An Empirical Analysis of Amazon EC2 Spot Instance Features Affecting Cost-Effective Resource Procurement

CHENG WANG, VMware Inc.
QIANLIN LIANG, Ele.me Inc.
BHUVAN URGAONKAR, Penn State University

Many cost-conscious public cloud workloads ("tenants") are turning to Amazon EC2's spot instances because, on average, these instances offer significantly lower prices (up to 10 times lower) than on-demand and reserved instances of comparable advertised resource capacities. To use spot instances effectively, a tenant must carefully weigh the lower costs of these instances against their poorer availability. Toward this, we empirically study four features of EC2 spot instance operation that a cost-conscious tenant may find useful to model. Using extensive evaluation based on historical spot instance data, we show shortcomings in the state-of-the-art modeling of these features that we overcome. As an extension to our prior work, we conduct data analysis on a rich dataset of the latest spot price traces collected from a variety of EC2 spot markets. Our analysis reveals many novel properties of spot instance operation, some of which offer predictive value whereas others do not. Using these insights, we design predictors for our features that offer a balance between computational efficiency (allowing for online resource procurement) and cost efficacy. We explore "case studies" wherein we implement prototypes of dynamic spot instance procurement advised by our predictors for two types of workloads. Compared to the state of the art, our approach achieves (i) comparable cost but much better performance (fewer bid failures) for a latency-sensitive in-memory Memcached cache and (ii) an additional 18% cost savings with comparable (if not better than) performance for a delay-tolerant batch workload.

CCS Concepts: • **Computing methodologies** → **Model development and analysis**; • **Computer systems organization** → **Cloud computing**;

Additional Key Words and Phrases: Spot instance features, resource procurement

ACM Reference format:
Cheng Wang, Qianlin Liang, and Bhuvan Urgaonkar. 2018. An Empirical Analysis of Amazon EC2 Spot Instance Features Affecting Cost-Effective Resource Procurement. *ACM Trans. Model. Perform. Eval. Comput. Syst.* 3, 2, Article 6 (March 2018), 24 pages.
https://doi.org/10.1145/3164538

1 INTRODUCTION

Amazon EC2 has been offering spot instances [6] since 2009, and a large segment of its "tenant" workloads has come to embrace these [27]. The appeal of spot instances lies in their low prices—

This article is an extended version of an ICPE '17 paper by the same authors [38].
Part of this research work was done when the first author was a Ph.D. student at Penn State University.
This research was supported in part by an NSF CNS 1717571 grant and an IBM faculty partnership award.
Authors' addresses: C. Wang, 6500 Riverplace Blvd, Building 6, Austin TX, USA, 78730; email: wangcheng@vmware.com; Q. Liang, 1518 Jinshajiang Road, Putuo District, Shanghai, P.R.China, 200333; email: qianlin.liang@ele.me; B. Urgaonkar, W371 Westgate Building, School of EECS, The Pennsylvania State University, University Park PA, USA, 16802; email: bhuvan@cse.psu.edu.

up to 1/10 that of on-demand instances of equivalent capacities. Unlike an on-demand instance, whose price changes slowly (over months or years), a spot instance has a highly dynamic price that may change as frequently as once every few minutes.[1]

From a tenant's point of view, an EC2 spot instance is a virtual machine (VM) that is cheaper than its on-demand or reserved counterpart but appears to have poorer availability. To use spot instances effectively, a tenant must be able to model well-relevant aspects of their operation. After submitting a bid, how long does it take for an instance to be ready for use? What is the expected lifetime of an instance? What would the costs be during its lifetime? What is the probability of simultaneous bid failures if the tenant places bids across spot markets? Finally, the tenant must combine these predictions with its application-specific trade-offs to devise online instance procurement algorithms.[2] Whereas such online procurement ("control") algorithms have been extensively researched recently (see Section 2.2), we find significant shortcomings in modeling and prediction of key features of spot operation upon which these algorithms rely. Overcoming these shortcomings is the goal of this study.

Research contributions.

- We consider four key features that we believe a tenant should model: (i) lifetime of an instance, (ii) average spot price during the lifetime, (iii) simultaneous revocations, and (iv) startup delay. In particular, for (i) and (ii), we show that the most commonly used prediction approach (based on cumulative distribution of spot prices) fails to capture the tenant's service contiguity, and design quantitative models and computationally efficient predictors (Section 3.1). For (iii), we find that the existing approach (based on correlations of spot price traces from different markets) overlooks the tenant's bids in its prediction and thus may fail to reflect the actual likelihood of simultaneous revocations, which our technique captures well (Section 3.2). Finally, our study provides valuable firsthand data and insightful results for evaluating the predictability of (iv) (Section 3.3), which is usually ignored by prior work, and the predictive power of "effective capacity" for spot prices (Section 3.4).
- We evaluate the efficacy of our modeling and prediction in two ways. First, by using extensive real-world data, we demonstrate that our proposed models and predictors outperform the baseline approaches that are commonly used in prior work (Section 3). Second, we take two real-world case studies and adapt resource procurement algorithms from related work to demonstrate how the tenants could leverage our models and predictors for cost-effective operations, as well as the improvements in performance and/or costs[3] (Section 4). Our approach offers (i) comparable cost but much better performance (fewer bid failures and lower tail latency) for a latency-sensitive in-memory Memcached workload and (ii) an additional 18% cost savings with comparable performance for a delay-tolerant batch workload.
- Our study reveals several novel insights about spot operation with implications for tenant control. Among our salient findings are the following. First, whereas raw spot prices are best considered nonstationary, most of our features exhibit short-term *temporal locality* (borrowing a phrase from caching and memory management) that can be leveraged for prediction.

[1]Although it may seem surprising to some, spot prices are known to exceed on-demand prices, sometimes significantly (see several examples later in Figure 2). We surmise that this may be part of mechanisms employed by EC2 to shed increasing load on its servers hosting spot instances. Others have also reported that EC2 spot pricing seems to have elements beyond simply reflecting supply-demand relationships [1].
[2]These trade-offs would be between costs on the one hand and overheads of possible revocations (either in the form of fault-tolerance mechanisms or loss of performance/correctness) on the other.
[3]All of our code and data is available at Github [22]. Our code includes both trace analysis and control implementation of tenant-side instance procurement on EC2.

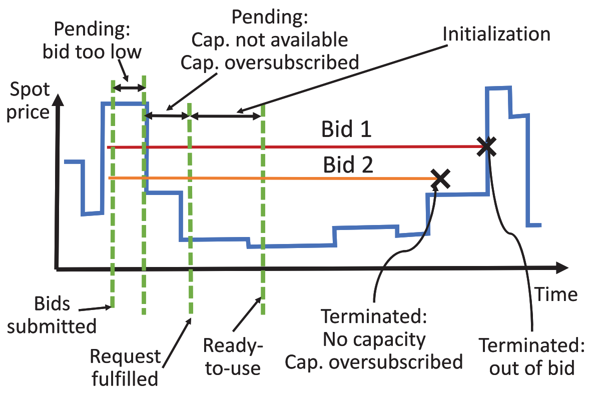

Fig. 1. Illustration of a spot instance operation using two hypothetical bids (Bid 1 and Bid 2).

Second, exploiting spatial locality simply via cross correlation of spot prices across markets could be misleading; it is important that the modeling of simultaneous revocations be conditioned on specific bid values. Third, we do not find significant statistical correlation between effective capacity measurements and the spot price, indicating the evolution of spot price likely depends on a spatially coarse (e.g., data center or availability zone wide) load metric.

- We extend our prior work [38], which conducts data analysis based on an outdated dataset (07/08/2015 to 10/06/2015) of spot price traces, by repeating our experiments using a richer dataset of the latest spot price traces collected from 03/12/2017 to 06/10/2017 across multiple EC2 global regions, availability zones, instance types, and operating system types. We find that the key insights based on the old dataset still hold, which further demonstrates the effectiveness of our proposed approaches. The dataset used in this article also reveals new insights that could lead to future directions. For example, the best choice of the history window size for most of the spot markets in the new dataset is at least twice as large as that in the old dataset. Further explorations of such behaviors may help improve the quality of our prediction approaches.

Outline. In Section 2, we discuss the life cycle of a spot instance and related work. In Section 3, we identify the key features and present our models and predictions. In Section 4, we provide real-world case studies to show the efficacy of our approaches. We conclude in Section 5.

2 BACKGROUND

2.1 Life of a Spot Instance

We show the key events in the life of a spot instance in Figure 1 (more details can be found at the AWS Web site [29]. We assume that Bid 1 and Bid 2 (with the former being higher) are placed at the time shown. After a tenant submits a bid, there can be a period during which its bid is strictly less than the spot price. It is quite well known that during such a period, the tenant's request for an instance is not granted (EC2 shows the status of the tenant's request as "pending: bid too low"). It is less well known, however, that there may sometimes be an additional delay[4] with the request status being "pending: capacity not available" or "pending: capacity oversubscribed" even after the

[4]There may in fact be even more reasons of delay. We discount these since these are mostly due to issues with the tenant's configuration. For example, the tenant may tell EC2 to launch a set of spot instances only if it can launch them all (also known as a launch group); the bid status would be "launch-group-constraints" if EC2 cannot launch all at the same time.

spot price has fallen below the bid. According to EC2, this happens if the spot market does not have available capacity or capacity is oversubscribed. After the request is "fulfilled," EC2 launches a spot instance and "initializes" it with the tenant-specified configuration, after which the instance is "ready to use."

In Figure 1, both bids are fulfilled at the same time, resulting in the two instances becoming ready to use at the same time. The following two ways for an instance to terminate are well known: (i) the tenant may use the instance until its needs are met and then voluntarily terminate the instance, or (ii) the bid may fall below the spot price that would cause EC2 to revoke the instance (shown for Bid 1 with a status of "terminated: out of bid"). A third less well-known way for an instance to terminate, however, is one shown occurring for Bid 2 wherein the instance is reclaimed by EC2 allegedly due to capacity scarcity (with a status of "terminated: out of capacity" or "terminated: oversubscribed"). Startup delays or involuntary terminations due to alleged capacity scarcity are aspects of the spot instance operation that bring additional complexity into their usage but have not been considered in related work.

When an instance is revoked by EC2, the tenant loses all of its local state (contents of main memory and local disks of the instance). EC2 does issue a warning to the tenant before revoking a spot instance (2 minutes prior to revocation). The tenant may choose to use this warning period to save some or all of that instance's local state. Finally, it must be noted the tenant is billed based on the spot price during the instance's lifetime (and not based on its bid).

2.2 Related Work

Spot Price Modeling/Prediction. There has been a plethora of related work that investigates effective modeling and prediction of spot prices. One line of work applies relatively simple approaches (in terms of the amount of historical data employed as well as the computational complexity of the model), such as autoregressive models [1, 25, 36, 43], to predict the short-term spot prices. However, since spot prices could vary at minute granularity, predictors that rely on such techniques may only be able to predict the near-future spot prices, which may fail to offer insights on how the spot price might evolve in the long run and is not suitable for cloud applications that need to provide long-lasting services.

