How to Bid the Cloud

Liang Zheng[*‡]   Carlee Joe-Wong[*]   Chee Wei Tan[†‡]   Mung Chiang[*]   Xinyu Wang[‡]
[*]Princeton University   [†]National University of Singapore [‡]City University of Hong Kong
{liangz, cjoe, chiangm}@princeton.edu   cheetan@alumni.caltech.edu   xinywang@cityu.edu.hk

ABSTRACT

Amazon's Elastic Compute Cloud (EC2) uses auction-based spot pricing to sell spare capacity, allowing users to bid for cloud resources at a highly reduced rate. Amazon sets the spot price dynamically and accepts user bids above this price. Jobs with lower bids (including those already running) are interrupted and must wait for a lower spot price before resuming. Spot pricing thus raises two basic questions: how might the provider set the price, and what prices should users bid? Computing users' bidding strategies is particularly challenging: higher bid prices reduce the probability of, and thus extra time to recover from, interruptions, but may increase users' cost. We address these questions in three steps: (1) modeling the cloud provider's setting of the spot price and matching the model to historically offered prices, (2) deriving optimal bidding strategies for different job requirements and interruption overheads, and (3) adapting these strategies to MapReduce jobs with master and slave nodes having different interruption overheads. We run our strategies on EC2 for a variety of job sizes and instance types, showing that spot pricing reduces user cost by 90% with a modest increase in completion time compared to on-demand pricing.

CCS Concepts

•Networks → **Network performance evaluation;** *Data center networks;* Cloud computing;

Keywords

Cloud pricing, spot instance, optimization.

1. INTRODUCTION

Cloud providers face an increasingly complicated problem of allocating their resources to different users. These resource allocations must take into account both available capacity within datacenter networks as well as individual jobs' requirements, which are sometimes specified in binding service level agreements. The large number of users submitting different types of jobs to any given cloud further complicates the problem, creating a highly dynamic environment as jobs are submitted and completed at different times [4].

1.1 Spot Pricing to Shape User Demand

While many works have considered the operational problem of scheduling jobs within a datacenter [8, 14, 22, 33], most have taken user demands as a given input. Yet user demand can be partially controlled by cloud providers' pricing [6]. The majority of today's pricing plans for cloud resources are variants on usage-based pricing, in which users pay a static per-unit price (per workload or per hour) to access cloud resources [21]. Usage-based pricing can affect overall demand levels, but does not even out short-term fluctuations [13]. To manage these fluctuations in demand for a fixed amount of available datacenter capacity, cloud providers could introduce more flexible pricing plans in which resources are priced according to real-time market demand [40]. Amazon's Elastic Compute Cloud (EC2) spot pricing [2] is an example of such a strategy.

Spot pricing creates an auction-based market for available cloud computing resources. Users can submit bids to the market at any time, using the spot price history to help decide how much to bid (i.e., their bid prices). The cloud provider sets the spot price at regular time intervals, e.g., every five minutes, depending on the number of bids received from users (demand) and how many resources are available (supply) at each time slot.[1] In each time slot, users' bids above the spot price are accepted, and those below it rejected. Running spot instances are terminated if their original bid prices fall below the new spot price, and re-launched only when their bids again exceed the spot price. Given that these interruptions may affect user utility, two natural questions then arise: *how might the provider set the spot price*, and *what prices should users bid*?

---

[1]These available resources include idle resources and those in use by launched spot instances.

We answer these questions in this work. Unlike most works on spot pricing, which consider only the provider's viewpoint [10, 17, 34, 38], we aim to develop both *a reasonable model for how the provider sets the spot prices* and *optimal bidding strategies for the user*. We do not seek to design the auction mechanism used by the provider, only to systematically model and estimate how the provider might set the spot prices before we develop users' optimal bidding strategies. These bidding strategies, in turn, depend not on the specific model of how providers choose the spot prices, but rather on the chosen spot prices themselves. They can thus be applied to other provider pricing models.

1.2 Research Challenges and Contributions

We can gain a basic statistical understanding of Amazon's prevailing spot prices by studying the two-month history made available by Amazon [1,15]. Though these statistics alone are sufficient for the user to deduce a bid price, they also give us some insights into how the provider sets the spot prices. By developing a model that relates the spot price to users' submitted bids, we can better understand the different factors considered by the provider in setting the spot prices.

Users' job completion times consist of three components: the time to execute the job, idle time while the spot price exceeds the user bid price, and the time to recover from job interruptions. A lower bid price reduces the cost of running the job during its execution time, but can extend the recovery time by inducing more job interruptions. This extra recovery runtime can result in a higher overall job cost, depending on the recovery time from each interruption (i.e., the job's "interruptibility") and the spot prices offered by the provider. Complicating this tradeoff is the fact that user jobs can have long runtimes spanning many changes in the spot price [15]. Users then face two key challenges:

1) Users must predict the spot prices in order to optimize their bids for a given job, not only in the next time slot but also for all future time slots until the job is completed.

2) Different jobs may have different degrees of interruptibility. Even within a single user job that requires many computational nodes, e.g., MapReduce's master/slave node model, different nodes can have different interruptibility requirements.

Figure 1 shows the basic architecture of a user client and its interaction with the cloud provider. The client calculates its bid price based on two inputs: user inputs on job characteristics and the historical distribution of the spot prices offered by the cloud provider. A price monitor keeps the spot price distribution up to date, and a job monitor at the provider tracks whether the job is ever outbid. The monitor also restarts users' jobs when the spot price falls below their bids.

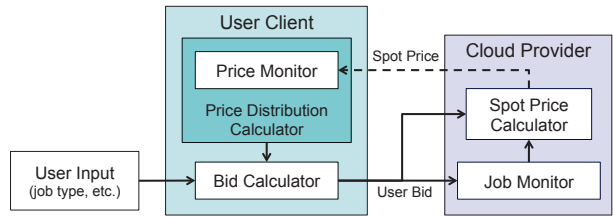We first give a brief overview of related work in Section 2 and relevant background on Amazon's EC2 in

Figure 1: Client and cloud provider interaction.

Section 3 before developing the following results:

A framework for understanding spot prices (Section 4): We develop a model to understand how the provider sets the spot price, using it to bound users' quality of service and testing it against empirical data.

Users' optimal bidding strategies (Section 5): Given a predicted spot price distribution, we derive optimal bidding strategies for different degrees of job interruptibility.

Adaptation to MapReduce jobs (Section 6): We adapt our bidding strategies for the master and slave nodes of MapReduce jobs and implement our bidding strategy for a MapReduce job.

Experiment on Amazon EC2 (Section 7): We run our client on a variety of EC2 instances and job types. The proposed bidding strategies substantially lower users' costs in exchange for modestly higher running times.

Our work considers spot pricing from the point of view of both the cloud provider and users, allowing us to better understand how providers set the spot prices and to develop bidding strategies customized to different user requirements. We discuss some limitations of our work in Section 8 before concluding the paper in Section 9. All proofs can be found in the Appendix.

2. RELATED WORK

Cloud scheduling and pricing. Many works have considered resource allocation in the cloud from a purely operational perspective [8, 14, 22, 33]. Others incorporate pricing considerations, e.g., dynamically allocating cloud resources, so as to maximize the provider's revenue [10, 17, 34] or social welfare [25, 38]. We construct a similar model but relate it to empirical bid prices and use it to develop bidding strategies for users. Joint user-provider interactions for cloud services are considered in [26], but auction-specific works on both provider and user actions are limited to statistical studies of historical spot prices [1, 15].

Game theoretic pricing. Spot pricing is a distributed solution to the problem of allocating cloud resources to users, which can be treated as a game between users. Some works have considered the Nash equilibrium of such a pricing game in cloud scenarios [3, 31, 32], although they do not consider optimal bids in a resource auction. More generally, auctions have been proposed as a solution to generic distributed allocation games over multiple time slots [20]. Users'

optimal bidding strategies in such auctions can be quite complex, particularly if multiple users try to optimize their bids at the same time [36]. We assume in this paper that users' bid optimizations do not significantly affect the distribution of the spot prices, and discuss the consequences of relaxing this assumption in Section 8.

Auctions and bidding. User bidding strategies for cloud auctions are much less studied than provider strategies. While some works have shown that users can reduce their costs by using spot rather than on-demand instances [28, 37], they only consider heuristic bidding strategies for single-instance jobs.

In general, auction frameworks assume that users' bids are determined by their valuations of the auctioned resource. Indeed, many works have studied the problem of designing online auctions to ensure truthful user bids [5, 11, 27, 29], including improvements to Amazon's spot pricing [35]. However, *in cloud scenarios, user valuations for the instance in a given time slot depend on (unknown) future spot prices.* While users may know their valuations for completing a job, job interruptions will increase the number of instance-time slots required to complete the job. This dependence differentiates our work from other auctions for computing or utility resources, e.g., auctions for smart grid electricity [9], secondary spectrum access [16], grid computing [19], or Internet data [24].

3. BACKGROUND

3.1 User Jobs and Instance Types

Throughout this work, we consider Infrastructure-as-a-service (IaaS) cloud services, which are essentially remote virtual machines (VMs) with CPU, memory, and storage resources [2]. We follow Amazon's terminology and use the term "instance" to denote the use of a single VM. Instances can be divided into several discrete types, each of which may have different resource capacities; users generally submit resource requests, or bids, separately for different instance types.

The simplest types of user jobs require only one instance and can be served by placing a single bid request for a given instance type. Parallelizable jobs, on the other hand, might require multiple instances running in parallel. The MapReduce framework is a common realization of this job structure. MapReduce divides job functions among a master node and several slave nodes. The master node assigns computational tasks to slave nodes and reschedules the tasks whenever a slave node fails. After all the tasks are finished on the slave nodes, the master node returns a result.

