Flexible Instance: Meeting Deadlines of Delay Tolerant Jobs in The Cloud with Dynamic Pricing

Xiaomeng Yi[1] Fangming Liu*[1] Zongpeng Li[2] Hai Jin[1]
[1]Key Laboratory of Services Computing Technology and System, Ministry of Education,
School of Computer Science and Technology, Huazhong University of Science and Technology
[2]University of Calgary, Canada

***Abstract*—A wide range of cloud computing jobs are delay tolerant up to a predefined deadline. Existing IaaS services offer either high cost and high fulfillment ratio or low cost without fulfillment ratio guarantee, where the fulfillment ratio is the ratio of job execution time to the time between job submission and completion. Neither of the services represents a cost-effective way to exploit job elasticity. This work proposes *flexible instance*, a cloud service where user-specified service fulfillment ratio, as a new pricing factor, is guaranteed by the provider to meet deadlines. Job elasticity is exploited by the provider to enhance resource utilization, by regulating demand fluctuation through computation arbitrage across the temporal domain. We leverage a two-stage pricing framework to agilely adapt cloud resource price to the demand-supply dynamics. The first stage uses an online strategy to reserve resources for each cloud user to guarantee its specified fulfillment ratio. We set the price of resources dynamically according to resource utilization, with a pricing curve $O(\ln p)$-competitive to the optimal fixed-price offline strategy in provider revenue. The second stage allows cloud users to submit a small budget to compete for extra service fulfillment ratio for execution speedup. A Nash bargaining framework is explored to achieve fairness, resource efficiency, and revenue maximization simultaneously. Extensive simulations driven by real-world traces show that *flexible instance* can reduce user cost for job execution while increasing provider revenue.**

Keywords*—*cloud computing; resource allocation; delay tolerant jobs; meeting deadline; pricing

I. Introduction

Batch processing as exemplified by big data analytics, video encoding, web crawling and DNA sequencing, represents a major type of workload in cloud computing. With big data analytics alone, the market is projected at \$125 billion in 2015 [1]. Batch processing jobs are often delay tolerant, up to a relatively loose deadline [2]. Within the specified deadline, users may be willing to pursue a speed up in job execution with a small payment [3]. Consider a video streaming service provider who leverages the cloud to transcode a batch of user uploaded videos into different resolutions. It takes the rented VMs 1 hour to finish the job. As long as all copies are available in 3 hours, service level agreements (SLAs) are satisfied. It may further be willing to pay a modest bonus for finishing the job earlier.

Batch processing jobs are naturally elastic in terms of execution. Their deadlines can be met, as long as a certain fulfillment ratio is guaranteed. The fulfillment ratio is defined as the ratio of job execution time to its total scheduling duration (*i.e.,* the time interval between submission and deadline). In the above example of video transcoding, a fulfillment ration of $1/3$ is required. However, currently, IaaS (Infrastructure-as-a-Service) providers offer two types of service in terms of service fulfillment ratio and pricing: high fulfillment ratio with high cost (*e.g., on-demand service*), and no guaranteed fulfillment ratio service with low cost (*e.g., spot instance service*, and *preemptible instance service*). Neither provides a flexible and cost-effective solution for batch processing jobs with deadline guarantees.

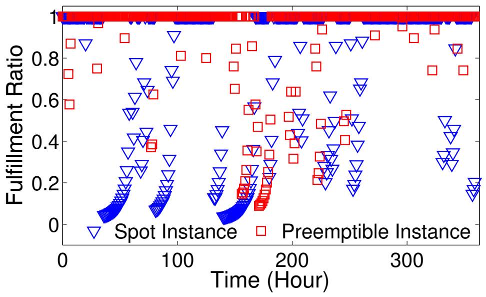

Fig. 1. Service fulfillment ratio of executing an 1-hour job in spot instance and preemptible instance in a fifteen-day period.

On-demand services such as Microsoft Azure [4] implement fixed charges per minute of VM usage, based on the instance's resource configuration. Such a simple pricing scheme fails to adapt to fluctuations in demand and supply, resulting in inefficiency in the cloud ecosystem. In contrast, Amazon EC2 spot instances [5] exploit an auction based pricing scheme. Job execution depends on the spot price set by the provider according to the dynamically changing demand-supply relation. Cloud users bid for their per-unit time VM usage. When the spot price is lower than the bid, the user runs its job in the VMs at the spot price. Otherwise, job execution is suspended until the spot price falls below the bid again. In the case of preemptible instance [6] in Google Compute Engine, cloud users are given a large price discount, while their launched instances can be preempted to spare resources for other cloud services. With both spot instance and preemptible instance, user jobs can be stopped by the provider at any time, and service fulfillment ratio is out of users' control.

As a real-world experiment, we continuously measure service fulfillment ratio of spot instance and preemptible instance when executing a 1-hour job in a fifteen-day period during July 7-21, 2015. In spot instance, we measure the fulfillment ratio of $m3.large$ instance running Linux operating system in the

*The Corresponding Author is Fangming Liu (fmliu@hust.edu.cn). The research was supported in part by a grant from the National Natural Science Foundation of China (NSFC) under grant No.61520106005, and by a grant from National 973 Basic Research Program under grant No.2014CB347800.

fulfillment ratio zone of $us\text{-}east\text{-}1a$, with the time averaged spot price as our bid. In preemptible instance, we launch three $n1\text{-}standard\text{-}1$ instances in zone $us\text{-}central1\text{-}a$ and measure their average fulfillment ratio. As illustrated in Fig. 1, in both cases the service fulfillment ratio is unstable, fluctuating within the range from 0.03 to 1 without any apparent pattern.

Although spot instance and preemptible instance charge substantially lower prices than on-demand service does, they come with no service fulfillment ratio guarantee. If the user resorts instead to on-demand service, the required VMs are provisioned with 100% fulfillment ratio but at a much higher price ($7\times$ spot instance price or $3.3\times$ preemptible instance price). For cloud users, a conceivable way of achieving desired fulfillment ratio is combining the existing on-demand service and spot instance. However, designing a cost-efficient strategy for such an approach is challenging, since it requires a precise prediction of spot instance prices, which is very difficult, if possible at all [7], [8]. Inaccurate predictions would impact the fulfillment ratio of spot instance services, which could potentially cause deadline violation. With the existing two types of services, the cloud provider can not realize the full profit potential from cloud users who wish to run delay tolerant jobs at a budget lower than that required by the on-demand service market.

We extend and complement the existing cloud market by proposing a new type of *flexible instance*, which is more flexible compared to the existing two type of services. In *flexible instance*, we use service fulfillment ratio as a pricing factor and provides guaranteed service fulfillment ratio to meet the deadlines of delay tolerant jobs. Compared with existing services, *flexible instance* has the following salient features:

- *User specified instance fulfillment ratio.* Cloud users estimate the minimum fulfillment ratio of launched instances to meet their respective job deadlines. Once a job is admitted, the cloud provider guarantees such minimum fulfillment ratio.
- *Fulfillment ratio based dynamic pricing.* Given user-specified VM types and fulfillment ratio, the provider calculates a corresponding price based on demand and supply, and let the user decide whether to accept that price.
- *Job acceleration with idle resource.* Each cloud user can submit a relatively small budget for purchasing extra instance fulfillment ratio, to expedite job execution. The provider may allocate idle resources in the cloud, if present, to cloud users according to their respective budget and resource demand.

Flexible instance helps the providers to gain profit by serving a wide range of potential users not covered by existing markets. Moreover, with user specified fulfillment ratio for meeting job deadlines, the provider can serve more users by strategically postponing the execution of some jobs during times when the instant resource demand surpasses cloud capacity and scheduling them to run on idle resources in off-peak times.

