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

Cheng Wang, Qianlin Liang, Bhuvan Urgaonkar
Pennsylvania State University
{cxw967,qxl5068,bhuvan}@cse.psu.edu

ABSTRACT

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 advertized resource capacities. To use spot instances effectively, a tenant must carefully weigh the lower costs of these instances against their poorer availability. Towards 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 both historical and current spot instance data, we show shortcomings in the state-of-the-art modeling of these features that we overcome. Our analysis reveals many novel properties of spot instance operation some of which offer predictive value while 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.

Keywords

Spot instance features, resource procurement

1. INTRODUCTION

Amazon EC2 has been offering *spot instances* [5] since 2009 and a large segment of its "tenant" workloads has come to embrace these [22]. The appeal of spot instances lies in their low prices - 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. In order 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 that these algorithms rely upon. Overcoming these shortcomings is the goal of this study.

Research Contributions:

- We consider four key features that we think a tenant should model: (i) lifetime of an instance, (ii) average spot price during 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 first-hand 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 which 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 demon-

[1]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].

[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.

ICPE'17, April 22-26, 2017, L'Aquila, Italy

DOI: http://dx.doi.org/10.1145/3030207.3030210

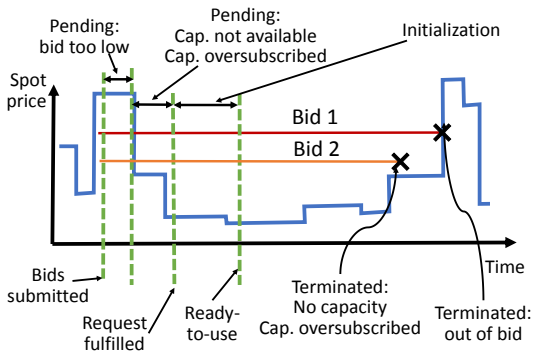

Figure 1: Illustration of spot instance operation using two hypothetical bids (Bid 1 and Bid 2).

strate 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. Amongst our salient findings are: (i) Whereas raw spot prices are best considered non-stationary, most of our features exhibit short-term *temporal locality* (borrowing a phrase from caching and memory management) that can be leveraged for prediction. (ii) 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. (iii) We do not find significant statistical correlation between effective capacity measurements and spot price, indicating the evolution of spot price likely depends on a spatially coarse (e.g., data center or availability zone wide) load metric.

Outline: In Section 2, we discuss the lifecycle 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 in [24]. 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 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 Fig. 1, both the 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 till 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 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 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: Prior work on spot price modeling/prediction ranges from relatively simple ones (in terms of the amount of historical data employed as well as the computational complexity of the model), e.g., auto-regressive models [1, 30, 36] or empirically measured cumulative distributions of key parameters [6, 10, 15, 19, 20, 26, 27], to more complex ones, e.g., employing Markovian models [14, 17, 21, 35], including adapting model parameters over time. Simple regressive models might fail to provide insights on how the spot price might evolve in the long run since it could change at a minute's granularity. 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. To our knowledge, one exception to the above models is [21] wherein the "sojourn time" of a spot instance procured via a particular bid is modeled and predicted 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 [26,27]. 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. The only one exception

[3] All our code and data is available at [18]. Our code includes both trace analysis and control implementation of tenant-side instance procurement on EC2.

[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. E.g., the tenant may tell EC2 to launch a set of spot instance only if it can launch them all (a.k.a. launch group); the bid status would be "launch-group-constraints" if EC2 cannot launch all at the same time.

is [13]. 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, e.g., delay-tolerance batch jobs [6, 12, 14, 17, 21, 26, 28, 35], video streaming [8], data caching [34]. On the one hand, 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, e.g., checkpointing, live-migration, 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 [23] 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 that we think a tenant should focus on. 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 cross-cutting our models is to express our features as quantities that are *conditioned on specific bids* chosen out of a small set of pre-selected 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 vs. different availability zones or geographic regions?
- *Capacity Measurements:* Do effective capacity measurements [32] have any predictive value? That is, do such measurements serve as faithful indicators of changes in spot prices or availability capacity in the concerned marketplace?

Using lessons learnt 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 90-day long spot price timeseries for 16 marketplaces for which we present such evaluation in this paper. We also plot the moving average and standard deviation of spot prices over a two-day sliding window in the same figures. These measurements show high variations especially in spot markets with frequent spikes, e.g., c3.2xlarge in us-west-1a and suggest that *raw* spot prices are best considered (highly) non-stationarity making questionable the efficacy of approaches in existing work (e.g., [1, 30, 36]) that rely on modeling them directly (i.e., implicitly assuming 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.

