Probabilistic Guarantees of Execution Duration for Amazon Spot Instances

Rich Wolski
University of California, Santa Barbara
UCSB Computer Science Department
Santa Barbara, CA 93106
rich@cs.ucsb.edu

John Brevik
California State University at Long Beach
CSULB Department of Mathematics
Long Beach, CA 90840
John.Brevik@csulb.edu

Ryan Chard
Argonne National Laboratory
Computing, Environment and Life Sciences
Argonne, IL 60439
rchard@anl.gov

Kyle Chard
University of Chicago and Argonne National Laboratory
Computation Institute
Chicago, IL 60637
chard@uchicago.edu

ABSTRACT

In this paper we propose DrAFTS – a methodology for implementing probabilistic guarantees of instance reliability in the Amazon Spot tier. Amazon offers "unreliable" virtual machine instances (ones that may be terminated at any time) at a potentially large discount relative to "reliable" On-demand and Reserved instances. Our method predicts the "bid values" that users can specify to provision Spot instances which ensure at least a fixed duration of execution with a given probability. We illustrate the method and test its validity using Spot pricing data *post facto*, both randomly and using real-world workload traces. We also test the efficacy of the method experimentally by using it to launch Spot instances and then observing the instance termination rate. Our results indicate that it is possible to obtain the same level of reliability from unreliable instances that the Amazon service level agreement guarantees for reliable instances with a greatly reduced cost.

CCS CONCEPTS

• **Networks** → **Cloud computing**; • **General and reference** → *Reliability*; • **Applied computing** → *Forecasting*;

KEYWORDS

cloud computing, service level agreements, cost optimization

ACM Reference format:
Rich Wolski, John Brevik, Ryan Chard, and Kyle Chard. 2017. Probabilistic Guarantees of Execution Duration for Amazon Spot Instances. In *Proceedings of SC17, Denver, CO, USA, November 12–17, 2017,* 11 pages.
DOI: 10.1145/3126908.3126953

SC17, Denver, CO, USA

DOI: 10.1145/3126908.3126953

1 INTRODUCTION

One of the fundamental tenets of cloud computing is that resources (*e.g.,* machines, network connectivity, storage, etc.) be characterized by their capacity and capability characteristics rather than their physical construction. Programmers and users reason about the use of cloud resources in terms of these characteristics (in principle) without regard to the physical infrastructure that is used to deliver them. For example, instances (virtual machines) available from Amazon's Elastic Compute Cloud (EC2 [4]) are advertised as rough equivalents to various models of physical processors (in terms of clock speed, cache size, etc.), but Amazon provides no guarantee that these specific processor models will be used to fulfill a specific user's request or are even available in their cloud. Instead, users enter into a "Service Level Agreement" (SLA), that quantifies the minimum capability that a particular request will receive.

Cloud computing vendors may offer different SLAs at different price points so that users can control the value transaction at a fine level of granularity. Vendors such as Amazon and Google [11] offer a preemptible tier of service where resources are offered (at a cheaper price) without a reliability SLA. Amazon offers these instances as "Spot instances" [3], for which reliability is based (in part) on the maximum amount of money a user agrees to pay for them. When the user makes a request for an instance having a specific set of characteristics (termed an "instance type"), he or she includes a "maximum bid price" indicating the maximum that the user is willing to be charged for the instance. Amazon creates a market for each instance type and satisfies the requests of the highest bidders. Periodically, Amazon recalculates the market price and terminates those instances whose maximum bid is below the new market price. The market-clearing mechanism is published, but individual user bids (and some of the other market parameters) are not. Thus, each user must devise an individual bidding strategy that meets his or her own reliability needs [15, 27].

Amazon also offers the *same* instance types under a fixed reliability SLA at a fixed price (*i.e.*, On-demand instances). Because the Spot instance market mechanism does not provide a way to guarantee how long an instance will run before it is terminated as part of an SLA, Spot market prices are often significantly lower—by up to an order of magnitude—than fixed prices for the same

instances with a reliability SLA. That is, because a user cannot determine a bid that will ensure a specific level of reliability in the Spot market, this uncertainty generally leads to lower prices. However, in many cases, our results indicate that users must often bid higher in the Spot tier than they would pay for a fixed price instance (which is covered by the Amazon reliability SLA) in order to get the same level of reliability. While a large body of work has investigated bidding strategies for optimizing the use of the Spot tier [12, 13, 16, 22, 24, 25, 29, 30], there is no published strategy of which we are aware that gives users the ability to determine at the time of a request "how high" they must bid to prevent their instances from being terminated.