Cloud service providers naturally aim to maximize the utilization of their cloud resource pool, to maximize their revenue. We propose a dynamic pricing framework for *flexible instance*, which provides guaranteed service fulfillment ratio to meet user-specified deadlines, fully utilizes off-peak resources to enhance resource efficiency and alleviate peak load, and simultaneously maximizes the provider's revenue. In particular, we adopt a two-stage pricing policy in the framework: a dynamic pricing scheme in the first stage to compute prices for job execution according to their resource usage and specified service fulfillment ratio guarantees; a competition based pricing strategy in the second stage to allocate idle resources for extra fulfillment ratio according to users' acceleration budgets.

In the first stage, we consider the problem of provider revenue maximization given a fixed resource pool and sequentially arriving cloud users, with resource demand and basic fulfillment ratio requirement revealed upon arrival. We exploit a carefully designed *pricing curve* to compute the payment of each user without *a prior* knowledge on its valuation of job execution. The pricing curve exploits a threat-based approach and is proven to guarantee a competitive ratio of $O(\ln p)$ against the optimal fixed price strategy, where p is the ratio between the upper and lower bounds of users' marginal valuation on cloud resources. For cloud users who accept the price in the first stage, their resource demands and acceleration budgets become input to the pricing problem in the second stage. Towards fairness and resource efficiency, we leverage Nash bargaining theory: cloud users are players in a cooperative game; each user bargains for its own resource usage, eventually jointly achieving maximum social utility. We demonstrate that the pricing scheme derived from the Nash bargaining solution, along with the resource allocation scheme, achieves resource efficiency, proportional fairness in competition, and maximal provider revenue. It is worth noting that the pricing scheme requires only the solution of Nash bargaining, which can be calculated efficiently without letting cloud users literally bargain with each other. We further conduct extensive simulations driven by real-world traces from a Google cluster [9] to show that *flexible instance* can significantly increase provider revenue while simultaneously reducing the cost of cloud users.

II. DESIGN PHILOSOPHY

Fig. 2 provides an overview of the *flexible instance* service. Cloud users participate in a two-stage pricing scheme to receive (a) a basic service fulfillment ratio from a reserved resource pool of fix capacity, and (b) a supplement fulfillment ratio through competing for transient idle resources.

Incentives to cloud users and the provider. *Flexible instance* aims to benefit both the provider and cloud users. For the provider, it brings a profit gain by serving a wide range of potential users not covered by existing markets, scheduling them to run on idle resources during off-peak times. For cloud users, it provides a new cost-effective service model. Compared to the on-demand market, *flexible instance* incurs a lower monetary cost since user jobs are often scheduled to execute on idle resources in off-peak hours. Meanwhile, *flexible instance* guarantees a basic service fulfillment ratio to meet job deadlines, which is lacking in spot instance and preemptible instance.

Service fulfillment ratio as a pricing factor. *Flexible instance* uses user specified service fulfillment ratio to extend the time range of job scheduling to job deadlines, and aggressively utilizes resources at off-peak times in the scheduling window

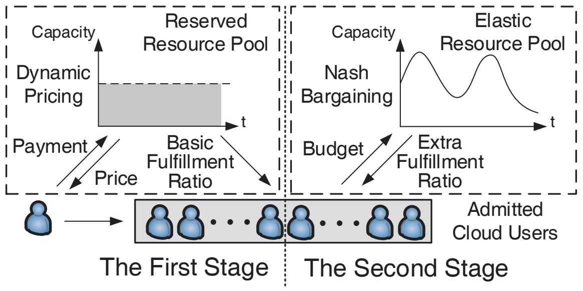

Fig. 2. The two-stage pricing framework for *flexible instance*.

to serve users, shedding peak load through temporal job shifting. It is hence natural to charge user jobs according to their flexibility to be scheduled to demand valleys. A simple measure of such flexibility is maximal tolerable delay. The larger the tolerable delay, the more likely a demand valley occurs within it. However, absolute delay tolerance alone is not a fair metric, when jobs have heterogeneous durations. Consider a 1-hour job and a 10-hour job both with a tolerable delay of 10 hours. The 1-hour job requires only 1 hour of execution in the next 11 hours, leaving ample scheduling flexibility. The 10-hour job poses a more strict requirement on cloud resources, with each 1-hour execution having only a 2-hour range for scheduling on average. We use service fulfillment ratio, which is the proportion of time a job takes for actual execution in its total scheduling duration (including tolerable delay and execution time), to calculate the price for each user.

Billing cycle and workflow. In practice, cloud service providers charge cloud users according to their resource usage by billing cycles. For example, Amazon EC2 [10] charges VM usage by hour, and Google Compute Engine [11] adopts a finer granularity in minutes. *Flexible instance* also implements discrete billing cycles, and conducts resource allocation and job scheduling for each billing cycle. Cloud users submit new jobs at the beginning of each cycle. For each user, according to the two-stage pricing scheme, as will be elaborated in Sec. III and Sec. IV, the provider calculates its payment as well as service fulfillment ratio in the current billing cycle.

A two-stage pricing framework. We leverage two different pricing stages in *flexible instance* with consideration of both provider's resource provisioning capacity and cloud users' elastic fulfillment ratio requirements. For a cloud user, its demand of service fulfillment ratio can be divided into two components: a basic demand in service fulfillment ratio to be guaranteed for meeting deadlines; and an elastic demand of accelerating job execution up to a limited budget. For the provider, we consider two types of resource in the cloud system, *i.e.*, a reserved resource pool with fixed capacity to guarantee basic service fulfillment ratio, and an elastic pool of idle resources with dynamically changing capacity due to demand fluctuation in other cloud services.

In the first pricing stage, we use the reserved resource pool to guarantee the basic fulfillment ratio of a cloud user, and calculate its payment accordingly. Users who accept the price further compete for transient idle resources within their respective budgets. In billing cycle t, let $P^b_n(t)$ be the price for user n to execute its job with guaranteed service fulfillment ratio, and $P^e_n(t)$ be the cost of an extra fulfillment ratio to expedite the execution. User n's total payment is:

$$P_n(t) = P^b_n(t) + P^e_n(t) \tag{1}$$

Given characteristics in resource supply and fulfillment ratio requirement in the two stages, two different pricing strategies are required. In the first stage, the capacity of resource pool is fixed and cloud users arrive on the fly. Considering the fluctuation of total user demands and the fixed resource capacity, the pricing should be dynamically adjusted to balance demand-supply. Furthermore, each user job has a deadline to meet, the resource transaction should be conducted online. Given the bundle of required VMs and the basic demand of service fulfillment ratio, the provider should compute the price for a user immediately upon its arrival. To dynamically adapt to the demand, we employ a well designed pricing curve, where the marginal resource price monotonically increases with resource utilization, to calculate prices for users in an online fashion.

In the second stage, the provider strives to glean revenue by granting idle resources to cloud users for additional service fulfillment ratio. An obvious approach is to auction such extra fulfillment ratio. However, since both resource and demand are dynamically changing, an auction should be conducted whenever the amount of idle resources changes or a cloud user arrives or departs. This may be infeasible in practice, since it requires frequently auction participation by cloud users. In *flexible instance*, we let each cloud user submit a time averaged budget for extra fulfillment ratio and leverage the Nash bargaining approach to allocate idle resources according to their respective budgets and resource usage. We only use the solution of Nash bargaining to make decisions on resource pricing and allocation. The solution can be efficiently computed by the provider according to the submitted budgets of cloud users; cloud users do not need to actually bargain with each other in the process. In Sec. IV we will show that the Nash bargaining approach can achieve fairness and resource efficiency while maximizing the provider's revenue.