3.2 EC2 Pricing

Amazon offers three types of pricing for instances: reserved, on-demand, and spot instances. Reserved instances guarantee long-term availability (e.g., over a year), and on-demand instances offer shorter-term usage (e.g., one hour on an instance). Both reserved and on-demand instances charge fixed usage-based prices. Spot instances, however, do not guarantee availability; users can use spot instances only if their bids exceed the spot price. Amazon generally updates the spot price every five minutes and encourages users to run interruptible jobs on spot instances.[2]

Spot instances allow two types of bids: one-time and persistent. One-time bids are submitted once and then exit the system once they fall below the current spot price. Thus, submitting a one-time bid runs the risk of having the job interrupted without completing. Persistent bids, however, are resubmitted in each time period until the job finishes or is manually terminated by the user. One-time bids allow for better control over bid completion times (e.g., users may default to on-demand instances if the jobs are not completed), while persistent bids allow the user to submit a bid request and then simply wait until the job finishes.

4. CLOUD PROVIDER MODEL

In this section, we develop an explanatory model for a cloud provider's offered spot prices. Though the model's primary utility for users is in explaining the spot price distribution used in Sections 5 and 6, we also derive some insights into the provider's behavior. In particular, since users' persistent requests are continually reconsidered if not satisfied, the provider might experience unsustainably large numbers of pending bids. We show that the number of pending bids remains bounded under reasonable conditions by formulating a model for a cloud provider to choose the spot prices in Section 4.1 and considering the effects on pending bids in Section 4.2. We validate our model in Section 4.3 by fitting it to two months' history of spot prices offered by Amazon.

Let us consider a series of discrete time slots $t \in \{1, 2, \dots\}$. At time slot t, the demand for a spot instance type is $L(t)$, and we use $\pi(t)$ to denote the spot price at time slot t. We restrict the spot price to not exceed the on-demand price $\bar{\pi}$ of the same instance type. We also impose the constraint $\pi(t) \geq \underline{\pi}$, where $\underline{\pi} \geq 0$ represents the provider's marginal cost of running a spot instance.

4.1 Revenue Maximization

When setting the spot price $\pi(t)$ in each time slot t, the cloud provider wishes to maximize its revenue $\pi(t)N(t)$, where $N(t)$ is the number of accepted bids (i.e., the system workload) and each successful bidder is charged only the spot price $\pi(t)$, regardless of the bid (s)he placed.[3] Other objectives, such as clearing the market, are also possible [10, 38, 39]; in fact, some stud-

---

[2] Amazon charges users separately for ingress and egress bandwidth to EC2. Since the required bandwidth is determined by the user job rather than the instance on which the job is run, we do not consider these costs.

[3] In practice N is a discrete integer, but for tractable analysis we take N continuous in our model.

ies have suggested that Amazon does not use revenue-maximizing spot prices [1]. Thus, we also include a capacity utilization term $\beta \log\left(1 + N(t)\right)$, which increases with $N(t)$. This term models the fact that the provider incurs a machine on/off cost for idle spot instances, giving it an incentive to accept more bids. We use a strictly concave function to penalize extremely heavy workloads, which can incur additional capacity costs.

At each time slot t, the provider chooses the spot price $\pi(t)$ so as to maximize the sum of the utilization term and revenue, $\beta \log\left(1 + N(t)\right) + \pi(t)N(t)$, subject to the constraint that $\pi(t)$ lie between the minimum and maximum prices ($\underline{\pi} \leq \pi(t) \leq \bar{\pi}$).[4] We denote this optimal price by $\pi^\star(t)$.

We now formulate the provider's optimization problem by using the probability distribution f_p to denote the distribution of bids received by the provider. For instance, if users' bid prices follow a uniform distribution, then $f_p(x) = 1/(\bar{\pi} - \underline{\pi})$. Defining $L(t)$ as the total number of bids submitted, the number of accepted bids is $N(t) = L(t)\frac{\bar{\pi}-\pi(t)}{\bar{\pi}-\underline{\pi}}$, or the fraction of submitted bids that exceed the spot price.[5] The provider then maximizes the sum of its revenue and utilization term:

$$\begin{aligned} \underset{\pi(t)}{\text{maximize}} \quad & \beta \log\left(1 + L(t)\frac{\bar{\pi} - \pi(t)}{\bar{\pi} - \underline{\pi}}\right) \\ & + \pi(t)L(t)\frac{\bar{\pi} - \pi(t)}{\bar{\pi} - \underline{\pi}} \\ \text{subject to} \quad & \underline{\pi} \leq \pi(t) \leq \bar{\pi}. \end{aligned} \quad (1)$$

In the rest of the paper, we use a uniform distribution for f_p, as is often used to model distributions of user valuations for computing services [30].

We solve (1) to find that the optimal spot price $\pi^\star(t)$ satisfies

$$L(t) = \frac{\bar{\pi} - \underline{\pi}}{\bar{\pi} - \pi^\star(t)}\left(\frac{\beta}{\bar{\pi} - 2\pi^\star(t)} - 1\right). \quad (2)$$

We can thus solve for the optimal solution to (1):

$$\begin{aligned} \pi^\star(t) = \max\Bigg\{ & \underline{\pi}, \; \frac{3}{4}\bar{\pi} + \frac{1}{2}(\bar{\pi} - \underline{\pi})\frac{1}{L(t)} \\ & -\frac{1}{4}\sqrt{\left(\bar{\pi} + 2(\bar{\pi} - \underline{\pi})\frac{1}{L(t)}\right)^2 + 8\beta(\bar{\pi} - \underline{\pi})\frac{1}{L(t)}} \Bigg\}. \end{aligned} \quad (3)$$

More weight on the utilization term (a higher β) leads to a lower spot price and more accepted bids. Since β will generally be small, we assume that $\beta \leq (L(t) +$

---

[4]In practice, we can emphasize the revenue over the utilization term by choosing a small scaling factor β. The provider can keep the number of accepted bids below its available capacity by increasing the minimum spot price $\underline{\pi}$ so that fewer bids are accepted.

[5]A higher spot price may cause more users to choose the more reliable on-demand instead of spot instances, which in turn affects user demand $L(t)$. Since the spot price is generally much lower than the on-demand price (cf. Section 7), we expect this effect to be negligible.
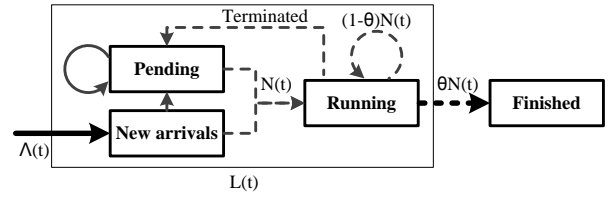

Figure 2: State transitions of spot instances. The solid and dashed arrow lines respectively represent the transitions at the start of the current and next time slots.

$1)(\bar{\pi} - 2\underline{\pi})$ and thus that $\pi^\star(t) > \underline{\pi}$ (the optimal spot price is above the minimum) in the rest of the paper.

In the above analysis, we considered user bids in a single time slot. However, bid resubmission may cause the spot price at time t to affect the prices in future time slots. We consider this dependency next.

4.2 Stable Job Queues

The dynamics of user requests (i.e., bids for an instance) consist of four distinct states: new arrivals, pending, running and finished, as shown in Figure 2.

At the beginning of time slot t, there are $L(t)$ bids for the spot resource remaining in the system, and $\Lambda(t)$ new bid arrivals. After the spot price is determined for this time slot, the $N(t)$ spot requests with the highest bid prices are successfully launched. The new arrivals with lower bid prices are then pended, and some pending requests remain pended until the next time slot.

After time slot t, we assume that a portion $\theta N(t)$ of the running instances are finished (including instances that have completed their jobs and terminated instances with one-time requests), while the other $(1 - \theta)N(t)$ instances are still running and will be considered together along with the pended bids as bids at the next time slot. If terminated, running instances with persistent requests revert to the pending state; hence, the requests in the pending state come from three sources: i) failed new arrivals, ii) failed pending requests, and iii) terminated running instances. The number of requests in the next time slot, $L(t+1)$, can thus be written in terms of the number of requests $L(t)$ in the previous time slot, along with $\Lambda(t)$ new arrivals and $\theta N(t)$ exiting spot instances: $L(t + 1) = L(t) - \theta N(t) + \Lambda(t)$. Note that $(1-\theta)N(t)$ running instances are included in these $L(t+1)$ requests. All requests compete with one another via bid prices. After solving (1) with $L(t+1)$, the cloud provider will terminate the running spot instances whose bid prices are lower than the optimal solution $\pi^\star(t+1)$, the spot price of the next time slot.

We assume that the $\Lambda(t)$ are independent and identically distributed (i.i.d.), following a distribution f_Λ with expected value λ and variance σ [34]. Writing $N(t)$ in terms of the spot price $\pi^\star(t)$, i.e., $N(t) = L(t)\frac{\bar{\pi}-\pi^\star(t)}{\bar{\pi}-\underline{\pi}}$, the number of submitted bids in each time slot satisfies

$$L(t + 1) = \left(1 - \theta\frac{\bar{\pi} - \pi^\star(t)}{\bar{\pi} - \underline{\pi}}\right)L(t) + \Lambda(t). \quad (4)$$

Note that $0 \leq \theta \leq 1$ and $\underline{\pi} \leq \pi^{\star}(t) \leq \bar{\pi}$ ensure a positive value of $L(t+1)$. Depending on the value of $\Lambda(t)$, $L(t+1)$ may be larger or smaller than $L(t)$.