In this paper we present a methodology for determining a minimized bid price that will ensure a fixed level of reliability in the Amazon Spot tier.[1] The methodology – called DrAFTS[2] – applies a non-parametric time-series forecasting technique to the pricing history for a specific instance type. Using forecasts generated by the technique, DrAFTS determines a minimized bid price that will ensure a given level of durability (expressed as the probability that an instance of this type will not be terminated before a given deadline) in the Amazon Spot tier. DrAFTS takes the probability and the deadline as parameters. Thus, a user who knows the duration over which an instance must persist can select a success probability that matches, or exceeds the reliability guarantee offered by Amazon as part of its fixed-price SLA. In this way, users of DrAFTS can get a *functional equivalent* of the fixed-price reliability (guaranteed at least probabilistically) while paying the lower Spot tier price.

It is possible for a user of the Amazon Spot tier to achieve a high level of reliability by simply bidding a large maximum value. However, the size of the bid defines the possible cost (*i.e.*, financial risk) associated with the transaction. The goal of DrAFTS is to minimize this risk while, at the same time, providing a probabilistic guarantee of reliability.

This paper makes the following contributions:

- It describes the DrAFTS methodology in terms of the statistical forecasting techniques it employs and the algorithm it uses to make probabilistic reliability predictions.
- It verifies the probabilistic guarantees and quantifies the risk mitigation provided by DrAFTS using "backtesting" and historical Amazon Spot tier pricing data.
- It demonstrates the effectiveness of DrAFTS by detailing the execution of both synthetic and "real-world" application workloads using DrAFTS-determined bids.
- It demonstrates how DrAFTS can be used to optimize the cost of using Amazon EC2.

It does so using a combination of after-the-fact analysis of price-history data and "live" experimentation with applications running in AWS. These latter experiments make use of a DrAFTS on-line service that has been operational in prototype form since late 2015 as part of the Aristotle project [5]. We also compare DrAFTS to other plausible methodologies for determining bids and find that it is the only method among those tested that is able to achieve a probabilistic durability guarantee. These contributions and our experiences with the DrAFTS service indicate that it is possible to make effective predictions of bounds on future price fluctuations in the Amazon Spot tier.

[1] Amazon terms its cloud offering "Amazon Web Services," commonly abbreviated as "AWS." We will use the term "Amazon Spot tier" or "Spot tier" to refer to the Amazon "EC2 Spot Instances" product – *cf.* https://aws.amazon.com/ec2/spot/.

[2] DrAFTS is an acronym for **Dur**ability **A**greements from **T**ime **S**eries.

2 AMAZON SPOT INSTANCES

To request an instance in the Amazon Spot tier, a user submits what amounts to a 4-tuple consisting of

$$(\textit{Region, Availability\_zone, Instance\_type, Max\_bid\_price}). \quad (1)$$

Amazon organizes its "Elastic Compute Cloud" (EC2) service (from which virtual machines may be rented) into independent *Regions*, each of which constitutes essentially a separate instantiation of the service. That is, EC2 resources are not shared across Regions. Each Region is further divided into *Availability Zones* (AZs), which define collections of resources with independent failure probabilities so that the joint probability of failure in multiple zones can be quantified. A virtual machine (termed an *instance*) launched by a user runs in a specific Region and AZ. In the Spot tier, the user must specify the Region and may specify the AZ, although if the AZ is missing from the request, Amazon will choose one (without regard for price). Note that the Region name is carried in the AZ name. For example, the *us-east-1* Region comprises five AZs named *us-east-1a*, *us-east-1b*, *us-east-1c*, *us-east-1d*, and *us-east-1e* respectively.

The instance type determines the nominal capabilities in terms of CPU, memory, and local storage capacity of the virtual machine that will be instantiated. For example an *m3.medium* instance type currently includes 1 "virtual" CPU, 3.75 gigabytes of memory, and 4 gigabytes of local disk storage. EC2 currently supports 57 different current generation instance types in the Spot tier, although not all types are available in all Regions and AZs.

A request to launch an instance in the Spot tier must include a maximum bid price, which determines the maximum hourly rate that the user is willing to pay for the instance.[3] This price is not revealed to the other users of the Spot tier. In addition, Amazon does not reveal the number of resources that are available. Instead, Amazon sets a *market price* for each AZ that is advertised to all users [1]. Requests carrying a maximum bid that is greater than the current market price are accepted and the instances to which they refer are initiated or are allowed to continue executing.