3.1 Features 1 & 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 for long enough to serve its needs. That is, 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 till 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 vs. performance trade-off. On the other hand, 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 non-bid 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 non-bid 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 [11]), such terminations may not be non-negligible. 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 with a bid that equals to 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

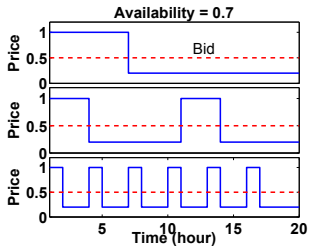

Figure 3: Pitfalls of CDF-based approaches used in prior work.

[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. E.g., 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].

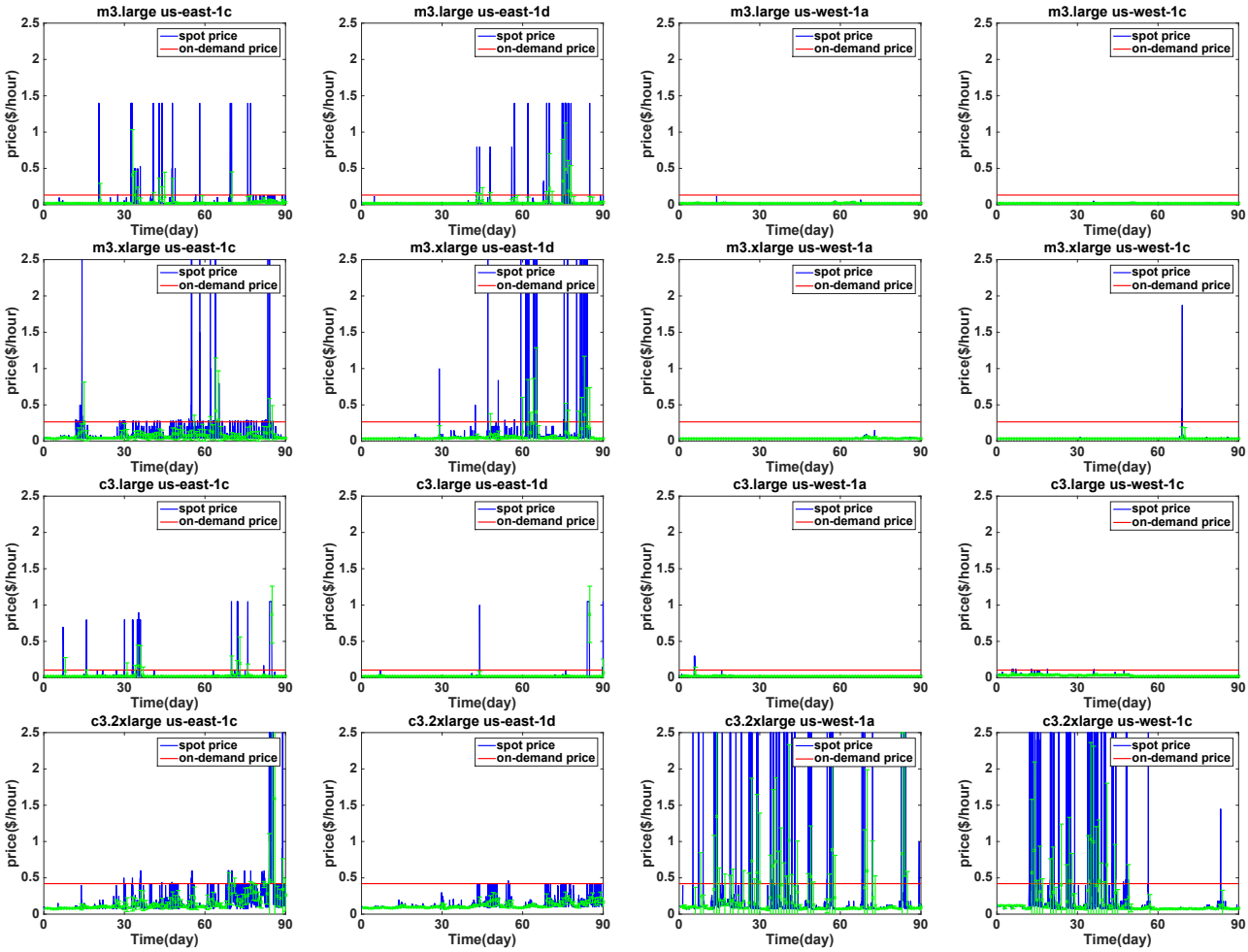

Figure 2: Sample spot price timeseries collected during the 90-day period (2015/07/08 to 2015/10/06) and are chosen due to their very different properties. The green "*" and error bars represent moving average and standard deviations over 2-day interval.

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. On the other hand, 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 these features, we find that effective models are offered by viewing them as random variables informed by empirically measured probability distributions in the recent past. A key requirement for such modeling to be effective lies in choosing the right amount of historical data as 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., 5th) 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

[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 time-scales (weeks to months), they might undergo more substantial changes.

