How to Bid the Cloud

Paper #114, 14 pages

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 this 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. We answer two basic questions faced by providers and users: how will the provider set the spot prices, and what prices should users bid? Computing users' bid strategies is particularly challenging: higher bid prices lessen the chance of interruptions, reducing the time overhead for recovering from interruptions, but can increase user costs. We address these questions and challenges in three steps: (1) modeling the cloud provider's behavior to infer the spot price and matching the model to historically offered prices, (2) deriving optimal bid strategies for different job requirements and interruption overheads, and (3) adapting this strategy 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, finding that spot pricing reduces user costs by 90% with modest increases in completion time compared to on-demand instances.

1. INTRODUCTION

With the recent growth in demand for cloud resources, today's cloud providers face an increasingly complicated problem of allocating resources to different users. These resource allocations must take into account both available capacity within datacenter networks as well as the specific job requirements, which are often specified in binding service level agreements negotiated with the user. 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 [10, 14, 23, 33], most have taken user demands as a given input and focused on operational concerns. Yet price setting gives cloud providers a measure of control over user demand [8]. However, 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 it does not even out short-term fluctuations [16]. To manage these fluctuations and the consequent excess demand for available datacenter capacity, cloud providers could introduce more flexible pricing plans in which resources are priced according to real-time market demand [1, 38]. Amazon's Elastic Compute Cloud (EC2) spot pricing [2] employs this strategy.

Spot pricing creates an auction for available resources in which users submit bids for resource instances. The cloud provider then sets a spot price at different times, depending on the number of bids received from users (i.e., demand) and how many idle resources are available at that time (i.e., supply). User bids above the spot price are accepted, but if a user's bid falls below the prevailing spot price, the user's launched instance is terminated until the bid again exceeds the spot price. User jobs can thus face interruptions in the course of completion, which may affect user utility. Two natural questions then arise: first, *how will the provider set the spot price*, and second, *what prices should users bid*?

We provide the answers to these questions in this work. Unlike most works on spot pricing, which consider only the provider's viewpoint [12, 19, 34, 37], we aim to both *accurately infer how the provider sets the spot prices* and then to *develop bidding strategies for the user*. Importantly, 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 in order to develop bid strategies for the user.

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 [6, 17]. However, explaining the spot prices with a model that relates the spot prices to users' submitted bids allows us to gain further insights into the effects of different factors

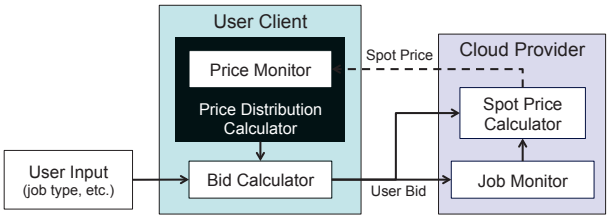


Figure 1: Client and cloud provider interaction.

like fully utilizing the available capacity and the performance that users receive from their bids.

Users bidding for spot resources face a tradeoff: bidding lower prices allows them to save money, but can also lengthen job runtimes by increasing the likelihood of interruptions. Even "interruptible" jobs can face extra delays when recovering from interruptions, e.g., from retrieving saved data and restarting the job. These delays can increase the job running time and thus its cost. Complicating this tradeoff is the fact that user jobs can have long runtimes spanning many changes in the spot price [17]. Users then face two key challenges:

1) Users must predict spot prices in order to optimize their bids for a given job, not only in the next time slot but also for all future timeslots until the job is completed. We leverage our estimation of the providers' behavior for this long-term prediction.

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 our client and its interaction with the cloud provider. The client calculates the bid price based on two types of inputs: user inputs on job characteristics, and the historical distribution of the prices offered by the cloud provider. A price monitor ensures that the price distribution is kept 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 relevant background information on Amazon's EC2 offerings and pricing in Section 2 before developing the following results:

A framework for setting the spot prices (Section 3): We develop a model to understand how the providers set spot prices, using it to bound users' quality of service and testing it against empirical data.

Users' optimal bid strategies (Section 4): Given a predicted spot price distribution, we derive optimal bid strategies for different degrees of job interruptibility.

Adaptation to MapReduce jobs (Section 5): We adapt our bid strategies for the master and slave nodes of MapReduce jobs and implement our bidding strategy on a MapReduce job.

Experiment on Amazon EC2 (Section 6): We run our client on a variety of EC2 instances and job types, finding that it substantially lowers users' costs in exchange for modestly higher running times.

