Providing Lifetime Service-Level-Agreements for Cloud Spot Instances

Alexander Pucher, Rich Wolski, Chandra Krintz
Department of Computer Science
University of California, Santa Barbara
{pucher, rich, ckrintz}@cs.ucsb.edu

***Abstract*—Spot instances are commonly offered by IaaS cloud providers to opportunistically utilize spare capacity and meet temporary user demand for additional resources. Although the availability of service SLAs is a core paradigm of cloud computing, spot instances in practice still come without any service quality guarantees. We aim to extend the spot instance service to provide a probabilistic SLA for eviction probability, based on the user estimate of the maximum expected instance lifetime. This probabilistic foundation simplifies reasoning about the spot instance service and enables providers to construct higher-level SLAs from them. For this to be possible, however, the statistical guarantees must be adhered to strictly, for a wide range of real-world workloads, at cloud scale. We propose a new approach to providing SLAs on the time-until-eviction for spot instances by employing Monte-Carlo simulation to compute the distribution of future spot instance lifetimes at current cloud utilization levels. We then show that an IaaS cloud scheduler can use the quantiles of such conditioned distributions to safely provision spot instance requests and maintain an SLA with a specific target eviction rate.**

I. Introduction

Cloud computing, in the form of Infrastructure as a Service (IaaS), has emerged as a new paradigm for Information Technology (IT) management of data center infrastructure. Under the IaaS cloud model, users request that data center resources be *provisioned* for their exclusive use via network-facing web service interfaces (APIs). “The Cloud” services these requests in a way analogous to the way in which e-commerce services operate: automatically and transactionally. Users interact only with the automated cloud services and requests are either fulfilled or denied immediately (possibly due to error) so that the user may retry if he or she desires to do so. The user-requested services are then typically delivered until cancellation subject to a *Service Level Agreement* (SLA).

In this paper, we examine the feasibility of using two classes of resources requests – *on-demand* and *spot* – to achieve greater resource utilization while providing statistical service level guarantees for *both* classes. We borrow this terminology from Amazon’s AWS [1] where *spot instances* are pre-emptable requests that may be terminated without warning if the “spot market” conditions warrant (i.e. the current market price exceeds the user’s bid), and *on-demand instances* are never pre-empted, but incur a higher per-minute occupancy cost than spot instances.

Our goal is to understand whether it is possible to use the spot instance service in a cloud to accept workload subject to a statistical guarantee on minimum time until pre-emption. In particular we

- demonstrate that it is possible to provide statistical guarantees on minimum spot instance lifetimes using production private cloud workload traces, and
- detail the effectiveness of co-scheduling on-demand workloads with spot workloads with these guarantees to utilize otherwise unused resource capacity.

This work complements previous investigations of public cloud spot instance services from the user perspective. Prior work develops bidding schemes based on spot instance price history of public clouds and looks at revenue optimization for hypothetical web services built on top of spot instances [2], [3], [4]. While these works demonstrate the economic viability of using spot instances in public clouds, they rely on use-case specific utility functions and assumptions about the spot instance workload, such as the ability to checkpoint.

In contrast, our work is motivated by production enterprise, research, and high-performance computing environments where a private cloud offers scalable, automated, SLA-governed service to its users. In these settings, resource utilization must be maximized, although resources are often over-provisioned to ensure an acceptable user experience in terms of response time and available capacity. Furthermore, unlike public cloud, enterprise private clouds do not typically charge a fee for usage. Rather, each user is given a a usage quota and must maximize resource utilization subject to quota limits.

In this setting, spot-instances can allow enterprise users to exceed their respective quotas by taking advantage of otherwise unused capacity. Because a spot-instance will be terminated if the capacity is needed to run an on-demand instance, users can run spot instances without a charge to their respective quotas. That is, the capacity for spot-instances is “scavenged” and then reclaimed when it is needed for on-demand instances.

From a cloud perspective, the uncertain duration of time that a spot-instance will run before it is terminated makes their use difficult. Our work uses on-line simulation to *predict* spot-instance lifetimes based on the recent history of cloud activity. In particular, we use a Monte-Carlo style [5] simulation (run every few minutes) to estimate the distribution of the time-until-eviction of spot instances from requests in the recent past. We use this non-parametric approach to compute the quantiles (i.e. percentiles) of the empirical distributions of spot instance lifetimes that are conditioned on the capacity currently available in the cloud. These quantiles then serve as a probabilistic lower bound on the future time-until-eviction which the user can interpret as a statistical “guarantee” of spot instance lifetime. For example, the lower 0.95 quantile indicates the minimum spot-instance lifetime that as user can expect with probability 0.95.

