Using Spot Instance SLAs for Reliable Cloud Federation

UCSB Technical Report Number 2015-02

Alexander Pucher, Rich Wolski, Chandra Krintz
University of California, Santa Barbara
Santa Barbara, CA 93106
{pucher, rich, ckrintz}@cs.ucsb.edu

ABSTRACT

Spot instances are a commonly offered by IaaS cloud providers to opportunistically utilize spare capacity and meet temporary user demand for additional resources at low cost. Although the availability of service SLAs is a core paradigm of cloud computing, spot instances typically come without any service quality guarantees. We aim to extend the spot instance service to provide SLAs for eviction probability, based on the user estimate of the maximum expected instance lifetime. In addition to providing users with better usability and ahead-of-time quality of service guarantees, this statistical certainty also opens the door to cloud-to-cloud federation of workloads. For this federation to be possible, however, the statistical guarantees must be adhered to strictly, for a wide range of real-world workloads, at cloud scale.

To this end, we propose a new approach to providing SLAs on the time-until-eviction for spot instances. We employ Monte-Carlo simulation to compute the quantiles of the conditional distributions of future spot instances for different available capacity levels. An IaaS cloud scheduler then uses these quantiles to determine when to provision federated requests in order to maintain an SLA at a specific target eviction probability for spot instances. We investigate the reliability of such SLA enforcement using synthetic and real-world traces, test its viability for cloud-to-cloud workload federation, and provide an in-depth analysis of trade-offs and cost factors of such federation.

Categories and Subject Descriptors

C.4 [**Computer Systems Organization**]: Performance of Systems; D.4.1 [**Operating Systems**]: process management—*Scheduling*

Keywords

Cloud Computing, Federation, Scheduling, Monte-Carlo simulation, Spot Instances, Quality-of-Service, Service-Level-Agreement

1. 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.

Further, to promote scaling, cloud resources are commoditized. Users cannot request specific machines, disks or network switches. Instead, each resource is characterized by a *Service Level Agreement* (SLA) that defines its performance, reliability and (occasionally) privacy characteristics. Thus cloud users transact with cloud web services for "levels of service" from the resources they wish to provision. The cloud either delivers these resources with at least the minimum level of service described in the SLAs or it rejects the request interactively in the same way an e-commerce site either accepts or rejects a purchase transaction – there is no queuing for resources as in large-scale batch systems.

In this paper, we examine the feasibility of using two classes of resources requests – *on-demand* requests and *spot* requests – to implement the federation of cloud workloads [4, 26, 10] for greater resource utilization. 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, 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 (possibly from another cloud) subject to a statistical guarantee on minimum time until pre-emption. In particular we

- investigate and demonstrate that it is possible to provide statistical guarantees on minimum spot instance lifetimes using production private cloud workload traces, and
- we detail the effectiveness of co-scheduling "native" on-demand workloads with "foreign" spot workloads as a way of using cloud federation to utilize otherwise unused resource capacity.

Our work focuses on private clouds because high quality,

scalable implementations are available [8, 22, 7]. Public clouds such as Amazon AWS [1], Google Cloud Platform [11], IBM SoftLayer [14] and Rackspace Cloud [24] hold both the details of their respective implementations and the quantitative characteristics of their respective workloads as trade secrets. Thus it is difficult to understand what techniques may ultimately be feasible.

In contrast, Eucalyptus [21] is a commercially successful private cloud the source code for which is available freely as open source [8]. Production Eucalyptus customers (as a way of supporting the Eucalyptus community) have chosen to make some of their internal enterprise workloads available in anonymized form [28, 30, 29, 23]. Thus, using Eucalyptus, it is possible to understand both the engineering feasibility associated with federation using spot instances as well as the effects on real-world production deployments. At the same time we note that Eucalyptus shares many architectural characteristics with other open source private cloud platforms such as Open Stack [22] and CloudStack [7]. Thus we believe the results contained herein to be generally applicable to the federation of private clouds in production computing settings.

This work complements previous investigations of federation by Grid and Supercomputing systems [9]. These previous efforts look at load sharing through federation as a way of increasing utilization but they do not do so subject to a verified SLA. Similarly, previous work studying the public cloud spot markets has investigated techniques that help users reason about spot instance lifetimes and pricing [2, 5, 19, 15]. Other work employs spot instances to facilitate federation but focuses on optimizing revenues (i.e. to tolerate evictions) or on the complexities of spot prices [26, 15].

In contrast, our work focuses on 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, but resources are often over-provisioned to ensure an acceptable user experience in terms of response time and available capacity.

To this end, we investigate an approach based on the use of a Monte-Carlo style [20] simulation to compute the quantiles (i.e. percentiles) of the conditional distributions of future spot instance lifetimes. This non-parametric approach generates distributions that are empirical and conditioned on the capacity occupied by both on-demand and previous spot instance workload. These quantiles then serve as the minimum future time-until-eviction guarantee provided to the request as an SLA at a specific target probability.

For example, the 0.01 quantile on the distribution of spot instance lifetimes represents the lifetime that is smaller than 99% of all lifetimes in the distribution. Thus, if that quantile can be estimated accurately and the estimate is stable for future requests, it can serve as a statistically guaranteed lower bound (with probability 0.99) that the next spot instance will experience. This estimate must be conditioned, however, on the capacity occupied by previously accepted workload at the time it is generated.

To be feasible, it must be possible to compute and recompute these distributions "on-the-fly" with sufficient regularity to compensate for changes in the underlying workloads. That is, we expect that cloud workloads are non-stationary time series that experience change points. The approach must produce distributions periodically from an on-going history of workload measurements to account for these change points. Also, the workload characteristics must be stable for periods of time that are "long" relative to the lifetimes of individual instances or the SLAs will not be met.

Our evaluation is based on trace-driven simulation with synthetic and recorded traces from IaaS clusters deployed in production [28]. 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-to-eviction estimates via Monte-Carlo simulation. We then apply these estimates in a cloud scheduler that is capable of maintaining an arbitrary SLA for maximum eviction probability of spot instances in a local cloud.