If too many user bids are continually re-submitted, the number of submitted bids $L(t)$ might diverge to infinity. To show that such a scenario does not occur, we first define the conditional Lyapunov drift as follows:

$$\Delta(t) \triangleq \frac{1}{2}L^2(t+1) - \frac{1}{2}L^2(t), \qquad (5)$$

or the change in the Lyapunov function $\frac{1}{2}L^2(t)$ over one time slot. Taking the conditional expectation, we have an upper bound on the Lyapunov drift (5):

PROPOSITION 1. *Suppose $\Lambda(t)$ follows a distribution with expected value λ and variance σ, and suppose that the spot prices $\pi^{\star}(t)$ are chosen according to (3). Then the conditional expectation of the Lyapunov drift is upper bounded: $\mathbb{E}\big(\Delta(t) \,|\, L(t)\big) \leq (\bar{\pi} - \underline{\pi})\lambda^2/(2\theta\bar{\pi}) + \sigma/2 - \epsilon L(t)$, where $\epsilon = \frac{\theta\lambda\bar{\pi}}{4(\bar{\pi}-\underline{\pi})}$.*

Proposition 1 implies that when (1) is used to compute the spot price, the queuing system is stable in the sense that the time-averaged queue size at any time t is uniformly bounded [23]. In fact, the number of requests can ultimately reach an equilibrium:

PROPOSITION 2. *The queue sizes of consecutive time slots are in equilibrium, i.e., $L(t+1) = L(t)$, if and only if the optimal spot prices $\pi^{\star}(t)$ satisfy*

$$\pi^{\star}(t) = h\big(\Lambda(t)\big) = \frac{1}{2}\left(\bar{\pi} - \frac{\beta}{1 + \frac{1}{\theta}\Lambda(t)}\right). \qquad (6)$$

We observe from (6) that $\pi^{\star}(t)$ is only in terms of the request arrivals at that time slot, so $\pi^{\star}(t)$, like $\Lambda(t)$, is i.i.d. at the equilibrium. Using this result, we can derive the probability density function (PDF) of the spot price in terms of f_{Λ}, the distribution (i.e., PDF) of $\Lambda(t)$.

PROPOSITION 3. *The probability density function of the spot price is given by:*

$$f_{\pi}(\pi) \simeq f_{\Lambda}(h^{-1}(\pi)), \qquad (7)$$

where $h^{-1}(\pi) = \theta\left(\frac{\beta}{\bar{\pi}-2\pi} - 1\right)$ is the inverse function of the function given in (6).

We can thus use the distribution of $\Lambda(t)$ at different time slots, f_{Λ}, to derive the distribution of the spot prices, f_{π}. Since we do not know the distribution of the bid arrivals, in the next section we instead test different distributions and compare their spot price predictions to the empirically observed spot prices.

4.3 Validation from Historical Spot Prices

We collected the spot price data for four instance types over August 14 – October 13, 2014, in the US Eastern region; we limit ourselves to a two-month dataset since Amazon only provides user access to spot price

history for the previous two months. The PDF of these prices is shown by the blue bars in Figure 3. We observe that they approximately follow a power-law or exponential pattern, indicating that the arrival process $\Lambda(t)$ is non-Poisson.[6] Furthermore, the shape of the PDFs across different instance types is consistent, though the spot prices are different, agreeing with [1]'s findings.

We next estimate the spot price PDF with (6) and (7) by assuming Pareto and exponential distributions for $\Lambda(t)$; other distributions could be used in a similar manner. We verify that the spot price distribution does not vary significantly over the day, e.g., due to more jobs submitted during the day, indicating that $\Lambda(t)$ can reasonably be assumed i.i.d. as in Section 4.2. The Kolmogorov-Smirnov test indicates that the distributions of the prices during the daytime and nighttime are similar with p-value > 0.01.

Pareto distribution. The PDF of a Pareto distribution is

$$f_{\Lambda}(\Lambda) = \frac{\alpha\Lambda_{min}^{\alpha}}{\Lambda^{\alpha+1}}, \quad \text{for } \Lambda \geq \Lambda_{min},$$

where $\Lambda_{min} = \theta\left(\frac{\beta}{\bar{\pi}-2\underline{\pi}} - 1\right)$ is derived from the monotonic relation between $\pi^{\star}(t)$ and $\Lambda(t)$ in (6).

Exponential distribution. The PDF of an exponential distribution is

$$f_{\Lambda}(\Lambda) = \frac{1}{\eta}e^{-\frac{1}{\eta}\Lambda}, \quad \text{for } \Lambda \geq 0.$$

We show the distribution fits in Figure 3; the η, α, β, and θ parameters were chosen to minimize the least-squares divergence between the estimated and empirical PDFs. We use the same β and θ values for each instance type since these parameters do not depend on the distribution of Λ. We observe that the two distributions fit the empirical data well, with a mean-squared error less than 10^{-6}. The small θ values indicate that few instances finish in each time slot due to the long running times of most jobs. For all instance types, the fitted Pareto and exponential distributions have finite mean and variance ($\alpha > 1$, $\eta > 0$). Thus, the conditions in Proposition 1 hold and the system is stable.

5. USER BIDDING STRATEGIES

We now derive users' bidding strategies for jobs running on a single instance, using the spot price PDF from Section 4 to predict future spot prices.[7] Though time series forecasting may be used instead, we note that users' job runtimes generally exceed one time slot, requiring predictions far in advance. Since the spot prices' autocorrelation drops off rapidly with a longer

---

[6]This observation is consistent with the findings in [18]. Thus, we can expect the cumulative distribution functions to have a knee, as was observed in [1].

[7]As noted in Section 1, the bidding strategies do not explicitly depend on the provider model in Section 4 but rather on the spot price's PDF.

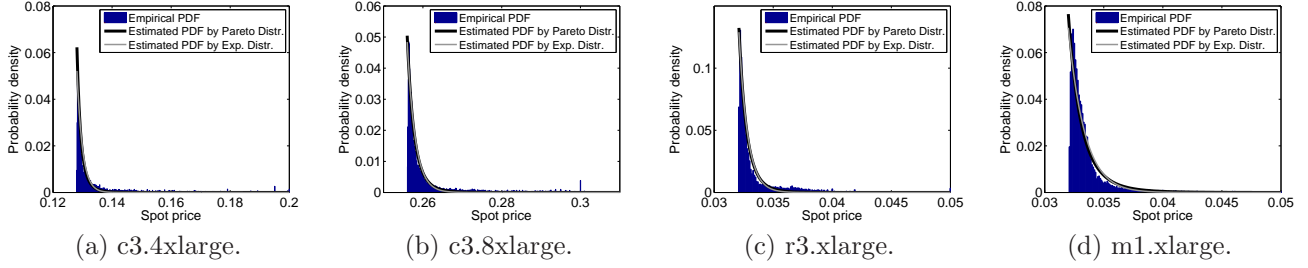(a) c3.4xlarge.　　　(b) c3.8xlarge.　　　(c) r3.xlarge.　　　(d) m1.xlarge.

Figure 3: Fitting the probability density function of Amazon spot prices in the US Eastern region by assuming Pareto and exponential distributions for $\Lambda(t)$. The fitted parameter values are $(\beta, \theta, \alpha, \eta) = (0.6, 0.02, 5, 0.00013)$, $(1.2, 0.02, 8, 7.1 \times 10^{-5})$, $(0.3, 0.02, 9.5, 0.000108)$, and $(0.3, 0.02, 5.2, 0.000204)$ for (a), (b), (c), and (d) respectively.

Table 1: Key terms and symbols.

Symbol	Definition
p	User bid price
π	Spot price
$\bar{\pi}$	On-demand price
$\underline{\pi}$	Minimum spot price
L	Demand for the spot instance
N	Number of launched spot instances
Λ	New request arrivals
t_k	Length of one time slot
T	Total job completion time
t_s	Job execution time (w/o interruptions)
t_r	Recovery time from an interruption
t_o	Overhead time to run multiple sub-jobs

lag time [1], such predictions are likely to be difficult. We discuss this point further in Section 8.

In this section, we first consider one-time requests (Section 5.1) and then persistent requests (Section 5.2) for a single instance; we consider jobs requiring multiple parallel instances in Section 6. We assume that users wish to minimize the cost of running their jobs. Users thus face a tradeoff between bidding lower prices and experiencing more interruptions, which lead to a longer runtime and potentially higher cost. We obtain the optimal bid prices by minimizing the user's cost subject to job interruptibility constraints (cf. Section 8).[8] As in Section 4, we consider a series of discrete time slots t and suppose the spot prices $\pi(t) = h(\Lambda(t))$, $t = 0, 1, \ldots$ are i.i.d. as in Proposition 2. We use p to denote the user's bid price and F_π to denote the cumulative distribution function of each spot price $\pi(t)$, corresponding to the PDF f_π in (7): $F_\pi(p)$ gives the probability that $p \geq \pi(t)$, i.e., the user's bid is accepted.

We summarize our model's notation in Table 1. Each job is characterized by its execution time t_s, or time required to complete without interruptions. We use T to denote the job's total completion time, i.e., the length of time between its submission and the time it finishes

---

[8]Including deadlines on the job completion times does not materially change our model, but we do not explicitly model this constraint. Users with hard job deadlines are more likely to use on-demand instances with guaranteed availability.

and exits the system. Since jobs with persistent requests may be interrupted, we suppose that persistent jobs are configured to save their data to a separate volume once interrupted and recover it upon resuming. Writing and transferring this data introduces a delay of t_r seconds per interruption. All prices are assumed to be in units of instance/hour, and all times given in units of hours.

5.1　One-time Requests

Since one-time requests are terminated as soon as the bid price falls below the spot price, we assume that users wish to minimize their expected job cost, subject to the constraint that jobs are not interrupted.

Job non-interruptibility. We first find the expected amount of time that a job will continue running without being interrupted:

$$t_k \sum_{i=1}^{\infty} i F_\pi(p)^{i-1} \big(1 - F_\pi(p)\big) = \frac{t_k}{1 - F_\pi(p)}. \quad (8)$$

Here the $F_\pi(p)$ terms represent the probability that $p \geq \pi(t)$, i.e., the request continues to run, and $1 - F_\pi(p)$ the probability that the job will be terminated. We thus find that $t_s \leq t_k / (1 - F_\pi(p))$: the expected amount of time that a job will keep running must exceed its execution time.