We evaluate our method based on trace-driven simulation with synthetic and recorded commercial traces from Eucalyptus [6], [7] IaaS clusters deployed in production systems [8]. We use the synthetic workload traces to demonstrate the theoretical efficacy of our approach and give an in-depth look at the steps required to produce accurate time-until-eviction estimates via Monte-Carlo simulation. We then apply this method in a cloud scheduler that is capable of maintaining an arbitrary SLA for maximum eviction probability of spot instances in an IaaS cloud even when faced with the adverse conditions of commercial production environments.

II. Methodology

The goals of our methodology are to define a method

- for predicting minimum lifetime of spot instances with configurable confidence bounds using historical observations of previous instance behavior, and
- for using these predictions in scheduler-level admission control to ensure that all accepted spot requests meet their target lifetime.

This latter requirement is consistent with current cloud abstractions in that requests are either accepted by the cloud (and thus subject to the advertised SLA) or rejected because the SLA cannot be met.

To predict minimum lifetime, we have developed a prediction utility that uses historical data from IaaS system logs of instance types (core counts) and instance start/stop events to construct empirical distributions of instance lifetimes conditioned on available cloud capacity via periodic Monte-Carlo simulation. From these lifetime distributions, the utility extracts the quantile associated with the SLA offered by the IaaS cloud for spot instances (e.g. a 95% or 99% confidence bound on the likelihood that the instance will not be evicted) to predict minimum lifetime for each level of available capacity.

For admission control, we assume that spot requests are accompanied by a *user-specified* lifetime (maximum) when submitted. Our IaaS scheduler uses (i) the quantile estimates for the SLA generated by the prediction utility, (ii) the instance size (also specified per request) and maximum lifetime from the user, and (iii) the currently available capacity of the system, to decide whether to admit a spot request. The scheduler evicts spot instances if/when an on-demand instance request is made and the cloud has insufficient capacity to service the request.

A. Scheduling Model

Instance requests (to either start or stop an instance) are routed to a scheduler (as implemented by IaaS infrastructures such as Eucalyptus [6], Open Stack [9], and Cloud Stack [10]) which handles admission control and placement of instances on physical resources in a cluster of "nodes." IaaS clouds typically define "instance types" that describe the resources that an instance will consume (CPU cores, memory, ephemeral disk storage, etc.). In the Eucalyptus systems (production and research) that we investigate in this work, we observe that the memory footprint associated with each instance type is such that the instance placement decision by the scheduler can be made strictly on core count.

When an instance is admitted, the scheduler makes a placement decision by selecting a node on which the instance will run. In this study, we use simple first-fit placement in favor of more complex approaches to highlight the impact SLA-aware admission control. If an on-demand instance is requested, and the scheduler cannot find a node with available capacity, the scheduler selects one or more spot instances to terminate (evict) so that the on-demand instance can be scheduled.

Further, our scheduler (like other Eucalyptus schedulers) assumes that the instance type definitions nest with respect to their core counts. For example, an empty 4-core node node is seen by the scheduler as having 1x 4-core slot, 2x 2-core slots, 4x 1-core slots, or 1x 2-core, 2x 1-core slots. The distinction between available cores and *available slots* is important when generating time-until-eviction estimates for different instance sizes and different cluster load levels.

B. Prediction

Past work has shown that cloud workloads can be highly variable and may not be easily described by single well-known distributions [11]. To address this problem we run a Monte-Carlo-style simulation on-line to generate the empirical distribution of expected spot instance lifetimes. However, we note that the time-until-eviction is affected by the capacity of the cloud that is occupied by un-evictable on-demand workload and other spot instances. Intuitively, if the cloud is relatively "empty", a spot-instance that is introduced will likely live longer than if the cloud is close to "full" capacity. Thus, our Monte-Carlo simulation produces a *set* of empirical distributions, one conditioned on each level of possible occupancy.

For example, a cloud with 100 cores has 101 possible occupancy levels: from 0 cores occupied to 100 cores occupied and each level of occupancy corresponds to a different distribution of spot-instance lifetimes. We use quantiles of these distributions to quote the expected lifetime to the scheduler during the admission control phase based on the current occupancy level at the time the spot-instance request is made. If the instance (based on its maximum lifetime specified by its user) is expected to be evicted with a higher probability than specified by the target probability (quoted as an SLA) for the cloud, it is rejected (not admitted). The cloud administrator is responsible for setting the SLA on eviction probability that is advertised to all cloud users.

The Monte-Carlo simulator generates a sample of "fictitious" spot-instance requests that using the recent cloud load history. It repeatedly chooses a random point in the history and simulates the arrival and eviction of a spot-instance, recording the occupancy level at he time the spot-instance starts and its time-until-eviction. Running faster than real time, it generates 10000 such samples and divides them into empirical distributions based on occupancy level.