Our work thus considers spot pricing from the point of view of both the cloud provider and user, leveraging an understanding of how providers set the spot prices to develop bidding strategies customized to different user requirements. We discuss some limitations of our work in Section 7 and related works in Section 8 before concluding the paper in Section 9. All proofs can be found in the Appendix.

2. BACKGROUND

2.1 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 [3]. 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 hardware and resource capacities; users generally submit 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 failure of a slave node occurs. After all the tasks are finished on the slave nodes, the master node returns a result.

2.2 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 bid exceeds the spot price. Amazon generally updates the spot price every five minutes and encourages users to run interruptible jobs on spot instances.[1]

Spot instances allow two types of bids: one-time and persistent. One-time bids are submitted once and then

[1]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.

exit the system once they fall below the current spot price. Thus, submitting a one-time bid takes 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.

3. CLOUD PROVIDER MODEL

In this section, we develop an explanatory model for cloud providers' offered spot prices. Though the model's primary use for users is in predicting the future spot price distribution for use in Sections 4 and 5, our model additionally yields interesting insights into the provider's behavior. We first formulate a model for choosing the revenue-maximizing prices in Section 3.1 and then consider its effects on pending bids in Section 3.2. In particular, since users' persistent requests are continually re-considered if not satisfied, the providers might experience unsustainably large numbers of pending bids. We show that the number of pending bids remains bounded under reasonable conditions. We validate our model in Section 3.3 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 spot instances 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.

3.1 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.[2] Since some studies have suggested that Amazon does not use revenue-maximizing spot prices [6], we also include an optional capacity utilization term $\beta \log(1 + N(t))$, 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.

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, $\beta \log\big(1 + N(t)\big) + \pi(t)N(t)$, subject to the constraint that $\pi(t)$ lie between the minimum and maximum prices ($\underline{\pi} \leq \pi(t) \leq \bar{\pi}$). We denote this optimal price by $\pi^\star(t)$. In practice, we would generally emphasize the revenue rather than utilization term by choosing a small scaling factor β. The provider can keep the number of accepted bids below its available capacity by adjusting the minimum spot price $\underline{\pi}$.

We now formulate the provider's optimization problem by using the probability distribution f_p to denote the distribution of bids received by the user. 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 then $N(t) = L(t)\frac{\bar{\pi} - \pi(t)}{\bar{\pi} - \underline{\pi}}$, or the fraction of submitted bids that exceed the spot price. The provider then maximizes the sum of its revenue and utilization term:

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

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, 31].

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

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

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

$$\pi^\star(t) = \max\left\{\underline{\pi},\ \frac{3}{4}\bar{\pi} + \frac{1}{2}(\bar{\pi} - \underline{\pi})\frac{1}{L(t)} - \frac{1}{4}\sqrt{\left(\bar{\pi} + 2\big(\bar{\pi} - \underline{\pi}\big)\frac{1}{L(t)}\right)^2 + 8\beta\big(\bar{\pi} - \underline{\pi}\big)\frac{1}{L(t)}}\right\}. \tag{3}$$

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 $\beta \leq (L(t) + 1)(\bar{\pi} - 2\underline{\pi})$ 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.

3.2 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

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

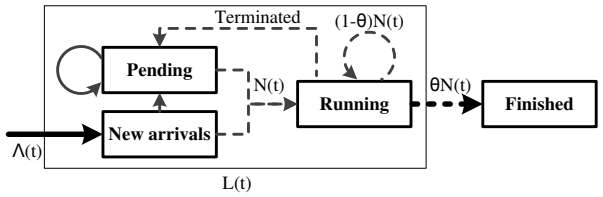


Figure 2: State transitions of spot instances. The solid and dashed arrow lines respectively represent the transitions at the start of the current and next time slots.

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: $L(t+1) = L(t) - \theta N(t) + \Lambda(t)$.

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]. Note that $N(t)$ can be written in terms of the spot price $\pi^\star(t)$: $N(t) = L(t)\frac{\bar{\pi}-\pi^\star(t)}{\bar{\pi}-\underline{\pi}}$, so the number of submitted bids in each time slot satisfies

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

Note that $0 \le \theta \le 1$ and $\underline{\pi} \le \pi^\star(t) \le \bar{\pi}$ ensure a positive value of $L(t+1)$. Now, 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 $L(t)$ can become large, possibly diverging to infinity over long timescales. To show that such a scenario does not occur, we 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 upper-bound the Lyapunov drift (5):

PROPOSITION 1 (STABILITY). *Suppose $\Lambda(t)$ follows some distribution whose expected value is λ and variance is 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: $\mathbb{E}\big(\Delta(t) \,|\, L(t)\big) \le (\bar{\pi}-\underline{\pi})\lambda^2/(2\theta\bar{\pi}) + \sigma/2 - \epsilon L(t)$, 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 [24]. 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. In the next section, we suppose that the system has reached this equilibrium and show that the spot prices offered by Amazon correspond to a Pareto distribution of the arrival rates $\Lambda(t)$.

