing scheme with cloud users and reservation methods with cloud providers, in which two types of priorities are utilized to design the priority aware pricing (i.e., PriorityPricing) and priority based reservation algorithms. Priority has been ignored in previous brokerage works while it can be used to largely increase broker's profit. The reason is that low priority requests can be delayed to the future to fully utilize the purchased reserved instances so that cost saving is gained. To evaluate the proposed pricing scheme and reservation methods, we conduct simulations by using a large-scale real-world trace.

Acknowledgment

This research is supported by National Key Research and Development Program under grant 2016YFB1000501, 863 Hi-Tech Research and Development Program under grant No. 2015AA01A203, and National Science Foundation of China under grants No. 61232008. This research is also supported by International Science & Technology Cooperation Program of China under grant No. 2015DFE12860 and supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research Program, under Contract DE-AC02-06CH11357. The Corresponding Author is Song Wu.

References

[1] Amazon Elastic Compute Cloud (EC2). [Online]. Available: http://aws.amazon.com/ec2/, 2017.
[2] Elastichosts. [Online]. Available: http://www.elastichosts.com/, 2017.
[3] W. Wang, D. Niu, B. Li, and B. Liang, "Dynamic cloud resource reservation via cloud brokerage," in *Proc. of ICDCS*, pp. 400–409, 2013.
[4] S. Niu, J. Zhai, X. Ma, X. Tang, and W. Chen, "Cost-effective cloud HPC resource provisioning by building semi-elastic virtual clusters," in *Proc. of SC*, pp. 1–12, 2013.
[5] M. Aazam, E. N. Huh, M. St-Hilaire, C. H. Lung, and I. Lambadaris, "Cloud customer's historical record based resource pricing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 7, pp. 1929–1940, 2016.
[6] A. N. Toosi, K. Vanmechelen, K. Ramamohanarao, and R. Buyya, "Revenue maximization with optimal capacity control in infrastructure as a service cloud markets," *IEEE Transactions on Cloud Computing*, vol. 3, no. 3, pp. 261–274, 2015.
[7] H. Xu and B. Li, "Dynamic cloud pricing for revenue maximization," *IEEE Transactions on Cloud Computing*, vol. 1, no. 2, pp. 158–171, 2013.
[8] More Google cluster data, google research blog. [Online]. Available: http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html, 2011
[9] H. Xu and B. Li, "Dynamic cloud pricing for revenue maximization," *IEEE Transactions on Cloud Computing*, vol. 1, no. 2, pp. 158–171, 2013.
[10] H. Jin, X. Wang, S. Wu, S. Di, and X. Shi, "Towards optimized fine-grained pricing of IaaS cloud platform," *IEEE Transactions on Cloud Computing*, vol. 3, no. 4, pp. 436–448, 2015.
[11] X. Wang, K. Wang, S. Wu, S. Di, K. Yang, and H. Jin, "Dynamic resource scheduling in cloud radio access network with mobile cloud computing," in *Proc. of IWQoS*, pp. 1–6, 2016.
[12] C. C. A. Vieira, L. F. Bittencourt, and E. R. M. Madeira, "Towards a PaaS architecture for resource allocation in IaaS providers considering different charging models," in *Proc. of GECON*, pp. 185–196, 2013.
[13] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Towards understanding heterogeneous clouds at scale: Google trace analysis," *Intel Science and Technology Center for Cloud Computing, Tech. Rep*, p. 84, 2012.
[14] Z. Zhou, F. Liu, H. Jin, B. Li, B. Li, and H. Jiang, "On arbitrating the power-performance tradeoff in SaaS clouds," in *Proc. of INFOCOM*, pp. 872–880, 2013.
[15] K. Wang, K. Yang, X. Wang, and C. Magurawalage, "Cost-effective resource allocation in C-RAN with mobile cloud," in *Proc. of ICC*, pp. 1–6, 2016.
[16] L. Zheng, C. Joe-Wong, C. W. Tan, M. Chiang, and X. Wang, "How to bid the cloud," in *Proc. of SIGCOMM*, pp. 71–84, 2015.
[17] G. Mankiw, *Principles of economics*, Sourth-Western Pub, 2008.
[18] W. Wang, B. Li, and B. Liang, "To reserve or not to reserve: Optimal online multi-instance acquisition in IaaS clouds," in *Proc. of ICAC*, pp. 13–22, 2013.
[19] W. Powell, *Approximate dynamic programming: Solving the curses of dimensionality*, John Wiley and Sons, 2011.
[20] S. Di, D. Kondo, and W. Cirne, "Characterization and comparison of cloud versus grid workloads," in *Proc. of CLUSTER*, pp. 230–238, 2012.
[21] L. M. A. Karlin, M. Manasse and S. Owicki, "Competitive randomized algorithms for nonuniform problems," *Algorithmica*, vol. 11, no. 6, pp. 542–571, 1994.
[22] AWS Case studies. [Online]. Available: http://aws.amazon.com/solutions/case-studies/,2017.
[23] S. Yang, Z. Murtaza, and L. Kang-Won, "Optimal bidding in spot instance market," in *Proc. of INFOCOM*, pp. 190–198, 2012.