TABLE I. LIST OF VARIABLES IN THE FIRST STAGE

	Input Variables
K	total number of user jobs in the cloud system
M	number of VM types in the cloud
L	number of computing resource types (resource dimensions)
α_{ml}	amount of type-l idle resource in a type-m VM
D_k	bundle of VMs required by user k to run its job
C^b_l	amount of type-l resource in the reserved resource pool
x^b_k	basic fulfillment ratio guarantee required by user job k
a_{kl}	amount of of type-l resource consumed to execute user job k
V_k	time averaged valuation of user k for job execution
	Decision Variables
P^b_k	price charged to user k for basic service fulfillment ratio

III. PRICING CURVE BASED PRICING FOR GUARANTEED FULFILLMENT RATIO

A. System model

We consider a cloud system with *dynamic resource provisioning* [12], where a pool of L types of resource (*e.g.*, CPU, memory, network bandwidth, storage) are dynamically assembled into M different types of VMs according to user demands. Each VM of type-m ($m \in \{1, \dots, M\}$) consists of α_{ml} units of type-l resource. The provider reserves a total amount of C_l^b units of type-l resource to guarantee the basic service fulfillment ratio for cloud users. The system runs in a time-slotted fashion, with each slot corresponding to a billing cycle. There are K users in the cloud, where user k $(k \in \{1, \ldots, K\})$ learns its demand at t_k. For ease of presentation, we assume that each cloud user submits only one job, and use k to denote both the user and its corresponding job hereafter. Our model can also cope with the case where a cloud user submits multiple jobs, by regarding each job as submitted by a distinct user.

Job k requires a VM bundle $D_k = (d_{k1}, \ldots, d_{kM})$ to execute concurrently, at a service fulfillment ratio of x_k^b, to meet the corresponding deadline. Here d_{km} is the number of type-m VMs required by user k. The time averaged valuation of user k for its job execution at a service fulfillment ratio of x_k^b is V_k, *i.e.*, user k is willing to pay up to V_k for each billing cycle from job submission to job completion. Once user k knows its demand, it submits its resource demand D_k and required basic service fulfillment ratio x_k^b to the provider. The provider decides the price P_k^b for user k's job in each billing cycle before its completion. The user then accepts or declines the price, based on its private valuation V_k of the job.

In each billing cycle, we reserve resources for each user according to its time averaged resource usage. Let $A_k = (a_{k1}, \ldots, a_{kL})$ be the resources user k consumes during its execution, where $a_{kl} = \sum_{1 \leqslant m \leqslant M} d_{km}\alpha_{ml}$ is the amount of type-l resource consumed by job k. Then the resources we reserve for user k in each billing cycle to guarantee its basic service fulfillment ratio can be represented as $x_k^b A_k$. For the provider, the total amount of resources reserved for all users should not exceed the capacity of the reserved resource pool, *i.e.*, $\sum_{1 \leqslant k \leqslant K} x_k^b a_{kl} \leqslant C_l^b$, $(l \in \{1, \ldots, L\})$.

Table I summarizes notation for ease of reference.

In the first pricing stage, our objective is to design a pricing strategy that maximize the provider's revenue from guaranteed fulfillment ratio. The problem is online by nature, since the provider needs to decide the price for each user upon its arrival. We pursue to design a strategy that is competitive to the optimal fixed price strategy, where the competitive ratio is defined as follows:

Definition 1. *In a period of T billing cycles, let $S(t)$ and $S_{opt}(t)$ be the provider revenue in billing cycle t from our pricing scheme and from the optimal fixed price strategy, respectively. The competitive ratio c in terms of revenue is the supremum of*

$$c = \frac{\sum_{t=1}^{T} S_{opt}(t)}{\sum_{t=1}^{T} S(t)} \tag{2}$$

B. Dynamic pricing mechanism

An intuitive approach to pricing guaranteed service fulfillment ratio is to set a fixed price for each unit of the reserved resources in every billing cycle. However, finding the optimal price that maximizes the provider's revenue requires the knowledge of the distribution of users' valuation on cloud resources, which is impractical to estimate precisely [13]. Moreover, a fixed price strategy fails to adapt to the fluctuation of user demands, thus the dynamics in demand-supply cannot be exploited to improve the provider's revenue.

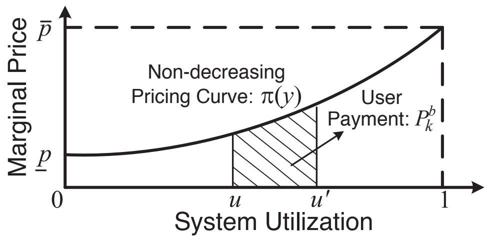

Fig. 3. The pricing curve based pricing scheme for the first stage.

In the first pricing stage, we use a dynamic pricing scheme for basic service fulfillment ratio. We calculate the price for each cloud user according to the resource utilization of the reserved resource pool in the cloud, as well as its time averaged resource demand. As illustrated in Fig. 3, we apply a *pricing curve* [14], [15] $\pi(y)$ to compute the marginal price of resource usage when the utilization of the reserved resource pool reaches y $(0 \leqslant y \leqslant 1)$. In other words, when the resource utilization is y, we set $\pi(y)$ as the price for one unit of resource. Given multiple resource types in the pool, we define the resource utilization of the cloud system as the utilization of the *critical resource*, *i.e.*, the resource that is most utilized, and is hence most likely to be the bottleneck that hinders the cloud from serving more users. Then, the payment of user k for its basic service fulfillment ratio is:

$$P_k^b = \int_u^{u'} \pi(y)dy \tag{3}$$

where u and u' are the utilization before and after resource reservation for job k, respectively.

After receiving a job requirement, the provider computes the amount of resources to reserve to guarantee basic fulfillment ratio. If the remaining resources are not sufficient, the requirement is rejected. Otherwise, the provider computes the price according to Eq. (3) and let the user decide whether to accept it. It is worth noting that in the first pricing stage, the resource price for a cloud user is fixed along time, although different cloud users may have difference prices for cloud resources.

Intuitively, when utilization is high, the marginal price should also be high so that spare resources are used to serve users with a higher valuation, for a higher revenue; when the utilization is low, the marginal price should also be low to attract more users to utilize the resource. Therefore, $\pi(y)$ should be non-decreasing. Furthermore, in practice, the marginal price should be no higher than price in the on-demand market, otherwise a user would switch to the latter for guaranteed fulfillment ratio. For each billing cycle, if a unit of resource l is not used, the provider can gain a residual value of $\underline{p}_l$ from energy and operational cost saved [14]. When the price falls below the residual value, the provider chooses not to serve users, to earn the residual value instead. Consequently, we assume that the marginal price of type-l resource is upper bounded by $\bar{p}_l$ and lower bounded by $\underline{p}_l$. Since $\pi(y)$ is a non-decreasing marginal price function in terms of the entire system utilization, it is upper bounded by $\bar{p} = \sum_{1 \leqslant l \leqslant L} C_l^b \bar{p}_l$ and lower bounded by $\underline{p} = \sum_{1 \leqslant l \leqslant L} C_l^b \underline{p}_l$, where C_l^b is the capacity of type-l resource in the reserved resource pool.

C. A $O(\ln p)$-competitive pricing curve

We next show that through a careful design of $\pi(y)$, our online pricing scheme can achieve a competitive ratio of $O(\ln p)$ to the optimal fixed price strategy in provider revenue, where $p = \overline{p}/\underline{p}$ is the ratio between upper and lower bounds of the marginal price.

Based on a *threat-based* approach from recent literature [14], we first investigate the design of the pricing curve if we pursue a competitive ratio of c. After that, we calculate the optimal competitive ratio c that can be achieved. Recall that the provider sells resources at the marginal price $\pi(y)$ when resource utilization reaches y. In the threat-based approach, $\pi(y)$ is set as low as possible without jeopardizing the competitive ratio c.