3.3 Characterizing the Spot Price

Leveraging the results in Section 3.2 (i.e., Proposition 2), 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).

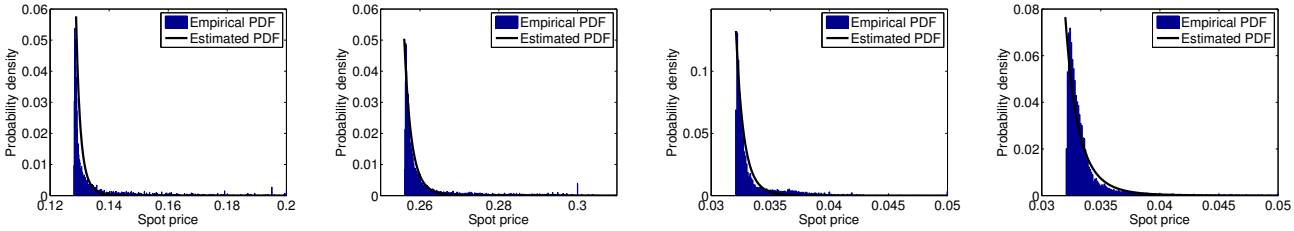
We can thus use the distribution of the $\Lambda(t)$ at different time slots, f_Λ, to derive the distribution of the spot prices, f_π. Since we do not know the distribution of the bid arrivals, we instead test different distributions and compare their spot price predictions to the empirically observed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

We collected the spot price data for four instance types from the 14th of August to the 13th of October in 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 follows a power-law or exponential pattern, indicating that the arrival process $\Lambda(t)$ is non-Poisson.[3] We thus use a Pareto distribution for $\Lambda(t)$ to estimate the PDF of the spot prices with (6) and (7).

The PDF of a Pareto distribution is

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

[3]This observation is consistent with the heavy-tailed distributions that many real-world processes follow [5].



(a) c3.4xlarge ($\beta = 0.6$, $\theta = 0.02$, $\alpha = 5$) (b) c3.8xlarge ($\beta = 1.2$, $\theta = 0.02$, $\alpha = 8$) (c) r3.xlarge ($\beta = 0.3$, $\theta = 0.02$, $\alpha = 9.5$) (d) m1.xlarge ($\beta = 0.3$, $\theta = 0.02$, $\alpha = 5.2$)

Figure 3: Fitting the probability density function of Amazon spot prices in the US Eastern region by assuming a Pareto distribution for $\Lambda(t)$. We fit our model to the prices for four different instance types.

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

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). In Figure 3, we use this distribution to fit the empirical PDF of the spot prices. The α, β, and θ parameters are chosen to minimize the least-squares divergence between the estimated and empirical PDFs. Figure 3 shows that the estimated probability density function using a Pareto distribution fits the empirical data well. We note that $\alpha > 1$ for all instance types, indicating that the distribution has a finite mean and variance. Thus, Proposition 1 holds and the system is stable. The non-negligible estimates for β indicate that Amazon's spot prices are set to increase both revenue and spare capacity utilization.

4. USER BIDDING STRATEGY

We now derive users' bid strategies for jobs running on a single instance, using the spot price PDF from Section 3 to predict future spot prices. 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 lag time [6], such predictions are likely to be difficult. We discuss this further in Section 7.

In this section, we first consider one-time requests (Section 4.1) and then persistent requests (Section 4.2). In general, we assume that users wish to minimize the cost of running their jobs.[4] Users thus face a trade-off between bidding lower prices and experiencing more interruptions, which lead to increased job runtime and can increase their cost. As in Section 3, we consider a series of discrete time slots t and suppose the spot prices $\pi(t) = h\big(\Lambda(t)\big)$, $t = 0, 1, \ldots$ 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 $p > \pi(t)$, i.e., that the user's bid is accepted.

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

4.1 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 the job will complete before being terminated. To find this expected cost, we first 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:

$$\mathbb{E}(\pi \,|\, \pi \leq p) = \frac{\int_{\underline{\pi}}^{p} x f_\pi(x)\mathrm{d}x}{\int_{\underline{\pi}}^{p} f_\pi(x)\mathrm{d}x} = \frac{\int_{\underline{\pi}}^{p} x f_\pi(x)\mathrm{d}x}{F_\pi(p)}, \qquad (8)$$

[4]Including deadlines on the 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.