Another line of work leverages the empirically measured cumulative distributions of key parameters [7, 13, 19, 23, 24, 31, 32, 44], or even more complex probablistic models (e.g., Markovian models [2, 18, 21, 26, 42]) while adapting model parameters over time. Models based on empirical distributions, although offering an improved treatment of longer-term properties than regressive models, usually discard valuable temporal information about the continuity of the spot price staying below different bid values. Therefore, they may fail to capture well the continuity of service availability, which is of great concern particularly for long-lived and "stateful" applications. We refer to such approaches as "CDF based" (short for cumulative density function based) and discuss the details of their limitations in Section 3.1.

As an extension to the CDF-based models, Song et al. [26] introduce the "sojourn time" of a spot instance procured via a particular bid and predict it via a semi-Markov chain. However, tenant control based on such multidimensional models would likely suffer scalability limitations when considering multiple spot markets with multiple bids for better availability and profitability.

Prior work models concerns arising from simultaneous revocation of spot instances across markets via cross correlations or correlation coefficients of raw spot price traces [31, 32]. In Section 3.2, we argue why this can be misleading and why it is important to consider simultaneous revocations conditioned on specific bid values. Finally, related work has ignored the startup delay of spot instances (which can be longer than that of on-demand instances) and its implications for control design and operation. To our knowledge, one exception is the work of Mao and Humphrey [17].

Even this, however, focuses on the boot time of instances, which is only part of the overall startup delay.

Cost-effective resource procurement with spot instances. A large body of related work provides cost-effective solutions for tenant-side procurement ("control") of spot instances, combined with on-demand and/or reserved instances, for different workloads or applications, such as delay-tolerance batch jobs [7, 12, 14, 16, 18, 21, 26, 31, 34, 42], HPC [8, 33], video streaming [10], and data caching [41]. In particular, Xu et al. [41] propose to replicate data contents on spot instances from multiple markets and only use spot instances to serve read requests while the write requests are served by on-demand instances that never fail. In our in-memory data caching case study (Section 4.1), to further reduce resource wastage and improve cost efficacy, we intelligently mix hot and cold data across spot and on-demand instances and minimize the performance degradation due to spot revocations. Similar to our case study of batch processing (Section 4.2), Sharma et al. [24] and Subramanya et al. [31] replicate batch jobs on spot or on-demand instances with higher consolidation degree and improve the performance of the backup copy of a job once its primary copy (on a cheap spot instance) fails. However, for tractability reasons, the prior work usually resorts to the aforementioned simple spot price prediction techniques in their resource procurement, which do not capture well the key feature we identify in our work (e.g., the continuity of service availability and simultaneous revocations across spot markets). Several fault-tolerance mechanisms have been explored to deal with bid failures, such as checkpointing, live migration, and replication, which are complementary to our work and can be incorporated with the key features we identify to provide better performance. EC2 itself provides a facility called *Spot Fleet* [28] for tenant procurement. However, the default bidding strategies are either evenly spread the spot requests across pools, which may not be cost effective, or only choose the pool with the lowest spot price, which may suffer from simultaneous bid failures.

3 USEFUL SPOT FEATURES

We describe four spot instance features on which we believe a tenant should focus. For each, we present a quantitative representation (a "model") and offer intuition for why it might be useful in online resource procurement ("control"). A key idea that is cross cutting our models is to express our features as quantities that are conditioned on specific bids chosen out of a small set of preselected values. For each feature, we explore one or more of the following properties that might be intuitively appealing for their potential in offering predictive value:

- *Temporal locality:* Does historical data offer useful hints about future evolution of this feature? If so, what is the right amount of historical data to consider?
- *Spatial locality:* How does the evolution of this feature in a market relate to that in others in the same versus different availability zones or geographic regions?
- *Capacity measurements:* Do effective capacity measurements [39] have any predictive value? In other words, do such measurements serve as faithful indicators of changes in spot prices or availability capacity in the concerned marketplace?

Using lessons learned from this exercise, we design computationally efficient predictors for our features. Finally, we evaluate our predictors on a large set of spot price marketplaces. Figure 2 shows a 90-day-long spot price time series for 16 marketplaces, which we evaluate in this work. We also plot the prices of their on-demand counterparts (with equal capacity) in the same figures. These measurements show high variations especially in spot markets with frequent spikes (e.g., r3.xlarge in us-east-1c) and suggest that raw spot prices are best considered (highly) nonstationarity, making questionable the efficacy of approaches in existing works (e.g., Ben-Yehuda et al. [1]; Wallace et al. [36]; Zhao et al. [43]) that rely on modeling them directly (i.e., implicitly assuming

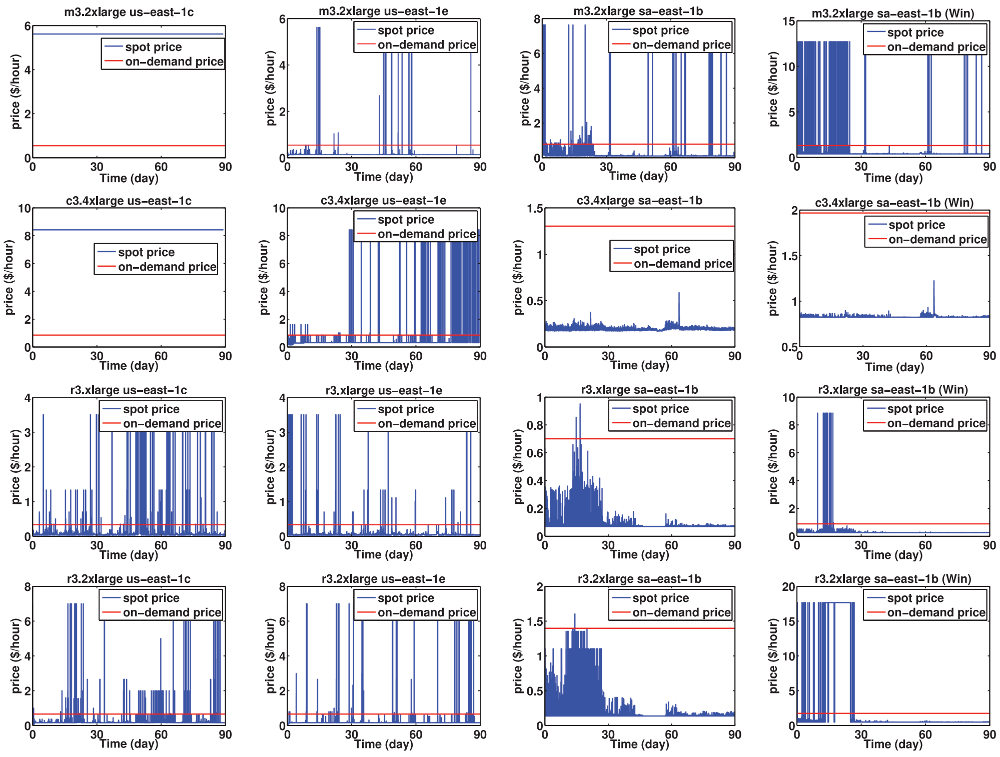

Fig. 2. Sample spot price time series collected during the 90-day period (03/12/2017 to 06/10/2017), chosen due to their very different properties. We show spot price traces from two EC2 global regions, us-east-1 (US East) and sa-east-1 (South America). Within each region, we pick different availability zones, denoted by the tail characters c, e, and b. For traces from the same availability zone, we use "Win" to represent Windows VMs (Linux otherwise). For each availability zone, we select four VM types: m3.2xlarge (general purpose), c3.4xlarge (compute optimized), and r3.xlarge and r3.2xlarge (memory optimized).

Table 1. VM Configuration and On-Demand Prices of the VM Types in Figure 2

VM Type	vCPU	RAM (GiB)	On-Demand Price ($/hour)			
			us-east-1c	us-east-1e	sa-east-1b	sa-east-1b (Win)
m3.2xlarge	8	30	0.532	0.532	0.761	1.265
c3.4xlarge	16	30	0.84	0.84	1.3	1.964
r3.xlarge	4	30.5	0.333	0.333	0.7	0.884
r3.2xlarge	8	61	0.665	0.665	1.399	1.767

Note: "Win" represents Windows VM instances (Linux otherwise).

temporal locality in raw prices). A key finding of our work (elaborated upon throughout the rest of this section) is that while assuming temporal locality in raw spot prices may not be reasonable, the features we identify (and which we find useful for control) do indeed show short-term temporal locality. The VM configurations and on-demand prices of the chosen VM types are shown in Table 1. We find that the on-demand prices of the same VM type could also vary across global regions and availability zones as well as different operating systems. In the same region, the spot market with a higher on-demand price may also have larger range of spot price variations.

3.1 Features 1 and 2: Lifetime and Average Price During Lifetime

We present our first two features together due to significant connections between their modeling and prediction.

A tenant would like a spot instance to be available long enough to serve its needs. In other words, it would be interested in the following question: how long is a successful bid going to last? To answer this, our first feature is concerned with the lifetime of a spot instance, which we define as the duration between when it becomes ready to use until its termination.[5] An effective model for this feature should not overestimate this quantity—doing so may render a control scheme overly optimistic in its estimation of the cost versus performance trade-off. However, underestimating it may lead to higher than desired costs. Based on Section 2.1, a spot instance could be terminated due to either bid failure ("terminated: out of bid") or nonbid failure ("terminated: not enough capacity" or "terminated: capacity oversubscribed"). We find the latter to be a rare event in today's EC2 spot markets. Therefore, in what follows, we ignore time to nonbid failure and only focus on time to bid failure in our analysis. It should be pointed out that in an alternate spot market (e.g., in a future cloud with higher data center utilization and/or one using alternate resource management policies [15]), such terminations may not be nonnegligible. Modeling of spot instance lifetime in such environments would be made especially complex because these two types of terminations may themselves not be independent (due to possible dependence through load on the data center).

Our second feature is the average spot price during an instance's lifetime. Since spot prices tend to be significantly smaller than on-demand prices (of equally sized instances) during periods when a bid is successful, and since EC2 charges a tenant based on the spot price during such periods (but not the bid), attempting to predict spot prices accurately is of little value. A visual inspection of the 90-day spot prices in Figure 2 and how these prices compare to a bid that equals the on-demand price clarifies this.[6] In particular, it suffices that we predict the average spot price during such periods with reasonable accuracy (since that is what will determine our costs).