2.1 Pricing

Amazon computes the market price so that the (hidden) supply is exhausted. It sorts the currently active maximum bids by value and allocates resources to maximum bids (taking into account request size) in descending order of bid value. The lowest maximum bid that corresponds to a "taken" resource determines the market price. It follows that, in principle, the market price changes whenever a new request is presented, when an active request is terminated by its user, or when the supply allocated to the resource pool by Amazon changes. In practice, we observe that many price changes and/or repeated price announcements occur with approximately a 5-minute periodicity, perhaps indicating that prices are adjusted according to a more deterministic schedule.

[3] Since the maximum bid is the only bid that a user submits, we will use the term "bid" and "maximum bid" interchangeably.

Instances that are running in the Spot tier are charged by the hour. When an instance is executing, its user is charged the current market price that occurs at the beginning of each hour of execution for that hour's duration. When the instance is terminated by its user, the user is charged for the complete hour of execution in which the termination occurs. That is, Amazon "rounds up" to the nearest hour when a user terminates an instance.

If the market price exceeds the maximum bid price for a running instance, the instance is terminated by Amazon; if the market price becomes *equal* to the maximum bid price, the instance may be terminated or may be left running. Thus, the duration that an instance will run before it is terminated is determined (assuming that the probability of hardware failure is negligible) by the time until the market price becomes greater than or equal to the maximum bid price for the instance.

Further, the difference between the market price computed at the beginning of each hour the instance runs (*i.e.*, the price the user will be charged) and the maximum bid price determines the worst-case financial risk associated with the instance. That is, the user "risks" paying up to the maximum bid price for each hour the instance executes but usually pays less.

2.2 Spot Instance Price Histories

Amazon makes up to 90 days of market price history for each instance type in each Region and AZ available for programmatic access. In this study, we have accumulated price histories for all AZs in the *us-east-1*, *us-west-1*, and *us-west-2* Regions spanning the period from October 2015 to April 2017.

Note also that Amazon prevents "herding" behavior in AZ selection by remapping AZ names on a user-by-user basis. Thus, different users selecting *us-east-1a*, for example, do not necessarily make requests from the same pool of resources. It is possible to compare market price histories from different users to determine a globally consistent AZ naming scheme. DrAFTS does not depend on this deobfuscation for its function, but using DrAFTS as a service does. In this work, when we have used the DrAFTS service to generate results, we have performed this deobfuscation manually.

Finally, we observe that the price updates are frequently (but not always) available on a 5-minute periodicity. Moreover, the Amazon web pages quote the cheapest price (across AZs) in each Region with a 5-minute update frequency. Thus it is likely (but not guaranteed) that any price quote will persist for approximately 5 minutes.

3 METHODOLOGY

From the perspective of a user of the Amazon Spot tier, DrAFTS attempts to find the lowest maximum bid price that ensures an instance will run for the specified duration before being terminated with probability at least as large as the probability associated with the user's desired reliability level. Note that DrAFTS bids provide statistical guarantees that are slightly more restrictive than the reliability SLAs currently provided by Amazon for its other classes of service[4] in that they are for *continuous* availability durations.

[4]Amazon offers several classes of service with respect to instances under the same SLA including "On-demand," and "Reserved" instances. Only instances in the Spot tier do not carry a reliability SLA. See http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-purchasing-options.html for a description of the current purchase options.

While these SLA guarantees are both probabilistic (the AWS SLA is not a deterministic guarantee), the nature of these SLAs is different. The Amazon SLA specifies a percentage of availability time that is cumulative over a fixed time period. As long as the instance appears available for a specified percentage of time within the time period (say, 99% in a month) the SLA is fulfilled. For example, one second of unavailability occurring in every non-overlapping 100-second period of time (technically) fulfills a 99% reliability guarantee. In contrast, the DrAFTS probability refers not to the *cumulative* availability but to the *continuous* availability of a specific instance request. To make the distinction clear, we will henceforth use the term "durability" to refer to the more restrictive form of probabilistic guarantee.

DrAFTS applies a non-parametric time-series forecasting technique (described in the next Subsection) to the history of prices from the Spot tier to determine its bid predictions. The methodology is a two-step process. In the first step, DrAFTS computes a time series of upper bounds on market prices at each moment in a price history. These bounds are probabilistic guarantees that the next value in the market price series will be less than the bounds and, thus, could have served as a maximum bid at each point in time. That is, this upper bound series is the series of smallest bids that would have guaranteed an instance would "survive" until the next market price update in the series. In the second step, it repeatedly increments this bid fractionally and computes a series of time durations until the bid would have been equaled or exceeded by market price. It then computes a probabilistic lower bound on this series of time durations. The process generates pairs of bids and lower-bounds on durations where each bid guarantees (at a minimum, probabilistically) the duration with which it is paired. Note that as bids get larger, the durations must increase monotonically for a fixed target probability according to the price-setting mechanism described in the previous section.