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 *over-estimation* 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 $L(b)$ *over-estimation rate* as the fraction of $L(b)$ predictions that result in over-estimation, 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 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.

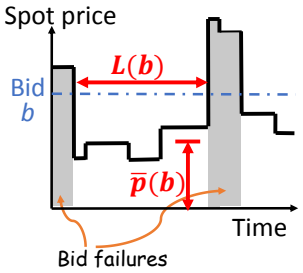

Figure 4: Illustration of features 1 and 2: lifetime of a spot instance and average spot price during lifetime.

Setup: We vary instance types, markets, bids, history window size H and show the assessment metrics f ($L(b)$ over-estimation rate) and ξ ($\bar{p}(b)$ relative deviation) in Table 1. 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 above metrics based on predictions of $L(b)$ and $\bar{p}(b)$ by using the empirical cumulative density function (CDF) of spot prices within H (updated dynamically), denoted as "CDF-based." 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: Under most of (market, bid) pairs, the best (lowest) f and ξ are below 10%, which we consider a reasonable demonstration of the efficacy of our predictors. We do see a small number of exceptions. E.g., for c3.large-c, even if we use 5th 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., m3.large-c), possibly due to the fact that the $L(b)$ and $\bar{p}(b)$ do not vary much when spot price exceeds bid.

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 5th 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.

[9]This is based on the discussions from [20, 26] and our observations that high spot price values are usually around multiples of on-demand prices. Although 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, in order to reduce bid failures and achieve better performance.

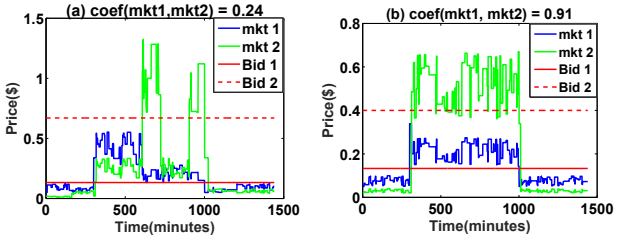

Figure 5: Synthetic examples with simultaneous revocations related to bids. "coef" is correlation coefficient.

Insights and implications: (i) 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, (ii) modeling of these features should be conditioned on specific bids; (iii) our approach outperforms CDF-based approaches by offering smaller over-estimation 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, e.g., [20, 26], 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: (a) the cross-correlation between the two markets' spot prices is low and (b) the cross-correlation is high. In (a) the tenant might be tempted to use both markets whereas in (b) 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 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 pre-specified 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

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

	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.large-c	$0.5d$	0.12	0.12	0.13	0.14	0.08	0.08	0.08	0.09	0.29	0.28	0.28	0.30	0.14	0.14	0.15	0.16
	$1d$	0.07	0.08	0.08	0.09	0.07	0.07	0.10	0.11	0.09	0.09	0.08	0.09	0.11	0.12	0.12	0.14
	$2d$	0.02	0.03	0.03	0.03	0.09	0.10	0.10	0.11	0.02	0.03	0.01	0.02	0.11	0.12	0.13	0.14
	$5d$	0.02	0.03	0.03	0.03	0.09	0.10	0.10	0.11	0.02	0.03	0.01	0.02	0.11	0.12	0.13	0.14
	$10d$	0.02	0.03	0.03	0.03	0.09	0.10	0.10	0.11	0.02	0.03	0.01	0.02	0.11	0.12	0.13	0.14
m3.large-d	$0.5d$	0.10	0.07	0.07	0.08	0.06	0.06	0.08	0.07	0.62	0.60	0.61	0.65	0.23	0.23	0.24	0.25
	$1d$	0.10	0.06	0.07	0.07	0.07	0.07	0.09	0.09	0.50	0.50	0.53	0.56	0.22	0.23	0.23	0.24
	$2d$	0.10	0.08	0.09	0.10	0.10	0.10	0.11	0.12	0.45	0.44	0.47	0.51	0.22	0.22	0.23	0.24
	$5d$	0.13	0.12	0.14	0.16	0.09	0.12	0.12	0.13	0.40	0.39	0.42	0.45	0.23	0.23	0.24	0.25
	$10d$	0.07	0.08	0.08	0.09	0.19	0.19	0.17	0.17	0.34	0.32	0.34	0.37	0.25	0.25	0.25	0.27
c3.large-c	$0.5d$	0.10	0.09	0.08	0.08	0.06	0.07	0.13	0.00	0.56	0.52	0.49	0.52	0.21	0.21	0.22	0.22
	$1d$	0.12	0.17	0.19	0.20	0.11	0.16	0.19	0.21	0.33	0.32	0.32	0.33	0.25	0.25	0.26	0.26
	$2d$	0.14	0.18	0.19	0.21	0.11	0.16	0.19	0.21	0.31	0.30	0.29	0.30	0.25	0.26	0.26	0.26
	$5d$	0.14	0.17	0.18	0.20	0.13	0.16	0.18	0.19	0.30	0.29	0.28	0.30	0.26	0.26	0.27	0.27
	$10d$	0.00	0.00	0.00	0.00	0.87	0.76	0.73	0.70	0.00	0.00	0.00	0.00	0.84	0.72	0.68	0.63
c3.large-d	$0.5d$	0.10	0.08	0.09	0.10	0.06	0.06	0.07	0.07	0.16	0.15	0.17	0.19	0.11	0.11	0.12	0.13
	$1d$	0.06	0.06	0.07	0.09	0.07	0.11	0.11	0.10	0.08	0.09	0.10	0.11	0.10	0.10	0.10	0.11
	$2d$	0.06	0.06	0.07	0.08	0.07	0.10	0.10	0.10	0.06	0.06	0.08	0.09	0.10	0.10	0.10	0.11
	$5d$	0.06	0.06	0.07	0.08	0.07	0.10	0.10	0.10	0.06	0.06	0.08	0.09	0.10	0.10	0.10	0.11
	$10d$	0.00	0.00	0.00	0.00	0.34	0.22	0.18	0.16	0.00	0.00	0.00	0.00	0.40	0.26	0.22	0.20