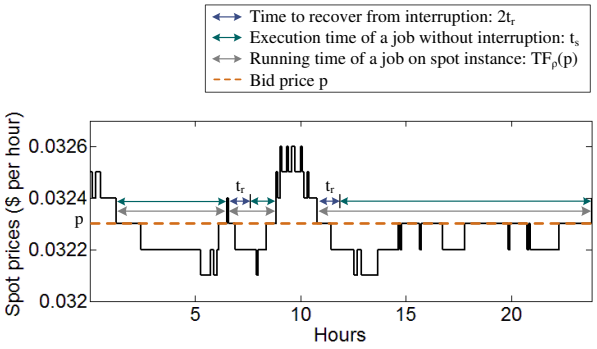


Figure 4: An example of job running times for the spot prices of a r3.xlarge-type instance in the US Eastern region on September 09, 2014.

which monotonically increases with p (cf. Appendix C). 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

$$\Phi_{so}(p) = t_s \mathbb{E}(\pi \,|\, \pi \leq p) = \frac{t_s \int_{\underline{\pi}}^{p} x f_\pi(x) \mathrm{d}x}{t_k F_\pi(p)}, \qquad (9)$$

i.e., the expected spot price, multiplied by the number of time slots it takes for the job to complete. 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.

The lowest possible bid price is determined by the constraint that the job completes before being terminated. We calculate this price by first finding the expected amount of time that a job will continue running without being interrupted:

$$t_k \sum_{i=1}^{\infty} i F_\pi(p)^{i-1}\big(1 - F_\pi(p)\big) = \frac{t_k}{1 - F_\pi(p)}. \qquad (10)$$

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

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

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

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 timeslots, which becomes more likely with a higher bid.

4.2 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 briefly discuss the implications of a job's interruptibility before finding the user's optimal bid price.

Job running time. The 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 the 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 $\mathbb{1}_\pi\big(\pi(t)\big)$ as an indicator function: $\mathbb{1}_\pi\big(\pi(t)\big) = 1$ if $p \geq \pi(t)$; otherwise, $\mathbb{1}_\pi\big(\pi(t)\big) = 0$. We then consider $\Big(\mathbb{1}_\pi\big(\pi(t)\big) - \mathbb{1}_\pi\big(\pi(t+1)\big)\Big)^2$, which equals 1 if $\mathbb{1}_\pi\big(\pi(t)\big) \neq \mathbb{1}_\pi\big(\pi(t+1)\big)$ (i.e., a transition happens) and 0 otherwise. The number of idle-to-running transitions in T/t_k time slots is then $\frac{1}{2}\sum_{k=0}^{T/t_k - 1}\Big(\mathbb{1}_\pi\big(\pi(t)\big) - \mathbb{1}_\pi\big(\pi(t+1)\big)\Big)^2$. We take the expectation to obtain

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

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

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

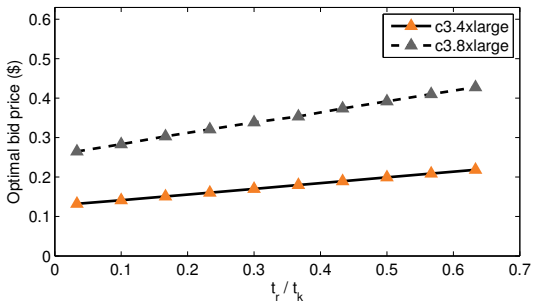


Figure 5: The optimal bid price $p^\star$ for persistent requests increases approximately linearly with recovery time t_r. We consider two different instance types; execution time is fixed as $t_s = 60s$.

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, we find that the job's running time is finite only if the recovery time is sufficiently small:

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

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. (10)): intuitively, the time to recover from an interruption should be smaller than the expected time on an instance between job interruptions. 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, $t_r < \min\left\{\frac{t_k}{1-F_\pi(p)}\right\} = t_k$, 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 (8) to find that the cost of a job with a persistent request is $\Phi_{sp}(p) = TF_\pi(p)\mathbb{E}(\pi \mid \pi \le p)$. The user's optimal bid price then satisfies

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

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 (cf. Appendix D), 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 to* (15) *is*

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

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 depends only 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. In Figure 5, we show that for the distributions of F_π that we derived in Section 3.3 (cf. Figure 3), the optimal bid price $p^\star$ increases approximately linearly t_r/t_k.