Moving to a realistic setting with a federating remote cloud and a local destination cloud, we extend our approach to production cluster hardware configurations and real world historic utilization traces. We discuss the modifications necessary to address sparseness of information in historical utilization data and challenging distributional properties of production traces, such as varying auto-correlation and the diverse population of instance types in real-world settings.

Our experiments show that it is possible to maintain SLAs on spot instance eviction probabilities in relatively adverse production settings with large, infrequent load spikes and varying workload patterns. We further demonstrate that this effectively enables the execution of the entire workload of one cloud on another using only spot instances, and that SLA-enforcement remains robust even in heavily resource-constrained environments. We conclude our investigation with a discussion of cost factors and trade-offs made in offering SLAs on minimum spot instance lifetimes.

The remainder of the paper is organized as follows. Section 2 describes or Monte-Carlo simulation and evaluation method in detail. Section 3 presents our results for federating synthetic and recorded production workloads between clouds, stress-tests the ability of our approach to make accurate predictions in resource-constrained environments and investigates the costs and trade-offs involved in offering SLAs on minimum spot instance lifetimes. We then overview related work in Section 4 and conclude with perspectives for future work in Section 5.

2. METHODOLOGY

Our goal is to federate workloads between private IaaS clouds using spot instances. In our view, a local IaaS cloud that is available for federation, provisions "native" instances on-demand and does not evict them (as is the case for most public and private IaaS clouds today). Remote IaaS clouds make federated requests to the local IaaS cloud, which either accepts (provisions) or rejects the requests as spot instances that can be evicted at any time.

To enable users and federation agents in remote IaaS clouds to reason about their use of such functionality, we also define a service level agreement (SLA) that provides a probabilistic guarantee on the minimum duration that a spot instance is likely to run before it is evicted, i.e. its minimum lifetime. This methodology defines two new capabilities:

- a method for predicting minimum lifetime of spot instances based on a target SLA, that uses historical observations of previous instance behavior, and
- a cloud scheduler that uses this predicted time-to-eviction for the specified SLA to perform admission control for

spot instances in a federated IaaS cloud.

To predict minimum lifetime, we have developed a prediction utility that can be incorporated into any IaaS cloud. The utility 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 federated spot instances (e.g. there is a 95% or 99% confidence bound on the likelihood that the instance will not be evicted), to make a prediction of minimum lifetime for each level of available capacity.

We assume that all federated requests are accompanied by *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 type and maximum lifetime submitted by the user, and (iii) the currently available capacity of the the system, to decide whether to schedule a federated request as a spot instance. The scheduler evicts spot instances if/when an on-demand instance request is made and the IaaS cloud has insufficient capacity to service the request

To evaluate our methodology (c.f. Section 3), we employ validated, trace-based simulation [23] using the Eucalyptus open source IaaS system [21] and synthetic and real-world traces from production Eucalyptus deployments. We next detail our scheduling and federation models, minimum lifetime prediction, eviction policy, and evaluation metrics in the subsections that follow.

2.1 Scheduling Model

The core responsibilities of the scheduler in an IaaS cloud are admission control, placement and pre-emption. Extensive research has shown that schedulers and their placement strategies play an important role for overall responsiveness and utilization of the cloud [13, 27, 25]. In this work, however, we focus on admission control and pre-emption, i.e. spot instance admission and eviction. To meet its time-to-eviction probability (SLA), the scheduler's admission control in this setting must be conservative, i.e. to reject federated requests that are likely to be evicted before they are terminated by the user. However, if admission control is overly conservative, unused capacity is wasted.

Instance requests (to either start or stop an instance) are routed to a scheduler (as implemented by Eucalyptus [21], Open Stack [22], and Cloud Stack [7]) which handles admission control and placement of instances on physical resources in a cluster of "nodes." IaaS clouds typically defines "instance type" 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. Instances cannot be split across multiple physical nodes. In this study, we use a simple first-fit placement in our scheduler that attempts to assign instances to nodes based on core count. If a on-demand instance is requested, and the scheduler cannot find a node with sufficient space to run the instance, 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. Slots do not span nodes, however, and 2 nodes with 1 core available each, do not translate to a 2-core slot. This distinction between available cores and *available slots* becomes important when generating time-to-eviction estimates for different instance sizes and different cluster load levels.

We model a workload in our system as a continuous stream of instance start and stop requests. Start requests come with a slot size (core count) and are flagged either "on-demand" or "spot" (federated). Federated requests also consist of the maximum eviction probability (SLA) and the maximum instance lifetime (execution duration) expectation as described previously. If there is insufficient capacity or the SLA of a federated request cannot be fulfilled, the scheduler rejects the request and does not start the instance.

2.2 Federation Model

Our scheduler offers differentiated classes of service for local workloads (on-demand instances) and federated workloads (evictable spot instances) for the cluster of nodes it controls. Federated workloads originate from remote clouds which can use this functionality to temporarily extend their available capacity by federating work to another IaaS cloud. Admitting federated requests as spot instances with SLA guarantees on eviction probability enables an IaaS cloud to utilize available capacity that would otherwise unused. SLA guarantees provide the owners of the federated workload with a probabilistic guarantee on the minimum amount of time an instance will run if admitted. If the SLA cannot be fulfilled, the request will be rejected immediately. This "fail-fast" approach allows users or remote clouds to decide whether they wish to wait and resubmit if and when conditions improve, or choose a different class of instance (e.g. the on-demand class) at a possible additional cost.

2.3 Prediction

Accurate prediction of the time-to-eviction for spot instances is the core prerequisite for concurrently meeting the user's SLA and maximizing cloud utilization. Past work has shown that cloud workloads can be highly variable and may not be easily described by single well-known distributions [29]. To address this problem we resort to Monte-Carlo simulation to generate the empirical distribution of expected spot instance lifetimes. However, we note that the time to 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 for each level of possible occupancy level.