Limitation of Prior Work. As discussed in Section 2.2, prior studies that rely on CDF-based modeling (even if dynamically updated) of spot price to predict instance lifetime may not be able to capture well service contiguity. We illustrate their pitfalls using Figure 3 via three synthetic traces. Although all traces have availability of 0.7 under the same bid, the bottom trace is clearly much worse than the top one since it incurs more frequent bid failures, which the CDF-based approaches would fail to distinguish. With such a model, a tenant might be tempted to use spot instances more aggressively than he should, which might cause performance degradation due to more frequent than desired service interruption. However, more complex statistical models usually result in control that suffer from scalability limitations (the "curses of dimensionality" of dynamic programming–based approaches).

Models and Temporal Locality-Based Prediction. For both of these features, we find that effective models are offered by viewing them as random variables informed by empirically measured

[5]Clearly, the lifetime defined in this manner could depend intimately on the time when a bid is placed. We do not consider this complexity in our work because it is conceptually simple to extend our characterization for this. For example, we could carry out our analysis separately for each hour of the day (or another appropriate time duration).

[6]Although it may seem surprising to some, spot prices are known to exceed on-demand prices, sometimes significantly (see several examples in Figure 2). We surmise that this may be part of mechanisms employed by EC2 to shed increasing load on its servers' hosting spot instances. Others have also reported that EC2 spot pricing seems to have elements beyond simply reflecting supply-demand relationships [1].

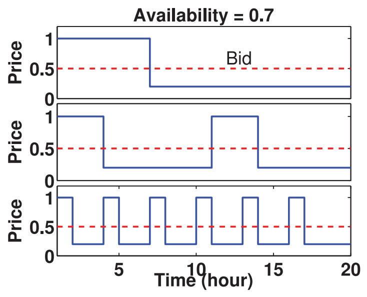

Fig. 3. Pitfalls of CDF-based approaches used in prior work.

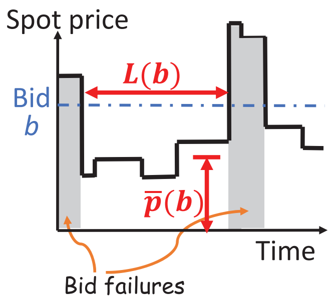

Fig. 4. Illustration of features 1 and 2: lifetime of a spot instance and average spot price during the lifetime.

probability distributions in the recent past. A key requirement for such modeling to be effective lies in choosing the right amount of historical data, which we describe momentarily. First, we introduce the definitions for the random variables we choose. We model as a random variable $L(b)$ the length of a *contiguous* period during which the spot price is less than or equal to a bid b. In other words, $L(b)$ captures the upper bound of the lifetime of a spot instance using bid b. We denote as $\bar{p}(b) = E[p_t|L(b)]$ a random variable for the average spot price p_t during a period when the bid b is successful[7], which serves to estimate the cost of a spot instance procured by placing a bid b. We set $L(b)$ and $\bar{p}(b)$ to 0 during the period when the bid fails. Figure 4 illustrates $L(b)$ and $\bar{p}(b)$.

Our prediction techniques assume temporal locality over a recent sliding time window (H most recent time slots, e.g., days) for making predictions of $L(b)$ and $\bar{p}(b)$. H must be chosen such that temporal locality[8] indeed holds for these quantities. Large $L(b)$ and small $\bar{p}(b)$ imply long service continuity and low costs, thereby encouraging the use of spot instances using bid b. We use a small percentile (e.g., fifth) of the recently constructed distribution of $L(b)$—denoted as $\hat{L}(b)$—as our prediction in the ongoing horizon. The reasoning behind this choice is that if the statistical properties of $L(b)$ do not change much over H, we expect that with a very high probability, bid b would be successful for at least $\hat{L}(b)$ time units. We use average of $\bar{p}(b)$ during the relevant H as its predictor (denoted as $\hat{\bar{p}}(b)$). Note that our approach would result in a control formulation wherein the number of state/control variables grows linearly with the number of (market, bid) pairs, whereas the semi-Markov model–based approach discussed in Section 2.2 has to discretize all state and control variables including spot prices from different markets, bids, sojourn time (from minutes to hours), and other application-specific variables, which results in optimization problems that suffer from scalability limitations.

Evaluation of Our Predictors. To evaluate our predictors, we introduce the following assessment metrics. We say that an *overestimation* of $L(b)$ has occurred when $\hat{L}(b) > L(b)$. This represents a scenario wherein the tenant was likely overly ambitious in using spot instances. We further define $L(b)$ *overestimation rate* as the fraction of $L(b)$ predictions that result in overestimation, denoted as $f(b)$. The assessment metric for $\hat{\bar{p}}(b)$ should capture the extent of its deviation from actual values. Therefore, we compute $\xi(b) = (\bar{p}(b) - \hat{\bar{p}}(b))/\bar{p}(b)$ and define as *relative deviation* of $\bar{p}(b)$ the mean

[7]We are overloading the term $L(b)$ to mean a contiguous duration when bid b is successful as well as its length. We are avoiding additional notational complexity since the distinction is very clear based on context.
[8]By temporal locality, we mean that over relatively short timescales (a day to a few days), the key features tend to change little, whereas over longer timescales (weeks to months), they might undergo more substantial changes.

Table 2. The Assessment Metrics $f(b)$ and $\xi(b)$ Under Different Bid Values and History Window Sizes (H in days)

	Bid	$f(b)$				$\xi(b)$				$f(b)$ CDF-based				$\xi(b)$ CDF-based			
H		7	14	21	28	7	14	21	28	7	14	21	28	7	14	21	28
m3.2xlarge-U	$0.5d$	0.07	0.07	0.04	**0.00**	0.01	0.01	0.00	**0.00**	0.17	0.16	**0.15**	0.15	0.01	0.01	0.01	**0.01**
	$1d$	0.11	**0.06**	0.07	0.06	0.04	**0.01**	0.02	0.02	0.19	0.14	**0.13**	0.13	0.03	0.01	0.01	**0.01**
	$2d$	0.11	**0.08**	0.10	0.09	0.02	**0.01**	0.02	0.02	0.17	0.16	**0.15**	0.16	0.02	0.01	0.01	**0.01**
	$5d$	0.13	**0.10**	0.11	0.10	0.03	**0.02**	0.03	0.04	0.18	0.16	**0.16**	0.18	0.03	0.02	0.02	**0.02**
	$10d$	0.13	**0.10**	0.11	0.10	0.03	**0.02**	0.03	0.04	0.18	**0.16**	0.16	0.18	0.03	0.02	0.02	**0.02**
m3.2xlarge-S	$0.5d$	0.08	0.07	0.06	**0.05**	0.06	0.11	0.12	**0.03**	**0.14**	0.14	0.16	0.21	0.06	**0.05**	0.05	0.05
	$1d$	0.10	0.08	0.05	**0.05**	0.06	0.08	0.07	**0.01**	0.21	**0.18**	0.19	0.19	0.10	0.08	0.08	**0.04**
	$2d$	0.13	0.10	0.06	**0.06**	0.68	0.61	0.07	**0.01**	0.21	0.21	0.20	**0.20**	0.15	0.13	0.08	**0.04**
	$5d$	0.13	0.10	0.06	**0.06**	0.51	0.42	0.07	**0.01**	0.19	**0.18**	0.18	0.20	0.14	0.12	0.08	**0.04**
	$10d$	**0.00**	0.00	0.00	0.00	1.03	1.00	**0.96**	0.98	**0.00**	0.00	0.00	0.00	**0.65**	0.68	0.72	0.76
r3.xlarge-U	$0.5d$	0.02	0.01	0.01	**0.00**	0.00	0.00	0.00	**0.00**	**0.15**	0.16	0.16	0.16	**0.18**	0.19	0.19	0.20
	$1d$	0.06	0.03	0.01	**0.00**	0.27	0.34	0.00	**0.00**	**0.18**	0.19	0.18	0.18	**0.22**	0.22	0.23	0.24
	$2d$	0.08	0.04	0.04	**0.01**	0.25	0.33	0.37	**0.00**	0.21	0.23	0.22	**0.20**	**0.21**	0.22	0.22	0.23
	$5d$	0.08	**0.03**	0.05	0.03	0.31	**0.30**	0.33	0.31	0.15	0.15	0.16	**0.11**	**0.25**	0.25	0.26	0.27
	$10d$	**0.08**	0.03	0.05	0.03	0.31	**0.30**	0.33	0.31	0.15	0.15	0.16	**0.11**	**0.25**	0.25	0.26	0.27
r3.xlarge-S	$0.5d$	0.06	0.06	**0.00**	0.00	0.02	0.01	0.00	**0.00**	0.15	0.13	0.05	**0.00**	0.02	0.01	0.00	**0.00**
	$1d$	0.04	0.01	**0.00**	0.00	**0.03**	0.03	0.03	0.03	0.05	0.03	0.00	**0.00**	0.02	0.01	0.01	**0.00**
	$2d$	0.00	0.00	0.00	0.00	**0.03**	0.03	0.03	0.03	0.00	0.00	0.00	0.00	0.01	0.01	0.01	**0.00**
	$5d$	0.00	0.00	0.00	0.00	**0.03**	0.03	0.03	0.03	0.00	0.00	0.00	0.00	0.01	0.01	0.01	**0.00**
	$10d$	0.00	0.00	0.00	0.00	**0.03**	0.03	0.03	0.03	0.00	0.00	0.00	0.00	0.01	0.01	0.01	**0.00**

Note: The shaded cells represent the best window size that minimizes $f(b)$ and $\xi(b)$. "-U" and "-S" represent the markets "us-east-1e" (US East) and "sa-east-1b" (South America). We round up the results and only show two digits after the decimal point due to space limitations. "CDF-based" $f(b)$ and $\xi(b)$ are computed based on predictions of $L(b)$ and $\bar{p}(b)$ using the CDF of spot prices.

value of $\xi(b)$ for all occurrences of $\bar{p}(b)$ in the relevant H. Lower values are better for both $f(b)$ and $\xi(b)$. We focus on the right choice of history window size (for model training), which turns out to be dependent on both the market and the bid. None of the prior works (to our knowledge) have analyzed these idiosyncracies.

Setup. We vary instance types, markets, bids, and history window size H and show the assessment metrics f ($L(b)$ overestimation rate) and ξ ($\bar{p}(b)$ relative deviation) in Table 2. The 90-day spot price traces are chosen from Figure 2, with bid b picked from $\{0.5d, d, 2d, 5d, 10d\}$, where d is the corresponding on-demand price.[9] As a baseline approach, we also present the preceding metrics based on predictions of $L(b)$ and $\bar{p}(b)$ by using the empirical CDF of spot prices within H (updated dynamically). For this baseline, $\hat{L}(b) = H \cdot Prob(p_t \leq b)$ and $\hat{\bar{p}}(b) = E[p_t | p_t \leq b]$. This baseline represents approaches commonly considered in related work.