3.1 Non-parametric Bounds Prediction

DrAFTS uses Queue Bounds Estimation from Time Series (QBETS) [20, 21], a non-parametric time series analysis method to determine bounds, dynamically, from observed price and durability duration series. QBETS adapts to changes in the underlying time series dynamics (both change-points and autocorrelations) making it useful in settings where forecasts are required from arbitrary data with widely varying characteristics.

A bounds forecast from this method requires three inputs:

(1) A time series of data.
(2) A quantile for which a confidence bound should be predicted ($q \in (0,1)$).
(3) The confidence level of the prediction ($c \in (0,1)$).

To estimate an upper bound on the q^{th} quantile of the time series, it treats each observation in the time series as a Bernoulli trial with probability q of success. If there are n observations, the probability of there being exactly k successes is described by a Binomial distribution (assuming observation independence) having parameters n and q. If Q is the q^{th} quantile of the distribution from

which the observations have been drawn, the equation

$$\sum_{j=0}^{k} \binom{n}{j} \cdot (1-q)^j \cdot q^{n-j} \quad (2)$$

gives the probability that no more than k observations are greater than Q. As a result, the k^{th} largest value in a sorted list of n observations gives an upper c confidence bound on Q when k is the smallest integer value for which Equation 2 is larger than c. Taking k to be the largest integer for which this formula is smaller than $1-c$ gives a lower confidence bound for Q.

The model described above assumes that the series is stationary. As a result, the "current" bound on the q^{th} quantile is also a prediction of the bound for the next observation. In practice, empirical time series taken from systems often exhibit change points and other forms of non-stationarity. The method attempts to correct for these events automatically to improve the accuracy of its forecasts.

Note that typical values of c for upper bounds are relatively close to 1, which makes the bound estimates conservative. The value returned as a bound prediction is larger than the true q^{th} quantile with probability c under the assumptions of the model. However, as a prediction based on confidence bounds, the degree to which it is larger is not estimated.

More succinctly, the implementation of the method sorts observations in a history of observations, and computes the value of k that constitutes an index into this sorted list that is either the upper c or lower c (user selectable) confidence bound on the q^{th} quantile. The methodology assumes that the time series of observations is ergodic, so that in the long run the confidence bounds are correct in a conservative sense. However, to improve prediction performance, the method also attempts to detect change points in the time series of observations so that it can apply this inference technique to only the most recent segment of the series that appears to be stationary.

Note also that the algorithm itself can be implemented efficiently if the time series state needed to determine change points is persistent so that it is suitable for on-line use. Details of this implementation as well as a fuller accounting of the statistical properties (including correction for autocorrelation via use of a table that captures the effect of the first autocorrelation on rare events) and detailed assumptions are presented in [19–21].

3.2 DrAFTS Prediction Methodology

DrAFTS uses QBETS to predict an *upper* bound on maximum bid price and a *lower* bound on the time the bid will be sufficient to prevent a termination due to market price. We term this time the "bid duration." It uses the square root of the desired target probability as the q^{th} quantile and, in the study, a value of 0.99 for the confidence level c. While other probability combinations are possible to reach the target probability, our experience has shown that using square roots strikes a good balance between keeping a bid low (*i.e.*, near the current price) and yielding a usable duration.

For example, to compute the DrAFTS prediction with probability 0.95 for an instance type at a particular moment in time, DrAFTS computes an upper bound prediction of the $q = 0.975$ quantile for all elements of the series up to that moment (roughly the square root of 0.95) and $c = 0.99$. The time series method returns an upper confidence bound on the 0.975 quantile of the next market price for *each* element of the time series. That is, DrAFTS creates a history of upper bound predictions, one for each point in the price history series where pricing data is available.

Based on this price history, DrAFTS then generates a series of durations for which the upper bound prediction would have remained above the market price. Each element of this series is the duration from when the prediction is made until the market price exceeds it. Thus, each prediction is paired with a duration until it is equal to or less than the market price.

DrAFTS then uses the time series method again to predict a lower confidence bound (again with $c = 0.99$) on the $(1-q)^{th}$ quantile of the new duration series. Note here that this prediction is based on the *conditional* probability that the price allows the instance to run in the first place.