The optimal price. To find the expected cost of a job with a one-time request, we need to find the expected price that a user must pay to use an instance in each time slot. If a user's bid at price p is accepted, this expected price is simply the expected spot price, or the expected value of all possible spot prices that are less than or equal to p:

$$\mathbb{E}(\pi \mid \pi \leq p) = \frac{\int_{\underline{\pi}}^{p} x f_\pi(x) \mathrm{d}x}{\int_{\underline{\pi}}^{p} f_\pi(x) \mathrm{d}x} = \frac{\int_{\underline{\pi}}^{p} x f_\pi(x) \mathrm{d}x}{F_\pi(p)}, \quad (9)$$

which monotonically increases with p (cf. Proposition 4's proof in the Appendix). Thus, as we would intuitively expect, the user must pay more as his/her bid price increases. The user's expected cost for this job is then the expected spot price, multiplied by the number of time slots it takes for the job to complete. Hence, we have the following cost minimization problem with
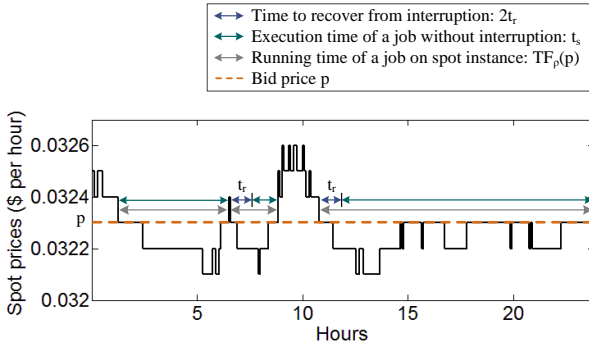
Figure 4: An example of job running times for the spot prices of an r3.xlarge-type instance in the US Eastern region on September 09, 2014.

a completion time constraint:

$$\begin{array}{ll} \underset{p}{\text{minimize}} & \Phi_{so}(p) = t_s \mathbb{E}(\pi \mid \pi \leq p) = \frac{t_s \int_{\underline{\pi}}^p x f_\pi(x) \mathrm{d}x}{F_\pi(p)} \\ \text{subject to} & \Phi_{so}(p) \leq t_s \bar{\pi}, \ t_s \leq \frac{t_k}{(1 - F_\pi(p))}, \ \underline{\pi} \leq p \leq \bar{\pi}. \end{array}$$
$$(10)$$

Since the expected spot price is monotonically increasing in the bid price, the user can minimize his or her expected cost by choosing the lowest possible bid price that satisfies the constraint that the job completes before being terminated:

PROPOSITION 4. *The optimal bid price for a one-time request is*

$$p^\star = \max\left\{\underline{\pi}, F_\pi^{-1}\left(1 - \frac{t_k}{t_s}\right)\right\}. \qquad (11)$$

We essentially wish to bid at the appropriate percentile of the spot price distribution F_π. As we would expect, the bid price increases as the number of time slots required to complete the job, t_s/t_k, increases: the job then needs to run for more consecutive time slots, which becomes more likely with a higher bid.

5.2 Persistent Requests

We now consider a job that places a persistent spot instance request. We begin by finding the total time that the job runs on the system, given a bid price, and then briefly discuss the implications of a job's interruptibility before deriving the user's optimal bid price.

Job running time. A job's total completion time T comprises two types of time slots: the running time, in which the job's bid price exceeds the spot price and the job actually runs on the instance, and the idle time. For a bid price p, the job's expected running time is $TF_\pi(p)$ (recall that $F_\pi(p)$ denotes the probability that the bid price $p \geq \pi(t)$, the spot price), and the idle time is then $T(1 - F_\pi(p))$. The running time can be further split into the execution time, t_s, and the additional running time required to recover from interruptions. We illustrate the job completion time in Figure 4, in which the bid price of $p = 0.0323$ is represented by an orange dashed line. The grey double-arrowed line represents the total

running time. Since the job is interrupted twice, the total recovery time is $2t_r$. Thus, for this example, we have $TF_\pi(0.0323) = 2t_r + t_s$.

To determine the total recovery time, we want to find the expected number of interruptions, i.e., times t in which the job runs on the system but was idle in the previous time slot ($p \geq \pi(t)$, $p < \pi(t-1)$). Note that the number of interruptions equals half of the total number of times the job's state changes between running and idle: each interruption requires both an idle-to-running and running-to-idle transition. We can thus find the expected total number of transitions and divide it by two. To do so, we define $\mathbb{I}_\pi\big(\pi(t)\big)$ as an indicator function: $\mathbb{I}_\pi\big(\pi(t)\big) = 1$ if $p \geq \pi(t)$; otherwise, $\mathbb{I}_\pi\big(\pi(t)\big) = 0$. We then consider $\big(\mathbb{I}_\pi\big(\pi(t)\big) - \mathbb{I}_\pi\big(\pi(t+1)\big)\big)^2$, which equals 1 if $\mathbb{I}_\pi\big(\pi(t)\big) \neq \mathbb{I}_\pi\big(\pi(t+1)\big)$ (i.e., a transition happens) and 0 otherwise. The number of idle-to-running transitions in T/t_k time slots is then $\frac{1}{2}\sum_{k=0}^{T/t_k - 1}\big(\mathbb{I}_\pi\big(\pi(t)\big) - \mathbb{I}_\pi\big(\pi(t+1)\big)\big)^2$. We take expectation to obtain

$$\begin{aligned} & \mathbb{E}\left(\frac{1}{2}\sum_{k=0}^{T/t_k - 1}\big(\mathbb{I}_\pi\big(\pi(t)\big) - \mathbb{I}_\pi\big(\pi(t+1)\big)\big)^2\right) \\ \overset{(a)}{=} & \frac{T}{t_k}\left(\mathbb{E}\big(\mathbb{I}_\pi\big(\pi(t)\big)\big) - \mathbb{E}\big(\mathbb{I}_\pi\big(\pi(t)\big)\mathbb{I}_\pi\big(\pi(t+1)\big)\big)\right) \\ = & \frac{T}{t_k}F_\pi(p)\big(1 - F_\pi(p)\big), \end{aligned}$$
$$(12)$$

where (a) is due to $\big(\mathbb{I}_\pi\big(\pi(t)\big)\big)^2 = \mathbb{I}_\pi\big(\pi(t)\big)$ since the value of $\mathbb{I}_\pi\big(\pi(t)\big)$ is either 1 or 0.

We now write the expected running time of the job as the sum of the recovery and execution times: $TF_\pi(p) = \left(\frac{T}{t_k}F_\pi(p)\big(1 - F_\pi(p)\big) - 1\right)t_r + t_s$. Simplifying the terms, the running time becomes

$$TF_\pi(p) = \frac{t_s - t_r}{1 - \frac{t_r}{t_k}\big(1 - F_\pi(p)\big)}, \qquad (13)$$

which decreases with p. As the bid price p increases, the job is less likely to be interrupted and will therefore have a shorter expected running time.

Job interruptibility. We can use the expected running time (13) to observe the effect of the recovery time parameter, t_r, on a job's feasibility for spot instances. Intuitively, spot instances are more effective for more "interruptible" jobs that can quickly recover from interruptions. In fact, a job's running time is finite only if the recovery time is sufficiently small:

$$t_r < \frac{t_k}{1 - F_\pi(p)}. \qquad (14)$$

The result in (14) can be obtained by requiring the denominator in (13) to be positive. We note that the upper bound to t_r is exactly the expected running time of a job without interruptions (cf. (8)): intuitively, the time to recover from an interruption should be smaller than the expected time on an instance between job in-

terruptions. We take (14) as a constraint on the bid price: if the job recovery time is high, the user should bid at a higher bid price in order to ensure that the job can complete. However, if the job recovery time is less than one time slot length, $t_r < \min_p \left\{ \frac{t_k}{1-F_\pi(p)} \right\} = t_k$, and a spot instance is feasible at any price.

The optimal bid price. We can now multiply the expected running time (13) with the expected spot price (9) to find that the cost of a job with a persistent request is $\Phi_{sp}(p) = T F_\pi(p) \mathbb{E}(\pi \mid \pi \leq p)$. The user's optimal bid price then solves the optimization problem

$$
\begin{aligned}
\underset{p}{\text{minimize}} \quad & \Phi_{sp}(p) = \frac{t_s - t_r}{1 - \frac{t_r}{t_k}\left(1 - F_\pi(p)\right)} \frac{\int_{\underline{\pi}}^{p} x f_\pi(x)\mathrm{d}x}{F_\pi(p)} \\
\text{subject to} \quad & \Phi_{sp}(p) \leq t_s \bar{\pi}, \ t_r < \frac{t_k}{1 - F_\pi(p)}, \ \underline{\pi} \leq p \leq \bar{\pi},
\end{aligned}
$$
(15)

where the first constraint ensures that the cost of running the spot instance is lower than the cost of running the job on an on-demand instance, and the second constraint ensures that the job is sufficiently interruptible. We use $p^\star$ to denote the optimal bid price to (15).

We now observe that the expected running time in (13) decreases with the bid price, while the expected spot price increases with the bid price. We find that the expected cost $\Phi_{sp}(p)$ first decreases and then increases with the bid price p, thus allowing us to solve for the optimal bid price $p^\star$:

PROPOSITION 5. *If the probability density function of the spot price monotonically decreases, i.e., $F_\pi(p)$ is concave, the optimal bid price solving (15) is*

$$
p^\star = \psi^{-1}\left(\frac{t_k}{t_r} - 1 \right),
$$
(16)

where $\psi^{-1}(\cdot)$ is the inverse function of

$$
\psi(p) = F_\pi(p) \left(\frac{\int_{\underline{\pi}}^{p} x f(x)\mathrm{d}x}{\int_{\underline{\pi}}^{p} (p - x) f(x)\mathrm{d}x} - 1 \right).
$$

We can observe from (16) that the optimal bid price does not depend on the execution time t_s: the cost is instead determined by the number of time slots needed for each recovery, t_r/t_k. While the execution time is fixed, a longer recovery time lengthens the total running time and thus the job cost.

6. BIDDING FOR MAPREDUCE JOBS

We now adapt Section 5's bidding strategies for single instances to parallelized MapReduce jobs, in which users bid for multiple instances at the same time. We first consider running only the slave nodes on spot instances and then consider running the full job (i.e., both master and slave nodes) on spot instances.[9] Though the

[9]While master nodes are often run on on-demand instances to guarantee that they will not be interrupted, our experiments in Section 7 show that, with sufficiently high bids, interruptions are rare even on spot instances.

job structure introduces an additional requirement on the master node's running time–the master node should keep running as long as the slave nodes are running–we show that this condition does not change the optimal bid for the master node as long as the job is sufficiently parallelized (i.e., multiple slave nodes run in parallel for a short amount of time).