Validation of predictor efficacy. As shown in Table 2, under most (market, bid) pairs, the best (lowest) f and ξ are below 10%, which we consider a reasonable demonstration of the efficacy of our predictors. For certain spot markets (e.g., r3.xlarge-S), since the spot price is below even half

[9]This is based on the discussions from Sharma et al. [24] and Subramanya et al. [31] and our observations that high spot price values are usually around multiples of on-demand prices. Although the spot price stays below the on-demand price most of the time as shown in Figure 2, the tenants may still bid at a price that is higher than the on-demand price to reduce bid failures and achieve better performance.

the on-demand price most of the time (although still dynamic), both our proposed approach and the CDF-based approach achieve good performance with almost 0% overestimation. However, we do observe a small number of exceptions when performing our analysis on older spot price traces. For example, for c3.large-c in Table 1 of Wang et al. [38] wherein the analysis is based on an older dataset (2015), even if we use fifth percentile as prediction for $L(b)$, f and ξ are much higher than those from other markets. It might be better not to use this market temporarily until we observe better predictability.

The "right choice" for history window size. We find that (i) the best choice of H varies across markets and bids, implying the necessity for considering different markets separately when determining history window size, instead of blindly choosing a single window size for all markets, and (ii) changing bid values may not affect f and ξ much (e.g., r3.xlarge-S), possibly due to the fact that the $L(b)$ and $\bar{p}(b)$ do not vary much when spot price exceeds bid.

In addition, comparing with the corresponding results in our prior work [38] (dataset from 2015) where $H = 7$ outperforms other choices for most of the (market, bid) pairs, we find that (i) the best choice tends to be larger window sizes, $H = 14$ or $H = 29$ as in Table 2, which indicates that the temporal locality of the spot prices from the new dataset can be exploited better by using longer windows, and (ii) the optimal values of $f(b)$ and $\xi(b)$ are lower (and thus better) than before, implying better predictability. More comprehensive understanding of such behaviors would rely on longer-term monitoring and collection of spot prices from a variety of spot markets, which we leave to our future work.

More generally, we find that the evolution of $L(b)$ is often not smooth, and regression-based models (one natural alternative to our approach) may not work well. Again, accurate prediction of $L(b)$ may not be necessary as discussed before. Our choice of prediction with the fifth percentile of $L(b)$ tends to be conservative such that higher probability of service contiguity can be achieved. If the application can tolerate more frequent service interruptions, higher percentiles or more aggressive prediction techniques could be used.

Insights and implications. First, there exists short-term temporal locality in $L(b)$ and $\bar{p}(b)$ and the history window size can be leveraged to improve the quality of prediction. Second, modeling of these features should be conditioned on specific bids. Third, our approach outperforms CDF-based approaches by offering a smaller overestimation rate of $L(b)$ and less relative deviation from the actual $\bar{p}(b)$.

3.2 Feature 3: Simultaneous Revocations

Limitations of Prior Work. When placing bids for multiple instances, a tenant may wish to avoid picking spot markets with "high" likelihood of simultaneous revocations (the spot instances may be terminated simultaneously due to coincident bid failures). Prior works, such as Sharma et al. [24] and Subramanya et al. [31], suggest bidding across markets where there are no significant statistical correlations among the "raw" historical spot price traces (referred to as the correlation coefficient– or coef-based approach). However, such raw correlations might be misleading. To appreciate this, let us consider illustrative examples in Figure 5. We generate synthetic spot prices for two markets: (i) the cross correlation between the two markets' spot prices is low and (ii) the cross correlation is high. In (i), the tenant might be tempted to use both markets, whereas in (ii), it may not want to use the two markets together at all, if its decision is only based on the raw correlation. However, it is obvious that the bid failures from the two markets are highly correlated under Bid 1 but not under Bid 2. Therefore, it may be imprudent for the tenants to make decisions solely based on the raw correlations without considering the actual bids. More specifically, what the tenant really needs are measurements of simultaneous revocations conditioned on bids. Furthermore, the statistical correlation of bid failures across markets may not be very informative for

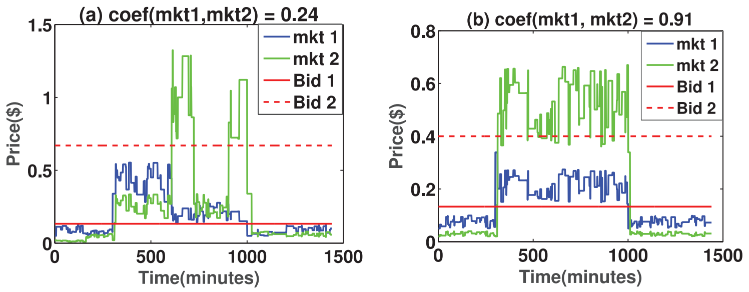

Fig. 5. Synthetic examples with simultaneous revocations related to bids. "coef" represents correlation coefficient.

decision making regarding bid placement. Instead, a tenant might find it more beneficial to learn the absolute time durations of simultaneous revocations (i.e., the total amount of time that a bid fails in two markets within the history window).

Model and Temporal Locality-Based Prediction. Based on these insights, a more informative metric that we propose is based on characterizing simultaneous revocations conditioned on prespecified bids. Under a given bid, we denote as A and B the sets of time periods when the bid fails in two spot markets under comparison,[10] respectively. Denote as $T(A)$ and $T(B)$ the corresponding lengths/sizes of A and B. $T(A \cap B)$ and $T(A \cup B)$ represent the time durations of coincident bid failures and total bid failures, respectively. $\frac{T(A\cap B)}{T(A\cup B)}$ reflects the probability that the bid fails in both markets when the bid already fails at one market. It is informative to look at both the durations of bid failures (e.g., $T(A)$) and $\frac{T(A\cap B)}{T(A\cup B)}$ when comparing markets. For example, even if $T(A)$ and/or $T(B)$ are relatively small compared to the history window size, if $\frac{T(A\cap B)}{T(A\cup B)}$ is high, which implies that markets (A,B) almost always fail together under the given bid, it may be better not to place bids in markets (A,B) simultaneously. However, even if both $\frac{T(A\cap B)}{T(A\cup B)}$ and $T(A \cap B)$ are small, we may use neither A nor B if $L(b)$ is also small in both markets. Tenants can combine such metrics with predicted $L(b)$ and $\bar{p}(b)$ to get a better understanding of simultaneous revocations for cost analysis.

Again, we find temporal locality useful for predicting simultaneous revocations. Similar to prediction of average price during the lifetime, we use the measured simultaneous revocations from a history window H as prediction for the near future. We vary the history window and explore the temporal locality. Our assessment metric is the rate of underestimation to the actual simultaneous revocations: the fraction of simultaneous revocation predictions that result in underestimation. From Table 3, we find that (i) the underestimation rate is low in general, implying good temporal locality for prediction; (ii) the history window size matters for different spot markets under different bids; and (iii) the underestimation rate could be 0, implying few bid failures in such markets under the particular bid. A tenant can use such information to decorrelate bid failures in his dynamic resource procurement.

Spatial Locality. Since EC2 regions and availability zones are geodistributed, some naturally appealing ideas for improving the (market, bid) selection are the following. Is there any spatial locality in spot prices? Are markets spread across regions/availability zones uncorrelated? Are such effects affected by bids? We show our measurements on simultaneous revocations across multiple regions, availability zones, and instance types in Table 4. We have several observations. First,

[10]Our analysis can be easily generalized to compare more than two markets.

Table 3. Underestimation Rate of Simultaneous Revocations

	H	3	7	14	21	28
r3.xl	0.5*d*	0.1018	0.1016	0.1105	0.1076	0.1205
	d	0.0703	0.0631	0.0693	0.0770	0.0858
	5*d*	0.0128	0.0139	0.0153	0.0174	0.0194
r3.2xl	0.5*d*	0.1915	0.1809	0.2033	0.1936	0.1749
	d	0.0132	0.0139	0.0152	0.0168	0.0188
	5*d*	0.0000	0.0000	0.0000	0.0000	0.0000

Note: We vary the history window H (days) and the bid price (from $0.5d$ to $5d$, where d is the on-demand price), and only show a subset of results here; the simultaneous revocation is computed for availability zones (us-east-1c, us-east-1e) for each instance type. The shaded cells represent the best H.

increasing the bid may decorrelate bid failures: even if spot prices of two markets always jump simultaneously, they do not usually reach the same high spot price. When the bid increases, one of the markets may experience less bid failures, whereas the other remains unaffected (possibly because the bid is not high enough). Second, increasing the bid may also increase the extent to which simultaneous revocation occurs. Based on the results in Table 3 of our prior work [38], as we increase the bid for c3.large, the total failure time $T(A \cup B)$ decreases quickly in markets (c,d) (highlighted), whereas $T(A \cap B)$ does not change much; thus, the fraction of time that concurrent bid failure occurs becomes larger. Third, since the properties of simultaneous revocations highly depend on markets and bids (and possibly also the history window size), simply comparing the raw statistical correlations of multiple markets' spot prices may not suffice and might even lead to faulty decision making. For example, the "coef" of the spot prices of c3.4xlarge (highlighted) in markets (b,w) can be considered as positively correlated, whereas it does not suffer from simultaneous revocations under the chosen bids. For c3.4xlarge in markets (c,e), 12.57% of the revocations are simultaneous revocations, whereas the "coef" is −0.0000. Therefore, it is crucially important to take into account the impact of bid values. Fourth, although the EC2 regions and availability zones are created for other types of failures (e.g., infrastructure failure), they unintentionally also amount to similar effects for spot bid failures. We choose 16 spot markets from two global regions, with multiple availability zones in each region, and show the likelihood of simultaneous revocations in Figure 6. Since the global regions are geographically isolated from each other while the availability zones within the same region are usually located close to each other, we expect that the likelihood of simultaneous revocations should be higher for spot markets in the same region than across regions, which is confirmed by our data analysis results (under both the traditional coef-based approach and our proposed approach).

Additionally, if we look at the simultaneous revocations across instance types but fix the availability zone (Table 5), we observe that in general the spot prices across different instance types are not highly correlated; even for the pair (c3.3xlarge, r4.8xlarge) that has the highest "coef," $\frac{T(A \cap B)}{T(A \cup B)}$ is still quite low under all bids. Such observation encourages the tenant to spread its bids across different instance types.