5. BIDDING FOR MAPREDUCE JOBS

We now adapt Section 4's bid 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.[5] Though the 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).

5.1 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 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.

[5]While master nodes are often run on on-demand instances to guarantee they will not be interrupted, our experiments in Section 6 show that with sufficiently high bids, interruptions are rare even on spot instances.

To find the 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.[6] Then the total running time satisfies:

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

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

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

Since all sub-jobs run simultaneously, the overall time to finish the parallelized job is $\max_{i=1,\dots,M} T_i F_\pi(p)$. Since all M sub-jobs are of equal size, we have

$$\min \max_{i=1,\dots,M} T_i F_\pi(p) = \frac{t_s + t_o - M t_r}{M\Big(1 - \frac{t_r}{t_k}\big(1 - F_\pi(p)\big)\Big)}. \tag{18}$$

Though the job is split into smaller pieces, the overall running time $\max_{i=1,\dots,M} T_i F_\pi(p)$ 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 4, the expected cost of running a job on M simultaneous instances is the sum of each instance's 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

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

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 like (15) in Proposition 5.

By comparing the costs at the optimal price for multiple bids and a single bid, 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] This overhead time may depend on M, the (fixed) number of sub-jobs.

5.2 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 bid 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 4.1.

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 5.1. 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 5.1 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:

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

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. (10) 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 6's experiments.

Table 2: EC2 instance types. Sizes are given in (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)

6. EXPERIMENTAL RESULTS

In this section, we first examine the optimal bid prices derived in Section 4 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 bid strategies. We then run a MapReduce example on spot instances to further highlight that our proposed bid strategy in Section 5 is adaptable to parallelized MapReduce jobs and can substantially lower the cost. 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.[7] We do not include the runtime of the bid price calculations in our measurements since it is under one minute, and can thus be easily run within one five-minute time slot.

6.1 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 use 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 create an Amazon Machine Image (AMI), where a shell script is 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 place all the spot requests using this AMI.

Results: We use 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 bid strategy for one-time requests can ensure reliability. Our bills show that the bid strategy for a one-time single bid can reduce up to 91% of user cost compared to running the on-demand instance. Figure 6 compares the cost of one-time requests on spot instances to the cost of on-demand instances for the instance types in Table 3. We also compare the actual costs to the expected cost from our analytical model; these analytical predictions closely match the experimental results.

[7]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.

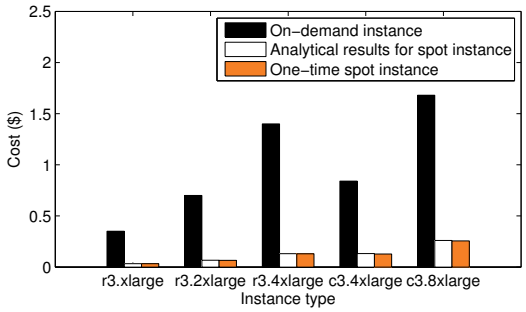


Figure 6: One-time spot instance requests substantially lower user cost compared to on-demand instances (on-demand and bid prices are as in Table 3).

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 somewhat longer completion times by placing persistent requests. In Figure 7, we plot the percentage difference in performance between the persistent and one-time bids.

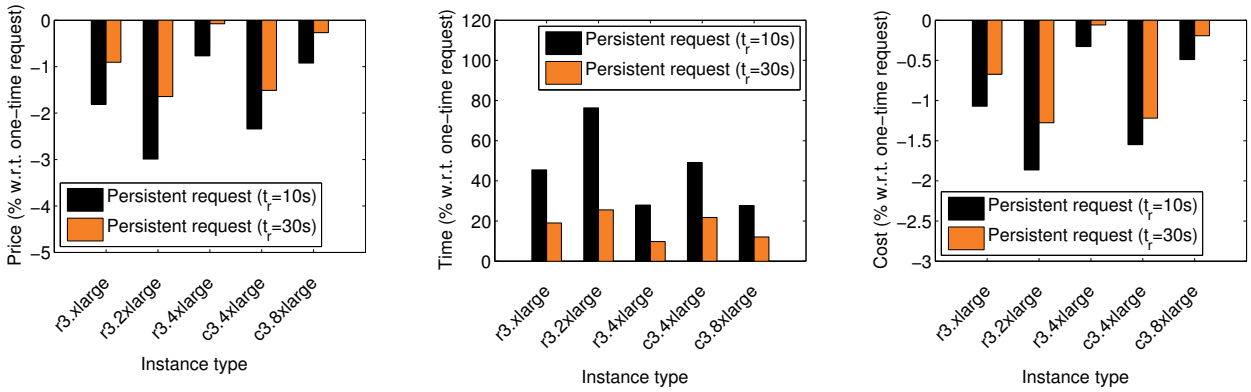
Figure 7(a) shows the percentage difference in price between the one-time and persistent requests for the five instance types. The negative values in this figure illustrate a lower optimal bid price for persistent requests: persistent requests can be interrupted. As we would expect from Section 4's results, longer recovery times ($t_r = 30$s rather than 10s) yield higher bid prices, since a higher bid price ensures that the job has sufficient time to execute after recovering from interruptions.