For example, a cloud with 100 cores has 101 different possible occupancy levels: from 0 cores occupied to 100 cores occupied. In practice, not all levels occur. To deal with this sparseness, we employ linear approximation between the quantiles of the distributions from surrounding capac-

ity levels for any excluded by the simulation.

We use quantiles of these distributions to quote the expected lifetime to the scheduler during the admission control phase based on the available capacity, i.e. capacity unoccupied at the time the request is fielded. If the instance is expected to be evicted with a higher probability than specified by its SLA, it is rejected (not admitted).

The Monte-Carlo simulator produces the empirical distribution of the time-to-eviction from the start of a spot instance. This distribution is further conditioned over the number of available slots at the time of the start request and the slot size of the instance. A single time-to-eviction sample is generated by re-running the recorded trace up until a random time stamp, then injecting a virtual spot instance, and continuing to process the remaining recorded requests. When the virtual spot instance is evicted eventually by an on-demand request, its lifetime is recorded. We repeat this process 10000 times in this work and assign each sample to a bucket based on the number of available instance slots at the time of the virtual instance start. We then extract quantiles from the samples in each bucket independently.

For example, assume the current number of available 2-core instance slots equals 4 (i.e. 92 of 100 cores of the cloud are occupied) and a spot instance request arrives with a 0.01 SLA and a maximum lifetime of 1000 seconds. This maximum lifetime will be compared to the 0.01 quantile of the samples for 2-core instances in the cases where 4 slots were available (i.e. conditioned on 92 cores occupied). If the 0.01 quantile is greater than 1000 seconds, the spot instance will be admitted and placed in the cluster and the number of 2-core slots will decrease to 3. Otherwise, the spot instance request is rejected immediately. The request would also have been rejected if the number of available slots for the spot instance is exhausted. That is, if a 12-core spot instance request arrives, but no node in the cloud has room for a 12-core instance, the request will be rejected.

2.4 Eviction Policy

The eviction policy affects the probabilistic bounds of the predictions made via our simulation. With an arbitrary eviction scheme, the eviction probability of a spot instance is not necessarily fixed at admission, but may vary wildly as new requests arrive at the cloud. In this paper we use a "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 decision making problem [16]. That is, the amount of work that is lost because of an eviction is minimized, which improves the user experience.

This policy has the additional benefit of being conservative with respect to the quoted SLA. Specifically, as additional spot instances enter the system it becomes less and less likely for an existing instance to be evicted as there is a growing number of new, lower priority instances in the system that can be evicted. Thus the SLA that is generated describes the probability that the spot instance will be evicted before its maximum lifetime *while it is the youngest job*. As time passes and other spot instances arrive, the probability that an older spot instance will be selected for eviction generally decreases (assuming the lifetime distributions are stable). Thus the YJF eviction policy further ensures that the quoted SLA will serve as a lower bound although the tightness of that bound relaxes as the instance continues to execute.

2.5 Evaluation Metrics

We evaluate the efficacy of our approach using validated, trace-based simulation using 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. We use one trace to simulate on-demand workload and a second trace for the federated workload. 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 has highest priority. After the SLA is fulfilled, a high number of completed spot instance requests is desirable to maximize utilization.

The admission ratio captures the decision making capability of the scheduler as well as the overall capacity of the system. A spot instance will be admitted *only if* the scheduler predicts that it will be able to exceed the requested lifetime associated with the spot instance with the probability associated with the SLA.

The eviction ratio captures the degree to which the probabilistic guarantees are being met among all spot instance that the scheduler has admitted. If, for example, the scheduler is offering a 0.01 SLA, the fraction of admitted spot instance that are evicted should be less than or equal to 0.01 over the entire population of spot instances.

3. 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 workload traces [28]. Our simulated clouds use a simple first-fit placement policy, the baseline or prediction-based admission control and a "youngest-job-first" eviction policy for spot instances.

The system is set up to simulate a single cloud installation. The local workload is modeled as consisting of two different classes of instances:

- **on-demand instances** which must be accepted if there is sufficient capacity in the cloud (potentially by evicting spot instances) and which cannot be evicted themselves, and
- **spot instances** which are launched opportunistically on unused cloud capacity, but which will be terminated without warning (i.e. evicted) if doing so is necessary to free enough capacity to accept the on-demand request.

Thus, the model is one in which spot instances "bottom feed' the available capacity but can be pre-empted by on-demand instances if the pre-emption makes accepting the on-demand request possible. Note that in our model, a spot-instance

Table 1: Parameters of synthetic log-normal on-demand and spot instance workloads

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

eviction is *only* triggered when an on-demand request would otherwise be rejected due to a lack of capacity. This two-level classification is designed to ensure that the cloud resources remain as heavily utilized as possible while giving priority to the on-demand class at all times.

We also assume that the users of spot instances request them along with a maximum lifetime so that the scheduler can determine whether an instance can be accepted without violating the SLA. Thus when a spot instance is submitted, the scheduler determines (by making a time-sensitive prediction of the time-to-eviction for that instance) how long it can guarantee the instance will be able to execute before it is evicted with a probability determined by the SLA. If the requested maximum lifetime is greater than that predicted lifetime, the spot instance start is rejected by the scheduler immediately.

Our goal is to determine the extent to which the users of the spot-instance class can be given a statistical guarantee of the lifetime that the instance will experience before it is evicted. Note that because on-demand instances pre-empt spot-instances but not vice versa, the SLA given to the on-demand class (in terms of instance lifetime) is unaffected by the presence of spot instances. We measure the performance of the admission control in terms of admission and eviction ratio of spot instances. In our model the SLA defines an upper bound (maximum) on the eviction ratio of admitted spot instances given their a-priori known maximum lifetime.