Heuristic for Efficiently Finding Marketplaces With Low Likelihood of Simultaneous Bid Failures. One possible way to exploit the observed spatial locality is through clustering of candidates formed by (market, bid) pairs. For example, similar to k-means clustering [9], we can interpret our model of simultaneous revocations ($\frac{T(A \cap B)}{T(A \cup B)}$) as the distance between two candidates. Then standard clustering algorithms can be applied to create clusters of candidates wherein

Table 4. Simultaneous Revocations Across Pairs of Markets

	Bid		$0.5d$				d				$5d$			
		coef	A	B	$A \cap B$	$\frac{A\cap B}{A\cup B}$	A	B	$A \cap B$	$\frac{A\cap B}{A\cup B}$	A	B	$A \cap B$	$\frac{A\cap B}{A\cup B}$
m3.2xlarge	c,e	0.0000	128310	976	976	0.0076	128310	352	352	0.0027	128310	246	246	0.0019
	c,b	0.0000	128310	5231	5231	0.0408	128310	2253	2253	0.0176	128310	1785	1785	0.0139
	c,w	0.0000	128310	2792	2792	0.0218	128310	1115	1115	0.0087	128310	1095	1095	0.0085
	e,b	-0.0088	5231	976	16	0.0026	2253	352	0	0.0000	1785	246	0	0.0000
	e,w	-0.0044	976	2792	49	0.0132	352	1115	0	0.0000	246	1095	0	0.0000
	b,w	0.0257	5231	2792	419	0.0551	2253	1115	41	0.0067	1785	1095	20	0.0033
c3.4xlarge	c,e	-0.0000	128491	16156	16156	0.1257	128491	11458	11458	0.0892	128491	11255	11255	0.0876
	c,b	0.0000	128491	0	0	0.0000	128491	0	0	0.0000	128491	0	0	0.0000
	c,w	0	128491	50	50	0.0004	128491	0	0	0.0000	128491	0	0	0.0000
	e,b	0.0624	0	16201	0	0.0000	0	11465	0	0.0000	0.0000	11262	0	0.0000
	e,w	-0.023	16201	50	9	0.0006	11465	0	0	0.0000	11262	0	0	0.0000
	b,w	0.4197	0	50	0	0.0000	0	0	0	0.0000	0	0	0	0.0000
r3.xlarge	c,e	0.0904	7629	2321	428	0.0449	5568	1330	271	0.0313	5102	983	203	0.0242
	c,b	-0.0187	7629	271	14	0.0018	5568	26	0	0	5102	0	0	0
	c,w	-0.0187	7629	109	7	0.0009	5568	3	0	0	5102	0	0	0
	e,b	-0.0039	271	2321	1	0.0004	26	1330	0	0	0	983	0	0
	e,w	0.0007	2321	369	8	0.003	1330	51	2	0.0008	983	47	0	0
	b,w	0.1788	271	369	38	0.0631	26	51	2	0.0066	0	47	0	0
r3.2xlarge	c,e	-0.005	6232	1804	325	0.0421	1531	1174	62	0.0086	688	909	0	0
	c,b	-0.0002	6232	1118	26	0.0035	1531	2	0	0	688	0	0	0
	c,w	0.0107	6232	17793	817	0.0352	1531	17647	235	0.0102	688	17647	133	0.0058
	e,b	-0.0099	1804	1118	10	0.0034	1174	2	0	0	909	0	0	0
	e,w	-0.0194	1804	17793	195	0.0101	1174	17647	87	0.0045	909	17647	41	0.0021
	b,w	0.2595	1118	17793	790	0.0436	2	17647	2	0.0001	0	17647	0	0

Note: Lowercase letters c, e, b, and w represent availability zones (markets) us-east-1c, us-east-1e, sa-east-1b, and sa-east-1b (Windows VM), respectively. The measurements are in minutes. The $T(.)$ operator is omitted for space. "Coef" is the coefficient of variation of two price traces. We set $\frac{A\cap B}{A\cup B} = 0$ when $A \cup B = 0$, which we interpret as no bid failures.

Table 5. Simultaneous Revocations Across Different Instance Types in Spot Market us-east-1e

	Bid	$0.5d$				d				$5d$			
	coef	A	B	$A \cap B$	$\frac{A\cap B}{A\cup B}$	A	B	$A \cap B$	$\frac{A\cap B}{A\cup B}$	A	B	$A \cap B$	$\frac{A\cap B}{A\cup B}$
m3.2xl, c3.4xl	−0.0145	976	16201	73	0.0043	352	11465	20	0.0016	246	11262	3	0.0002
m3.2xl, r3.xl	0.0079	976	2321	104	0.0326	352	1330	7	0.0031	246	983	0	0
m3.2xl, r3.2xl	−0.0037	976	1804	26	0.0094	352	1174	18	0.0085	246	909	0	0
c3.4xl, r3.xl	0.0536	16201	2321	646	0.0361	11465	1330	343	0.02	11262	983	262	0.0125
c3.4xl, r3.2xl	0.0508	16201	1804	653	0.0376	11465	1174	370	0.0218	11262	909	213	0.0126
c3.4xl, r4.8xl	0.2822	16201	6738	3034	0.1524	11465	5263	2467	0.1312	11262	5235	2467	0.0313
r3.xl, r3.2xl	0.0128	7629	6232	763	0.0583	5568	1531	34	0.0037	5102	688	24	0.0029

candidates in the same cluster have higher probability of simultaneous bid failures. A tenant can simply spread its choice of (market, bid) across clusters to reduce correlated bid failures.[11]

[11]Picking instances from multiple marketplaces (especially those geographically distributed) is of course more complex. For example, there may be concerns related to communication between instances over a WAN or issues of proximity to

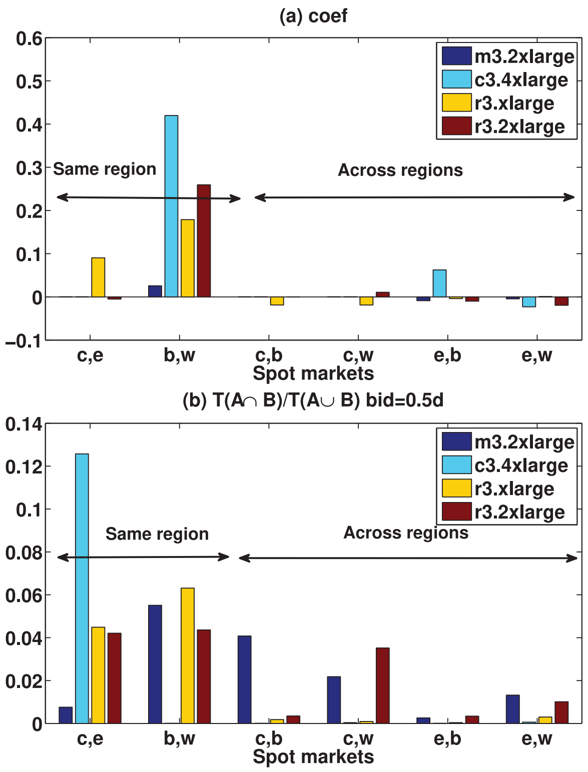

Fig. 6. Likelihood of simultaneous revocations measured using different metrics. (a) Using coef. (b) Using the proposed approach with bid= $0.5d$, where d is the on-demand price.

Insights and implications. First, temporal locality can be employed for predicting simultaneous revocations. Second, modeling and prediction of simultaneous revocations should take into account bids; only looking at the raw cross correlation may lead to faulty understanding/prediction of simultaneous bid failures. Third, there is spatial locality of spot bid failures across regions, availability zones, and different instance types, which can better help the tenant decorrelate bid failures.

3.3 Feature 4: Time to Start

The *time to start* is an important feature for a tenant's resource procurement. For example, during unexpected flash crowds, if the tenant wants to allocate new spot instance but finds that the spot bid status is still *pending* after a long waiting time, the application performance might be severely degraded. EC2's official documentation only provides rough estimates of maximum instance boot times (corresponding to the duration labeled "Initialization" in Figure 1), which range from 1 to 5 minutes [5]. However, recall from Figure 1 that there can be additional contributors to spot instance startup delay of two types: (i) a period when the bid is lower than the spot price and (ii) a period (even after the spot price has become lower than the bid) during which EC2 makes the tenant wait (allegedly) due to a lack of capacity at its end.

tenants that we cannot consider in our work [40]. Actual decision making would need to consider these against the issues that we do model here.

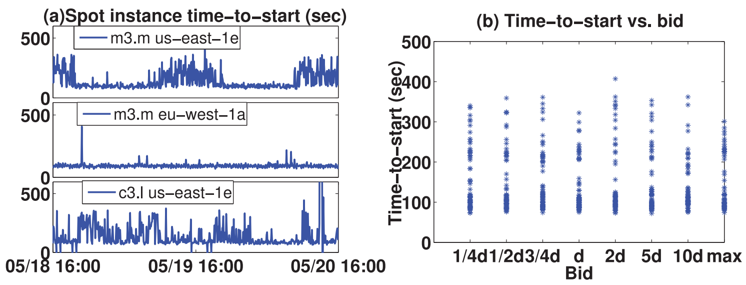

Fig. 7. (a) Spot instance time to start. (b) Spot instance time to start versus bids. We request one instance every 5 minutes across 2 days from 2016/05/18 16:00 to 2016/05/20 16:00 with bids uniformly chosen from $\{\frac{1}{4}d, \frac{1}{2}d, \frac{3}{4}d, d, 2d, 5d, 10d, max\}$, where d is the on-demand price and max is the maximum bid allowed. The time to start is set to 0 if the bid is less than the spot price. The largest two values in the bottom trace of (a) are 2,800+ seconds.

Limitations of Prior Work. To the best of our knowledge, this feature has been ignored by related work. The only research work that explores a limited aspect of this issue is that of Mao and Humphrey [17], which focuses only on instance boot-up times. For the spot instance, they observe no significant correlation between the spot price and the time to start; however, they do not explore/report the temporal locality and the predictive property of the time to start.

Model and Temporal Locality-Based Prediction. Of course, (i) depends on the bid placed by the tenant—a higher bid will exceed the spot price sooner than a lower bid. We find that for relatively high bids (greater than or equal to the price of the comparable on-demand instance), this type of delay can be ignored for all practical purposes. However, for lower bids (that certain cost-conscious tenants may prefer), we do not find any patterns that can be generalized readily for useful prediction.

To see the predictability of (ii), we choose instance types and several spot markets across different time zones and bid spot instances every 5 minutes over 2 days. The bids are uniformly chosen from $\{\frac{1}{4}d, \frac{1}{2}d, \frac{3}{4}d, d, 2d, 5d, 10d, max\}$, where d is the on-demand price and max is the maximum bid allowed. We report a subset of our results in Figure 7. In some cases (top and middle traces shown in Figure 7(a)), we find time-of-day–like behavior or small variance around a fixed value. This seems to depend very much on the market with no other obvious predictive indicators (more in a particular region or availability zone or instance type, etc.) In such marketplaces, a tenant may be able to exploit such predictability. In others, however, a tenant may have to resort to working with worst-case values. For example, in the bottom trace in Figure 7(a), the largest two samples are greater than 2,800 seconds.

We also explore the relationship between the time to start and the bid. Intuitively, higher bids should give the tenant higher priority for resource procurement, thus a shorter time to start, as EC2 might make more profit by serving higher bids first. However, we do not observe a statistically significant correlation between the time to start and different bid values (Figure 7(b)).