6.1 Bidding for Slave Nodes Only

We first consider running only the slave nodes of a MapReduce job on several spot instances in parallel. We suppose that the job is split into M sub-jobs of equal size, each corresponding to one instance request. We now wish to calculate the optimal (i.e., cost-minimizing) bid prices for these sub-jobs; since we assume that all sub-jobs are bidding for the same type of spot instance, the bid price should be the same for all of them. We again find the total cost by multiplying the expected running time of the job by the expected spot price.

To find a job's expected running time, we denote each sub-job i's total time in the system as T_i. Since splitting a job results in additional overhead, e.g., due to message passing between the sub-jobs, we use t_o to represent a constant additional overhead time from splitting the job.[10] Then the total running time satisfies:

$$
\sum_{i=1}^{M} T_i F_\pi(p) = \sum_{i=1}^{M} \left(\frac{T_i F_\pi(p)\left(1 - F_\pi(p)\right)}{t_k} - 1 \right) t_r + t_s + t_o,
$$

i.e., it is the sum of the recovery, execution, and overhead times. Hence, we can extend the result for a single persistent bid in (13) as

$$
\sum_{i=1}^{M} T_i F_\pi(p) = \frac{t_s + t_o - M t_r}{1 - \frac{t_r}{t_k}\left(1 - F_\pi(p)\right)}.
$$
(17)

Since all of the sub-jobs run simultaneously, the overall time of the parallelized job execution is $\max_{i=1,\ldots,M} T_i F_\pi(p)$. All of the M sub-jobs are of equal size, so

$$
\max_{i=1,\ldots,M} T_i F_\pi(p) = \frac{t_s + t_o - M t_r}{M\left(1 - \frac{t_r}{t_k}\left(1 - F_\pi(p)\right)\right)}.
$$
(18)

Users can minimize this cost by choosing the optimal number of slave nodes, M. Although the job is split into smaller pieces, the overall running time $\max_{i=1,\ldots,M} T_i F_\pi(p)$ is larger than t_s/M, or the running time without interruptions or overhead. Thus, distributing the job across M instances shortens the completion time as compared to a single instance only if the overhead time is sufficiently small. Comparing the completion times in both cases, we find that using multiple instances shortens the completion time if $t_o < (M-1)t_k/(1 - F_\pi(p))$.

As in Section 5, the expected cost of running a job on M simultaneous instances is the sum of each instance's

[10]This overhead time may depend on M, the (fixed) number of sub-jobs.

expected running time, multiplied by the expected spot price: $\Phi_{mp} = \sum_{i=1}^{M} T_i F_\pi(p)\mathbb{E}(\pi \mid \pi \le p)$. The user then minimizes this cost by solving the following optimization problem:

$$\begin{aligned}
\underset{p}{\text{minimize}} \quad & \Phi_{mp}(p) = \frac{t_s + t_o - M t_r}{1 - \frac{t_r}{t_k}\left(1 - F_\pi(p)\right)} \frac{\int_{\underline{\pi}}^{p} x f_\pi(x)\mathrm{d}x}{F_\pi(p)} \\
\text{subject to} \quad & \Phi_{mp}(p) \le t_s\bar{\pi}, \ \underline{\pi} \le p \le \bar{\pi},
\end{aligned}$$
(19)

where the first constraint ensures that the cost is lower than that of running the job on an on-demand instance. Comparing (19) to bidding for a single persistent request in (15), we see that (19) can be solved similarly to (15) in Proposition 5.

By comparing the costs for multiple bids and for a single bid at the optimal bid prices, we find that when the overhead time is sufficiently small ($t_o < (M-1)t_r$), bidding for multiple spot instances can both lower the cost and shorten the job's overall running time. In contrast, running the job on an on-demand instance will reduce the running time but increase the cost.

6.2 Bidding for Master and Slave Nodes

Running a MapReduce job entirely on spot instances requires us to treat master and slave nodes separately; for instance, we might bid on different instance types for the master and slave nodes, since the slave nodes will likely have higher computing requirements. We thus develop two separate bidding strategies:

1) **Master node:** Since the master node has to be available at all times to manage slave node failures and to periodically check the status of tasks at the slave nodes, we do not allow any interruptions for the master node. We thus place a one-time request for a single spot instance, as in Section 5.1.

2) **Slave nodes:** MapReduce requires many slave nodes to process large jobs, but allows slave nodes to be interrupted. Thus, we place a persistent request for each slave node using the strategy in Section 6.1. The bid prices for the slave nodes must be determined jointly with that of the master node, since the master node's running time should exceed the slave nodes'. We note from Section 6.1 that if many simultaneous bids are submitted, the slave nodes' running time will decrease, shortening the required master node running time.

We can formally describe these strategies in the following optimization problem:

$$\begin{aligned}
\underset{p_v, p_m}{\text{minimize}} \quad & \Phi_{so}(p_m) + \Phi_{mp}(p_v) \\
\text{subject to} \quad & \frac{t_k}{1 - F_\pi^m(p_m)} \\
& \ge \frac{1}{F_\pi^v(p_v)}\left(\frac{t_s + t_o - M t_r}{1 - \frac{t_r}{t_k}\left(1 - F_\pi^v(p_v)\right)} - \frac{(M-1)t_k}{1 - F_\pi^v(p_v)}\right), \\
& \underline{\pi} \le p_v \le \bar{\pi}, \ \underline{\pi} \le p_m \le \bar{\pi},
\end{aligned}$$
(20)

Table 2: EC2 instance types. Sizes are given as (vCPU, memory in GiB, SSD storage in GB).

	m3	r3	c3
.xlarge	4, 15, 1x32	4, 30.5, 1x80	4, 7.5, 2x40
.2xlarge	8, 30, 2x80	8, 61, 1x160	8, 15, 2x80
.4xlarge	–	16, 122, 1x320	16, 30, 2x160
.8xlarge	–	–	32, 60, 2x320

where p_m and p_v denote the bid prices of the master node and slave nodes respectively, and $F_\pi^m(\cdot)$ and $F_\pi^v(\cdot)$ denote the spot prices' cumulative distribution functions for the master and slave node instance types. The first constraint in (20) ensures that the master node runs longer than any of the slave nodes (cf. (8) and (18)), where the righthand side of the inequality represents the worst-case completion time of the M parallel sub-jobs. We use $p_m^\star$ and $p_v^\star$ to denote the optimal bid prices for (20)'s optimization problem.

We can solve (20) by noting that, aside from the first constraint, p_m and p_v are independent variables. Thus, we can set their optimal values $p_m^\star$ and $p_v^\star$ respectively as the optimal bid prices for a one-time single instance request (Proposition 4) and for multiple persistent requests (as in (19)). The first constraint is satisfied if the user submits sufficiently many simultaneous requests for the slave nodes. In practice, this minimum number of nodes, which we denote as $M^\star$, can be as low as 3 or 4, as we show in Section 7's experiments.

7. EXPERIMENTAL RESULTS

In this section, we first examine the optimal bid prices derived in Section 5 on Amazon EC2 spot instances. By comparing the price charged per hour, total job completion time and final cost, we illustrate the tradeoff of using different bidding strategies. We then run a MapReduce example on spot instances to further highlight that our proposed bidding strategy in Section 6 is adaptable to parallelized MapReduce jobs and can substantially lower their costs. Table 2 summarizes the instance types that we use in our experiments. The `m3`, `r3`, and `c3` prefixes denote balanced, memory-optimized, and compute-optimized instances respectively. We test a variety of instance types and sizes and repeat each experiment ten times for each instance type; all performance graphs are shown as averages.[11] We do not include the runtime of the bid price calculations in our measurements since it is much smaller than one minute, and can thus be easily run within one five-minute time slot. For example, the bid price calculations for the `c3.4xlarge` instance with around 1MB of spot price history take 11.305 seconds for a one-time and 4.365 seconds for a persistent bid on a standard laptop.

---

[11]To ensure accuracy, we use our bills from Amazon to calculate the job costs. Since Amazon does not break the bills into individual jobs, we do not report the cost of each job.

7.1 Single-Instance Bids

Parameter setting: We consider a job that needs one hour (i.e., $t_s = 1h$) to be executed without interruption. The optimal bid prices of five Amazon spot instances (`r3.xlarge`, `r3.2xlarge`, `r3.4xlarge`, `c3.4xlarge` and `c3.8xlarge`) are listed in Table 3 for a one-time request and persistent requests with recovery times $t_r = 10s$ and $t_r = 30s$. We used the spot price history for the two months immediately prior to the experiments to calculate these prices.

Experiment setup: To simulate an exact one hour running time of a spot instance, we created an Amazon Machine Image (AMI), where a shell script was added to `/etc/rc.local` so that a one-hour count-down program will run once the instance is launched using this AMI. In addition, this program writes instance launched time as a sequence of items into Amazon DynamoDB, from which we can obtain the instance status (first run or restarted from interruption) and simulate a recovery time if the instance is interrupted. We then placed all spot requests using this AMI.

Results: We used the optimal bid prices for one-time requests (Table 3) to bid the associated spot instance at random times of the day. None of our experiments were interrupted, verifying that our bidding strategy for one-time requests can ensure reliability. Our bills show that this bidding strategy can reduce user costs by up to 91% compared to running the job on an on-demand instance. Figure 5 compares the cost of one-time requests on spot instances to that of on-demand instances for the instance types in Table 3. We also compare the actual costs (with actual spot price $\hat{\pi}$) to the expected cost from our analytical model; the analytical predictions closely match the experimental results.

We additionally compare against a heuristic called the best offline price in retrospect (shown as $\tilde{p}$ in Table 3 and the grey bars in Figure 5), in which we numerically search the spot price history of the last 10 hours for the minimal price that consistently exceeds the spot prices for one hour. This retrospective price is lower than the actual bid price in some cases, meaning the one-time spot instance would have been terminated if we had bid using this price: 10 hours of history is insufficient to predict the future prices.