C. Eviction Policy

The eviction policy affects the conditional spot-instance lifetime distributions generated by the simulation (but not the correctness of the method). Many policies are possible but each has an impact on user experience. In this paper we chose a simple "Youngest-Job-First" (YJF) eviction policy. Choosing the "youngest" (i.e. the spot instance that has started most recently) to evict among the candidate spot instances is an attempt to minimize the "regret" associated with an eviction in this online

TABLE I: Parameters of synthetic log-normal on-demand and spot instance workloads

	VM arrival	VM duration	VM cores	mean util.
on-demand	$\mu = 4, \sigma = 1$	$\mu = 6, \sigma = 1.5$	1	21.77
spot	$\mu = 4, \sigma = 1$	$\mu = 6, \sigma = 1.5$	1	21.95

decision making problem [12]. That is, the amount of work that is lost because of an eviction is minimized.

D. Evaluation Metrics

To evaluate the system we use trace-based simulation with both synthetic and production traces taken from private Eucalyptus IaaS clouds. We replay each trace in its entirety and we log each individual state change in the simulated system. In each case, the simulator uses separate traces for spot-instance and on-demand instance requests. We then generate summary statistics and evaluate our solution using two metrics:

- eviction ratio of spot instances
  $evicted = evictions/admissions$
- admission ratio of spot instances
  $admitted = admissions/requests$

The enforcement of the target SLA probability has highest priority. After the SLA is fulfilled, a high number of completed spot instance requests is desirable to maximize utilization.

III. Results

Our experiments are run in simulation, based on our previous work on validated simulation of private IaaS clouds. We use both, synthetic traces and anonymized production traces obtained from Eucalyptus IaaS cloud installations. For reproducibility we assume instant start and stop of instances in the traces and rely on a publicly available set of anonymized commercial production traces [8].

A. Prediction with synthetic traces

To outline our approach and show its basic behavior we compare a scenario with an SLA-unaware scheduler and the SLA-aware scheduler using multiple different SLA levels using 10-day synthetic traces (Parameters in Table I). Our initial setup uses a single platform (IaaS cluster configuration) and synthetic on-demand and spot request traces. The platform contains 8 nodes with 4 cores each, for a total of 32 cores. As a rough estimate based on mean utilization the platform should be able to support the on-demand trace plus half the spot instance trace. We use a log-normal distribution to approximate the long-tailed empirical distribution of instance life times.

Note that there is a trade-off between the probabilistic guarantee given to the user and the fraction of spot-instance requests that can be accepted by the scheduler. Greater "certainty" associated with a spot-instance SLA (in the form of a lower eviction probability) implies that fewer spot-instance requests can be accepted (to decrease the possibility that an eviction will be necessary).

To illustrate this trade-off, show the ratio of admitted spot instances, as well as the eviction ratio of spot instances in Figure 1. The x-axis shows different SLA probabilities, starting with the no-guarantees baseline on the left and then increasingly stringent SLAs of 0.25, 0.10, 0.05 and 0.01. The y-axis shows the fraction of admitted spot-instance requests instances in gray and the fraction of evicted spot instances in black. The SLA-aware scheduler meets the SLA in all cases (the eviction fraction is less than the advertised guarantee level), at the cost of preemptively rejecting an higher fraction of spot instances for stricter SLAs. The measured SLA probabilities are in fact stricter than the target SLA probabilities, showing that the predictions of time-until-eviction made by the simulation are conservative.

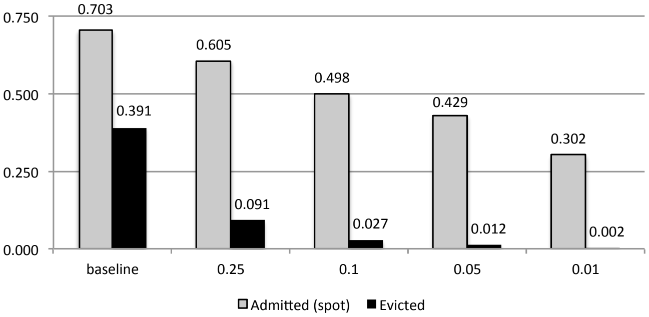

Fig. 1: Ratio of admitted and evicted spot instances with synthetic log-normal traces.

The most visible improvement is the step from the no-guarantees baseline to the 0.25 eviction ratio SLA. While the baseline admits 70% of all requested spot instances, 39% of the admitted spot instances are evicted before completion. The 0.25 SLA in contrast admits 60% of all requested spot instances, but only 9% of the admitted spot instances are evicted. Subsequent decreases in the demanded maximum eviction rate of spot instances decrease the number of admitted spot instances as well, but consistently (conservatively) achieve the SLA eviction probabilities.

This experiment outlines the setup of our simulation driven approach to enforce guaranteed levels of eviction probabilities in a controlled environment. In the next section we discuss the simulation method in-depth.