To derive a closed-form expression of $\pi(y)$, we first investigate its inverse function $Q(z) = \pi^{-1}(z)$, which represents the total resource utilization in the reserved resource pool when the marginal price reaches z. We will focus on deriving $Q(z)$ by setting up a number of equations according to the threat-based strategy. Let $V(z)$ be the payment of all users when resource utilization is $Q(z)$. Then $Q(z)$ can be derived based on the following conditions:

$$Q(z) = 0 \text{ and } V(z) = 0, \quad \forall z \leqslant c\underline{p} \tag{4}$$
$$Q(\overline{p}) = 1 \tag{5}$$
$$V'(z) = zQ'(z), \quad \forall z \in [c\underline{p}, \overline{p}] \tag{6}$$
$$\frac{z}{c} = (1 - Q(z))\underline{p} + V(z), \quad \forall z \in [c\underline{p}, \overline{p}] \tag{7}$$

Recall that the provider gains a residual value $\underline{p}$ if all the reserved resources are not used. Eq. (4) implies that the provider should not serve any user with a price lower than $c\underline{p}$ since the competitive ratio is not violated anyway: any pricing strategy would result in a revenue lower than $c\underline{p}$, while not selling any resource results in a revenue of exactly $\underline{p}$. As illustrated in Eq. (5), when the marginal price reaches its upper bound, all resources should be sold to users. In Eq. (6), $V'(z)$ is the provider's marginal revenue when the marginal price is z, and $Q'(z)$ is the amount of resources sold at price z. Eq. (7) represents the worst case when the marginal price reaches z in a billing cycle. A competitive ratio of c guaranteed in each billing cycle also holds for the entire T billing cycles. Recall that the provider calculates payments for new users upon their arrival. The marginal price being z indicates there is one user who values marginal resource usage at z and all the other users who reject the provider's price has a valuation lower than z. Therefore, the resource price in a fixed price strategy that sells all the resources would be no higher than z. So the total revenue of an optimal fixed price strategy is at most z where all the users valuate marginal resource usage at exactly z and all the reserved resources are sold. By setting the revenue target z/c equal to the total collected payment $V(z)$ and the residual value of remaining resources $(1 - Q(z))\underline{p}$, a competitive ratio of c is guaranteed.

By solving the four equations (4)-(7) we can derive the solution of $Q(z)$, and compute $\pi(y)$ by inverting $Q(z)$: $\pi(y) = \underline{p}(1 + (c-1)e^{cy})$. The maximum achievable c for the threat-based strategy to be feasible can be derived from the following equation: $c = \ln \frac{p-1}{c-1}$, where p is the ratio between the upper bound and lower bound of marginal price. We can see that $c = O(\ln p)$; consequently, the pricing curve based pricing scheme achieves a competitive ratio of $O(\ln p)$ in revenue.

TABLE II. LIST OF VARIABLES IN THE SECOND STAGE

	Input Variables
$N(t)$	total number of jobs in the second stage in billing cycle t
D_n	bundle of VMs required by user n to run its job
a_{nl}	amount of of type-l resource consumed to execute user job n
B_n	user n's time averaged budget for extra fulfillment ratio
$C_l(t)$	amount of available type-l resource in the two resource pools
$X(t)$	the space of achievable fulfillment ratio vector for the jobs
$X_0(t)$	the vector of initial utilities that users acquire in the first stage
	Decision Variables
$X_n^e(t)$	extra service fulfillment ratio allocated to user n in cycle t
$P_n^e(t)$	price charged to user n for extra service fulfillment ratio

IV. A Nash Bargaining based Approach to Pricing Elastic Fulfillment ratio

Beyond the pricing curve based pricing scheme that reserves a fixed amount of resources for basic fulfillment ratio, *flexible instance* further helps the provider gain a profit from transient idle resources. Cloud users who accepted the price in the first stage compete for additional elastic resource. We exploit an asymmetric Nash Bargaining framework [16] to derive a competition based pricing scheme, in which the resource allocation result achieves fairness and efficiency, and maximizes provider revenue under the proposed pricing scheme.

In a billing cycle t, let $N(t)$ be the number of users who accept the payment in the first pricing stage, where user n ($n \in \{1, \ldots, N(t)\}$) requires a VM bundle $D_n = (d_{n1}, \ldots, d_{nM})$ at a basic fulfillment ratio x_n^b. Recall that only users who accept the price in the first stage will participate in the second stage, therefore, $\sum_{1 \leqslant t \leqslant T} N(t) \leqslant K$, where K is the number of all the cloud users. Each user n submits a parameter B_n as its time averaged budget for the excess fulfillment ratio beyond x_n^b. For each type of resource l, the sum of available resource units in the reserved resource pool and elastic resource pool is $C_l(t)$. The resources that user job n consumes during its execution are $A_n = (a_{n1}, \ldots, a_{nL})$ where $a_{nl} = \sum_{1 \leqslant m \leqslant M} d_{nm}\alpha_{ml}$. Let x_n be the amount of fulfillment ratio finally assigned to user n, *i.e.*, $x_n(t) = x_n^b + x_n^e(t)$ where x_n^b and $x_n^e(t)$ are the basic and extra fulfillment ratio of job n. Then the resource consumption of user n is $A_n x_n(t)$. The same pricing scheme is applied in all billing cycles; we focus on the t^{th} cycle and drop t from the notation hereafter.

Desired pricing and allocation properties. For the second-stage pricing scheme, there are several desirable properties from both cloud users and the provider sides. For cloud users, a basic pricing property is that they should never be charged a price higher than their respective budget. Moreover, proportional fairness requires that the extra fulfillment ratio each user obtains is proportional to its budget, so that a user would never be penalized for reporting a higher budget. From the provider's point of view, it is desirable to make full utilization of its idle resources to serve users by increasing their fulfillment ratio, so as to mitigate system load during demand peaks. Therefore, resource efficiency is desired. Specifically, the resource allocation scheme should achieve

Pareto-optimality, where a user cannot increase its fulfillment ratio without sacrificing the fulfillment ratio of other users. Last but not least, the provider naturally wishes to maximize its revenue by serving cloud users with idle resources.

Utilizing Nash bargaining solution for resource allocation. To achieve the aforementioned properties of resource allocation for additional fulfillment ratio, inspired by [17], we resort to asymmetric Nash bargaining to obtain a Nash bargaining solution, from which we derive the pricing and allocation scheme subject to user resource demands and budgets. In the Nash Bargaining game, multiple players enter the game with an initial utility and a utility function. Throughout the game, they cooperate to achieve a win-win solution, where the Nash product as the social utility gains is maximized. This is similar to our resource allocation problem, where each user job competes for extra fulfillment ratio from idle resources with an initial basic fulfillment ratio, and the provider aims to maximize the joint profit attained at all cloud users. Note that the Nash bargaining solution can be calculated efficiently without requiring the players to actually engage in a physical bargaining process. We only utilize the solution of Nash bargaining to allocate resource.

In our context of resource allocation for extra fulfillment ratio, the game can be described as follows: N user jobs act as players, and compete for the limited idle resources in the cloud. Since we allocate idle resources to jobs to increase their fulfillment ratio, we use the acquired service fulfillment ratio x_n as the utility for each user job n. Each job is assigned an initial fulfillment ratio, which is their required basic fulfillment ratio guaranteed in the first pricing stage. Since all the players have their respective budgets, we apply the asymmetric weighted Nash bargaining solution and assign them with different contributions to the social welfare by using the exponentiation of the utility gains, *i.e.*, $(x_n - x_n^b)^{B_n}$. Next, we will demonstrate that the Nash bargaining solution derives a pricing and resource allocation strategy that achieves resource efficiency, proportional fairness while maximizing the provider revenue.