We now use the results of the one-time requests for each instance type as a baseline to illustrate that users can further lower their cost in exchange for longer completion times by placing persistent requests. In Figure 6, we plot the percentage difference in performance between the persistent and one-time bids.

Figure 6(a) shows the percentage difference in price between the one-time and persistent requests for the five instance types. The negative values in this figure illustrate a lower optimal bid price for persistent requests: persistent requests can be interrupted. As we would expect from Section 5's analysis, longer recovery times ($t_r = 30s$ rather than 10s) yield higher bid prices, since
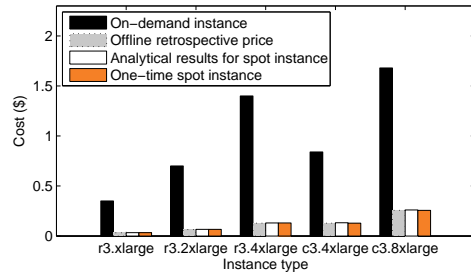

Figure 5: One-time spot instance requests substantially lower user cost compared to on-demand instances (on-demand and bid prices are as in Table 3).

a higher bid price ensures that the job has sufficient time to execute after recovering from interruptions.

Conversely, as shown in Figure 6(b), the completion time of a one-hour job with a persistent request is longer than that of a one-time request. The difference in these completion times includes both the recovery time after the instance is interrupted and the idle time during which the user's bid price is lower than the spot price. Interestingly, the job with longer recovery time t_r has a shorter completion time, though it is still longer than the completion time with a one-time request. We can observe from Table 3 that a longer recovery time leads to a higher optimal bid price, which can consequently increase the likelihood of satisfying the spot price and lead to less idle time.

As shown in Figures 6(a) and 6(b), the price charged per hour is slightly lower, but the completion time is longer for the persistent requests compared to the one-time requests. This makes the final cost underdetermined: Figure 6(c) demonstrates that persistent bids lead to lower overall costs. Smaller recovery times ($t_r = 10s$ versus 30s) have higher completion times but yield lower costs due to their lower bid prices (the higher completion time is mostly due to more idle time, for which the user is not charged, rather than extra time running on the instance). Thus, persistent requests can reduce user cost but lengthen the job completion time.

We also compare against simply bidding the 90th percentile spot price. Compared to our optimal prices, bidding the (larger) 90th percentile price yields a much smaller decrease in cost, despite slightly higher bid prices in Figure 6(a) and lower completion times in Figure 6(b). The jobs running on `r3.4xlarge` with recovery time $t_r = 30s$ instead have a longer completion time and lower bid price, but higher cost, with the 90th percentile spot price. This comparison is consistent with the fact that our bid prices are chosen to minimize users' costs.

7.2 MapReduce Jobs

We implemented our optimal bidding strategy to run Hadoop MapReduce jobs on the Common Crawl Dataset [7] using Amazon Elastic MapReduce (EMR). This dataset, which is hosted on Amazon's Simple Storage Service, maintains an open repository of web crawl data that is accessible to the public for free. In this experiment,

Table 3: Optimal bid prices for a single-instance job on spot instances.

Instance type	$\bar{\pi}$	$\underline{\pi}$	One-time bid				Persistent bid ($t_r = 10s$)		Persistent bid ($t_r = 30s$)	
			$p^\star$	$\mathbb{E}(\pi \mid \pi < p^\star)$	$\tilde{p}$	$\hat{\pi}$	$p^\star$	T	$p^\star$	T
r3.xlarge	$0.35	$0.0321	$0.0374	$0.0331	$0.0324	$0.033	$0.0332	1.4549h	$0.0355	1.1903h
r3.2xlarge	$0.70	$0.0646	$0.0795	$0.0669	$0.0644	$0.066	$0.0661	1.7638h	$0.0711	1.2558h
r3.4xlarge	$1.40	$0.1286	$0.1430	$0.1304	$0.1288	$0.130	$0.1327	1.2798h	$0.1422	1.0976h
c3.4xlarge	$0.84	$0.1281	$0.1669	$0.1324	$0.1283	$0.128	$0.1322	1.4917h	$0.1413	1.2180h
c3.8xlarge	$1.68	$0.2561	$0.2903	$0.2604	$0.2564	$0.256	$0.2648	1.2767h	$0.2831	1.1209h
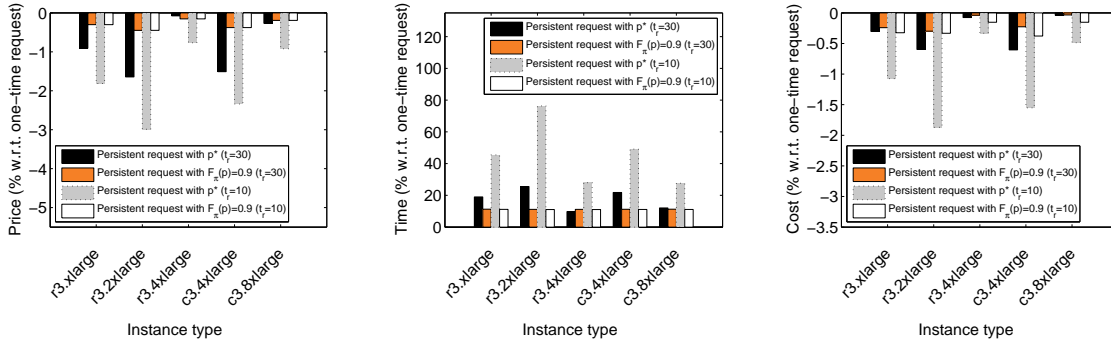
(a) Expected spot price. (b) Job completion time. (c) Total job cost.

Figure 6: Persistent requests yield lower costs but longer completion times compared to one-time requests. The bid prices for each instance type are given in Table 3.

Table 4: Optimal bid prices and actual costs for a MapReduce job.

Client Setting	Master node			Slave nodes			
	Instance type	$p_m^\star$	Actual cost	Instance type	$p_v^\star$	$M^\star$	Actual cost
CS1	c3.xlarge	$0.133	$0.10	m3.2xlarge	$0.070	5	$0.90
CS2	m3.xlarge	$0.101	$0.13	m3.2xlarge	$0.070	5	$1.03
CS3	m3.xlarge	$0.102	$0.13	r3.2xlarge	$0.071	3	$0.51
CS4	r3.xlarge	$0.042	$0.13	m3.2xlarge	$0.070	5	$0.58
CS5	r3.xlarge	$0.042	$0.13	r3.2xlarge	$0.071	3	$0.64

we ran the well-known "Common Crawl Word Count" example that counts the frequency of words appearing on the Common Crawl Corpus.

Parameter setting: We consider a word count job with recovery time $t_r = 30s$ and overhead time $t_o = 60s$. Using the bidding strategy proposed in Section 6, we list in Table 4 the optimal bid prices and the number of slave nodes for five client settings with different instance types for the master node and the slave nodes.

Experiment setup: We placed a one-time request for a single spot instance for the master node and persistent requests for multiple spot instances for the slave nodes. Since the master node just distributes the raw data and tracks the status of each task, it does not require a high-performance instance; we therefore bid on instances with better CPU performance for the slave nodes.

Results: We compare the completion time and cost of running MapReduce jobs on on-demand and spot instances for the five client settings. We use the optimal bid prices listed in Table 4 to place the spot instance

bids. Using the on-demand instance as a baseline, our bills from Amazon show that the bidding strategy for MapReduce jobs can reduce up to 92.6% of user cost with just a 14.9% increase of completion time.

In Figure 7, we show the completion time and the cost of the five client settings for on-demand and spot instances. Again, our experimental results closely approximate the analytical results from our model. As we would expect, the job completion time is longer on the spot instances than on the on-demand instances (Figure 7(a)), while the costs of the jobs running on the spot instances are much lower than those on the on-demand instances (Figure 7(b)).

In Table 4, we break down the actual costs of the master and slave nodes for each client setting of the MapReduce job. The cost of the master node is 10% to 25% of the slave node cost, indicating that users may wish to pay a higher spot price for the master node: this price does not significantly affect the overall cost, and helps ensure that the master node is not interrupted.
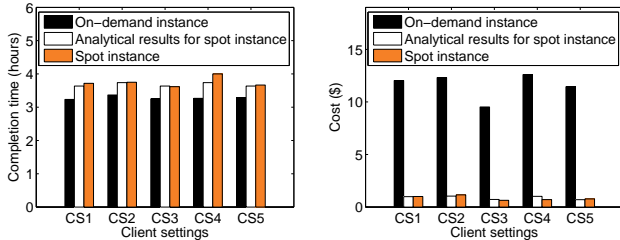
(a) Job completion time. (b) Job cost.

Figure 7: MapReduce jobs can save about 90% of user cost but have a 15% longer completion time on spot compared to on-demand instances.

8. DISCUSSION

Like all work based on analytical models, our work has some limitations. We discuss five important ones here and suggest ways to address them in future work.

Provider objectives. We have assumed that the cloud provider optimizes the sum of a concave utilization term and its revenue, where the concave shape of the utilization term is due to the cost of running large numbers of instances while maintaining users' quality-of-service. We could more explicitly account for user satisfaction by taking the social welfare to be the provider's objective function [25, 38]. While our current formulation matches well with the observed spot prices, including other factors in the provider's spot price optimization problem may shed more light on the provider's behavior. We emphasize, however, that our bidding strategies depend only on the spot prices, not on the provider's objectives or optimization problem.

Temporal correlations. We assume i.i.d. job arrivals that induce i.i.d. spot prices at the equilibrium. However, empirical studies have found temporal correlation in cloud workloads, possibly inducing positive temporal correlation in the spot prices [12]. In fact, a study of the spot prices in 2010 shows the presence of limited autocorrelation for consecutive time slots [1]. Incorporating these correlations into users' spot price predictions may improve their bidding strategies and outcomes: this correlation would likely reduce the degree to which the spot price changes in consecutive time slots. Thus, the user's job would be interrupted less often, leading to lower job running times and costs.