To understand the degree to which spot-instances (governed by a statistical guarantee of minimum lifetime offered as an SLA) could be used to implement workload federation, we use cloud workload traces to understand what happens when the workload requested on one cloud is run using only spot instances on another (while it runs its own on-demand workload). The investigation begins with synthetic workloads using known parameters that have been set to ensure that results could be obtain in a timely manner. It then continues with production workload traces garnered from commercial cloud entities [23, 28]. Finally, because the commercial clouds may have been over provisioned by their owners, we investigate the impact of reducing the hardware capacity. That is, to stress test the algorithm's ability to make predictions in resource-constrained environments while using production workload traces.

3.1 Federation of synthetic traces

To show the theoretical efficacy of the approach we compare a federation scenario with an SLA-unaware scheduler and the SLA-aware scheduler using multiple different SLA levels. The initial setup uses a single platform (IaaS cluster configuration), an on-demand trace and a federated spot instance trace. 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 federated spot instance trace. We use a log-normal distribution to approximate the long

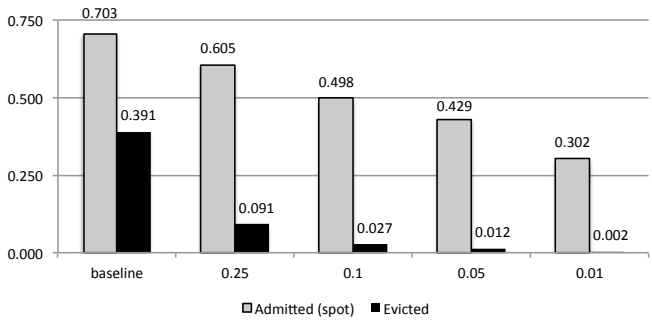

Figure 1: Ratio of admitted and evicted spot instances with synthetic log-normal traces.

tailed empirical distribution of instance life times. The parameters for generating the synthetic traces can be found in Table 1.

For this first experiment, all instances have uniform capacity requirements of 1 core each. The Monte-Carlo simulation is re-run every 6 hours of simulated time on the entire trace observed by the scheduler so far. The spot trace starts 24 hours after the on-demand trace to allow for a warm-up period and the experiment covers a total time period of 240 hours (10 days). The simulator takes approximately 30 seconds to compute the empirical distributions needed to describe the 32 different occupancy levels.

We 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 levels, starting with the no-guarantees baseline on the left and then increasingly stringent SLAs of 0.25, 0.10, 0.05 and 0.01 maximum eviction ratio to the right. The y-axis shows the fraction of admitted 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 guaranteed level), at the cost of preemptively rejecting an higher fraction of spot instances for stricter SLAs. The measured SLAs are in fact better than the target SLAs, since the estimates of the time-to-eviction made by the simulation are conservative (c.f. Subsection 3.2.1).

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 already produces a comparably low 9% eviction rate. Subsequent decreases in the demanded maximum eviction rate of spot instances decrease the number of admitted spot instances as well, but consistently achieve the lower target rates.

The experiment demonstrates the potential of a simulation driven approach to successfully enforce guaranteed levels of eviction probabilities in a controlled environment. If these results can be extended to production environments, it could effective enable ahead-of-time eviction probability guarantees for spot instances with well-defined lifetime limits. Spot instances are either rejected at start-up with fail-fast semantics or have statistical certainty about their completion ratio which simplifies users' reasoning about the system.

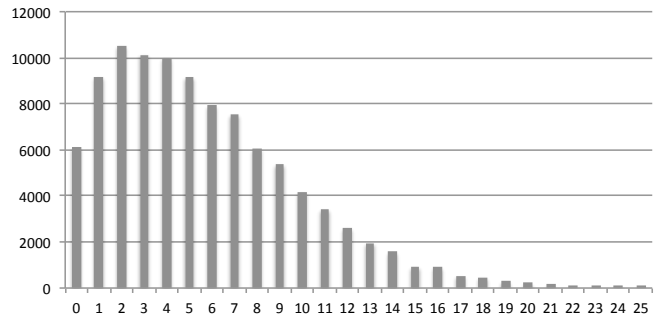

Figure 2: Number of time-to-eviction samples per available-slots bucket.

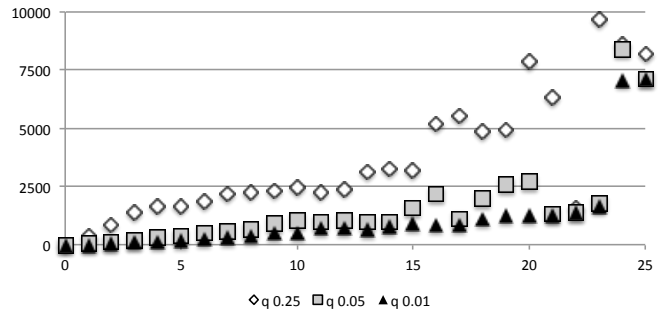

Figure 3: Quantiles of time-to-eviction per available-slots bucket.

3.2 Conditional Distributions and Sample Size

As described in Section 2.3, the prediction is generated from an empirical distribution conditioned on the number of available instance slots at the time of the request. 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. Note 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. Thus to compute the conditional lifetime conditioned on 1 core being available, the sample of cases where only 1 core is available may be small.

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.

Figure 3 shows the quantiles of the conditional distribution of times-to-eviction. The x-axis again shows the condition, while the y-axis indicates the time-to-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 of the graph (for 15 free slots) 0.05 of the time-to-eviction samples are 1500 seconds or less, and 0.25 of them are 3200 seconds or less.

A holistic 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 high SLAs, such as 0.05 or 0.01. With increasing slot count (decreasing cluster utilization) a smoothly changing, and mostly increasing estimate of the time-to-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).

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. 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.

3.2.1 *Conservative estimates*

We note that for many of the distributions, the quantile estimate is conservative. That is, the observed fraction of SLA violations is far lower than the SLA could permit and still remain valid. The result of this conservativeness is that fewer spot instances are admitted than could be if a more accurate estimate were available, however, the SLA itself is not violated.