Note that Amazon may or may not terminate an instance when its maximum bid is exactly *equal* to the market price. QBETS assumes that the bound on the desired target quantile is contained in the observed time series. Because the methodology is attempting to use only the most recently relevant history, it is possible that the upper bound on the market price is equal to the current market price (exactly).

To account for the possibility that the current Spot price is equivalent to the upper bound prediction, DrAFTS *adds* \$0.0001 (the smallest cost increment allowed by the Spot tier interface) to each upper bound prediction so that it must be larger than the quoted market price returned in all cases. This premium ensures that DrAFTS predicts the minimum time until a Spot instance is *eligible* to be terminated because of price rather than the time until the Spot price absolutely exceeds the maximum bid. This estimate is a conservative lower bound on the time until the instance actually will be terminated. We refer to this bound on the time until an instance *may* be terminated as the **durability** of the prediction.

3.3 Performance

For all experiments described in Section 4 (except those in Subsection 4.3) each DrAFTS maximum bid was computed using the previous 3 months pricing data requiring approximately 2 minutes to generate using server class machines. In a production setting, the predictor state can be updated incrementally (in a few milliseconds) whenever a new price data point is available.

For the experiments described in Subsection 4.3 we also wanted to experiment with DrAFTS's operation as a stand-alone web service. Our goal was to understand how DrAFTS might function as a decision-support tool or cloud-based service available to application scheduling programs. The service is available at http://predictspotprice.cs.ucsb.edu. The DrAFTS service implementation operates asynchronously. It periodically queries the Amazon price-history API and computes a set of maximum-bid predictions for each instance type and AZ. To access these bids, clients use a Representational State Transfer (REST) [10] API, via which they can request a set of DrAFTS maximum bids for a specific instance type and AZ. The service computes duration predictions associated with increasing maximum bid values in increments of 5% for both the 0.95 and 0.99 probability levels. It starts with the smallest predicted bid that can guarantee any duration with the specified probability and computes bid predictions up to 4 times this minimum value

in increments of 5%. It is currently configured to recompute all bid predictions every 15 minutes. Note the service does not yet deobfuscate AZs perfectly (it uses a heuristic) and therefore, in this work we preconfigured the AZ mapping for each client.

4 RESULTS

We validate the effectiveness of the DrAFTS method for determining maximum bids in the Spot tier using four sets of experiments that span the operational lifetime of the DrAFTS service. We first evaluate DrAFTS's ability to correctly satisfy the requested durability guarantee for many combinations of instance type, AZ, and Region. We then explore DrAFTS's efficacy using a lower target probability and instantiating a large number of identical instances for a fixed time period. In the third set of experiments, we incorporate DrAFTS bid predictions into a production analysis platform and explore its ability to reduce cost and risk when provisioning instances. Finally, we show the cost savings that would have resulted from using DrAFTS as part of a simple cost-optimization strategy.

4.1 Correctness

We term a bid to be *correct* when it is sufficient to prevent the instance for which it is made from being terminated by Amazon due to a change in market price. We measure the correctness of the overall methodology though *backtesting*. To do so, we repeatedly choose a time at random in the market price history for each combination of AZ and instance type and run the DrAFTS algorithm with a specific target probability p using the data before that time in the history. We then choose a random instance duration and compute the DrAFTS-predicted maximum bid. Finally, we test whether this bid would have prevented a termination by Amazon by computing the time from that point in the history until the predicted bid price is greater than or equal to the observed market price. If the duration of the instance is longer than the interval until the market price is greater than or equal to the predicted maximum bid, the prediction succeeds in preventing a termination. Otherwise, it fails. We record the fraction of successes from a suitably large set of such experiments and compare it to the success probability p supplied to DrAFTS in the test. If the success fraction is greater than or equal to p, DrAFTS would have appeared to be functioning "correctly" in terms of its ability to provide a probabilistic guarantee to a fictitious user who had submitted the random instances over the time period that has been "backtested."

We treat each combination of AZ and instance type as a separate category of resource. This categorization is necessary because users of the Spot tier must decide on and specify which Region, AZ, and instance type to use. Thus while it may be possible to achieve an overall success fraction that meets the probability target across all possible combinations, users require that DrAFTS meet its probability target for *each* combination separately. We tested all instance types available from the *us-east-1*, *us-west-1*, and *us-west-2* Regions.[5] Amazon reported 9 total AZs were available in these three Regions. There were 53 different instance types at the time of the study, but not all instance types are available from all AZs. The total number of combinations of AZ and instance type we backtested was 452.