Let $X \subset \mathcal{R}^N$ be the space of achievable fulfillment ratio vector for N jobs in the cloud, where $\mathcal{R}^N$ is the set of all allocation strategies. Given the corresponding initial utility for job n, x_n^b, we represent the vector of initial utilities of all user jobs as X_0=$(x_1^b, x_2^b, \ldots, x_N^b) \in X$. Since each x_n is a closed domain, the allocation space X is a convex and closed set. Let G=$\{x | x \in X, x_n \geqslant x_n^b\}$ be the subset of allocation strategies that enable users to achieve at least their basic service fulfillment ratio. Assume that G is nonempty, then (G, X_0) is a bargaining game, where users with initial utility vector X_0 bargain to achieve an eventual solution in G.

Definition 2. *A mapping $\mathcal{S}$: $(G, x_0) \to \mathcal{R}^N$ is a Nash bargaining solution if it satisfies: $S(G, x_0) \in G$, Pareto optimality, symmetry, scale independence, and independence of irrelevant alternatives [18].*

Let J=$\{x_n | x \in G, x_n > x_n^b\}$ be the set of jobs that can achieve strictly higher than basic service fulfillment ratio. The first stage of our framework guarantees that there are sufficient resources to provide basic fulfillment ratio to all users, *i.e.*, $\sum_{n=1}^N a_{nl} x_n^b < C_l; l \in \{1, \ldots, L\}$, so set J is nonempty. Since X is a convex and compact subset of $\mathcal{R}^N$, we have the following theorem [19]:

Theorem 1. *There exists a Nash bargaining solution and the elements of the vector $x = \mathcal{S}(G, x_0)$ solve the following optimization problem:*

$$\max_{x_n} \quad \prod_{x_n \in J} (x_n - x_n^b)^{B_n} \tag{8}$$

Theorem 1 implies that there exists a Nash bargaining solution in our resource allocation game for extra fulfillment ratio. The convex optimization above has a unique solution equivalent to the Nash bargaining solution. Eq. (8) illustrates the form of the joint profit in the bargaining game, which is the product of the utility gains of all the players and can be maximized by the Nash bargaining solution. The objective function (8) is equivalent to the objective $\max_{x_n} \sum B_n \ln(x_n - x_n^b), \forall x_n \in J$.

Considering the constraints for each x_n and the total amount of cloud resources $C_l = C_l^b + C_l^e$, we can formulate the optimization problem as follows:

$$\max_{x_n} \quad \sum_{n=1}^N B_n \ln(x_n - x_n^b) \tag{9}$$

$$\text{s.t.} \quad \sum_{n=1}^N a_{nl} x_n \leqslant C_l, \qquad l \in \{1, ..., L\} \tag{10}$$

$$x_n \geqslant x_n^b, \qquad n \in \{1, ..., N\} \tag{11}$$

$$x_n \leqslant 1, \qquad n \in \{1, ..., N\} \tag{12}$$

where constraint (10) ensures the capacity is not violated for any type of resource. Constraint (11) ensures that basic service fulfillment ratio is guaranteed. Constraint (12) expresses the fact that the fulfillment ratio can be no larger than 1. By solving the optimization problem (9), we obtain the following theorem:

Theorem 2. *There exist $\mu_l \geqslant 0$ $(l \in \{1, \ldots, L\})$ such that*

- *If $B_n = 0$ then $x_n = x_n^b$.*
- *If $B_n > 0$ and $\sum_{l=1}^L \mu_l a_{nl} > 0$ then*

$$x_n = x_n^b + \min\{1 - x_n^b, \frac{B_n}{\sum_{l=1}^L \mu_l a_{nl}}\} \tag{13}$$

where for each $l \in \{1, \ldots, L\}$, $(\sum_{n=1}^N a_{nl} x_n - C_l)\mu_l = 0$.

- *If $\sum_{l=1}^L \mu_l a_{nl} = 0$, then $x_n = 1$.*

where x_n is the unique Nash bargaining solution for the optimization problem.

Proof: Recall that the allocation space X is nonempty, convex and compact. Define

$$f(x) = \sum_{n=1}^N B_n \ln(x_n - x_n^b) \tag{14}$$

then $f(\cdot)$ is strictly concave.

Since the constraints are linear in x_i and the objective function is continuously differentiable, the first-order Karush-Kuhn-Tucker (KKT) conditions are necessary and sufficient for optimality [20].

Let $\mathcal{L}(x, \lambda, \beta, \mu)$ denote the Lagrangian where $\lambda_n \geqslant 0$ $(n = 1, \ldots, N)$, $\beta_n \geqslant 0$ $(n = 1 \ldots, N)$, and $\mu_l \geqslant 0$ $(l = 1, \ldots, L)$ denote the Lagrange multipliers associated with the constraints corresponding to x_n^b and resource capacity respectively. Then $\mathcal{L}(x, \lambda, \beta, \mu)$ can be represented as:

$$\mathcal{L}(x, \lambda, \beta, \mu) = f(x) - \sum_{n=1}^{N} \lambda_n (x_n^b - x_n) - \sum_{n=1}^{N} \beta_n (x_n - 1) - \sum_{l=1}^{L} \mu_l (\sum_{n=1}^{N} a_{nl} x_n - C_l) \qquad (15)$$

The first-order necessary and sufficient conditions include:

$$B_n + (\lambda_n - \beta_n - \sum_{l=1}^{L} \mu_l a_{nl})(x_n - x_n^b) = 0; \qquad n \in \{1, \ldots, N\} \qquad (16)$$

$$(x_n - x_n^b)\lambda_n = 0; \quad \lambda_n \geqslant 0; \quad n \in \{1, \ldots, N\} \qquad (17)$$

$$(x_n - 1)\beta_n = 0; \quad \beta_n \geqslant 0; \quad n \in \{1, \ldots, N\} \qquad (18)$$

$$(\sum_{n=1}^{N} a_{nl} x_n - C_l)\mu_l = 0; \quad \mu_l \geqslant 0; \quad n \in \{1, \ldots, N\} \qquad (19)$$

Since $\sum_{n=1}^{N} a_{nl} x_n^b \leqslant c_l^b < C_l$ is guaranteed in the first stage, the constraints $x_n \geqslant x_n^b$ are nonactive and hence $\lambda_n = 0$ for all $n \in \{1, \ldots, N\}$. Furthermore, $\beta_n = 0$ if $x_n < 1$, otherwise $x_n = 1$. By deriving x_n from (16), we obtain the result as stated in Theorem 2. ■

The Lagrange multiplier μ_l can be interpreted as the implied cost associated with type-l resource. It reflects the marginal cost of fulfillment ratio improvement for a user using resource type-l. Accordingly, for each user n, the cost of improving its service fulfillment ratio by one unit is $\sum_{l=1}^{l} a_{nl}\mu_l$. The resulting fulfillment ratio allocation scheme is the bargaining result at price μ_l $(1 \leqslant l \leqslant L)$. Based on the above solution, we design a natural pricing structure. Let P_n^e denote the price charged to user n $(n = 1, 2, \ldots, N)$ in the second stage, then

$$P_n^e = (x_n - x_n^b) \sum_{l=1}^{L} a_{nl}\mu_l \qquad (20)$$

From Eq. (20) and Eq. (19), we can observe that the pricing scheme resulted from the Nash Bargaining framework charges cloud users according to the extent of their competition in resource usage. A higher total user demand on type-l resource leads to a higher marginal cost in u_l.

From the pricing scheme and resource allocation scheme derived from the Nash bargaining solution, we can observe that each user's fulfillment ratio increase is proportional to its bid B_n. Resource efficiency is achieved since the Nash bargaining solution achieves Pareto-optimality. Furthermore, users will never be charged a price greater than their bids. We next prove that the Nash bargaining solution maximizes provider revenue obtained in the pricing scheme in Eq. (20).