Insights and implications. It is important to model this feature, which has not been addressed in related work. However, we do not find generally useful evidence for temporal locality like we do for our features 1 through 3. Therefore, tenants may be forced to work conservatively with this feature (e.g., allowing for a large delay just to be safe based on high percentiles of previous observations of the time to start) or the worst-case observations.

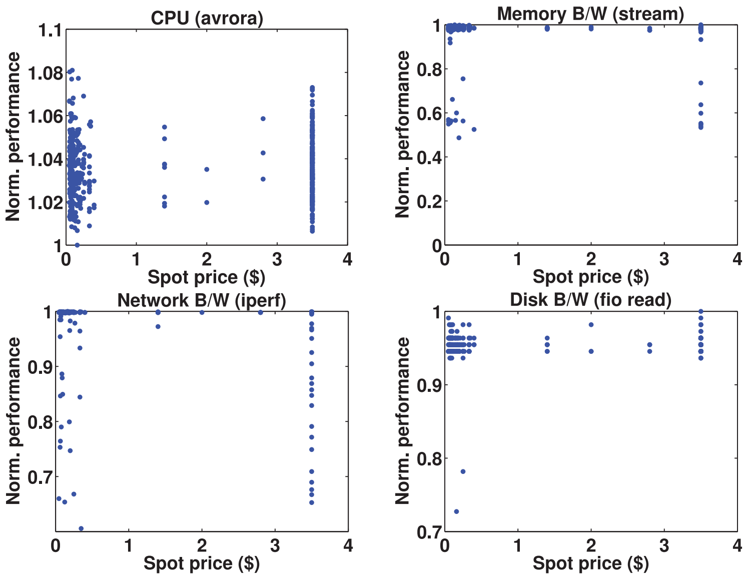

Fig. 8. Scatter plots of 24-hour microbenchmark performance measurements on a single r3.xlarge instance versus the spot price from us-east-1d. Avrora is a CPU-intensive benchmark [3]; STREAM [30], iPerf [11], and fio [11] are commonly used benchmarks that measure the bandwidths of memory, network, and disk I/O. The performance measurements are normalized with respect to the best performance samples in the experiments. The spot price varies from $\frac{1}{7}$ to 10 times the on-demand price.

3.4 Do Effective Capacity Measurements Provide Predictive Power?

EC2 claims that the spot price is set based on its supply/demand relationship [4]. Therefore, a natural thought is this: can we further improve the predictability of spot prices if we can somehow infer the current resource demand versus the capacity status in the spot pool? For example, when there are too many tenants' spot requests and the spot pool becomes "congested," it is highly possible that the "effective" capacity, as opposed to the advertised capacity, perceived by the tenant could be lower or have higher variations (reflected by degraded application performance) than when the spot pool has enough unused capacity. In particular, we have the following hypotheses. First, is there any predictive power in effective capacity versus the spot price? Second, is there any predictive power in effective capacity versus sudden revocations? Such relationships, if observed, can be exploited by the tenants to improve their prediction of spot prices (or of features 1 through 3).

To test our hypotheses, we conduct experiments on several EC2 spot markets wherein we record effective capacity variations based on application performance measurements from commonly used microbenchmarks. Due to space limitations, we only show a subset of our measurement results in Figure 8. As illustrated by the scatter plots of normalized performance versus the spot price, we do not observe enough predictive power in dynamic effective capacity (reflected by the varying performance measurements) for spot prices. We further infer that the evolvement of spot prices may not be reflected by local congestion signals, or that even when there is not enough capacity (implied by the peak price periods), EC2 would rather kick off spot instances by raising the spot prices instead of sacrificing capacity/performance.

We do not see a significant performance/capacity difference across on-demand and spot instances with the same resource configuration. Our interpretation is that EC2 might have provided the same guarantee of performance and capacity isolations for both on-demand and spot instances. However, when the on-demand resource pool lacks capacity, EC2 might revoke spot instances to make room for the more profitable on-demand instances. We leave more comprehensive and extensive comparison studies of on-demand versus the spot instance to our future work.

4 CASE STUDIES

The goal of our case studies is not to devise fundamentally novel resource procurement (i.e., "control") algorithms. Recall from Section 2 that such control algorithms have received a lot of attention recently for a variety of workload types. Instead, we are interested in evaluating the cost/performance improvements that our modeling and prediction techniques can offer to this existing body of work. Given this, we adapt control algorithms in related work to use our modeling and prediction for two real-world applications: (i) an in-memory Memcached-based data store and (ii) a homegrown synthetic batch processing workload. Both of these workloads possess the following property, making the use of spot instances suitable for them: the failure of an instance may only degrade their performance but does not affect their correctness. In both cases, our control formulations attempt to minimize operational costs while maintaining specified application-level performance guarantees using a combination of spot and on-demand instances.

4.1 Case Study I: In-Memory Data Store

We consider the problem of cost-effective operation for a Memcached-based (a popular in-memory key-value data store [20]) caching tier within a larger data storage application. As a typical mode of operation for such an application, we assume that the entire working set needs to be kept in memory for satisfactory performance. When a spot instance assigned to the caching tier is lost due to a bid failure, the back-end database serves misses. When servicing misses, the requested data be inserted into caching tier and stale data is evicted based on an LRU policy when there is not enough memory capacity.

Control Design. We formulate an online optimization problem that exploits predictability within the workload (request arrival rate $\hat{\lambda}_t$ and working set size $\hat{M}_t$) and spot price features ($L(b)$ and $\bar{p}(b)$)) to determine (i) how many and which on-demand and spot instances to procure/deallocate and (ii) how to partition the overall working set (itself dynamic) among on-demand and spot instances. Implicit in this decision making are the markets from which to pick spot instances and the bids to place. Alternative approaches, such as data replication across multiple markets [41], are complementary to our work.

We view on-demand instances as special spot markets with $L(b) = \infty$ and $\bar{p}(b)$ equal to the corresponding on-demand price. This allows us to conveniently represent all different markets in a unified way. Denote as $s \in S$ a spot market and as b a bid picked from B_s (a set of preselected bid values depending on the market s). Denote as N_t^{sb} and $\tilde{N}_t^{sb}$ the existing number of instances and extra instances to procure/deallocate from market s under bid b at the beginning of time slot t, respectively. Denote as x_t^{sb} the fraction of working set kept in market s under bid b. Denote as m^s and c^s the amount of RAM and number of vCPUs for instance in market s. We present our optimization formulation as follows:

$$\min_{\tilde{N}_t^{sb}, x_t^{sb}} \sum_{s \in S} \sum_{b \in B_s} [\hat{\bar{p}}_t^s(b)(N_t^{sb} + \tilde{N}_t^{sb})T + \frac{\alpha x_t^{sb} \hat{M}_t}{\hat{L}^s(b)} T + \beta \max\{0, -\tilde{N}_t^{sb}\}]$$

$$\text{s.t.} \sum_{s \in S} \sum_{b \in B_s} x_t^{sb} = 1$$

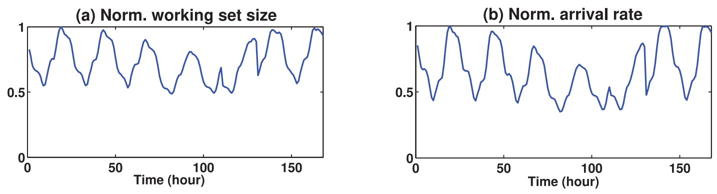

Fig. 9. Normalized working set size and request arrival rates from the Wikipedia access trace [35]. Normalization is done against the maximum value in each trace.

$$\sum_{s \in S'} x_t^{sb} \geq \xi, S' = \{\text{OD}\}$$
$$x_t^{sb} \hat{M}_t \leq (N_t^{sb} + \tilde{N}_t^{sb}) m^s, \forall s \in S, b \in B_s$$
$$\phi(\hat{\lambda}_t x_t^{sb}, (N_t^{sb} + \tilde{N}_t^{sb}) c^s) \leq l^{TGT} \forall s \in S, b \in B_s,$$

where {OD} is the set of on-demand instance types. In the objective function, the first term represents the resource costs that depend on the predicted average spot price $\hat{\bar{p}}_t^s(b)$ during the optimization window T; the second term is the bid failure penalty, a decreasing function of the predicted $L(b)$ in a spot market, implying a certain "loss rate"; the last term reflects the resource deallocation penalty that depresses performance oscillation due to overly frequent workload rebalancing among markets. α, β reflect the weights of each term in the objective. The first constraint leads to a full partition of the working set, and the second constraint forces that at least ξ percent of the working set should be placed on on-demand instances to ensure service availability if all spot markets fail. We use the third constraint to guarantee enough RAM capacity in each market to hold its portion of the working set. Finally, the last constraint enforces an application-specific latency target l^{TGT} where $\phi(\hat{\lambda}_t x_t^{sb}, (N_t^{sb} + \tilde{N}_t^{sb}) c^s)$ is a function capturing the relationship between latency, arrival rates, and the number of vCPUs. In addition, $\phi(.)$ can be obtained by various techniques (e.g., regression using empirical measurements).

Implementation. We implement a two-timescale hierarchical resource procuring framework. At a coarser timescale (e.g., hours), an online optimizer updates the predictions of $L(b)$ and $\bar{p}(b)$ with history window H identified to be appropriate for the chosen markets in Section 3 and solves our optimization problem. At a finer timescale (e.g., every 10 minutes), each existing VM instance collects performance statistics and makes local control decisions to continue meeting performance targets in the face of prediction errors, such as unpredictable flash crowds, under/overprovisioning of VM instances by the online optimizer, and system noise (e.g., VMs do not get enough capacity due to the cloud's resource overbooking or statistical multiplexing). In the case of bid failures, we start new on-demand instances with the same capacity as the failed instances and redirect the requests accordingly. More details about our control design and implementation (including an additional aspect about how we coordinate reactive fine-grain vertical scaling and coarse-grain VM scaling) can be found in our prior work [37].

Experiment Design. We assume that our tenant uses a single spot instance type across two availability zones with bids of $b = d, 5d$ (denoted as b_1, b_2, respectively) in each zone, where d is the on-demand price. We evaluate our approach using a variety of 90-day spot price traces taken from Wang et al. [38]. We generate our workload by scaling the dynamic arrival rates λ_t and working set size M_t from the Wikipedia access trace [35] (Figure 9). We find that both λ_t and M_t can be well captured via AR(2) models with R-squared equal to 0.99 and 0.94, respectively.

Table 6. Cost Savings of Different Strategies Compared Against

	m3.l		m3.xl		c3.l		c3.2xl	
	e	w	e	w	e	w	e	w
Cost Savings (%)								
BL-CDF	67	72	51	59	62	68	42	46
PROP	58	72	45	59	61	68	42	26
Number of Bid Failures								
BL-CDF	23	0	28	1	3	0	9	49
PROP	13	0	10	0	4	0	2	6

Note: Lowercase letters e and w represent experiments using all spot price traces from us-east and us-west, respectively.