B. Conditional Distributions and Sample Size

The scheduler computes conditional distributions for all possible core counts on a fixed duty schedule (every 6 hours of trace time in the previous experiments) based on the history of on-demand and spot instance behavior it has observed so far. This gives rise to the property that the sample sizes for "rarely" occurring conditions may be small. For example, if the cloud is moderately loaded, the number of examples where all but one of the cores is busy might occur infrequently or not at all.

To provide an in-depth insight in the behavior of the Monte-Carlo simulation, we provide an exemplary intermediate results at the 9 day mark of our synthetic trace experiment for single-core instance slots. Figure 2 shows the number of samples generated on the y-axis for each condition on the x-axis (available slot count). In our specific example, 2 to 4 open slots are encountered the most frequently, with about 10000 samples each. High open slot counts, which correspond to low cluster utilization, are increasingly uncommon. Based on the number of samples we expect predictions for common cases to be highly accurate, while infrequently occurring cases will be based on empirical distributions estimated from small samples.

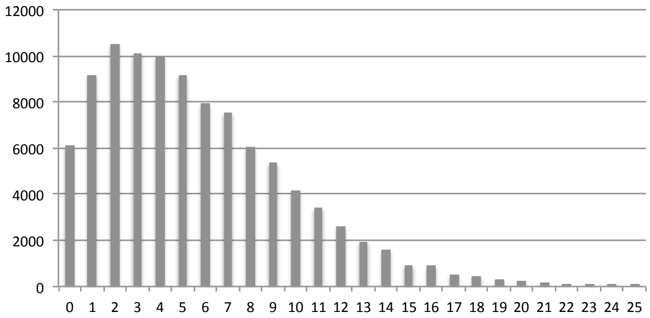

Fig. 2: Number of time-until-eviction samples per available-slots bucket for a synthetic log-normal simulation run. Frequently encountered load levels (left) have many samples, corner cases (right) have few. The shape of the histogram depends on the historic workload.

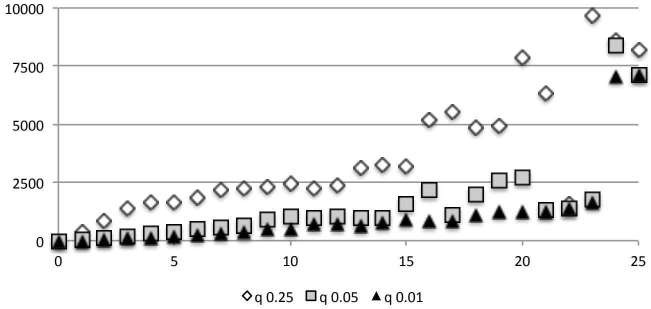

Fig. 3: Quantiles of time-until-eviction per available-slots bucket for a synthetic log-normal simulation run. Predictions for common load levels (left) can be made with high confidence, predictions for infrequent ones (center) are rough estimates that become increasingly erratic for corner cases (right).

Figure 3 shows the quantiles of the conditional distribution of times-until-eviction. The x-axis again shows the condition, while the y-axis indicates the time-until-eviction as estimated by a quantile. The estimates to the left correspond to the buckets with high sample count in Figure 2, whereas the estimates to the right decrease in sample count. The 0.25 quantile lies above the 0.10 quantile, followed by the 0.01 quantile. These quantiles estimate the minimum time a spot instance is expected to survive with the corresponding probability. For example, from the figure, 0.01 of the instances that are started when there are 15 free slots run for 900 seconds or less before being evicted. In the same column (for 15 free slots), 0.05 of the time-until-eviction samples are 1500 seconds or less, and 0.25 of them are 3200 seconds or less.

A combined look on the counts per bucket in Figure 2 and the corresponding quantiles in Figure 3 also provides an insight into the reliability of conditional estimates. Buckets 0 to 14 each have over 1000 samples each to determine quantiles from. This is generally enough for low SLA probabilities, such as 0.05 or 0.01. With increasing slot count (decreasing cluster utilization) a smoothly changing, and mostly increasing estimate of the time-until-eviction can be observed. Buckets 15 to 20 still have over 200 samples each, which is enough for rough estimates, but a look back at the quantiles shows that changes from bucket to bucket already become erratic. Estimates for 21 available slots and over appear extremely infrequently in our synthetic trace. Their samples are mostly artifacts from the initial warm-up period and as such, their estimated quantiles are not reliable (but also hardly used).

TABLE II: Mapping of recorded commercial production traces and their original hardware platforms from the data set collection [8] to experiments in this paper.

Name	Source	Organization	Workload	Nodes
A	DS2	Medium	bursts	7 x 8 cores
B	DS3	Medium	bursts	7 x 12 cores
C	DS5	Large	variable	31 x 32 cores
D	DS6	Large	constant	31 x 32 cores

Since we are using a synthetic trace based on log-normal distributions for arrival time and instance lifetime, this specific example could be described analytically as well. However, for arbitrary traces, as found in production environments, this is challenging to impossible depending on the typical usage of the cluster. Monte-Carlo simulation offers a way to estimate arbitrary empirical distributions and can be tailored to achieve the desired degree of prediction accuracy.