Theorem 3. *The Nash bargaining solution maximizes the providers revenue under the pricing scheme in Eq. (20).*

Proof: Revenue maximization at the provider can be formulated as:

$$\max_{x_i} \quad \sum_{i=1}^{N} \sum_{l=1}^{L} a_{nl}\mu_l (x_n - x_n^b) \qquad (21)$$

$$\text{s.t.} \quad x_n^b \leqslant x_n, \qquad n \in \{1, \ldots, N\} \qquad (22)$$

$$x_n \leqslant 1, \qquad n \in \{1, \ldots, N\} \qquad (23)$$

$$\sum_{n=1}^{N} a_{nl} x_n \leqslant C_l, \qquad l \in \{1, \ldots, L\} \qquad (24)$$

Its Largangian $\mathcal{L}(x, \delta, \eta, \kappa)$ can be presented as:

$$\mathcal{L}(x, \delta, \eta, \kappa) = \sum_{i=1}^{N} \sum_{l=1}^{L} a_{nl}\mu_l (x_n - x_n^b) - \sum_{n=1}^{N} \delta_n (x_n^b - x_n) - \sum_{n=1}^{N} \eta_n (x_n - 1) - \sum_{l=1}^{L} \kappa_l (\sum_{n=1}^{N} a_{nl} x_n - C_l) \qquad (25)$$

The first-order necessary and sufficient conditions are:

$$-\sum_{l=1}^{L} a_{nl}\mu_l + \sum_{l=1}^{L} a_{nl}\kappa_l = 0; \quad n \in \{1, \ldots, N\} \qquad (26)$$

$$(x_n - x_n^b)\delta_n = 0; \quad \delta_n \geqslant 0; \quad n \in \{1, \ldots, N\} \qquad (27)$$

$$(x_n - 1)\eta_n = 0; \quad \eta_n \geqslant 0; \quad n \in \{1, \ldots, N\} \qquad (28)$$

$$(\sum_{n=1}^{N} a_{nl} x_n - C_l)\kappa_l = 0; \quad \kappa_l \geqslant 0; \quad n \in \{1, \ldots, N\} \qquad (29)$$

By taking $\delta_n = \lambda_n$, $\eta_n = \beta_n$, and $\kappa_l = \mu_l$, we can see that the Nash bargaining solution in Theorem 2 also satisfies the first-order Karush-Kuhn-Tucker necessary and sufficient conditions in (26)-(29). ■

V. Performance Evaluation

We conduct trace-driven simulation studies based on large scale real-world traces (Google cluster usage trace [9]) to evaluate the performance of our dynamic pricing framework for service fulfillment ratio. The trace contains information on resource usage from 933 users in a Google cluster, collected during a period of 29 days.

A. Preprocessing and Simulation Settings

In the dataset, resource usage information of users are recorded in the form of *jobs* and *tasks*. Each job consists of several tasks, and each task has a demand on computing resources such as CPU and memory. We take each job in the dataset as a cloud user. The resource demand of each user is calculated as the sum of resource demands of all the tasks in the corresponding job.

We generate for each user its required VM bundle, job submission time, and job execution time from collected information of resource demands, submission time, and completion time of all jobs, in a five-day period from the trace. Each billing cycle is set to 1 minute. Resource usage in the trace is normalized by the amount of resources in the largest physical server in the cluster. We assume that the largest server has 32 CPU cores with 128GB memory. Accordingly, we can get the actual resource demand of each job. Then we calculate

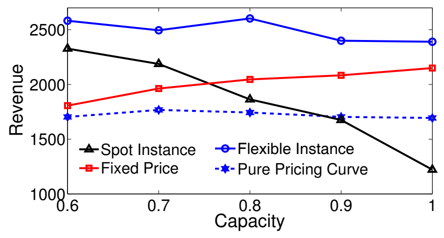

Fig. 5. Provider revenue under different pricing strategies with different capacity of reserved resource pool.

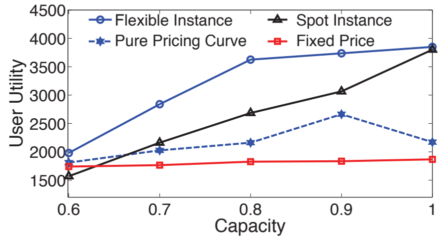

Fig. 6. Total user utility under different pricing strategies with different capacity of reserved resource pools.

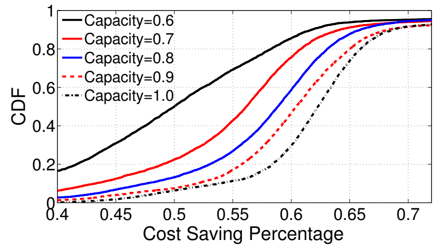

Fig. 7. The CDF of user cost saving percentage compared to on-demand service.

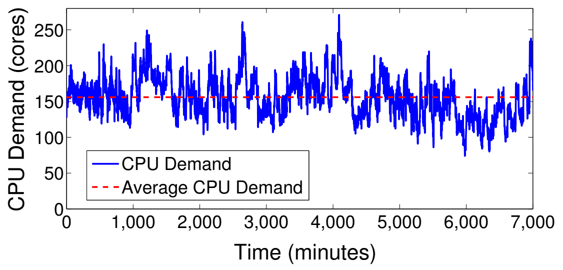

Fig. 4. CPU usage in Google cluster trace in a 5 day period.

the bundle of VMs requested by each user, as the most cost effective combination of VMs from two instance families (*i.e.*, "High CPU" and "High memory") in the Google Compute Engine. Fig. 4 depicts the aggregated CPU demand of all VM bundles required by cloud users. We can observe strong fluctuation in total resource demand, which calls for a proper pricing and allocation scheme to optimize resource utilization.

In the simulation, we set the basic service fulfillment ratio of a user uniformly distributed in $[0, 1]$, modeling delay tolerance of batch workloads. To model cloud users who don't have a sufficient budget for on-demand services, we set user valuation of job execution under basic service fulfillment ratio to be η times the cost of using on-demand service to execute their jobs, with η taken uniformly from $[0, 1]$. An extra budget of 10% of the valuation is available to speed up job execution.

We evaluate the performance of different pricing strategies with different capacity of the resource pool. We normalize the capacity of reserved resource pool, with the largest capacity to be the time averaged CPU and memory demand of all users. We calculate the upper bound of the pricing curve value $\overline{p}$ as the price of the combination of "High CPU" and "High Memory" on-demand instances providing the same capacity of the reserved resource pool. The lower bound $\underline{p}$ is set as 10% of the upper bound.

B. Comparing to Single Stage Strategies

We compare the performance of the proposed *flexible instance* with the following three single stage strategies, in terms of provider revenue, user utility, and resource utilization: a pure pricing curve based strategy that is identical to the first stage of our pricing framework (Pure Pricing Curve), a fixed price strategy with VMs with full fulfillment ratio at half price of the on-demand service (Fixed Price), and a straightforward implementation of spot instance (Spot Instance). In the spot instance strategy, we greedily allocate resources to cloud users who have the highest valuation on resource usage in each minute until the resource pool is exhausted. All users are charged at the valuation of the last admitted user in the process. For fair comparison, we set the capacity of elastic resource pool as the amount of idle resources in the reserved resource pool, *i.e.*, no idle resources from other services are considered.

Provider Revenue. Fig. 5 illustrates the revenue of provider under different pricing strategies with different capacity of the reserved resource pool. We can observe that *flexible instance* outperforms other strategies in revenue. Pure Pricing Curve reaches a stable revenue with different resource capacities, since it is adaptive to the demand-supply relation. However, such stable revenue is relatively low, since it does not glean revenue from idle resources in the system. Revenue achieved by Spot Instance decreases as the capacity of resource pool increases. This is because in spot instance, the service price is set as the lowest valuation among all the admitted users. A larger resource pool admits more users, thus resulting in a lower service price. Compared to *flexible instance*, Fixed Price earns less revenue since its price does not adapt to the fluctuation of total user demand.