We speculate that the effect is due to the long-tailed nature of the log-normal distribution. Compared to a more symmetric distribution, because a relatively few instances have long lifetimes, a larger number of instances have relatively short lifetimes. Thus, a small sample size for some conditions in the Monte-Carlo simulation leads to a conservative estimate of the quantile since fewer "long" jobs are likely to have occurred.

3.3 Challenges of 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 [30, 29, 23]. and the traces themselves are available as part of a collection from [28]. Table 2 shows the mapping of data sets from the collection to experiments in

Table 2: Mapping of production traces from the data set collection [28] 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

this paper, together with a short description of their workload and platform properties. We chose this diverse set of workloads and platform sizes to investigate the applicability of the simulation-based approach to SLA enforcement.

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 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, the real world traces start abruptly without a ramp-up period. As with the synthetic traces we start the spot instance trace with a delay of 24 hours to allow the scheduler to warm up.

A visual inspection of the bursts in the real-world traces shown in Figure 4, Figure 5 and Figure 6 (trace A similar to B) 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 correctly capture this auto-correlation via replaying the actual observed trace in our Monte-Carlo

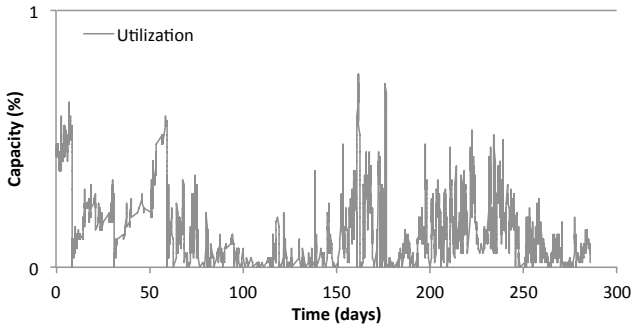

Figure 4: Production trace B as executed on its native platform shows highly variable load and bursts of large requests as well.

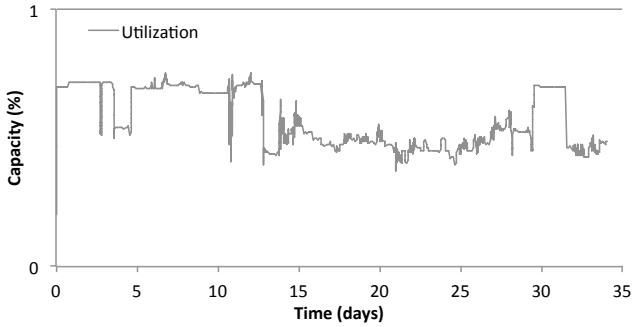

Figure 5: Production trace C as executed on its native platform shows a mixed pattern of load with constant plateaus and periods with higher variability.

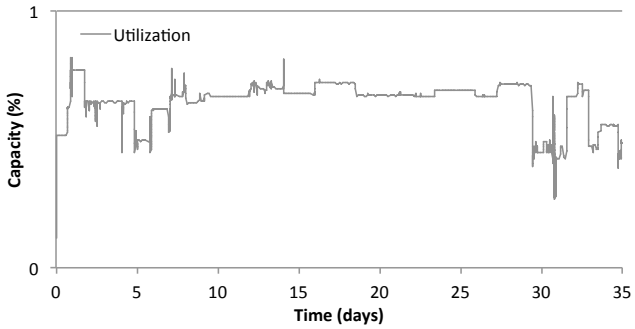

Figure 6: Production trace D as executed on its native platform shows a mostly constant load with a few spikes.

simulation rather than re-sampling the input distribution. The fundamental assumption of our Monte-Carlo approach is that the behavior of the trace can be modeled as a distribution between re-computation intervals. With this, we expect unprecedented bursts or load-levels to cause SLA violations in the short run that are compensated for by more conservative admission control in the long run.

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 3.5) 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. Fortunately, 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.

3.4 Federation of production traces

We perform the evaluation with production traces in two parts. The first part 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 the federation of "A" as spot instances onto "B" as on-demand trace and 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 part of the evaluation investigates the federation of the more constant workloads "C" and "D". 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.

Table 3: Results of workload federation with production workloads 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 4: Results of workload federation 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

The results are summarized in Table 3 for the baseline, while the results for the SLA-aware scheduler are presented in Table 4. The SLA-aware scheduler meets the threshold, while the baseline scheduler, unsurprisingly, 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 cloud-to-cloud federation. 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.

3.5 Federation with platform scaling

In this section we explore the limits of the Monte-Carlo approach to computing the conditional lifetime quantiles for spot-instance lifetimes and its robustness in increasingly resource constrained environments. In this experiment we execute "A-B", "B-A", "C-D" and "D-C" again, but vary the size of the underlying platform from N to $N-3$ nodes for "A-B" and "B-A". For the larger platforms in "C-D" and "D-C" where platform C and D each have 31 nodes, we vary the numbers from N to $N-15$ in steps of 5 nodes. The target SLA remains at 0.01.

Figures 7 and 8 show the results for scaled-down platforms of A-B and B-A, respectively. Figures 9 and 10 shows the same for the C-D and D-C configurations. This experiment demonstrates the robustness of the simulation in fulfilling its target SLA. While the baseline scheduler does not meet the SLA in any single case, the SLA-aware approach succeeds in all cases but one ("B-A N-1") and in this case it is close (the eviction fraction is 0.012 when the target fraction should be no bigger than 0.01). An in-depth look a the single miss of the SLA-aware scheduler shows a single digit number of evictions. This is an artifact of our constrained trace duration and is expected to even out over longer duration. That is, it is so close to the 0.01 target that we believe, in a longer trace, it would eventually drop below the 0.01

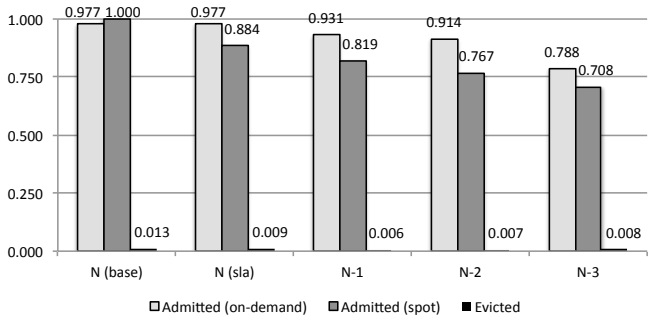

Figure 7: Admission and eviction ratios of on-demand and spot instances for A-B down-scaled. Non-SLA base, marked 'N(base)' for $N = 7$ nodes in the first column compared with 0.01 SLA with full and reduced node counts $N = [7, 6, 5, 4]$ in the other columns.