Table 1: The assessment metrics $f(b)$ and $\xi(b)$ under different bid values and history window sizes (H in days). The shaded cells represent the best window size that minimizes $f(b)$ and $\xi(b)$. "-c" and "-d" represent the markets. We round up the results and only show two digits after the decimal point due to space limit. "CDF-based" $f(b)$ and $\xi(b)$ are computed based on predictions of $L(b)$ and $\bar{p}(b)$ using CDF of spot prices.

	H	3	7	14	21	28
c3.2xl	$0.5d$	0.1778	0.1563	0.1286	0.1426	0.1270
	d	0.1353	0.1129	0.0894	0.1008	0.0971
	$5d$	0.0789	0.0799	0.0842	0.0952	0.0892
c3.1	$0.5d$	0.0242	0.0126	0.0000	0.0000	0.0000
	d	0.0141	0.0000	0.0000	0.0000	0.0000
	$5d$	0.0000	0.0000	0.0000	0.0000	0.0000

Table 2: Under-estimation rate of simultaneous revocations. We vary the history window H (days) and only show a subset of results here; the simultaneous revocation is computed for (us-west-1a, us-west-1c) for each instance type. The shaded cells represent the best H.

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. E.g., even if $T(A)$ and/or $T(B)$ are relatively small compared with the history window size, if $\frac{T(A\cap B)}{T(A\cup B)}$ is high, which implies markets (A,B) almost always fail together under the given bid, it may be better not to place bids in markets (A,B) simultaneously. On the other hand, 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 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 *under-estimation to the actual simultaneous revocations*: the fraction of simultaneous revocation predictions that result in under-estimation. From Table 2, we find that (i) the under-estimation rate is low in general, implying good temporal locality for prediction, (ii) history window size matters for different spot markets under different bids and (iii) the under-estimation rate could be 0, implying few bid failures in such markets under the particular bid. A tenant can use such information to de-correlate bid failures in his dynamic resource procurement.

Spatial Locality: Since EC2 regions and availability zones are geo-distributed, some naturally appealing ideas for improving the (market,bid) selection are: Is there any spatial locality in spot prices? Are markets spread across regions/availability zones un-correlated? Are such effects affected by bids? We show our measurements on simultaneous revocations across multiple regions, availability zones and instance types in Table 3. We have several observations: (i) Increasing bid may de-correlate bid failures: even if spot prices of two markets always jump simultaneously, they don't 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). (ii) Increasing bid may also increase the extent to which simultaneous revocation occurs, e.g., as the total failure time $T(A\cup B)$ decreases in markets (c,d) of c3.large (highlighted in Table 3), the fraction of time that concurrent bid failure occurs becomes less. (iii) Since the properties of simultaneous revocations highly depend on markets and bids (and possibly also 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. It is crucially important to take into account the impact of bid values. (iv) 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.

Additionally, if we look at the simultaneous revocations across instance types but fix the availability zone (Table 4), we observe that in general the spot prices across different instance types are not highly correlated; even for the pair (c3.large, c3.2xlarge) 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. However, although both $T(A)$ and $T(B)$ decrease