For each combination we generated 300 Spot tier requests beginning at random times between October 2016 and December 2016. Each request had a duration drawn from a uniform random distribution between 0 and 12 hours in length. For each request, we computed the maximum bid using each method and then determined whether that bid would have been sufficient to prevent Amazon from terminating the request.

Table 1 shows the percentage of AZ-instance type combinations that achieved < 0.99, 0.99, or 1.0 success fraction for four different methods of computing a bid: DrAFTS, On-demand price, AR(1) and the empirical cumulative distribution function (CDF).

Method	Correctness fractions		
	<0.99	0.99	1
DrAFTS	0.2%	27.0%	72.8%
On-demand	37%	12%	51%
AR(1)	29%	17%	54%
Emperical-CDF	6%	62%	32%

Table 1: *Backtested correctness fractions for all instance types in us-east-1, us-west-1, and us-west-2 using different methods to compute the maximum bid, October 1, 2016 through December 1, 2016, with a sample size of 300 and a random instance duration between 0 and 12 hours.*

4.1.1 DrAFTS correctness. The DrAFTS target probability was set to 0.99 with confidence bound $c = 0.99$. When using DrAFTS , this target was met in all but 0.2% of the combinations. The 1 (out of 452) that "failed" to achieve a success fraction of at least 0.99 had a success fraction of 0.98.

Note that due to autocorrelation in the price data it is possible for DrAFTS to fail to meet its probability goal (but almost to meet it) for contiguous periods of time due to random variation. That is, the methodology finds the upper bound from the "population" but depending on the random time stamps chosen, it is possible to see success fractions that are slightly below the probability target for short periods. Thus we did not expect to see a success fraction of 1.0 but something near it, as shown in the table. Indeed, to test the hypothesis that the one case in which DrAFTS failed to achieve a 0.99 or better guarantee, we re-ran the simulations for the one failure separately using a different random number seed and the success probability for this second run was 0.99. With the durability guarantee set at 0.99, DrAFTS is correct as long as the miss fraction is 0.01 or less due to random variation and 1 out of 452 is well below this fraction.

4.1.2 Comparison with On-Demand. The On-demand price is the hourly price a user must pay for an instance to obtain the Amazon reliability SLA. Currently, that SLA guarantees 99.95% instance availability over the course of 1 month or the user is entitled to a 10% refund [2]. If the availability is less than or equal to 99%, the refund is 30%. Amazon sets On-demand prices by Region. That is, a user pays the same On-demand price in each AZ within a Region. Table 1 shows that the On-demand price, when used as a maximum bid, does not ensure a correctness fraction of 0.99 for many of the

[5] We conducted the experiments with an ordinary Amazon user account. When this account queried AWS for the available AZs, it reported that 4 were available in *us-east-1*, 2 were available in *us-west-1*, and 3 were available in *us-west-2*.

possible AZ and instance-type combinations (approximately 37% of those tested).

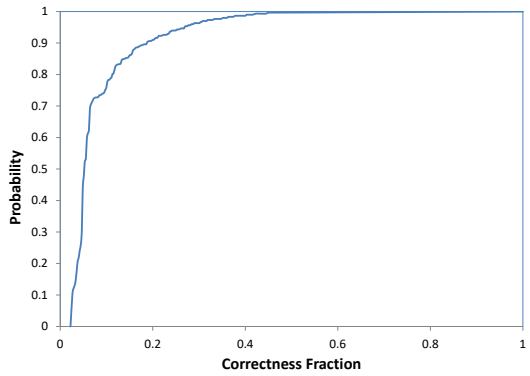

Figure 1: *CDF of Backtested DrAFTS correctness fractions less than* 0.99 *for all instance types in us-east-1, us-west-1, and us-west-2 using the On-demand price as a maximum bid October 1, 2016 through December 1, 2016 with a sample size of* 300 *and a random instance duration of* 12 *hours.*

Further, many of the success fractions when the On-demand price was used as a maximum bid are substantially smaller than 0.99. Figure 1 shows the empirical CDF of the correctness fractions that were less than 0.99 generated by the backtesting experiment with the On-demand price as the maximum bid. Indeed, some of the success fractions were even zero. That is, the On-demand price for these combinations of AZ and instance type was *never* sufficient to prevent a termination due to price. For example, the *cg1.4xlarge* instance type, in the *us-east-1* Region had a price of $2.1 at the time of the experiment. The smallest price we observed when backtesting *cg1.4xlarge* in what appeared to our test account as *us-east-1c* was $2.10010. That is, the Spot tier market price of a *cg1.4xlarge* in *us-east-1c* was at least one tenth of a cent higher than the On-demand price for each of the 300 randomly generated instances. We cannot yet determine whether this phenomenon (which we observed for several other combinations) is a natural consequence of the Amazon Spot tier market-making mechanism or an artifice designed to discourage the use of the Spot tier for those specific combinations. However, the overall results indicate that the On-demand price does not determine a maximum bid that guarantees the same level of durability for the instance durations and time period we tested.