Risk-averseness. We suppose that users choose their bid prices so as to minimize their expected costs, subject to constraints on the expected runtimes. Risk-averse users may also wish to minimize the variance in costs and runtimes, so as to ensure that particularly bad outcomes do not occur. For example, we might choose the bid price so as to minimize the expected cost subject to an upper bound on the cost variance. Other users might allow their jobs' completion time to have a small probability of exceeding a job deadline. Instead of bounding the expected completion time, we could constrain the user's bid price so that the probability of exceeding this deadline is lower than a given small threshold.

Task dependence. Some tasks within a job cannot proceed before other tasks have been completed. While we do not account for such dependent tasks in our model, we can in practice bid on these tasks only after the tasks that they depend on have been completed. Thus, we will not bid on idle tasks that are waiting for other tasks to finish.

Collective user behavior. The bidding strategies that we have developed in Sections 5 and 6 assume that an individual user's bid price will not measurably affect the provider's spot price. While our experiments in Section 7 show that this assumption holds for a single user, it may not hold if multiple users begin to optimize their bidding strategies, which might affect the distribution of the submitted bids. To study this scenario, we can assume that users with a distribution of jobs optimize their bids and use Section 4's model to derive the effect on the provider's offered spot price.

9. CONCLUSION

Spot pricing opens up an auction-based market in which cloud providers can dynamically provision data center resources to meet user demand and users can develop bidding strategies that lower their cloud resource costs. In this work we first consider providers' setting of the spot prices, developing an optimization model for the provider and comparing its results to observed spot prices. We then answer the question of how users should bid for cloud resources. Since spot instances do not guarantee their availability, we consider the trade-off between bidding higher prices to avoid interruptions (for one-time requests) and bidding lower prices to save money (for persistent requests).

We adapt these bidding strategies to MapReduce jobs with master and slave nodes. Finally, we run our bidding client on Amazon EC2 to verify that our analytical results accurately approximate the real-time experimental results. Our bidding strategies can reduce users' costs by around 90% on a variety of instance types, with modest increases in the job completion times.

Spot pricing has been offered by Amazon for many years, yet the heavy-tailed spot price distributions that we observe suggest that few users bid for spot instances. We thus conjecture that many users are concerned by the possibility of too many job interruptions. Our paper suggests that these concerns might be misplaced: by tailoring their bids to their jobs' interruption constraints, users can save a substantial amount of money with spot instances without excessive job interruption.

APPENDIX

Proof of Proposition 1.

PROOF. Substituting (4) into (5), the Lyapunov drift can be expressed as $\Delta(t) = \frac{1}{2}\big((1 - \theta\frac{\bar{\pi} - \pi^{\star}(t)}{\bar{\pi} - \underline{\pi}})L(t) + \Lambda(t)\big)^2 - \frac{1}{2}L^2(t)$. We now bound the expectation of this quantity by:

$$
\begin{aligned}
&\mathbb{E}\big(\Delta(t)\,|\,L(t)\big)\\
&\overset{(a)}{\leq} \frac{1}{2}\Big(-\frac{\theta\bar{\pi}}{\bar{\pi}-\underline{\pi}} + \frac{1}{4}\big(\frac{\theta\bar{\pi}}{\bar{\pi}-\underline{\pi}}\big)^2\Big)L^2(t)\\
&\quad + \big(1 - \frac{1}{2}\frac{\theta\bar{\pi}}{\bar{\pi}-\underline{\pi}}\big)L(t)\mathbb{E}\big(\Lambda(t)\big) + \frac{1}{2}\mathbb{E}\big(\Lambda^2(t)\big)\\
&\overset{(b)}{=} \frac{1}{2}\Big(-\frac{\theta\bar{\pi}}{\bar{\pi}-\underline{\pi}} + \frac{1}{4}\big(\frac{\theta\bar{\pi}}{\bar{\pi}-\underline{\pi}}\big)^2\Big)L^2(t)\\
&\quad + \big(1 - \frac{1}{2}\frac{\theta\bar{\pi}}{\bar{\pi}-\underline{\pi}}\big)\lambda L(t) + \frac{1}{2}(\sigma + \lambda^2)\\
&\leq \big(1 - \frac{1}{4}\frac{\theta\bar{\pi}}{\bar{\pi}-\underline{\pi}}\big)\max_{L(t)}\Big\{-\frac{1}{2}\frac{\theta\bar{\pi}}{\bar{\pi}-\underline{\pi}}L^2(t) + \lambda L(t)\Big\}\\
&\quad - \frac{1}{4}\frac{\theta\lambda\bar{\pi}}{\bar{\pi}-\underline{\pi}}L(t) + \frac{1}{2}(\sigma + \lambda^2)\\
&\overset{(c)}{=} \frac{\bar{\pi}-\underline{\pi}}{2\theta\bar{\pi}}\lambda^2 + \frac{1}{2}\sigma - \frac{1}{4}\frac{\theta\lambda\bar{\pi}}{\bar{\pi}-\underline{\pi}}L(t),
\end{aligned}
$$

where (a) is due to $\pi(t) \leq \frac{1}{2}\bar{\pi}$, (b) is due to $\mathbb{E}(\Lambda^2(t)) = \text{Var}(\Lambda(t)) + \big(\mathbb{E}(\Lambda(t))\big)^2$ and the equality in (c) holds when $L(t) = \frac{\bar{\pi}-\underline{\pi}}{\theta\bar{\pi}}\lambda$. □

Proof of Proposition 2.

PROOF. We first prove that (6) is a necessary condition for $L(t+1) = L(t)$. We rewrite (4) as

$$
L(t) = \frac{\bar{\pi} - \underline{\pi}}{\theta(\bar{\pi} - \pi^{\star}(t))}\Lambda(t). \tag{21}
$$

Solving the system of equations (2) and (21) yields (6).

We then prove that (6) is a sufficient condition for $L(t+1) = L(t)$. From (4), we can derive

$$
\begin{aligned}
&L(t+1) - L(t)\\
&\overset{(a)}{=} -\theta\Big(\frac{\beta}{\bar{\pi} - 2\pi^{\star}(t)} - 1\Big) + \Lambda(t)\\
&\overset{(b)}{=} \theta\Big(\frac{\beta}{\bar{\pi} - \big(\bar{\pi} - \frac{\beta}{1+\frac{1}{\theta}\Lambda(t)}\big)} - 1\Big) + \Lambda(t) = 0,
\end{aligned}
$$

where (a) and (b) are obtained by respectively substituting (2) and (6). Thus, $L(t+1) = L(t)$. □

Proof of Proposition 4.

PROOF. By taking the first-order derivative of $\Phi_{so}(p)$ in the objective of (10), we have

$$
\partial\Phi_{so}(p)/\partial p = \frac{t_s}{(F_{\pi}(p))^2}f_{\pi}(p)\Big(-\int_{\underline{\pi}}^{p}xf_{\pi}(x)\mathrm{d}x + pF_{\pi}(p)\Big).
$$

By letting $g(p) = -\int_{\underline{\pi}}^{p}xf_{\pi}(x)\mathrm{d}x + pF_{\pi}(p)$, we have $\partial g(p)/\partial p = F_{\pi}(p) > 0$. So $g(p)$ increases with p. Combining the fact that $g(\underline{\pi}) = -\underline{\pi}f_{\pi}(\underline{\pi}) + \underline{\pi}F_{\pi}(\underline{\pi}) = 0$, the nonnegativity and monotonic increasing of $g(p)$ lead to $\partial\Phi_{so}(p)/\partial p > 0$. Therefore, $\Phi_{so}(p)$ also increases with p. Minimizing $\Phi_{so}(p)$ is equivalent to finding the minimum p in its feasible set. From $t_k/\big(1 - F_{\pi}(p)\big)$, we have $p \geq F_{\pi}^{-1}\big(1 - \frac{t_k}{t_s}\big)$ due to the monotonic property of $F_{\pi}(p)$. Thus, the minimum of the feasible set is the larger one of $\underline{\pi}$ and $F_{\pi}^{-1}\big(1 - \frac{t_k}{t_s}\big)$. □

Proof of Proposition 5.

PROOF. By taking the first-order derivative of $\Phi(p)$ in (15), we have

$$
\partial\Phi_{sp}(p)/\partial p = \frac{(t_s - t_r)f_{\pi}(p)\big((1-\frac{t_r}{t_k}) + 2\frac{t_r}{t_k}F_{\pi}(p)\big)}{\big((1-\frac{t_r}{t_k})F_{\pi}(p) + \frac{t_r}{t_k}\big(F_{\pi}(p)\big)^2\big)^2}g(p),
$$

where

$$
g(p) = -\int_{\underline{\pi}}^{p}xf_{\pi}(x)\mathrm{d}x + p\frac{(1-\frac{t_r}{t_k})F_{\pi}(p) + \frac{t_r}{t_k}\big(F_{\pi}(p)\big)^2}{(1-\frac{t_r}{t_k}) + 2\frac{t_r}{t_k}F_{\pi}(p)}.
$$

Note that first three terms before $g(p)$ in $\partial\Phi_{sp}(p)/\partial p$ are positive. To show the positivity of $\partial\Phi_{sp}(p)/\partial p$, we take first-order derivative of $g(p)$ and then have $\partial g(p)/\partial p = \frac{(1-\frac{t_r}{t_k})F_{\pi}(p) + \frac{t_r}{t_k}\big(F_{\pi}(p)\big)^2}{\big((1-\frac{t_r}{t_k}) + 2\frac{t_r}{t_k}F_{\pi}(p)\big)^2}\Big((1-\frac{t_r}{t_k}) + 2\frac{t_r}{t_k}\big(F_{\pi}(p) - pf_{\pi}(p)\big)\Big)$.