C. Prediction with production traces

To study the utility of Monte-Carlo-based SLA enforcement in a more realistic setting, we use four different traces obtained from independent Eucalyptus IaaS production installations for our experiments. The origin of these traces is documented in [13], [11], [14]. and the traces themselves are available as part of a collection from [8]. Table II shows the mapping of data sets from the collection to experiments in this paper, together with a short description of their workload and platform properties.

Compared to synthetic traces there are a number of important differences. First, instance starts show temporal auto-correlation. These “bursts” of instance starts are more extreme than ones observed in synthetic log-normal traces. Second, the behavior of users changes over time and causes change points which the empirical distribution derived via Monte-Carlo simulation only picks up over longer time frames. Third, instance sizes are no longer uniform and traces contain instances with slot sizes between 1 and 30 cores.

To facilitate the experiments with real world traces, two modifications are made to the Monte-Carlo simulation. First, we expect that our randomization approach may not generate starting points needed for all conditional core-utilizations needed, especially in the beginning of the experiment where data samples are scarce. To avoid rejecting spot instances unnecessarily due to a perceived lack of information, we linearly approximate quantiles of unobserved conditional distributions between observed “neighboring” distributions.

For example, if the empirical distributions conditioned over 20 slots and 18 slots are available, while there are no samples for 19 slots, the quantiles for 19 available slots are generated by linear approximation between the the matching quantiles of the neighbors. For example, the 0.01 quantile for 19 slots would then be calculated as $q(0.01|19) = (q(0.01|18) + q(0.01|20))/2$. In the case where multiple conditions are missing, we fit a line to the two endpoints in the range of missing values and use it to approximate the quantiles between. Additionally, the extreme

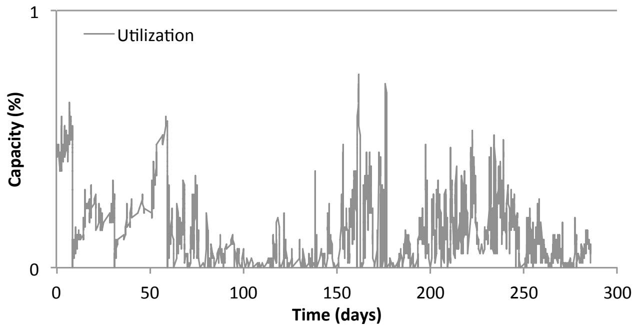

Fig. 4: Production trace B as executed on its native platform shows highly variable load and bursts of large requests as well.

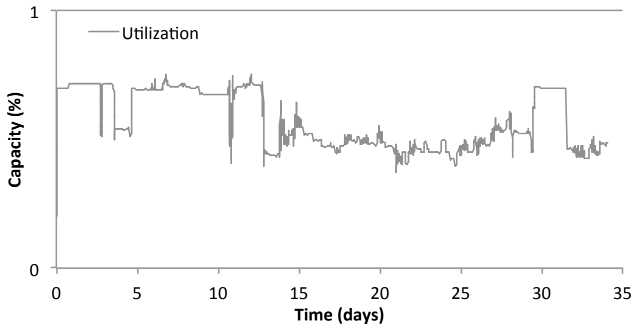

Fig. 5: Production trace C as executed on its native platform shows a mixed pattern of load with constant plateaus and periods with higher variability.

points of zero and full utilization need to be populated with useful data. We chose an impossible value for expected life time before eviction if there are no slots available for a given capacity and conservatively use the quantiles for the lowest known cluster utilization as values for zero utilization as well.

Second, we start the real world traces after a delay of 24 as we do for the synthetic traces. Such a delay allows the scheduler to "warm up."

A visual inspection of the real-world traces shown in Figures 4 and 5 shows significant spikes at irregular intervals. If an on-demand spike in load appears in an environment already loaded with spot instances, we expect to see a high number of correlated evictions, possibly leading to a violation of the SLA in the short-term. If these correlated evictions are not compensated for in the long-term by conservatively maintaining a capacity buffer, these short-term violations will sum up to an SLA violation over the course of the whole trace. We try to capture this auto-correlation by replaying the actual observed trace in our Monte-Carlo simulation rather than re-sampling the input distribution. That is, we choose random locations in the trace, but then replay the trace from those periods to include auto-correlation effects.

We take the same approach to handling change points in the production time series traces. The Monte-Carlo simulation that computes the empirical conditional distributions is re-run every 6 hours of trace time to capture changes that may have occurred in the underlying dynamics. The the Monte Carlo simulation with production traces takes no more than 300 seconds (5 minutes) to generate the empirical distributions. Thus, in a production implementation, it would be possible to make these estimates every 6 hours "on-the-fly" as part of the cloud's typical operation.