User Utility and Cost Saving. We calculate the utility of each user as its valuation of the service minus payment. We set the service valuation to be 0 if a user job cannot complete before deadline. Fig. 6 shows the total utility of all users under different pricing strategies. *Flexible instance* consistently outperforms other alternatives. Compared with Pure Pricing Curve, by accelerating job execution with idle resources during off-peak periods, *flexible instance* executes user jobs faster, thus admitting more jobs in demand peaks. By serving a larger user population, *flexible instance* leads to a higher total utility. Spot Instance achieves a lower user utility with small resource capacity, since it does not provide any guarantee in terms of service fulfillment ratio, which may result in deadline violation to cloud jobs. User utility from Fixed Price stays stable as resource capacity increases, since it fails to attract more user demands in a higher resource supply.

We further investigate the cost saving for each user by using *flexible instance*, instead of on-demand service to execute their jobs. Fig. 7 depicts the cost saving percentage *flexible instance* brings to cloud users with different capacity of reserved resources. We can observe that the cost saving percentages increase as the capacity of reserved resource increases. The reason is that *flexible instance* adapts to the demand-supply relation. Bigger resource capacity implies more sufficient supply, therefore, leads to a lower service price, which brings a larger amount of cost saving.

Resource Utilization. Fig. 8 depicts the fluctuation of CPU utilization in a one-day period. *Flexible instance* achieves a

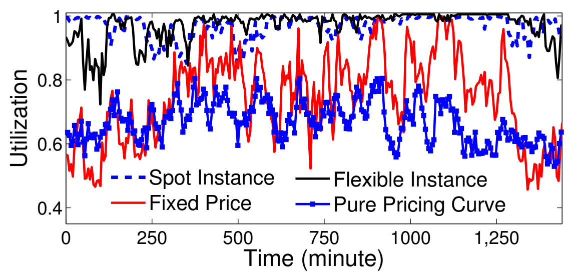

Fig. 8. A comparison of CPU utilization under different pricing strategies.

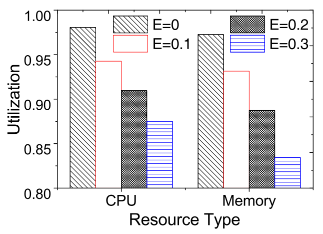

Fig. 10. Resource utilization with different elastic resource capacity.

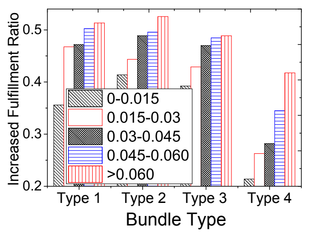

Fig. 11. The increased fulfillment ratio of different VM bundles under different acceleration budgets.

similar resource utilization to Spot Instance, which aggressively admits user jobs to utilize cloud resources. Thanks to the Nash bargaining solution in *flexible instance*, high resource efficiency is achieved when allocating idle resources to cloud users. Pure Pricing Curve and Fixed Price miss the opportunity of idle resource utilization and have lower resource utilization.

C. Impact of Parameters

Basic Fulfillment ratio. We evaluate the provider's revenue when the basic service fulfillment ratio required by cloud users are drawn from uniform distributions in the range of $[0.1, 0.3]$, $[0.3, 0.5]$, $[0.5, 0.7]$, and $[0.7, 0.9]$, respectively, with results depicted in Fig. 9. We can observe that the instant provider revenue in each minute fluctuates across time, due to the fluctuation of user demands. However, the provider constantly earns a higher revenue from cloud users with lower basic service fulfillment ratio. This is because that jobs with lower fulfillment ratio offer more space for job schedule maneuver. With the two pricing strategies working in concert, *flexible instance* can efficiently exploit the delay tolerant nature of cloud jobs to enhance resource utilization for the cloud service provider.

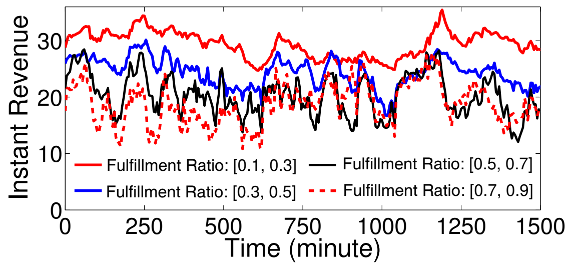

Fig. 9. The instant revenue of the provider with cloud users drawn from different fulfillment ratio distribution.

Capacity of Elastic Resources. We assume the capacity of elastic resources is uniformly distributed in $[0, E]$ across time, where E is normalized by the time averaged CPU and memory demand of all users. Fig. 10 illustrates the time averaged resource utilization of the reserved and elastic resource pools with different values in E, in *flexible instance*. Resource utilization gradually decreases with the increase of elastic resource capacity. Nonetheless, *flexible instance* can efficiently allocate idle resource to cloud users, and the overall utilization of both CPU and memory remain higher than 80% in all the cases.

Budget for Acceleration. We investigate 4 typical types of VMs bundle most popular to cloud users, and analyze the amount of increased fulfillment ratio acquired from different acceleration budgets. We divide the distribution of budgets into 5 ranges and present the average increased fulfillment ratio in each range in Fig. 11. We can observe that for each type of VM bundle, larger budgets lead to a higher increased fulfillment ratio, agreeing with the characteristics of the Nash bargaining solution.

VI. Related Work

Efficient trading mechanism design for cloud resources has been a focal point in the recent cloud computing literature. Substantial research has been devoted to solutions that maximize provider revenue or social welfare in different service models of cloud computing. Xu *et al.* [13] design a dynamic pricing mechanism to cope with stochastic demand and perishable resources in the cloud, with the target of maximizing the expected long-term revenue for the provider. Wang *et al.* [21] design a truthful multi-unit combinatorial auction for pricing VMs in the cloud, which is held round by round in each decision interval, without considering users' persistent demand for VMs over a period of time. In [22] and [14], users' continuous demand on VMs are processed instantaneously with truthful online auctions. However, these works consider either a single type of VM or constant capacity of available resources. Considering dynamic resource provisioning, [23] and [24] formulate the VM pricing problem as a combinatorial auction and design truthful mechanisms for winner determination. Shi *et al.* [25] propose an online auction framework for the cloud service that has a time varying capacity and dynamically assembles multiple types of resources into heterogeneous VMs, with each user having a total budget throughout a time span. Zhao *et al.* [26] and Roh *et al.* [27] investigate the dynamic pricing problem for a geo-distributed cloud where user jobs can be distributed to datacenters located in different regions with different resource capacity and power cost. Our work differs from the these existing researches in two aspects: (a) we price service fulfillment ratio to exploit the delay tolerance nature of batch cloud jobs, while guaranteeing their deadlines. (b) To regulate demand fluctuation, we propose a fair and efficient mechanism to let users compete for the transient idle resources in the off-peak periods to enhance utilization and alleviate peak loads through arbitrage of computation power.

Towards reducing monetary cost of renting VMs, a large body of recent research has been devoted to smart usage of existing pricing schemes offered by cloud service providers. To exploit the discount given to long term VM renting, Niu *et al.* [28] propose a semi-elastic cluster computing model for organizations to reserve and dynamically resize their VM cluster. Wang *et al.* [29] further design a brokerage service to multiplex user demand and perform long term VM reservation. Spot instance service is utilized in [30] to save the cost of running MapReduce workflows. Song *et al.* [7] studies the scenario where a broker utilizes low cost spot instances to save the cost of serving users. Zheng *et al.* [8] further investigate spot instance in not only the biding strategy of cloud users but also the pricing strategy for the provider. Rather than exploiting existing pricing schemes of cloud services, our work aims to design a new type of service which is resource efficient and cost effective for delay-tolerant jobs with deadlines.