Conversely, as shown in Figure 7(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 when 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 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 7(a) and 7(b), for the persistent requests, the price charged per hour is lower but completion time is longer compared to the one-time requests. This makes the final cost underdetermined: Figure 7(c)

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 ($t_r = 30s$)	
			$p^\star$	$\mathbb{E}(\pi \mid \pi < p^\star)$	$p^\star$	T	$p^\star$	T
r3.xlarge	\$0.35	\$0.0321	\$0.0374	\$0.0331	\$0.0332	1.4549h	\$0.0355	1.1903h
r3.2xlarge	\$0.70	\$0.0646	\$0.0795	\$0.0669	\$0.0661	1.7638h	\$0.0711	1.2558h
r3.4xlarge	\$1.40	\$0.1286	\$0.1430	\$0.1304	\$0.1327	1.2798h	\$0.1422	1.0976h
c3.4xlarge	\$0.84	\$0.1281	\$0.1669	\$0.1324	\$0.1322	1.4917h	\$0.1413	1.2180h
c3.8xlarge	\$1.68	\$0.2561	\$0.2903	\$0.2604	\$0.2648	1.2767h	\$0.2831	1.1209h



(a) Expected spot price. (b) Job completion time. (c) Total job cost.

Figure 7: Persistent requests yield lower cost 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 for a MapReduce job.

Client Setting	Master node		Slave nodes		
	Instance Type	$p_m^\star$	Instance Type	$p_v^\star$	$M^\star$
CS1	c3.xlarge	\$0.133	m3.2xlarge	\$0.070	5
CS2	m3.xlarge	\$0.101	m3.2xlarge	\$0.070	5
CS3	m3.xlarge	\$0.102	r3.2xlarge	\$0.071	3
CS4	r3.xlarge	\$0.042	m3.2xlarge	\$0.070	5
CS5	r3.xlarge	\$0.042	r3.2xlarge	\$0.071	3

demonstrates that persistent bids lead to lower overall costs. Smaller recovery times ($t_r = 10$s versus 30s) yield lower costs due to their lower bid prices; though their completion times are higher, 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.

6.2 MapReduce Jobs

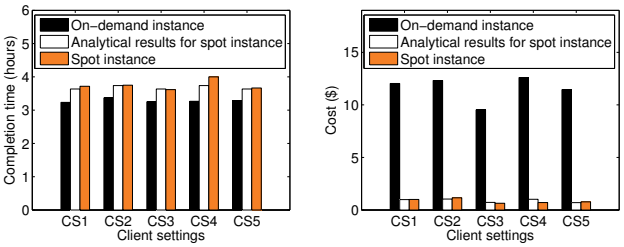
We now implement our optimal bidding strategy to run Hadoop MapReduce jobs on the Common Crawl Dataset [9] using Amazon Elastic MapReduce (EMR). This dataset, which is hosted on Amazon's Simple Storage Service, maintains an open repository of web crawl data that is accessible to the public for free. In this experiment, we run 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 bid strategy proposed in Section 5, 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 place 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 high-performance instances; 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 bid strategy for MapReduce jobs can reduce up to 92.6% of user cost with just a 14.9% increase of completion time.

In Figure 8, 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 instance (Figure 8(a)), while the cost of the job running on the spot instance is much lower than that on the on-demand in-



(a) Job completion time. (b) Job cost.

Figure 8: MapReduce jobs can save about 90% of user cost but have a 15% longer completion time on spot compared to on-demand instances.

stance (Figure 8(b)).

7. DISCUSSION

Like all work based on analytical models, our work has some limitations. We discuss four 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 utilization term and its revenue. However, in practice the provider can have other concerns, such as users' social welfare [26,37] or the electricity costs for running instances. While our current formulation matches well with the observed spot prices, including these factors in the provider's spot price optimization problem may shed more light on the provider's behavior.