The third difference of real-world traces over to our synthetic ones are non-uniform instance capacities. This has two major implications: first, Monte-Carlo simulation must consider different instance sizes and second, placement decisions for on-demand instances made at any time may have consequences later in the trace. Because the scheduler attempts to find space for an on-demand instance and only evicts when there is insufficient capacity, the presence of spot-instances can change where the scheduler places on-demand instances. As a result, because an instance cannot span nodes, it could be that the introduction of spot instances increases the "fragmentation" of the available core capacity and, hence, affects the ability to run on-demand instances. However, while spot-instances might cause the scheduler to reject an on-demand instance it would have otherwise accepted (due to fragmentation effects) all of the on-demand instances that are accepted receive the SLA guarantees that they would have without spot instances present. This effect (detailed in Subsection III-E) is small for the production workloads we study but grows as the cloud runs closer to capacity.

The conditional distribution of expected lifetimes therefore effectively becomes conditioned over instance capacity (taking into account fragmentation effects) in addition to available slot count. The conditioning over instance capacity does not increase the amount of data required for accurate estimates as we can re-run the same recorded trace with different virtual instance sizes. An increasingly diverse population of instance types therefore leads to a linear increase in computational effort for Monte-Carlo simulations, but not to a relative reduction of estimation accuracy. In practice, we do not expect this to be a severe problem due to the embarrassingly parallel nature of Monte-Carlo simulation.

D. SLA-aware co-scheduling of production traces

Having addressed the issues associated with generating predictions for production traces the question remains whether these modifications allow effective admission control for varying types of production workloads. In this section we investigate the efficacy of our approach for co-scheduling different production workloads while maintaining an SLA for both, on-demand and spot instances.

We perform the evaluation with production traces in two parts and pair up our production traces based on similar platform sizes. The first combination uses highly variable workloads, "A" as on-demand trace and "B" as spot instance trace. The specifications of the physical cloud platform are taken from "A", which contains 7 nodes with 12 cores each. We refer to this configuration as "A-B". We use the inverse notation "B-A" to describe co-scheduling of "A" as spot instances in addition to "B" as on-demand trace and "B"'s physical platform, which contains 7 nodes with 8 cores each. In both cases, we set the SLA to 0.01 eviction rate and we compare the results of the SLA-aware scheduler ("sla") with the SLA-unaware baseline scheduler ("base").

The second combination investigates the co-scheduling of the more constant workloads "C" and "D" with larger platforms

TABLE III: Results of co-scheduled workloads with production traces without SLA enforcement. In all cases the eviction ratio is greater than 0.01.

Baseline	A-B	B-A	C-D	D-C
admitted (on-demand)	0.977	1.000	1.000	0.997
admitted (spot)	1.000	0.850	0.943	0.963
evicted	0.013	0.024	0.016	0.013

TABLE IV: Results of co-scheduled workload with production workloads with SLA-aware scheduler, fulfilling the 0.01 eviction SLA (equivalent to a 0.99 survival ratio)

SLA-aware	A-B	B-A	C-D	D-C
admitted (on-demand)	0.977	1.000	1.000	0.999
admitted (spot)	0.884	0.757	0.491	0.278
evicted	0.009	0.000	0.002	0.006

of 31 nodes each. The experiments are defined analogously to the first part and we refer to them as “C-D” and “D-C”.

An important side-note is that A contains a number of instances requiring 12 cores each, while the platform of B only provides a maximum of 8 cores per node. This practically lowers the load impact of A as spot trace over its impact as on-demand trace on its native platform, as high-core-count instances are rejected by the scheduler due to the physical limits of the platform.

The results are summarized in Table III for the baseline, while the results for the SLA-aware scheduler are presented in Table IV. The SLA-aware scheduler meets the threshold, while the baseline scheduler misses in all cases. The modifications discussed in the previous section allow the SLA-aware scheduler to successfully handle production traces. The results are, however, close due to low overall utilization of the underlying cluster hardware. In fact, the mean utilization of on-demand and spot traces combined is 26.62 cores. This compares to a platform capacity of 84 cores for A and 56 cores for B. This degree of under-utilization is typical for clouds over-provisioned to meet peak demand. Reducing the under-utilization is a prime goal of co-scheduling. In order to demonstrate the efficacy of our approach in more resource constrained scenarios, we perform a platform down-scaling experiment in simulation in the next section.

E. SLA-aware co-scheduling with platform scaling

In this section we stress-test our approach to computing the conditional quantiles for spot-instance lifetimes in increasingly resource constrained environments. We use setups “A-B” and “C-D” again, but vary the size of the underlying platform from N to $N-3$ nodes for “A-B” and from N to $N-15$ in steps of 5 nodes for “C-D”. This corresponds to a reduction in node count by about half while the request rate stays constant and target SLA on eviction remains at 0.01. The inverse experiments “B-A” and “D-C” show similar results and are skipped for brevity.