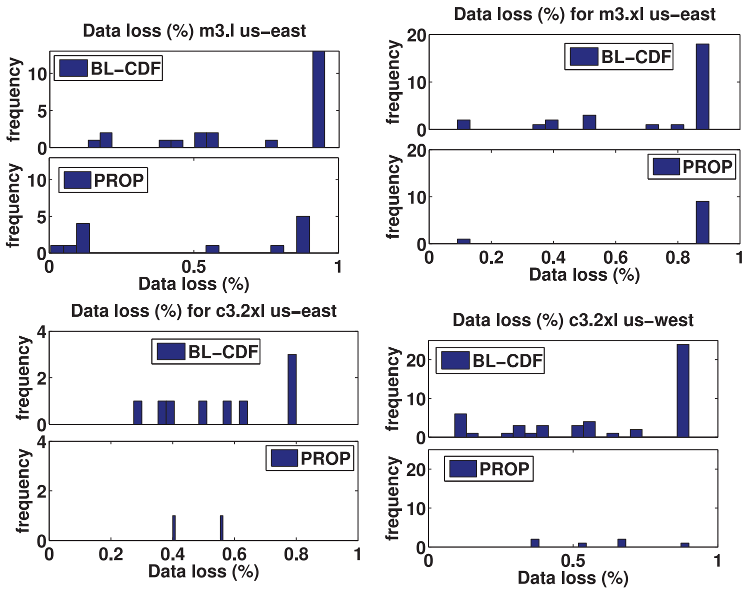

Fig. 10. Histograms of data loss (normalized against actual working set size) during bid failures. Cases not shown here are those without significant difference.

Baselines. Denote as "PROP" our proposed online approach. We further create two baselines to compare against: (i) "BL-OD," in which all data are stored on on-demand instances (no bidding), and (ii) "BL-CDF," predicting $L(b)$ and $\bar{p}(b)$ via the CDF-based approach (see Section 3). In all baselines, the workload partition on spot instances across markets under different bids is determined by the same online optimizer.

Trace-Driven Simulation. We conduct experiments with a variety of spot price traces and show the cost savings (against BL-OD) and performance (reflected by data loss due to bid failures) under different strategies in Table 6 and Figure 10. First, we observe that in the region us-west where the spot prices are quite low and bid failures are rare events, PROP and BL-CDF have almost the same cost savings (up to 72%) and same number of failures, which is because the spot prices in these cases have good temporal locality. Second, we find that in region us-east where the spot

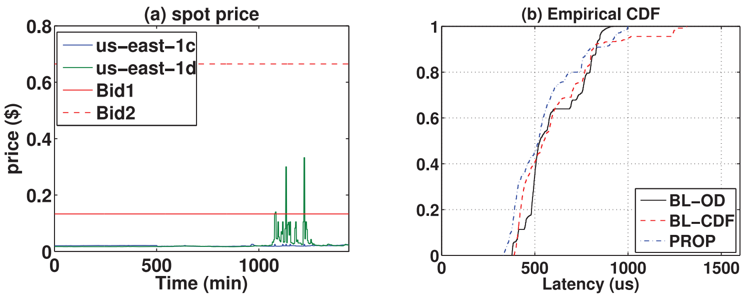

Fig. 11. (a) 24-hour spot price of m3.large from two markets with Bid1 = d and Bid2 = $5d$. (b) The CDF of latencies.

prices fluctuate a lot and bid failures occur quite often, PROP offers less but still comparable cost savings compared to BL-CDF. Recall our discussion in Section 2.2 and Figure 3 that the CDF-based approach makes the tenant tempted to use spot instances more aggressively even if there are short-lived but frequent spikes in spot prices, thereby achieving lower costs than PROP. However, this comes at the expenses of many more bid failures. Third, not only does PROP lead to less bid failures, it also has much less data loss during each bid failure than BL-CDF. As Figure 10 shows, BL-CDF has more frequent 90% data loss (out of working set) in all four cases than PROP. This is possibly because PROP's prediction of the key features are closer to the actual values, which further demonstrates better temporal locality than the CDF-based approach.

Experiments With Prototype on EC2. To explore operation in the real world (especially application performance that may be affected by factors that our simulations may not capture), we conduct experiments with Memcached on our prototype system on EC2 using a 24-hour subset from the workload trace in Figure 9 and 24-hour spot price traces taken from m3.large in us-east (Figure 11(a)). Three bid failures occur under Bid 1 in us-east-1d. For BL-CDF, the third failure is avoided after it updates prediction of $L(b)$ and $\bar{p}(b)$. PROP does not incur any bid failures.

We show the application performance under different strategies in Figure 11(b). We find that the three strategies offer comparable performance up to 90th percentile latencies. Below the 90th percentile, sometimes BL-CDF and PROP could outperform BL-OD, as they could use more spot instances to serve the requests without incurring higher costs than BL-OD. Beyond the 90th percentile, BL-OD offers the best performance possibly because it does not spread the working set among multiple markets, which depresses the latency oscillation due to working set repartition. Meanwhile, Prop is able to beat BL-CDF and gets closer to BL-OD (but with much less cost), whereas BL-CDF has worse performance due to bid failures.

Key Insights. PROP offers less but still comparable cost savings with BL-CDF. However, PROP is able to achieve much better performance in terms of both less bid failures and less data loss during bid failures, particularly in markets with higher variations. Since the in-memory data store is usually considered a performance-sensitive/latency-critical application, the tenant may desire "always-on"/"service contiguity" more than cost savings and prefer PROP to BL-CDF.

4.2 Case Study II: Batch Processing

We leverage a fault-tolerance mechanism, replicating computation, to optimize the cost versus performance trade-off for a tenant running delay-tolerant batch jobs.

Algorithm Design. When a batch job arrives, we put a "primary copy" of it on a spot instance and a "backup copy" on an on-demand instance. Jobs on the spot instances are guaranteed to have enough resource capacity (regular capacity) for their normal execution, whereas the on-demand instance capacities are oversubscribed so that the backup copies only get a small portion out of their own regular capacities. Therefore, primary copies are expected to finish sooner than backup copies. Since spot instances are generally much cheaper than on-demand instances and the on-demand capacities are oversubscribed, the expected costs would be much lower than if we run the jobs only on on-demand instances and with oversubscription. If the spot instance is not revoked by the time when the primary copy finishes, we will terminate the backup copy to save costs and make more room for new jobs; otherwise, we will boost the performance of the backup copies whose primary copies have failed by allowing them to use more computing resources temporarily.

By default, we place one primary copy per vCPU in the primary pool (spot) but at most four backup copies per vCPU in the backup pool (on-demand). As an initial step toward a more comprehensive solution, we apply a simple yet effective strategy for *primary copy placement*. Upon the arrival of a new job j, find all valid (market, bid) pairs that satisfy $\hat{L}(b) \geq \hat{l}_j$ as candidates, where $\hat{l}_j$ is the predicted execution time of job j if given regular capacity. Then randomly choose a candidate with $\bar{p}(b)$ less than or equal to the n-th smallest $\bar{p}(b)$ to execute the primary copy. For *backup copy placement*, we compute an index for each instance in the backup pool, which is a function of the probability of simultaneous revocations of each existing job on that instance versus the new job. Then we choose the instance with the lowest index value, which indicates less probability of simultaneous revocations and probably offers more capacity headroom for the backup copy of the new job for performance boosting when unexpected bid failure occurs. If the lowest index exceeds a certain threshold, a new on-demand instance will be allocated.

To further improve performance when bid failure occurs, we boost the CPU capacity allocated to a backup copy once its corresponding primary copy fails. More specifically, we use "Linux Cgroups" inside a backup instance to change the "CPU share" of the backup copies that share the same vCPU. The CPU share reflects the relative percentage of one core that is allocated to a process. We set the initial CPU share of each backup copy to 256 so that all backup copies have the same CPU capacity if their corresponding primary copies do not fail. Once a primary copy fails, we boost the CPU share of its backup copy to 1024 so that this backup copy could use as much as four times the CPU capacity of other backup copies that are not boosted. It is still possible that more than one backup copies on the same core need to boost performance at the same time; however, the probability of such events can be reduced greatly by considering our proposed simultaneous revocations in the backup placement strategy described earlier.

Experimental Setup. We use m3.xlarge (four vCPUs) across us-east-1c (s_1) and us-east-1d (s_2) with bids $b_1 = d, b_2 = 5d$, where d is the on-demand price. We mark four markets: s_1b_1, s_1b_2, s_2b_1, and s_2b_2. The 3-month price traces are chosen from Wang et al. [38]. We use an exponential distribution to generate the interarrival time of jobs, with $\lambda = 10$ jobs per hour. The lengths of the jobs are uniformly selected from the range [30, 300] (minutes). The jobs are CPU intensive, with little memory I/O and no network traffic.

Our baselines. We create BL-OD, which does not use spot instances and runs one job per vCPU on on-demand instances without replication. To compute the index for a pair of jobs (existing vs. new), we use the summation of probabilities of simultaneous revocations, which is calculated via $\frac{T(A \cap B)}{T(A \cup B)}$ for our approach PROP, and via "coefficient of variation" for another baseline BL-COEF (mimicking the approach used by prior work [24, 31]).

Trace-Driven Simulation. We conduct trace-driven simulation using the 3-month spot price traces to demonstrate the long-term benefit of our proposed approach. Figure 12 shows the cost breakdown and CDF of job execution time under different strategies.

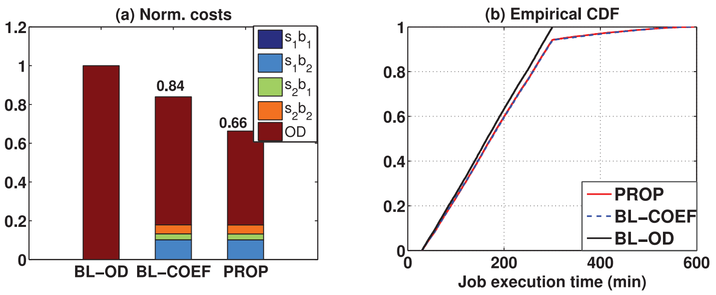

Fig. 12. Cost breakdown (a) and CDF of job execution time (b). The CDFs of BL-CDF and PROP overlap.