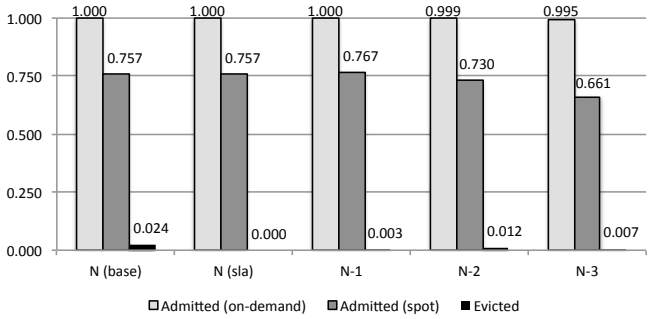

Figure 8: Admission and eviction ratios of on-demand and spot instances for B-A down-scaled. Non-SLA base, marked 'N(base)' for $N = 7$ nodes in the first column compared with 0.01 SLA with full and reduced node counts $N = [7, 6, 5, 4]$ in the other columns.

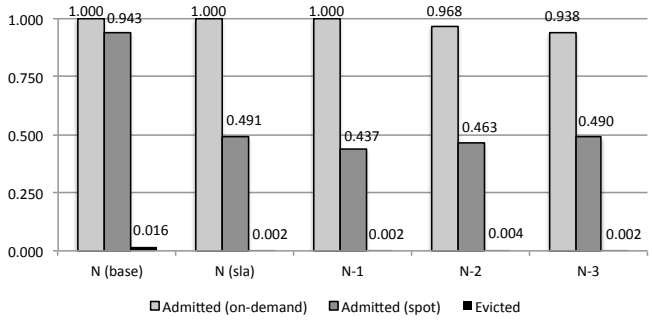

Figure 9: Admission and eviction ratios of on-demand and spot instances for C-D down-scaled. Non-SLA base, marked 'N(base)' for $N = 31$ nodes in the first column compared with 0.01 SLA with full and reduced node counts $N = [31, 26, 21, 16]$ in the other columns.

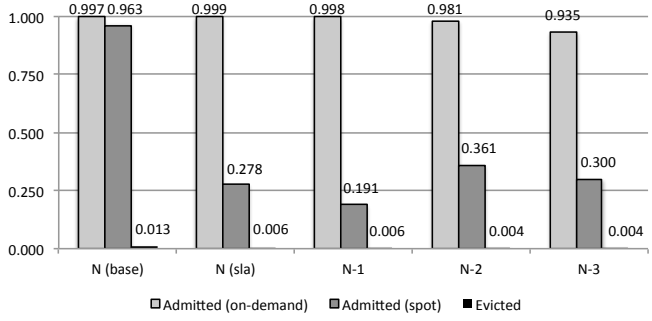

Figure 10: Admission and eviction ratios of on-demand and spot instances for D-C down-scaled. Non-SLA base, marked 'N(base)' for $N = 31$ nodes in the first column compared with 0.01 SLA with full and reduced node counts $N = [31, 26, 21, 16]$ in the other columns.

target threshold.

Thus, while on-demand instance requests cannot completely be fulfilled, the on-demand rejection fractions for the production traces is 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, the fraction of admitted on-demand instances that are admitted is 0.997 or higher. Indeed, the fraction practically is 1.0 in all cases *except* in Figure 7. Here, both the SLA-aware scheduler (our methodology) and our reproduction of the extant Eucalyptus scheduler (marked "N(base)") reject approximately 2.3% of the on-demand instances. The Eucalyptus scheduler, at the time these traces were generated, had a bug due to a race condition that would cause it to over-provision cores on occasion which our trace-driven simulator does not reproduce. Thus we are unable to determine if this loss is due to fragmentation or due to our trace-driven simulator's rejection of on-demand instances that the *in situ* scheduler would have accepted due to the bug.

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. "extra space") 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-to-eviction remain valid.

Another interesting observation is that for the down-scaling experiments the ratio of admitted spot instances at times increases as the cluster size decreases, for example "C-D $N-3$âĂŹâĂŹ and "D-C $N-2$âĂŹâĂŹ. An in-depth look at the simulation unveils that the rejected on-demand instances came in batches and with high core counts per instance. Their rejection opens up substantial capacity in the cluster. Furthermore, the inopportune placement of a spot instance can lead to a scattered placements of other, typically small, on-demand instances, which block 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.

3.6 Costs of SLA enforcement

Finally, we investigate the costs involved in providing probabilistic SLAs for maximum spot instance eviction probability. Obvious cost factors are the effort spent on collecting utilization traces and the periodical execution of the Monte-Carlo simulation. Historical traces are collected in the form

of log files already, and hence, do not generate additional effort. The Monte-Carlo simulation is light-weight and can be tuned to trade off accuracy for computational cost. Considering the experiments in this paper executed on commodity laptop hardware in reasonable time frames, we argue that this overhead in insignificant compared to the scale and compute-power available in IaaS clouds.