	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.large	b,c	0.0968	2590	1347	607	0.1823	1386	35	6	0.0042	900	6	6	0.0067
	c,d	0.1102	1347	2391	58	0.0158	35	1730	24	0.0138	6	1489	0	0
	d,e	0.0594	2391	10204	727	0.0613	1730	4517	329	0.0556	1489	0	0	0
	b,c'	0.0748	2590	0	0	0	1386	0	0	0	900	0	0	0
	c,a'	-0.0120	1347	7	0	0	35	0	0	0	6	0	0	0
	a',c'	-0.0846	0	0	0	0	0	0	0	0	0	0	0	0
m3.xlarge	b,c	0.2268	13314	103	54	0.0040	7050	0	0	0	376	0	0	0
	c,d	0.2109	103	3408	32	0.0092	0	1070	0	0	0	774	0	0
	d,e	0.0859	3408	279	5	0.0014	1070	9	0	0	774	9	0	0
	b,c'	0.0027	13314	84	0	0	7050	50	0	0	376	4	0	0
	c,a'	0.2001	103	5	0	0	0	0	0	0	0	0	0	0
	a',c'	0.1462	5	84	0	0	0	50	0	0	0	4	0	0
c3.large	b,c	0.6614	1459	3725	1121	0.2759	1103	2340	976	0.3956	1059	2274	921	0.3818
	c,d	0.7728	3725	1408	1211	0.3088	2340	1200	1189	0.5057	2274	1194	1184	0.5184
	d,e	0.4177	1408	298	251	0.1725	1200	284	238	0.1910	1194	279	233	0.1879
	b,c'	-0.0766	1459	2487	60	0.0154	1103	470	0	0	1059	0	0	0
	c,a'	-0.0158	3725	565	51	0.0120	2340	470	0	0	2274	0	0	0
	a',c'	0.3268	565	2487	335	0.1233	470	124	22	0.0385	0	0	0	0
c3.2xlarge	b,c	0.1456	6291	22998	5036	0.2076	1589	3917	946	0.2075	511	1152	18	0.0109
	c,d	0.3704	22998	8039	6272	0.2533	3917	14	0	0	1152	0	0	0
	d,e	0.3315	8039	7115	2350	0.1935	14	756	0	0	0	0	0	0
	b,c'	-0.0238	6291	7593	144	0.0105	1589	4011	8	0.0014	511	2899	0	0
	c,a'	0.0125	22998	9727	1915	0.0622	3917	7233	301	0.0277	1152	5047	20	0.0032
	a',c'	0.3680	9727	7593	4407	0.3413	7233	4011	2609	0.3021	5047	2899	1504	0.2335

Table 3: Simultaneous revocations across different pairs of markets (b,c,d,e) from region us-east and (a',c') from region us-west under different bids. 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.

as we increase the bid from $0.5d$ to d for (m3.large,m3.xlarge), we notice that $\frac{T(A \cap B)}{T(A \cup B)}$ increases from 0.0044 to 0.1304, which re-emphasizes our key finding that analysis of simultaneous revocations should take into account the bids instead of blindly looking at the "coef" as done by the prior work.

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 the k-means clustering [7], 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 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].

Insights and implications: (i) Temporal locality can be employed for predicting simultaneous revocations. (ii) 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. (iii) 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

Time to start is an important feature for 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 min [4]. 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.

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 [13], which focuses only on instance boot up times. For spot instance, they observe no significant correlation between spot price and time-to-start; however, they do not explore/report the temporal locality and the predictive property of 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 min over two days. The bids are uniformly chosen from $\{\frac{1}{4}d, \frac{1}{2}d, \frac{3}{4}d, d, 2d, 5d, 10d, max\}$, wherein d is the on-demand price and max is the maximum bid allowed. We report a subset of our results in Figure 6. In some cases (top and middle traces shown in Figure 6(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

[11]Picking instances from multiple marketplaces (especially those geographically distributed) is of course more complex. E.g., there may be concerns related to communication between instances over a WAN or issues of proximity to tenants that we can not consider in our work [33]. An actual decision-making would need to consider these against the issues that we do model here.

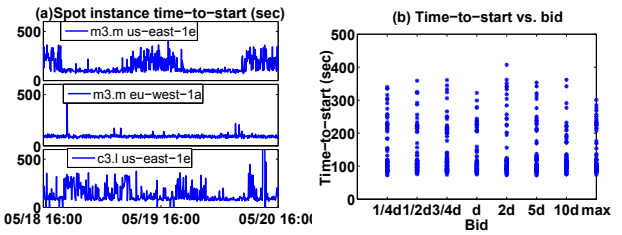

Figure 6: (a) Spot instance time-to-start. (b) Spot instance time-to-start vs. bids. We request one instance every 5 min across two days from 2016/05/18 16:00 to 2016/05/20 16:00 with bids uniformly chosen from $\{\frac{1}{4}d, \frac{1}{2}d, \frac{3}{4}d, d, 2d, 5d, 10d, max\}$, wherein d is the on-demand price and max is the maximum bid allowed. The time-to-start is set to 0 if bid is less than spot price. The largest two values in the bottom trace of (a) are 2800+ seconds.

(e.g., 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 6(a), the largest two samples are greater than 2800 seconds.