Temporal correlations. We assume i.i.d. job arrivals that yield similarly i.i.d. spot prices at the equilibrium. However, empirical studies have found that cloud workloads exhibit temporal correlation, possibly inducing some temporal correlation in the spot prices [15]. In fact, a study of the spot prices in 2010 shows the presence of limited autocorrelation for consecutive time slots [6]. Incorporating these correlations into users' spot price predictions may improve their bid strategies.

Risk-averseness. We suppose that users choose their bid prices so as to minimize their expected costs, subject to constraints on the expected runtimes. However, 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 instance, we might calculate the worst-case (i.e., maximum) job running time in Sections 4 and 5 and choose the bid prices so as to minimize the resulting worst-case cost.

Collective user behavior. The bidding strategies that we have developed in Sections 4 and 5 assume that an individual user's bid price will not measurably affect the provider's spot price. While our experiments in Section 6 show that this assumption holds for a single user, it may not hold if multiple users begin to optimize their bid 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 3's model to derive the effect on the provider's offered spot price.

8. RELATED WORK

Cloud scheduling and pricing. Many works have considered resource allocation in the cloud from a purely operational perspective [10, 14, 23, 33]. Others incorporate pricing considerations, e.g., dynamically allocating cloud resources, so as to maximize the provider's revenue [12, 19, 34] or social welfare [26, 37]. We construct a similar model but relate it to empirical bid prices and use it to develop bid strategies for users. Joint user-provider interactions for usage-based pricing are considered in [32], but auction-specific works on both provider and user actions are limited to statistical studies of historical spot prices [6, 17].

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, 36], they only consider heuristic bidding strategies for single-instance jobs.

In general, auction frameworks assume that users' bids are determined by their valuation of the auctioned resource. Indeed, many works have studied the problem of designing online auctions to ensure truthful user bids [7, 13, 22, 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 valuation for completing a job, *job interruptions* will increase the number of instance-time slots required to complete the job. This consideration differentiates our work from other auctions for computing or utility resources, e.g., auctions for smart grid electricity [11], secondary spectrum access [18], grid computing [20], or Internet data [25].

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 bid strategies that lower their cloud resource costs. In this work we first consider providers' setting of the spot prices, developing a revenue-maximization model for the provider and comparing its results to the historical 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 tradeoff between bidding higher prices to avoid interruptions (for one-time requests) and bidding lower prices to save money (for persistent requests).

To show the validity of this tradeoff and our bid strategies' adaptability to specific job requirements, we adapt these bid strategies to MapReduce jobs with mas-

ter 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. We visualize the resulting price per hour, job completion time and final cost for both on-demand and spot instances for a variety of instance types. Our bid strategies can reduce users' costs by around 90%, with modest increases in job completion times. Appropriate bidding strategies thus allow users to significantly reduce the cost of running their jobs while successfully managing job interruptions.

APPENDIX

A. PROOF OF PROPOSITION 1

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

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

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

B. 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

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

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

C. PROOF OF PROPOSITION 4

PROOF. By taking the first-order derivative of $\Phi_{so}(p)$ in (9), we have

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

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 $g(p)$ leads 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 $p\ge F_\pi^{-1}\big(1-\frac{t_k}{t_s}\big)$ due to the monotonic property of $F_\pi(p)$. Thus, the minimum of the feasible set is the larger one of $\underline{\pi}$ and $F_\pi^{-1}\big(1-\frac{t_k}{t_s}\big)$. □

D. PROOF OF PROPOSITION 5

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

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

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 $F_\pi(p)-pf_\pi(p)\ge 0$. We then have $\partial g(p)/\partial p\ge 0$. 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

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

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

10. REFERENCES

[1] Agmon Ben-Yehuda, O., Posener, E., Ben-Yehuda, M., Schuster, A., and Mu'alem, A. Ginseng: market-driven memory allocation. In *Proceedings of the 10th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments* (2014), ACM, pp. 41–52.

[2] Amazon. EC2 spot instance. *online at: http://aws.amazon.com/ec2/purchasing-options/spot-instances/* (2015).

[3] Amazon. Elastic Compute Cloud (Amazon EC2). *online at: http://aws.amazon.com/ec2/* (2015).

[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] Barabasi, A.-L. The origin of bursts and heavy tails in human dynamics. *Nature 435*, 7039 (2005), 207–211.

[6] Ben-Yehuda, O. A., Ben-Yehuda, M., Schuster, A., and Tsafrir, D. Deconstructing Aamazon EC2 spot instance pricing. *ACM Trans. on Economics and Computation 1*, 3 (2013), 1–16.