A comparison between SLA-aware scheduling and the SLA-unaware baseline also gives an indication of the costs of providing a spot instance SLAs over a no-guarantees approach. While the SLA-unaware approach does not meet the 0.01 threshold, it still achieves a respectable 0.05 to 0.02 with our specific workloads. It admits and completes a substantially higher absolute number of spot instance requests, which would have lead to better utilization than using the SLA-aware approach. Hence, with the benefit of hindsight, the provider could have offered a 0.05 SLA without SLA-aware scheduling and QoS-driven rejections. At no point this SLA is guaranteed however – the 0.05 value is simply the number observed from the trace. Alternatively, in the SLA-aware approach the Monte-Carlo simulation is run repeatedly to allow the system to adapt to changing conditions.

To provide the benefits of usability and statistical certainty of an SLA the scheduler must forego opportunities for additional spot instance starts, making the rejections due to conservative estimates an opportunity cost. The most drastic example is the down-scaling experiment "D-C" shown in Figure 10. The baseline scheduler admits 0.96 of all spot instances and merely evicts a 0.02 fraction of these, whereas the SLA-aware scheduler meets the 0.01 SLA but at the cost of only admitting a mere 0.28 of all spot instances. Although the difference seems substantial, it shrinks when the actual amount of work done is considered. In this specific case, the SLA-aware scheduler still completes a 0.54 fraction of the total spot work (defined as the completed spot instances' $lifetime * cores$). The "C-D" experiment shows similar properties.

We argue that this cost is acceptable in an economic context, as spot instance SLAs and federation offer streams for additional utility that would not be accessible without guarantees. Even more so, as the provider can still offer no-guarantees spot instances in addition to on-demand and SLA-spot instances to maximize utilization.

4. RELATED WORK

Spot instances were first employed in 2009, by the "de facto" standard IaaS system Elastic Compute Cloud (EC2) as part of Amazon Web Services (AWS) [3]. Spot instances allow IaaS users to rent virtual machines at a variable hourly rate that is dictated by a spot market (VM supply and demand) and that is bounded by a user's upper bound (bid). Spot instance prices are typically significantly lower than *on-demand* instances but do not provide a guarantee (SLA) on their lifetime: spot instances can be terminated (evicted) at any time. 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 [6]. Spot instances enable IaaS providers to utilize temporarily available resource capacity.

Another technique for better utilizing IaaS cloud capacity is cloud federation. Cloud federation has been the focus of much research in the literature [4, 26, 10]. Inter-cloud federation [4] proposes an architectural framework and defines the mechanisms and policies for distributing load across multiple clouds using dynamic coordination to achieve locality and high QoS levels. The idea of federating computer resources is not new and is successfully employed in many Computational Grid systems [9]

Goiri et al. [10] develop theoretical models for cloud providers to maximize revenue via opportunistic in- or outsourcing of requests based on price and utilization. In [26] the authors specifically leverage spot instances to facilitate federation. In particular, they investigate optimal policies for workload federation among providers based on the proportions of spot instances in the user workload and relative scarcity of resources within the federation. Their investigation, however, relies purely on synthetic workloads to explore properties of cloud federations. Our work is similar in that it studies cloud federation via spot instances, but uses production-grade workload traces to evaluate the real-world efficacy of the federation approach.

Other researchers have studied pricing models and user experience (QoS) for spot instances. The authors in [15] model pricing as a mixture of multiple Gaussian distributions. This work reveals the challenges with modeling analytically, empirically observed phenomena in the cloud and shows how significantly complexity grows with the addition of cloud attributes. Our work circumvents the necessity for fitting a model to the cloud workload by using Monte-Carlo replay of recorded historical traces. Andrzejak et al. [2] model the trade-offs between spot instance bids and realized execution time to achieve probabilistic deadline guarantees for long-running jobs with check-pointing. While the approach applies to generic batch jobs, the it relies on the ability to checkpoint progress periodically. Our approach similarly starts out by providing probabilistic bounds on spot instance lifetimes based on empirical traces, but diverges by providing guarantees for an entire population of instances rather than for meeting job deadlines. Additionally, our work does not require check-pointing abilities, and hence, applies to arbitrary jobs that can be described with an a-priori known maximum duration.

In [19] the authors investigate a hypothetical service provider running a QoS-sensitive web service purely on spot instances. The work is similar to ours with respect to executing an entire service workload as spot instances only and guaranteeing an SLA to end-users. The primary focus of this prior work however, is revenue maximization. As such SLA-violations are acceptable in resource-constrained situations and readily traded off for additional income. Similarly, [5] investigates a service running purely over spot instances. The authors show that existing SLAs capture only part of the observed variation typical in cloud environments, and propose a new utility-based approach to SLAs – user satisfaction. In our work, we also address servicing an entire workload with spot instances, but provide a new type of SLA on spot instance eviction probability. While user satisfaction depends on the specific end-application, guarantees on eviction probability improve usability for cloud users as it simplifies reasoning about the system for arbitrary applications and, as a consequence, allows remote clouds to reliably federate workloads with statistical guarantees.

HPC applications can be executed in the cloud using spot instances, but doing so can significantly complicate application and system design and performance due to preemp-

tion. In [18] the authors compare the performance of traditional HPC clusters and the recently introduced EC2 high-performance clusters in terms of turnaround time and cost. Although dedicated HPC hardware is still superior in terms of performance, the queue wait times are found to substantially increase turnaround time, which may make offloading to the cloud a user's preferred choice. The work in [17] further explores the potential of running HPC workloads in the cloud, by using redundancy and check-pointing to decrease costs and mitigate spot instance eviction. There have been numerous case studies that explore the cloud as cost-effective replacement for dedicated HPC resources. An example for this is [12] which profiles HPC applications suitable for federation into cloud.

Our approach to cloud federation via spot instances is different from existing work as we provide an SLA on the maximum spot instance eviction probability which simplifies reasoning about the system as whole – it does not impose a fixed revenue or QoS model, but rather provides a foundation for custom probabilistic utility models. Our SLA enforcement is based on estimates of the time-to-eviction of spot instances, which are generated via Monte-Carlo simulation from historical traces directly, rather than from an analytical model that is fit to the workload first. Finally, our approach to estimation and SLA enforcement is demonstrably robust in an evaluation against a broad variety of real-world workload traces obtained from private IaaS clouds deployed in production.