We also explore the relationship between time-to-start vs. bid. Intuitively, higher bids should give the tenant higher priority for resource procurement, thus shorter time-to-start, since EC2 might make more profit by serving higher bids first. However, we do not observe a statistically significant correlation between time-to-start and different bid values (Figure 6(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 1-3. 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.

3.4 Do Effective Capacity Measurements Provide Predictive Power?

EC2 claims that the spot price is set based on its supply/demand relationship [3]. Therefore, a natural thought is: Can we further improve the predictability of spot prices if we can somehow infer the current resource demand vs. 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 advertized 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: (i) Is there any predictive power in effective capacity vs. spot price? (ii) Is there any predictive power in effective capacity vs. sudden revocations? Such relationships, if observed, can be exploited by the tenants to improve their prediction of spot prices (or of features 1-3).

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 micro-benchmarks. Due to space limit, we only show a subset of our measurement results in Figure 7. As illustrated by the scatter plots of normalized performance vs. 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 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 vs. 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 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 [16]) 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/de-allocate 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 [34], 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, as b a bid picked from B_s (a set of pre-selected 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/de-allocate 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.

	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.l, m3.xl	0.1357	1046	328	6	0.0044	24	28	6	0.1304	6	0	0	0
m3.l, c3.l	0.0146	1566	3631	11	0.0021	987	2335	3	0.0009	6	2274	0	0
m3.l, c3.2xl	0.1939	35	41898	29	0.0007	6	17538	0	0	6	1167	0	0
m3.xl, c3.l	0.1153	431	3348	0	0	28	2317	0	0	0	1552	0	0
m3.xl, c3.2xl	0.2870	60	29769	56	0.0019	0	15715	0	0	0	1158	0	0
c3.l, c3.2xl	0.3214	2327	45478	1987	0.0434	2306	18261	1492	0.0782	0	1167	0	0

Table 4: Simultaneous revocations across different instance types in us-east-1c.

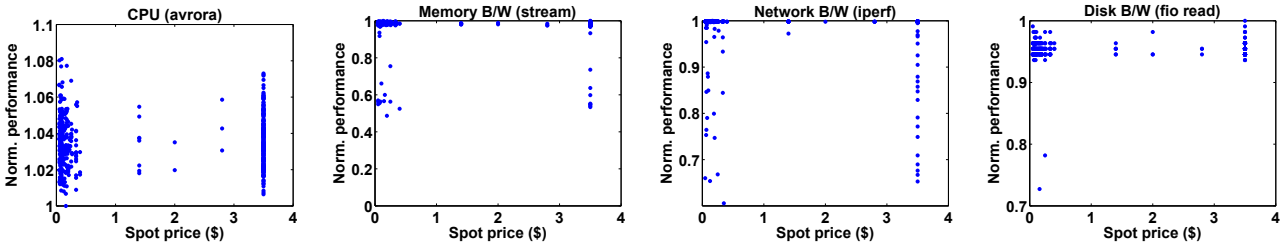

Figure 7: Scatter plots of 24-hour micro-benchmark performance measurements on a single r3.xlarge instance vs. spot price from us-east-1d. avrora is a CPU-intensive benchmark [2]; STREAM [25], iperf [9] and fio [9] are commonly used benchmarks that measure the bandwidths of memory, network and disk I/O. The performance measurements are normalized w.r.t. the best performance samples in the experiments. The spot price varies from $\frac{1}{7}$ to 10 times the on-demand price.

We present our optimization formulation as follows:

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

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

$$\sum_{s \in S'} x_t^{sb} \geq \xi, \quad S' = \{\text{OD}\}$$

$$x_t^{sb} \hat{M}_t \leq (N_t^{sb} + \tilde{N}_t^{sb}) m^s, \quad \forall s \in S, b \in B_s$$

$$\phi(\hat{\lambda}_t x_t^{sb}, (N_t^{sb} + \tilde{N}_t^{sb}) c^s) \leq l^{TGT} \quad \forall s \in S, b \in B_s$$

where $\{\text{OD}\}$ is the set of on-demand instance types. In the objective function, the first term represents the resource costs which 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 re-balancing 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 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} wherein $\phi(\hat{\lambda}_t x_t^{sb}, (N_t^{sb} + \tilde{N}_t^{sb}) c^s)$ is a function capturing the relationship between latency, arrival rates and the number of vCPUs. $\phi(.)$ can be obtained by various techniques, e.g., regression using empirical measurements.

We update 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 solve our optimization problem once every hour. In 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 can be found in our technical report [31].

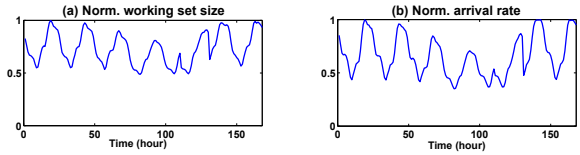

Figure 8: Normalized working set size and request arrival rates from Wikipedia access trace [29]. Normalization is done against the maximum value in each trace.