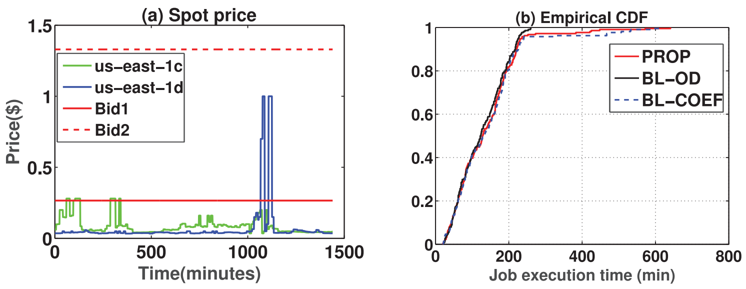

Fig. 13. (a) 24-hour spot price of m3.xlarge from two markets with Bid1 = d and Bid2 = $5d$ (d, on-demand price). (b) The CDF of job execution time under different strategies.

We have several observations. First, PROP saves as much as 33.7% and 17.7% costs compared to BL-OD and BL-COEF (Figure 12(a)), respectively. This is because BL-OD only uses on-demand instances that are expensive and BL-COEF does not compute probability of simultaneous revocations conditioned on bids, thereby determining the backup placement conservatively and using more on-demand instances. For example, if two jobs' primary copies are in the same spot market but under different bids, BL-COEF would consider them to fail simultaneously with a probability of 1, which may not be true if the spot price falls between the two bids and only one of the jobs fails. Second, from Figure 12(b), we find that BL-OD offers the best performance since there is no bid failure. BL-COEF and PROP has almost the same CDF of job execution time, implying that PROP captures the simultaneous revocations well and provides similar capacity headroom for performance boosting when failures occur compared to the conservative BL-COEF.

Experiments With Prototype on EC2. To demonstrate the efficacy of our proposed approach in a real-world setting, we deploy a prototype system on EC2 with 24-hour spot price traces of m3.xlarge (Figure 13(a)) and conduct real-time experiments. A bid failure occurs at around 1,100th minute under Bid 1. Since the predicted $L(b)$ is smaller for us-east-1c, two markets, s_1b_1 and s_1b_2 (both in us-east-1c), are excluded by our algorithm in this experiment.

We show the performance under different strategies in Figure 13(b). We observe that the relative performance of all strategies are similar from trace-driven simulation to the real-world experiment. However, we notice that the performance of PROP is better (although not much) than BL-COEF for jobs that are affected by bid failures. This is possibly because BL-COEF is not only conservative but may also be misleading: even if the "coef" is low for two spot markets, the probability of simultaneous revocation could become high depending on the specific bids.

Key Insights. For batch jobs that can tolerate bid failure–induced delay, our approach can save more costs by applying simultaneous revocation features while still providing comparable (if not better than) performance with the traditional approach, which neglects the impact of bids.

5 CONCLUSION

In this article, we identified four key features of spot instance operation that a tenant should model. Using extensive empirical evaluation based on both historical and current spot instance data, we showed shortcomings in the state of the art that our model and prediction overcome. We further demonstrated the efficacy of our proposed approaches using two real-world case studies via both trace-driven simulation and system prototyping on EC2.

REFERENCES

[1] O. A. Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir. 2011. Deconstructing Amazon EC2 spot instance pricing. In *Proceedings of CloudCom'11.*

[2] A. Andrzejak, D. Kondo, and S. Yi. 2010. Decision model for cloud computing under SLA constraints. In *Proceedings of MASCOTS '10.*

[3] Avrora. 2016. Home Page. Retrieved February 19, 2018, from http://users.cecs.anu.edu.au/~steveb/research/research-infrastructure/dacapo-benchmarks.

[4] AWS. 2015. Building Price-Aware Applications Using EC2 Spot Instances,. Retrieved February 19, 2018, from https://aws.amazon.com/blogs/aws/category/ec2-spot-instances/.

[5] AWS. 2016. EC2 Boot Time. Available at http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ComponentsAMIs.html.

[6] AWS. 2016. Amazon EC2 Spot Instances. Retrieved February 19, 2018, from http://aws.amazon.com/ec2/spot-instances/.

[7] Y. Gong, B. He, and A. C. Zhou. 2015. Monetary cost optimizations for MPI-based HPC applications on Amazon clouds: Checkpoints and replicated execution. In *Proceedings of SC'15.*

[8] W. Guo, K. Chen, Y. Wu, and W. Zheng. 2015. Bidding for highly available services with low price in spot instance market. In *Proceedings of HPDC'15.*

[9] J. A. Hartigan and M. A. Wong. 1979. Algorithm as 136: A *k*-means clustering algorithm. *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 28, 1, 100–108.

[10] J. He, Y. Wen, J. Huang, and D. Wu. 2014. On the cost-QoE tradeoff for cloud-based video streaming under Amazon EC2's pricing models. *IEEE Transactions on Circuits and Systems for Video Technology* 24, 4, 669–680.

[11] iPerf. 2016. Home Page. Retrieved February 19, 2018, from https://iperf.fr/iperf-download.php.

[12] I. Jangjaimon and N. F. Tzeng. 2015. Effective cost reduction for elastic clouds under spot instance pricing through adaptive checkpointing. *IEEE Transactions on Computers* 64, 2, 396–409.

[13] B. Javadi, R. Thulasiramy, and R. Buyya. 2011. Statistical modeling of spot instance prices in public cloud environments. In *Proceedings of UCC'11.*

[14] D. Jung, J. Lim, H. Yu, J. Gil, and E. Lee. 2014. A workflow scheduling technique for task distribution in spot instance-based cloud. In *Ubiquitous Information Technologies and Applications.* Lecture Notes in Electrical Engineering, Vol. 280. Springer, 409–416.

[15] G. Kesidis, B. Urgaonkar, N. Nasiriani, and C. Wang. 2016. Neutrality in future public clouds: Implications and challenges. In *Proceedings of HotCloud'16.*

[16] S. Khatua and N. Mukherjee. 2013. Application-centric resource provisioning for Amazon EC2 spot instances. In *Proceedings of Euro-Par'13.*

[17] M. Mao and M. Humphrey. 2012. A performance study on the VM startup time in the cloud. In *Proceedings of IEEE CLOUD'12.*

[18] A. Marathe, R. Harris, D. Lowenthal, B. R. de Supinski, B. Rountree, and M. Schulz. 2014. Exploiting redundancy for cost-effective, time-constrained execution of HPC applications on Amazon EC2. In *Proceedings of HPDC'14.*

[19] M. Mattess, C. Vecchiola, and R. Buyya. 2010. Managing peak loads by leasing cloud infrastructure services from a spot market. In *Proceedings of HPCC'10.*

[20] Memcached. 2016. Home Page. Retrieved February 19, 2018, from https://memcached.org/.

[21] I. Menache, O. Shamir, and N. Jain. 2014. On-demand, spot, or both: Dynamic resource allocation for executing batch jobs in the cloud. In *Proceedings of ICAC'14.*

[22] Github. 2016. Spot Characterization Code and Data. Available at https://github.com/patiner/spot_characterization.git.

[23] P. Sharma, D. Irwin, and P. Shenoy. 2016. How not to bid the cloud. In *Proceedings of HotCloud'16.*

[24] P. Sharma, S. Lee, T. Guo, D. Irwin, and P. Shenoy. 2015. SpotCheck: Designing a derivative IaaS cloud on the spot market. In *Proceedings of EuroSys'15*.

[25] V. K. Singh and K. Dutta. 2015. Dynamic price prediction for Amazon spot instances. In *Proceedings of HICSS'15*.

[26] Y. Song, M. Zafer, and K. Lee. 2012. Optimal bidding in spot instance market. In *Proceedings of INFOCOM'12*.

[27] AWS. 2015. Featured Customer Testimonials. Retrieved February 19, 2018, from https://aws.amazon.com/ec2/spot/testimonials/.

[28] AWS. 2016. How Spot Fleet Works. Retrieved February 19, 2018, from http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/spot-fleet.html.

[29] AWS. 2016. Spot Request Status. Retrieved February 19, 2018, from http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/spot-bid-status.html.

[30] STREAM. 2016. STREAM: Sustainable Memory Bandwidth in High Performance Computers. Retrieved February 19, 2018, from http://www.cs.virginia.edu/stream/.

[31] S. Subramanya, T. Guo, P. Sharma, D. Irwin, and P. Shenoy. 2015. SpotOn: A batch computing service for the spot market. In *Proceedings of SoCC'15*.

[32] S. Subramanya, A. Rizk, and D. Irwin. 2016. Cloud spot markets are not sustainable: The case for transient guarantees. In *Proceedings of HotCloud'16*.

[33] M. Taifi, J. Y. Shi, and A. Khreishah. 2011. SpotMPI: A framework for auction-based HPC computing using Amazon spot instances. In *Proceedings of ICA3PP'11*.

[34] S. Tang, J. Yuan, and X.-Y. Li. 2012. Towards optimal bidding strategy for Amazon EC2 cloud spot instance. In *Proceedings of IEEE CLOUD'12*.

[35] G. Urdaneta, G. Pierre, and M. Van Steen. 2009. Wikipedia workload analysis for decentralized hosting. *Elsevier Computer Networks* 53, 11, 1830–1845.

[36] R. M. Wallace, V. Turchenko, M. Sheikhalishahi, I. Turchenko, V. Shults, J. L. Vazquez-Poletti, and L. Grandinetti. 2013. Applications of neural-based spot market prediction for cloud computing. In *Proceedings of IDAACS'13*.

[37] C. Wang, A. Gupta, and B. Urgaonkar. 2016. Fine-grained resource scaling in a public cloud: A tenant's perspective. In *Proceedings of IEEE CLOUD'16*.

[38] C. Wang, Q. Liang, and B. Urgaonkar. 2017. An empirical analysis of Amazon EC2 spot instance features affecting cost-effective resource procurement. In *Proceedings of ICPE'17*.

[39] C. Wang, B. Urganokar, A. Gupta, L. Chen, R. Birke, and G. Kesidis. 2016. Effective capacity modulation as an explicit control knob for public cloud profitability. In *Proceedings of ICAC'16*.

[40] Z. Wu, M. Butkiewicz, D. Perkins, E. Katz-Bassett, and H. V. Madhyastha. 2013. SPANStore: Cost-effective geo-replicated storage spanning multiple cloud services. In *Proceedings of SOSP'13*.

[41] Z. Xu, C. Stewart, N. Deng, and X. Wang. 2016. Blending on-demand and spot instances to lower costs for in-memory storage. In *Proceedings of INFOCOM'16*.

[42] M. Zafer, Y. Song, and K.-W. Lee. 2012. Optimal bids for spot VMS in a cloud for deadline constrained jobs. In *Proceedings of Cloud'12*.

[43] H. Zhao, M. Pan, X. Liu, X. Li, and Y. Fang. 2012. Optimal resource rental planning for elastic applications in cloud market. In *Proceedings of IPDPS'12*.

[44] L. Zheng, C. Joe-Wong, C. W. Tan, M. Chiang, and X. Wang. 2015. How to bid the cloud. In *Proceedings of SIGCOMM'15*.

Received June 2017; revised October 2017; accepted November 2017