Due to the positivity and monotonic decreasing property of $f_{\pi}(p)$ (cf. Figure 3), $F_{\pi}(p)$ is concave and $F_{\pi}(p) - pf_{\pi}(p) \geq 0$. We then have $\partial g(p)/\partial p \geq 0$. Thus, $g(p)$ monotonically increases with p, so is $\partial\Phi_{sp}(p)/\partial p$. By the fact that $g(\underline{\pi}) < 0$ and $g(\bar{\pi}) > 0$, $\partial\Phi_{sp}(p)/\partial p$ increases monotonically from a negative value to a positive value, i.e., $\Phi_{sp}(p)$ first decreases and then increases with p. Hence, $\Phi_{sp}(p)$ is minimized when $\partial\Phi_{sp}(p)/\partial p = 0$, i.e., $g(p) = 0$. Letting $g(p) = 0$, we thus deduce

$$
\begin{aligned}
&\int_{\underline{\pi}}^{p}xf_{\pi}(x)\mathrm{d}x = p\frac{(1-\frac{t_r}{t_k})F_{\pi}(p) + \frac{t_r}{t_k}\big(F_{\pi}(p)\big)^2}{(1-\frac{t_r}{t_k}) + 2\frac{t_r}{t_k}F_{\pi}(p)}\\
&\Rightarrow\quad \psi(p) = F_{\pi}(p)\Big(\frac{\int_{\underline{\pi}}^{p}xf(x)\mathrm{d}x}{\int_{\underline{\pi}}^{p}(p-x)f(x)\mathrm{d}x} - 1\Big) = \frac{t_k}{t_r} - 1,
\end{aligned}
$$

which leads to (16). In addition, $\Phi(p^{\star}) < \Phi(\bar{\pi}) = (t_s - t_r)\mathbb{E}(\pi\,|\,\pi \leq \bar{\pi}) \leq t_s\bar{\pi}$ so the constraints in (15) are satisfied at optimality. □

10. REFERENCES

[1] AGMON BEN-YEHUDA, O., BEN-YEHUDA, M., SCHUSTER, A., AND TSAFRIR, D. Deconstructing Amazon EC2 spot instance pricing. *ACM Trans. on Economics and Computation 1*, 3 (2013), 1–16.

[2] AMAZON. EC2 spot instance, 2015. http://aws.amazon.com/ec2/purchasing-options/spot-instances/.

[3] ARDAGNA, D., PANICUCCI, B., AND PASSACANTANDO, M. A game theoretic formulation of the service provisioning problem in cloud systems. In *Proc. of WWW* (2011), ACM, pp. 177–186.

[4] ARMBRUST, M., FOX, A., GRIFFITH, R., JOSEPH, A. D., KATZ, R., KONWINSKI, A., LEE, G., PATTERSON, D., RABKIN, A., STOICA, I., ET AL. A view of cloud computing. *Communications of the ACM 53*, 4 (2010), 50–58.

[5] BLUM, A., SANDHOLM, T., AND ZINKEVICH, M. Online algorithms for market clearing. *Journal of the ACM (JACM) 53*, 5 (2006), 845–879.

[6] BUYYA, R., YEO, C. S., AND VENUGOPAL, S. Market-oriented cloud computing: Vision, hype, and reality for delivering IT services as computing utilities. In *Proc. of IEEE HPCC* (2008), IEEE, pp. 5–13.

[7] COMMON CRAWL. Common Crawl Corpus. *online at: http://commoncrawl.org/* (2015).

[8] DOGAR, F. R., KARAGIANNIS, T., BALLANI, H., AND ROWSTRON, A. Decentralized task-aware scheduling for data center networks. In *Proc. of ACM SIGCOMM* (2014).

[9] FANG, X., MISRA, S., XUE, G., AND YANG, D. Smart grid – The new and improved power grid: A survey. *IEEE Communications Surveys & Tutorials 14*, 4 (2012), 944–980.

[10] FENG, G., GARG, S., BUYYA, R., AND LI, W. Revenue maximization using adaptive resource provisioning in cloud computing environments. In *Proc. of ACM/IEEE Grid* (2012), pp. 192–200.

[11] FRIEDMAN, E. J., AND PARKES, D. C. Pricing WiFi at Starbucks: Issues in online mechanism design. In *Proc. of ACM EC* (2003), ACM, pp. 240–241.

[12] GUENTER, B., JAIN, N., AND WILLIAMS, C. Managing cost, performance, and reliability tradeoffs for energy-aware server provisioning. In *Proc. of IEEE INFOCOM* (2011), IEEE, pp. 1332–1340.

[13] HA, S., SEN, S., JOE-WONG, C., IM, Y., AND CHIANG, M. TUBE: time-dependent pricing for mobile data. In *Proc. of ACM SIGCOMM* (2012).

[14] JAIN, N., MENACHE, I., NAOR, J. S., AND YANIV, J. Near-optimal scheduling mechanisms for deadline-sensitive jobs in large computing clusters. *ACM Trans. on Parallel Computing 2*, 1 (2015), 3.

[15] JAVADI, B., THULASIRAM, R. K., AND BUYYA, R. Statistical modeling of spot instance prices in public cloud environments. In *Proc. of IEEE/ACM UCC* (2011), IEEE/ACM, pp. 219–228.

[16] JIA, J., ZHANG, Q., ZHANG, Q., AND LIU, M. Revenue generation for truthful spectrum auction in dynamic spectrum access. In *Proc. of ACM MobiHoc* (2009), pp. 3–12.

[17] JIN, H., WANG, X., WU, S., DI, S., AND SHI, X. Towards optimized fine-grained pricing of IaaS cloud platform. *IEEE Trans. on Cloud Computing* (2014), to appear.

[18] JUAN, D.-C., LI, L., PENG, H.-K., MARCULESCU, D., AND FALOUTSOS, C. Beyond poisson: Modeling inter-arrival time of requests in a datacenter. In *Advances in Knowledge Discovery and Data Mining*. Springer, 2014, pp. 198–209.

[19] KANG, L., AND PARKES, D. C. A decentralized auction framework to promote efficient resource allocation in open computational grids. In *Proc. Joint Workshop on The Economics of Networked Systems and Incentive-Based Computing* (2007).

[20] KUTANOGLU, E., AND WU, S. D. On combinatorial auction and Lagrangean relaxation for distributed resource scheduling. *IIE Trans. 31*, 9 (1999), 813–826.

[21] LAMPE, U., HANS, R., SELIGER, M., AND PAULY, M. Pricing in infrastructure clouds–an analytical and empirical examination. In *Association for Information Systems Conference* (2014).

[22] LEE, G., CHUN, B.-G., AND KATZ, R. H. Heterogeneity-aware resource allocation and scheduling in the cloud. *Proc. of HotCloud* (2011), 1–5.

[23] LEONARDI, E., MELLIA, M., NERI, F., AND AJMONE MARSAN, M. Bounds on average delays and queue size averages and variances in input-queued cell-based switches. In *Proc. of IEEE INFOCOM* (2001), vol. 2, IEEE, pp. 1095–1103.

[24] MACKIE-MASON, J., AND VARIAN, H. Pricing the Internet. In *Public Access to the Internet*, B. Kahin and J. Keller, Eds. Prentice-Hall, Englewood Cliffs, NJ, 1995.

[25] MENACHE, I., OZDAGLAR, A., AND SHIMKIN, N. Socially optimal pricing of cloud computing resources. In *Proc. of VALUETOOLS* (2011), ICST, pp. 322–331.

[26] NAIR, J., SUBRAMANIAN, V. G., AND WIERMAN, A. On competitive provisioning of cloud services. *ACM SIGMETRICS Performance Evaluation Review 42*, 2 (2014), 30–32.

[27] NG, C., PARKES, D. C., AND SELTZER, M. Virtual worlds: Fast and strategyproof auctions for dynamic resource allocation. In *Proc. of ACM EC* (2003), ACM, pp. 238–239.

[28] POOLA, D., RAMAMOHANARAO, K., AND BUYYA, R. Fault-tolerant workflow scheduling using spot instances on clouds. *Procedia Computer Science 29* (2014), 523–533.

[29] PORTER, R. Mechanism design for online real-time scheduling. In *Proc. of ACM EC* (2004), ACM, pp. 61–70.

[30] SEN, S., JIN, Y., GUÉRIN, R., AND HOSANAGAR, K. Modeling the dynamics of network technology adoption and the role of converters. *IEEE/ACM Trans. on Networking 18*, 6 (2010), 1793–1805.

[31] TENG, F., AND MAGOULÈS, F. A new game theoretical resource allocation algorithm for cloud computing. In *Advances in Grid and Pervasive Computing*. Springer, 2010, pp. 321–330.

[32] UPADHYAYA, P., BALAZINSKA, M., AND SUCIU, D. How to price shared optimizations in the cloud. *Proc. of the VLDB Endowment 5*, 6 (2012), 562–573.

[33] VAN DEN BOSSCHE, R., VANMECHELEN, K., AND BROECKHOVE, J. Cost-optimal scheduling in hybrid IaaS clouds for deadline constrained workloads. In *Proc. of IEEE CLOUD* (2010), IEEE, pp. 228–235.

[34] WANG, P., QI, Y., HUI, D., RAO, L., AND LIU, X. Present or future: Optimal pricing for spot instances. In *Proc. of IEEE ICDCS* (2013).

[35] WANG, Q., REN, K., AND MENG, X. When cloud meets eBay: Towards effective pricing for cloud computing. In *Proc. of IEEE INFOCOM* (2012), IEEE, pp. 936–944.

[36] WELLMAN, M. P., MACKIE-MASON, J. K., REEVES, D. M., AND SWAMINATHAN, S. Exploring bidding strategies for market-based scheduling. In *Proc. of ACM EC* (2003), ACM, pp. 115–124.

[37] YI, S., ANDRZEJAK, A., AND KONDO, D. Monetary cost-aware checkpointing and migration on Amazon cloud spot instances. *IEEE Trans. on Services Computing 5*, 4 (2012), 512–524.

[38] ZHANG, L., LI, Z., AND WU, C. Dynamic resource provisioning in cloud computing: A randomized auction approach. In *Proc. of IEEE INFOCOM* (2014).

[39] ZHANG, Q., ZHU, Q., AND BOUTABA, R. Dynamic resource allocation for spot markets in cloud computing environments. In *Proc. of IEEE/ACM UCC* (2011), IEEE/ACM, pp. 178–185.

[40] ZHOU, Y., AND WENTZLAFF, D. The sharing architecture: sub-core configurability for IaaS clouds. In *Proc. of ACM ASPLOS* (2014), ACM, pp. 559–574.