5. CONCLUSIONS AND FUTURE WORK

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 small and medium-size clouds, workload consolidation has its limitations due to smaller and more specialized customer bases. Cloud federation has been proposed to improve the efficiency of these smaller clouds by offloading peak demand within a federation and increasing mean utilization. The use of spot instances for this purpose seems economically 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-to-eviction of spot instances from historical workload traces via Monte-Carlo simulation. This effectively enables cloud providers to offer an a-priori SLA on the eviction probability of spot instances. A user or remote cloud can request spot instances for a fixed lifetime with quantitative bounds on eviction probability and receive an immediate response from the destination cloud of whether it can provide the desired quality of service, or not. Given the availability of these SLAs we demonstrate that production workloads of an entire cloud could be executed on another cloud using only spot instances, while maintaining the target SLA.

As part of future work, we are focused on addressing the limitations of our approach and enabling its effective use in production IaaS deployments. First, our method generates conservative estimates of the time-to-eviction for both synthetic and production traces, which translates into unused capacity and lost potential revenues (an opportunity cost). We are interested in identifying ways of reducing this conservativeness while still maintaining the SLA. With reduced conservativeness the impact of inaccuracies in corner cases and after change-points may increase as short-term eviction bursts take longer to compensate for. To alleviate this issue, we can extend our method to provide bounds on the estimated time-to-eviction quantiles as well. Such bounds can be used to dynamically adjust the degree of conservativeness to the degree of uncertainty under the current load conditions. Second, although we employ production traces and IaaS platforms for our evaluation, the implementation of this approach in a physical test bed poses interesting research challenges such as automatic selection of instances for federation, shepherding of spot instances with unknown duration via dynamic extension and migration, managing of spot instance populations with mixed SLAs, and supporting hybrid (public-private) cloud settings.

6. ACKNOWLEDGMENTS

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

7. 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*, pages 257–266, Aug 2010.

[3] 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/`.

[4] R. Buyya, R. Ranjan, and R. N. Calheiros. Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services. In *Proceedings of the 10th International Conference on Algorithms and Architectures for Parallel Processing - Volume Part I*, ICA3PP'10, pages 13–31, Berlin, Heidelberg, 2010. Springer-Verlag.

[5] 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*, HPDC '11, pages 229–238, New York, NY, USA, 2011. ACM.

[6] 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*, HotCloud'10, pages 7–7, Berkeley, CA, USA, 2010. USENIX Association.

[7] CloudStack. [Online; accessed Aug-2014] `http://cloudstack.apache.org/`.

[8] Eucalyptus Systems Inc. `http://www.eucalyptus.com`, June 2013.

[9] I. Foster. Globus toolkit version 4: Software for service-oriented systems. In H. Jin, D. Reed, and

W. Jiang, editors, *Network and Parallel Computing*, volume 3779 of *Lecture Notes in Computer Science*, pages 2–13. Springer Berlin Heidelberg, 2005.

[10] I. Goiri, J. Guitart, and J. Torres. Characterizing cloud federation for enhancing providers' profit. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 123–130, July 2010.

[11] Google Cloud Platform. [Online; accessed Aug-2014] `https://cloud.google.com/`.

[12] A. Gupta, L. V. Kalé, D. S. Milojicic, P. Faraboschi, R. Kaufmann, V. March, F. Gioachin, C. H. Suen, and B.-S. Lee. Exploring the performance and mapping of hpc applications to platforms in the cloud. In *Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '12, pages 121–122, New York, NY, USA, 2012. ACM.

[13] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, pages 295–308, Berkeley, CA, USA, 2011. USENIX Association.

[14] IBM SoftLayer. [Online; accessed Aug-2014] `http://www.softlayer.com/`.

[15] 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*, pages 219–228, Dec 2011.

[16] Y. Mansour. Regret minimization and job scheduling. In J. van Leeuwen, A. Muscholl, D. Peleg, J. PokornÃ¡, and B. Rumpe, editors, *SOFSEM 2010: Theory and Practice of Computer Science*, volume 5901 of *Lecture Notes in Computer Science*, pages 71–76. Springer Berlin Heidelberg, 2010.

[17] 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 *Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing*, HPDC '14, pages 279–290, New York, NY, USA, 2014. ACM.

[18] A. Marathe, R. Harris, D. K. Lowenthal, B. R. de Supinski, B. Rountree, M. Schulz, and X. Yuan. A comparative study of high-performance computing on the cloud. In *Proceedings of the 22Nd International Symposium on High-performance Parallel and Distributed Computing*, HPDC '13, pages 239–250, New York, NY, USA, 2013. ACM.

[19] 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*, pages 296–303, Sept 2011.

[20] Monte Carlo Method. `http://en.wikipedia.org/wiki/Monte\_Carlo\_method`.

[21] 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*, pages 124–131. IEEE, 2009.

[22] OpenStack. [Online; accessed Aug-2014] `http://www.openstack.org/`.

[23] 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.

[24] Rackspace Cloud. [Online; accessed Aug-2014] `http://www.rackspace.com/cloud/`.

[25] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes. Omega: Flexible, scalable schedulers for large compute clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems*, EuroSys '13, pages 351–364, New York, NY, USA, 2013. ACM.

[26] A. Toosi, R. Calheiros, R. Thulasiram, and R. Buyya. Resource provisioning policies to increase iaas provider's profit in a federated cloud environment. In *High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on*, pages 279–287, Sept 2011.

[27] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, pages 5:1–5:16, New York, NY, USA, 2013. ACM.

[28] R. Wolski and J. Brevik. `http://www.cs.ucsb.edu/~rich/workload`, June 2013.

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

[30] R. Wolski and J. Brevik. Using Parametric Models to Represent Private Cloud Workloads. *IEEE Transcations on Services Computing*, 4(7):714–725, October 2014.