Figures 6 and 7 show the results for scaled-down platforms of A-B and C-D, respectively. This experiment demonstrates the robustness of the approach in fulfilling its target SLA. While the baseline scheduler does not meet the SLA in any single case, the SLA-aware approach works consistently with visible differences in behavior.

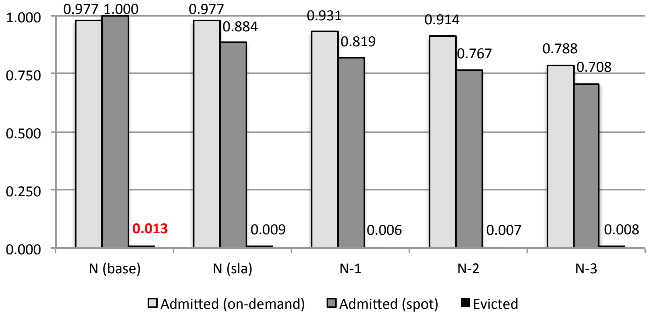

Fig. 6: Admission and eviction ratios of on-demand and spot instances for A-B down-scaled. Non-SLA base, marked ‘N (base)’ for $N = 7$ nodes in the first column compared with 0.01 SLA with full and reduced node counts $N = [7, 6, 5, 4]$ in the other columns.

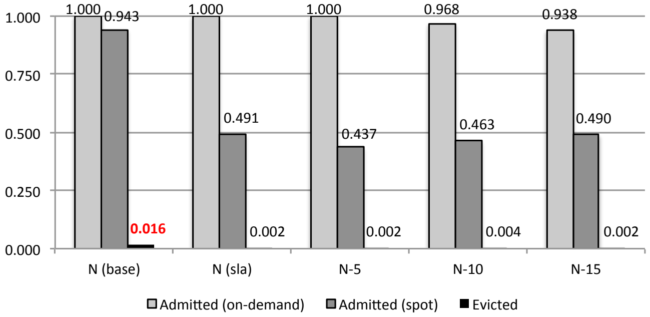

Fig. 7: Admission and eviction ratios of on-demand and spot instances for C-D down-scaled. Non-SLA base, marked ‘N (base)’ for $N = 31$ nodes in the first column compared with 0.01 SLA with full and reduced node counts $N = [31, 26, 21, 16]$ in the other columns.

Thus, while on-demand instance requests cannot completely be fulfilled in increasingly constrained environments, the on-demand rejection fractions for the production traces are small. In each figure, the column labeled N represents a replay of the production workload using the number of nodes and cores that were present when the trace was gathered (i.e. the production scenario). In the cases where our methodology offers an SLA on spot-instance lifetime in the same environment, the fraction of admitted on-demand instances is equal to the baseline.

As a result, we conclude that the success of the predictions for the real-world production traces is not due to a lack of utilization (i.e. an abundance of extra capacity) in over provisioned production clouds. Shrinking these clouds does cause some of the observed production workload to be rejected, but the generated predictions of the time-until-eviction remain valid.

An additional observation is that for the down-scaling experiments the ratio of admitted spot instances may increase as the cluster size decreases (e.g. “C-D $N-3$). An in-depth look at the simulation reveals that the rejected on-demand instances come in batches and with high core counts per instance. Their rejection due to capacity constraints leaves some additional capacity for spot instances. Furthermore, the inopportune placement of a spot instance does indeed lead to “fragmentation” and at times blocks the placement of large on-demand instances later on. While in our synthetic workloads the on-demand trace was completely unaffected by the spot trace, real-world traces are measurably impacted by the presence of spot instances.

IV. Related Work

Spot instances were first employed in 2009 as part of Amazon Web Services (AWS) [15]. Spot instances in public clouds are typically available at a rate significantly lower than that of on-demand instances as they allow providers to opportunistically utilize spare capacity. They do not, however, provide a guarantee (SLA) on their lifetime: spot instances can be terminated (evicted) at any time, whereas on-demand instances provide a 99.95% SLA on their availability once started.

Due to their unreliability but low cost they are typically used as opportunistic accelerators [16]. Previous research has also studied pricing models and user experience (Quality of Service) for services built entirely on spot instances. The authors in [17] model pricing as a mixture of multiple Gaussian distributions and reveal the challenges with modeling analytically, empirically observed phenomena in the cloud. Andrzejak et al. [2] model the trade-offs between spot instance bids and realized execution time to achieve probabilistic deadline guarantees for long-running, check-pointable jobs. In [3] the authors investigate a hypothetical service provider running a QoS-sensitive web service purely on spot instances, with a focus on revenue maximization. Similarly, [4] investigates a service running purely over spot instances and finds that existing SLAs capture only part of the observed variation typical in cloud environments.