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 Figure 2. We generate our workload by scaling the dynamic arrival rates λ_t and working set size M_t from Wikipedia access trace [29] (Figure 8). 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.

Baselines: Denote as "PROP" our proposed online approach. We further create two baselines to compare against: (i) "BL-OD": all data are stored on on-demand instances (no bidding). (ii) "BL-CDF": Predicting $L(b)$ and $\bar{p}(b)$ via the CDF-based approach (cf. 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-saving (against BL-OD) and performance (reflected by data loss due to bid failures) under different strategies in Table 5 and Figure 9. 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 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 make the tenant tempted to use spot instance 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 much 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 9 shows, BL-CDF has 13 and 22 times of 90% data loss (out of working set) in the two cases whereas PROP only has 5 and 1 times at the same data loss level, respectively. 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.

	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

Table 5: Cost savings of different strategies compared against BL-OD. 'e' and 'w' represent experiments using all spot price traces from us-east and us-west, respectively.

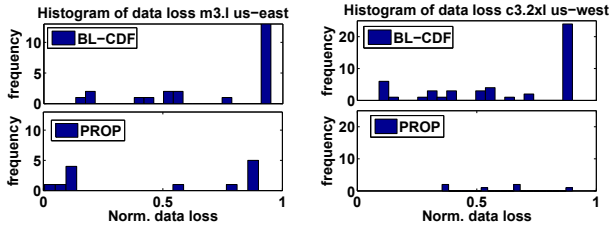

Figure 9: Histograms of data loss (norm. against actual working set size) during bid failures.

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 8, and 24-hour spot price traces taken from m3.large in us-east (Figure 10(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.

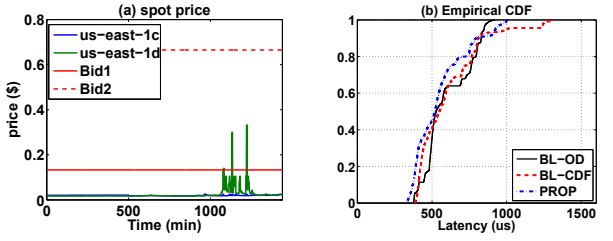

Figure 10: (a) 24-hour spot price of m3.large from two markets with Bid1= d and Bid2= $5d$. (b) The CDF of latencies.

We show the application performance under different strategies in Figure 10(b). We find that the three strategies offer comparable performance up to 90%ile latencies. Below 90%ile, sometimes BL-CDF and PROP could outperform BL-OD since they could use more spot instances to serve the requests without incurring higher costs than BL-OD. Beyond 90%ile, BL-OD offers the best performance possibly because it does not spread working set among multiple markets which depresses the latency oscillation due to working set re-partition. Meanwhile, Prop is able to beat BL-CDF and gets closer to BL-OD (however with much less costs) 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 in-memory data store is usually considered as a performance-sensitive/latency-critical application, the tenant may desire "always-on"/"service contiguity" more than cost saving 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 vs. 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 over-subscribed so that the backup copies only get a small portion out of their own regular capacities. Therefore, primary copies would finish sooner than backup copies. Since spot instances are in general much cheaper than on-demand instances and the on-demand capacities are over-subscribed, the expected costs would be much lower than if we run the jobs only on on-demand instances and with over-subscription. If the spot instance is not revoked by the time when the primary copy finishes, we terminate the backup copy to save costs and make more room for new jobs; otherwise, we boost the performance of the backup copies whose primary copies have failed by allowing them to use more 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 towards 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 the valid (market, bid) pairs that satisfy $\hat{L}(b) \geq \hat{l}_j$ as candidates, wherein $\hat{l}_j$ 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 (e.g., summation) of the probability of simultaneous revocations of each existing job on that instance vs. 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. We leave more details and advanced performance enhancements to our technical report [31].

Experimental Setup: We use m3.xlarge (4 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 three-month price traces are shown in Figure 2. We use an exponential distribution to generate the inter-arrival 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 [20, 26]).

Trace-driven Simulation: We conduct trace-driven simulation using the three-month spot price traces to demonstrate the long-term benefit of our proposed approach. Figure 11 shows the cost break-down and CDF of job execution time under different strategies.

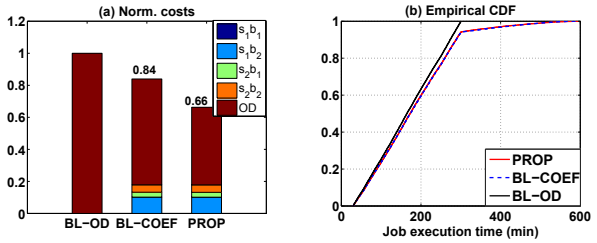

Figure 11: (a) Cost break-down and (b) CDF of job execution time. The CDFs of BL-CDF and PROP overlap.