4.1.3 Comparison with other methodologies. Comparing DrAFTS to other predictive methodologies for Spot pricing is problematic in that the other approaches reported in the literature do not predict probabilistic bounds (to be used as a durability guarantee) nor do they typically provide a parametric model from which quantiles may be determined. One exception is the work by Yehuda et al. [1] in which the authors show that Spot prices may be well-modeled by an AR(1) [6] series. If so, then each price value is the sum of a fraction of the previous price value and a normal random variable. Using this formulation, it is possible to estimate the bounds of such empirically with a sufficiently large sample size.

Yehuda et al. note that an AR(1) model is only applicable for segments of the price time series that are stationary and that the series they examine contain many such segments. That is, a long-term price series is composed of separate but contiguous segments, each of which can be well modeled by an AR(1) model. Note: in a republished version of this article the authors state that recent market prices are no longer consistent with an AR(1) model [1]. However, they show this change for a single series. Our work explores many more instance types, across AZs, and with more samples. We see several series that are, in fact, well-modeled by an AR(n) process and some that are not.

To compare the efficacy of this approach to DrAFTS, we modified the software to use an AR(1) model in conjunction with the same non-parametric binomial change-point detection method used by DrAFTS. The software treats segments of the time series between change points as stationary AR(1) time series and computes the target quantile based on this AR(1) assumption for each series. To make a durability prediction at an arbitrary point in time, the system uses the quantile from the series beginning at the change point detected most recently before that point in time. This quantile is treated as a bound on the series for future values. Thus this approach uses an AR(1) model in place of the non-parametric QBETS to determine bounds. Note that without change-point detection, the comparison would unfairly penalize the AR(1) approach since it only holds for regions of stationarity in the price series.

Table 1 shows the use of an AR(1) model to be similar in quantity to using the On-demand price. The approach meets the probabilistic target for some of the AZ and instance-type combinations, but for many (29%) it fails to do so. Interestingly, these results do not contradict the results in the Yehuda et al. study. In our results, the AR(1) method correctly determines a guarantee for the same AZ and instance-type combinations that Yehuda and his colleagues tested. However, DrAFTS is correct, in terms of a durability guarantee, for all combinations, indicating that some of them may not be as accurately modeled as an AR(1) (or indeed AR(n)) series.

Finally, one methodology that has been suggested for determining a bid price is to use the empirically determined quantile from the observed price series as a bid. Table 1 also shows the correctness fractions when the empirical CDF is used to determine the bid for a 0.99 durability target. In this case, a large fraction of the AZ and instance-type combinations meet the 0.99 target, but 6% (26 out of 452 combinations) show a lower success rate. While this methodology is simple to implement and understand, it does not provide the guarantee and level of certainty that is provided by DrAFTS.

4.2 Instance Launch Experiments

To further test the efficacy of DrAFTS, we used its bid predictions to launch instances in the Spot tier. To control expense, we chose inexpensive instance types and a duration of 3300 seconds (5 minutes less than 1 hour). This latter decision stems from early experimentation in which the time between when our experimental apparatus decided to terminate an instance and the actual termination time recorded by Amazon could take up to 5 minutes. As a result, durations of close to an hour would occasionally be charged for two hours as the recorded termination "rolled over" the one hour mark.

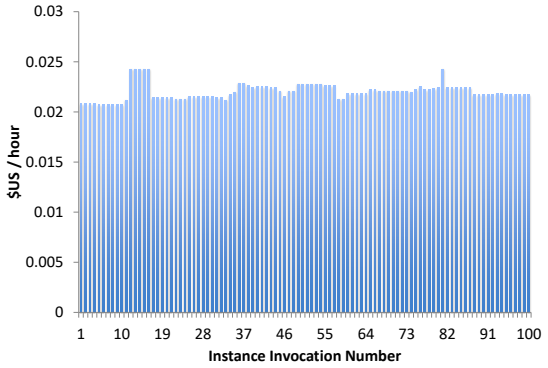

Figure 2: *DrAFTS Maximum Bids, p=0.95, 100 experimental instance launches of type c4.large in the us-east-1 Region, November 15 through November 22, 2015.*