In our work, we also service complex commercial workloads with spot instances, but side-step manual analytical modeling via Monte-Carlo simulation to provide a powerful new type of SLA on spot instance eviction probability for arbitrary jobs with bounded lifetime. While revenue and user experience depend on the specifics of the end-application, guarantees on eviction probability simplify reasoning about the system as a whole and allow providers to use it as foundation for custom SLA models.

V. Conclusions

Core economic drivers of cloud computing are the simplification of infrastructure management for clients, and increased utilization of hardware for providers by consolidation of different workloads. For private enterprise clouds, workload consolidation has its limitations due to smaller and more specialized user bases. A spot instance service model is promising, but comes with the challenge of reasoning about the implications of using evictable instances.

In this paper, we present a novel approach to providing spot instances with probabilistic SLAs on their minimum lifetime, which offers a generic solution to estimating costs and availability guarantees. We base the SLAs on estimating the time-until-eviction of spot instances from historical workload traces via Monte-Carlo simulation, which effectively enables cloud operators to offer an SLA on spot instance eviction probability. Users can request spot instances for a fixed lifetime with quantitative bounds on eviction probability and receive an immediate response from the cloud provide of whether it can provide the desired quality of service, or not. We demonstrate the robustness of our approach to providing guarantees with commercial workload traces and evaluate its efficiency for increasing utilization via workload co-scheduling.

Acknowledgments

This work was funded by NSF (CNS-0905237, CNS-1218808, and ACI-0751315) and NIH (1R01EB014877-01).

References

[1] "Amazon Web Services home page," http://aws.amazon.com/.

[2] A. Andrzejak, D. Kondo, and S. Yi, "Decision model for cloud computing under sla constraints," in Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on, Aug 2010, pp. 257–266.

[3] M. Mazzucco and M. Dumas, "Achieving performance and availability guarantees with spot instances," in High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on, Sept 2011, pp. 296–303.

[4] J. Chen, C. Wang, B. B. Zhou, L. Sun, Y. C. Lee, and A. Y. Zomaya, "Tradeoffs between profit and customer satisfaction for service provisioning in the cloud," in Proceedings of the 20th International Symposium on High Performance Distributed Computing, ser. HPDC '11. New York, NY, USA: ACM, 2011, pp. 229–238. [Online]. Available: http://doi.acm.org/10.1145/1996130.1996161

[5] "Monte Carlo Method," http://en.wikipedia.org/wiki/Monte_Carlo_method.

[6] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The eucalyptus open-source cloud-computing system," in Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on. IEEE, 2009, pp. 124–131.

[7] Eucalyptus Systems Inc., "http://www.eucalyptus.com," Jun. 2013. [Online]. Available: http://www.eucalyptus.com

[8] R. Wolski and J. Brevik, "http://www.cs.ucsb.edu/~rich/workload," Jun. 2013. [Online]. Available: http://www.cs.ucsb.edu/~rich/workload

[9] "OpenStack," [Online; accessed Aug-2014] "http://www.openstack.org/".

[10] "CloudStack," [Online; accessed Aug-2014] "http://cloudstack.apache.org/".

[11] R. Wolski and J. Brevik, "Using parametric models to represent private cloud workloads," University of California, Santa Barbara, Tech. Rep. UCSB-CS-2013-05, August 2013, http://128.111.41.26/research/tech_reports/reports/2013-05.pdf.

[12] Y. Mansour, "Regret minimization and job scheduling," in SOFSEM 2010: Theory and Practice of Computer Science, ser. Lecture Notes in Computer Science, J. van Leeuwen, A. Muscholl, D. Peleg, J. Pokorn, and B. Rumpe, Eds. Springer Berlin Heidelberg, 2010, vol. 5901, pp. 71–76. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-11266-9_6

[13] R. Wolski and J. Brevik, "Using Parametric Models to Represent Private Cloud Workloads," IEEE Transcations on Services Computing, vol. 4, no. 7, pp. 714–725, October 2014.

[14] A. Pucher, E. Gul, C. Krintz, and R. Wolski, "Using Trustworthy Simulation to Engineer Cloud Schedulers," in Cloud Engineering (IC2E), 2015 IEEE International Conference on, March 2015.

[15] "Announcing Amazon EC2 Spot Instances," [Online; accessed Aug-2014] "http://aws.amazon.com/about-aws/whats-new/2009/12/14/announcing-amazon-ec2-spot-instances/".

[16] N. Chohan, C. Castillo, M. Spreitzer, M. Steinder, A. Tantawi, and C. Krintz, "See spot run: Using spot instances for mapreduce workflows," in Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing, ser. HotCloud'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 7–7. [Online]. Available: http://dl.acm.org/citation.cfm?id=1863103.1863110

[17] B. Javadi, R. Thulasiram, and R. Buyya, "Statistical modeling of spot instance prices in public cloud environments," in Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on, Dec 2011, pp. 219–228.