While significant progress has been made to design efficient trading mechanisms for cloud services, delivering performance guaranteed service is only beginning to receive attention in the cloud context. Guo *et al.* [17] utilize the Nash bargaining approach to perform bandwidth allocation in cloud datacenter networks, so as to guarantee the bandwidth requirements of VMs while fairly sharing the residual idle bandwidth. In [31], Xu *et al.* explore the hardware heterogeneity and interference among VMs in IaaS clouds. Accordingly they design a VM provisioning framework that provide cloud users with VMs that have predictable performance. Niu *et al.* [32] investigate a framework that utilizes a hybrid cloud to deal with flash crowds in web applications. Using Lyapunov optimization techniques in request distribution, the solution can handle user requests with bounded response time meanwhile achieving cost-efficiency for the service provider. In our work, we design *flexible instance* to ensure that delay tolerant jobs can meet their deadlines when their scheduling flexibility are exploited to persuit resource efficiency for the service provider and cost efficiency for cloud users.

VII. Conclusion

This work represents a first attempt towards pricing service fulfillment ratio in IaaS clouds. The proposed *flexible instance* service complements existing cloud services by utilizing idle resources in the cloud to serve delay tolerant user jobs while guaranteeing a basic service fulfillment ratio to meet their corresponding deadlines. To fully utilize the idle resources and guarantee fulfillment ratio, we proposed a two-stage pricing framework. To agilely adapt resource prices to the demand-supply relation and to maximize provider revenue, we use pricing schemes derived from a well designed pricing curve and the Nash bargaining solution, respectively, in the two stages. Through large scale simulations driven by real-world cloud usage traces, we demonstrate that the *flexible instance* service can achieve cost-efficiency for batch job execution while improving the provider's revenue.

References

[1] IDC reveals worldwide big data and analytics predictions for 2015. [Online]. Available: http://www.idc.com/getdoc.jsp?containerId=prUS25329114

[2] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *Proc. of ACM SC*, 2011.

[3] Using windows azure to speed up genome research and save millions of dollars. [Online]. Available: http://windowswideopen.com/category/strata/

[4] Microsoft Azure Virtual Machine. [Online]. Available: https://azure.microsoft.com/en-us/services/virtual-machines/

[5] Amazon EC2 Spot instances. [Online]. Available: http://aws.amazon.com/ec2/purchasing-options/spot-instances/

[6] Preemptible instances in google compute engine. [Online]. Available: https://cloud.google.com/compute/docs/instances/preemptible

[7] Y. Song, M. Zafer, and K.-W. Lee, "Optimal bidding in spot instance market," in *Proc. of IEEE INFOCOM*. IEEE, 2012.

[8] L. Zheng, C. Joe-Wong, C. W. Tan, M. Chiang, and X. Wang, "How to bid the cloud," in *Proc. of the 2015 ACM SIGCOMM*. ACM, 2015.

[9] C. Reiss, J. Wilkes, and J. Hellerstein. "Google Cluster-Usage Traces". [Online]. Available: http://code.google.com/p/googleclusterdata/

[10] Amazon EC2 instance purchasing options. [Online]. Available: http://aws.amazon.com/ec2/purchasing-options/

[11] Google Compute Engine. [Online]. Available: https://cloud.google.com/compute/pricing

[12] Q. Zhang, M. F. Zhani, R. Boutaba, and J. L. Hellerstein, "Harmony: Dynamic heterogeneity-aware resource provisioning in the cloud," in *Proc. of IEEE ICDCS*. IEEE, 2013.

[13] H. Xu and B. Li, "Dynamic cloud pricing for revenue maximization," *IEEE Transactions on Cloud Computing*, 2013.

[14] W. Shi, C. Wu, and Z. Li, "RSMOA: A revenue and social welfare maximizing online auction for dynamic cloud resource provisioning," in *Proc. of IEEE IWQoS*, 2014.

[15] R. Lavi and N. Nisan, "Competitive analysis of incentive compatible on-line auctions," in *Proc. of ACM EC*. ACM, 2000.

[16] H. Boche, M. Schubert, N. Vucic, and S. Naik, "Non-symmetric nash bargaining solution for resource allocation in wireless networks and connection to interference calculus," in *Proc. of European Signal Processing Conference*, 2007.

[17] J. Guo, F. Liu, J. C. S. Lui, and H. Jin, "Fair network bandwidth allocation in iaas datacenters via a cooperative game approach," *IEEE/ACM Transactions on Networking (TON)*, vol. 24, no. 2, pp. 873–886, April 2016.

[18] N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani, *Algorithmic game theory*. Cambridge University Press, 2007.

[19] H. Yaïche, R. R. Mazumdar, and C. Rosenberg, "A game theoretic framework for bandwidth allocation and pricing in broadband networks," *IEEE/ACM Transactions on Networking (TON)*, 2000.

[20] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2009.

[21] Q. Wang, K. Ren, and X. Meng, "When cloud meets ebay: Towards effective pricing for cloud computing," in *Proc. of IEEE INFOCOM*, 2012.

[22] H. Zhang, H. Jiang, B. Li, F. Liu, A. Vasilakos, and J. Liu, "A framework for truthful online auctions in cloud computing with heterogeneous user demands," *IEEE Transactions on Computers*, no. 1, pp. 1–1, 2015.

[23] S. Zaman and D. Grosu, "A combinatorial auction-based mechanism for dynamic vm provisioning and allocation in clouds," *Cloud Computing, IEEE Transactions on*, vol. 1, no. 2, pp. 129–141, 2013.

[24] M. M. Nejad, L. Mashayekhy, and D. Grosu, "Truthful greedy mechanisms for dynamic virtual machine provisioning and allocation in clouds," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 26, no. 2, pp. 594–603, 2015.

[25] W. Shi, L. Zhang, C. Wu, Z. Li, and F. C. Lau, "An online auction framework for dynamic resource provisioning in cloud computing," in *Proc. of ACM SIGMETRICS*, 2014.

[26] J. Zhao, H. Li, C. Wu, Z. Li, Z. Zhang, and F. Lau, "Dynamic pricing and profit maximization for clouds with geo-distributed datacenters," in *Proc. of IEEE INFOCOM*, 2014.

[27] H. Roh, C. Jung, W. Lee, and D.-Z. Du, "Resource pricing game in geo-distributed clouds," in *Proc. of IEEE INFOCOM*, 2013.

[28] S. Niu, J. Zhai, X. Ma, X. Tang, and W. Chen, "Cost-effective cloud hpc resource provisioning by building semi-elastic virtual clusters," in *In Proc. of IEEE SC*, 2013.

[29] W. Wang, D. Niu, B. Li, and B. Liang, "Dynamic cloud resource reservation via cloud brokerage," in *Proc. of IEEE ICDCS*, 2013.

[30] N. Chohan, C. Castillo, M. Spreitzer, M. Steinder, A. Tantawi, and C. Krintz, "See spot run: using spot instances for mapreduce workflows," in *In Proc. of USENIX HotCloud*, 2010.

[31] F. Xu, F. Liu, and H. Jin, "Heterogeneity and interference-aware virtual machine provisioning for predictable performance in the cloud," *IEEE Transactions on Computers*, vol. PP, no. 99, pp. 1–1, 2015.

[32] Y. Niu, B. Luo, F. Liu, J. Liu, and B. Li, "When hybrid cloud meets flash crowd: Towards cost-effective service provisioning," in *Proc. of IEEE INFOCOM*. IEEE, 2015.