We have several observations. First, PROP saves as much as 33.7% and 17.7% costs compared to BL-OD and BL-COEF (Figure 11(a)), respectively. This is because BL-OD only uses on-demand instances which 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 probability of 1, which may not be true if spot price falls between the two bids and only one of the jobs fails. Second, from Figure 11(b), we find that BL-OD offers best performance since there is no bid failure. BL-COEF and PROP has almost the same CDF of job execution time, implying PROP captures the simultaneous revocations well and provides similar capacity headroom for performance boosting when failures occur compared to the conservative BL-COEF.

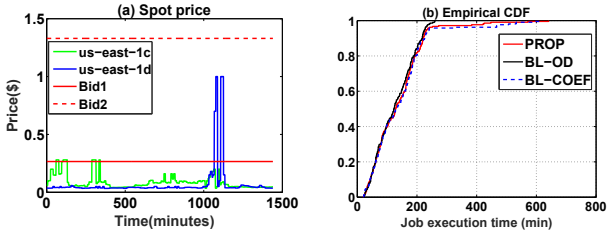

Figure 12: (a) 24-hour spot price of m3.xlarge from two markets with Bid1= d and Bid2= $5d$ (d: on-demand price). (b) The CDF of job execution time under different strategies.

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 12(a)) and conduct realtime experiments. A bid failure occurs at around 1100-th 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 12(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 (though 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 paper, 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.

6. ACKNOWLEDGEMENT

This research was supported, in part, by NSF CAREER 0953541 grant and an IBM faculty partnership award. We gratefully acknowledge this support as well as the reviewers' feedback.

7. REFERENCES

[1] O. A. Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir. Deconstructing amazon ec2 spot instance pricing. In *Proc. of CloudCom'11*, 2011.

[2] Avrora, 2016. http://dacapobench.org/benchmarks.html.

[3] Building price-aware applications using ec2 spot instances, 2015. https://aws.amazon.com/blogs/aws/category/ec2-spot-instances/.

[4] Ec2 boot time, 2016. http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ComponentsAMIs.html.

[5] EC2 spot, 2016. http://aws.amazon.com/ec2/spot-instances/.

[6] Y. Gong, B. He, and A. C. Zhou. Monetary cost optimizations for mpi-based hpc applications on amazon clouds: Checkpoints and replicated execution. In *Proc. of the SC'15*, 2015.

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

[8] J. He, Y. Wen, J. Huang, and D. Wu. On the cost–qoe tradeoff for cloud-based video streaming under amazon ec2's pricing models. *Circuits and Systems for Video Technology, IEEE Transactions on*, 2014.

[9] iPerf, 2016. https://iperf.fr/iperf-download.php.

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

[11] G. Kesidis, B. Urgaonkar, N. Nasiriani, and C. Wang. Neutrality in future public clouds: Implications and

challenges. In *HotCloud'16*, 2016.

[12] S. Khatua and N. Mukherjee. Application-centric resource provisioning for amazon ec2 spot instances. In *Euro-Par 2013 Parallel Processing*. 2013.

[13] M. Mao and M. Humphrey. A performance study on the vm startup time in the cloud. In *Proc. of IEEE CLOUD'12*, 2012.

[14] A. Marathe, R. Harris, D. Lowenthal, B. R. de Supinski, B. Rountree, and M. Schulz. Exploiting redundancy for cost-effective, time-constrained execution of hpc applications on amazon ec2. In *HPDC'14*, 2014.

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

[16] Memcached, 2016. https://memcached.org/.

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

[18] Spot characterization code and data, 2016. https://github.com/patiner/spot_characterization.git.

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

[20] P. Sharma, S. Lee, T. Guo, D. Irwin, and P. Shenoy. Spotcheck: Designing a derivative iaas cloud on the spot market. In *Proc. of EuroSys'15*, 2015.

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

[22] Spot instance: featured customer testimonials, 2015. https://aws.amazon.com/ec2/spot/testimonials/.

[23] Spot fleet, 2016. http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/spot-fleet.html.

[24] Spot bid status, 2016. http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/spot-bid-status.html.

[25] STREAM, 2016. http://www.cs.virginia.edu/stream/.

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

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

[28] S. Tang, J. Yuan, and X.-Y. Li. Towards optimal bidding strategy for amazon ec2 cloud spot instance. In *Proc. of IEEE CLOUD'12*, 2012.

[29] G. Urdaneta, G. Pierre, and M. Van Steen. Wikipedia workload analysis for decentralized hosting. *Elsevier Computer Networks*, 53(11), 2009.

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

[31] C. Wang, Q. Liang, and B. Urgaonkar. An empirical analysis of amazon ec2 spot instance features affecting cost-effective resource procurement. Technical report, CSE TR-16-006, Penn State University. http://www.cse.psu.edu/research/publications/tech-reports/2016/CSE-16-006.pdf, 2016.

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

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

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

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

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