[7] Blum, A., Sandholm, T., and Zinkevich, M. Online algorithms for market clearing. *Journal of the ACM (JACM) 53*, 5 (2006), 845–879.

[8] Buyya, R., Yeo, C. S., and Venugopal, S. Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In *Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications* (2008), IEEE, pp. 5–13.

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

[10] Dogar, F. R., Karagiannis, T., Ballani, H., and Rowstron, A. Decentralized task-aware scheduling for data center networks. In *Proceedings of ACM SIGCOMM* (2014).

[11] Fang, X., Misra, S., Xue, G., and Yang, D. Smart gridÑthe new and improved power grid: A survey. *IEEE Communications Surveys & Tutorials 14*, 4 (2012), 944–980.

[12] Feng, G., Garg, S., Buyya, R., and Li, W. Revenue maximization using adaptive resource provisioning in cloud computing environments. In *Proceedings of the 2012 ACM/IEEE 13th International Conference on Grid Computing* (2012), pp. 192–200.

[13] Friedman, E. J., and Parkes, D. C. Pricing wifi at starbucks: issues in online mechanism design. In *Proceedings of the 4th ACM conference on Electronic commerce* (2003), ACM, pp. 240–241.

[14] Ghodsi, A., Zaharia, M., Hindman, B., Konwinski, A., Shenker, S., and Stoica, I. Dominant resource fairness: Fair allocation of multiple resource types. In *Proceedings of NSDI* (2011), vol. 11, pp. 24–24.

[15] 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.

[16] Ha, S., Sen, S., Joe-Wong, C., Im, Y., and Chiang, M. TUBE: time-dependent pricing for mobile data. In *Proceedings of ACM SIGCOMM* (2012).

[17] Javadi, B., Thulasiram, R. K., and Buyya, R. Statistical modeling of spot instance prices in public cloud environments. In *Proc. of IEEE UCC* (2011), pp. 219–228.

[18] 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.

[19] Jin, H., Wang, X., Wu, S., Di, S., and Shi, X. Towards optimized fine-grained pricing of iaas cloud platform. *IEEE Transactions on Cloud Computing PP* (2014).

[20] 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).

[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] Lavi, R., and Nisan, N. Online ascending auctions for gradually expiring items. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms* (2005), Society for Industrial and Applied Mathematics, pp. 1146–1155.

[23] Lee, G., Chun, B.-G., and Katz, R. H. Heterogeneity-aware resource allocation and scheduling in the cloud. *Proceedings of HotCloud* (2011), 1–5.

[24] 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 *Proceedings of IEEE INFOCOM* (2001), vol. 2, IEEE, pp. 1095–1103.

[25] 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.

[26] Menache, I., Ozdaglar, A., and Shimkin, N. Socially optimal pricing of cloud computing resources. In *Proceedings of the 5th International ICST Conference on Performance Evaluation Methodologies and Tools* (2011), ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), pp. 322–331.

[27] Ng, C., Parkes, D. C., and Seltzer, M. Virtual worlds: Fast and strategyproof auctions for dynamic resource allocation. In *Proceedings of the 4th ACM Conference on Electronic Commerce* (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 *Proceedings of the 5th ACM conference on Electronic commerce* (2004), ACM, pp. 61–70.

[30] Ren, S., Park, J., and Van Der Schaar, M. Entry and spectrum sharing scheme selection in femtocell communications markets. *IEEE/ACM Transactions on Networking 21*, 1 (2013), 218–232.

[31] Sen, S., Jin, Y., Guérin, R., and Hosanagar, K. Modeling the dynamics of network technology adoption and the role of converters. *IEEE/ACM Transactions on Networking 18*, 6 (2010), 1793–1805.

[32] Urgaonkar, B., Kesidis, G., Shanbhag, U. V., and Wang, C. Pricing of service in clouds: optimal response and strategic interactions. *ACM SIGMETRICS Performance Evaluation Review 41*, 3 (2014), 28–30.

[33] Van den Bossche, R., Vanmechelen, K., and Broeckhove, J. Cost-optimal scheduling in hybrid iaas clouds for deadline constrained workloads. In *Proceedings of the IEEE International Conference on Cloud Computing* (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] 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.

[37] Zhang, L., Li, Z., and Wu, C. Dynamic resource provisioning in cloud computing: A randomized auction approach. In *Proc. of IEEE INFOCOM* (2014).

[38] Zhou, Y., and Wentzlaff, D. The sharing architecture: sub-core configurability for iaas clouds. In *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems* (2014), ACM, pp. 559–574.