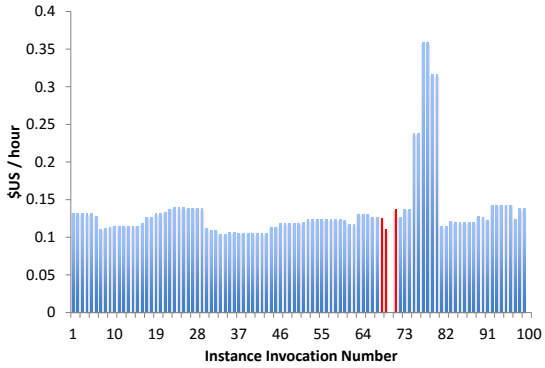

Figure 3: *DrAFTS Maximum Bids, p=0.95, 100 experimental instance launches of type c3.2xlarge in the us-west-1 Region, January 7 through January 14, 2016.*

In each experiment, a script computed the DrAFTS maximum bid that would ensure a 3300 second duration with probability $p = 0.95$. We chose 0.95 (instead of the 0.99 described in the previous subsections) so that we could initiate approximately 100 instances and observe a meaningful failure count. Rather than choosing a single AZ, however, we allowed the experiment to choose the AZ in a specified Region that currently had the lowest predicted price upper bound. That is, we used the predicted price upper bound for each AZ in a given Region as a "fitness function" so that financial risk associated with each experiment would be minimized.

For a given Region the script repeatedly computed the current price upper bound and DrAFTS bid in each AZ, chose the AZ that had the lowest predicted price upper bound, and requested an instance from that AZ with the corresponding bid. Once Amazon reported the instance as being in the "running" state, the script would then pause for 3300 seconds and afterwards interrogate Amazon to determine whether the instance was still running. If not it recorded a failure, otherwise it recorded a success.

Further, we designed each experiment to take place over the course of a week and to run approximately 100 instances during that week. To prevent Amazon from detecting a regular periodicity and performing some unseen optimization on our behalf, we varied the time between experiments by selecting an inter-experiment interval from a normal distribution with a mean of 2748 seconds and a standard deviation of 687 seconds.

Figure 2 shows a time series of DrAFTS maximum bids recorded for the week between November 15 and November 22, 2015, for the *c4.large* instance type in the *us-east-1* Region. In the figure, the x-axis shows the instance launch number and the y-axis depicts the value, in U.S. dollars, of the DrAFTS-determined maximum bid. We chose this Region and instance type believing that the generally low hourly price in the Spot tier would induce variability due to its popularity. However all 100 instances completed successfully (*i.e.* were not terminated due to price). Backtesting this combination along with the AZ selection methodology revealed that at the 0.95 target probability level, DrAFTS predictions were relatively conservative. Often, the backtested results exhibited a success fraction greater than 0.99 making it plausible that a test consisting of 100 instance launches would contain no failures.

In Figure 3 we show similar results for a second experiment where backtesting showed that DrAFTS would be less conservative at the 0.95 level in the *us-west-1* Region for the *c3.2xlarge* instance type in January of 2016. The experiment recorded 4 failures over the course of the week (instance invocation numbers 69 through 72 shown in dark red in the figure), which is consistent with the target success probability of 0.95 we had chosen. Further, the four failures occurred "back to back," further lending credence to the hypothesis that the failure percentage using DrAFTS shown in Table 1 is due to autocorrelation in the price data. Finally, the third failure of the four was not a price termination but rather a failure of the instance to launch due to the bid being below the current market price.

We conducted three similar experiments using different instances types and Regions (during non-overlapping week-long time periods) with nearly identical results (graphs omitted for brevity). In all cases the DrAFTS bids achieved the target guarantee probability.

4.3 Application-Driven Experiments

As a more realistic test of DrAFTS efficacy, we integrated it with the Globus Galaxies platform [17]—a production, cloud-hosted analysis platform that underlies the Globus Genomics service [18]. The platform enables users to define and execute workflows composed of various analysis applications. Executed workflows are decomposed into individual jobs, which are then queued for execution. Importantly, most jobs are tolerant of delays and users are therefore willing to incur increased execution time (due to instance revocation) to obtain lower costs. The platform implementation includes a *provisioner* [7] that monitors the job queue and provisions instances in the Spot tier to execute individual applications. The platform is a particularly good example for the DrAFTS methodology as it includes approximate computational profiles—descriptions of the requirements of a particular application (*e.g.*, CPU and memory requirements) and estimated execution time [8]. However, these profiles are used only to select suitable instance types. Indeed, before the work described herein, there was no reliable way to use the execution times to influence bid determinations.

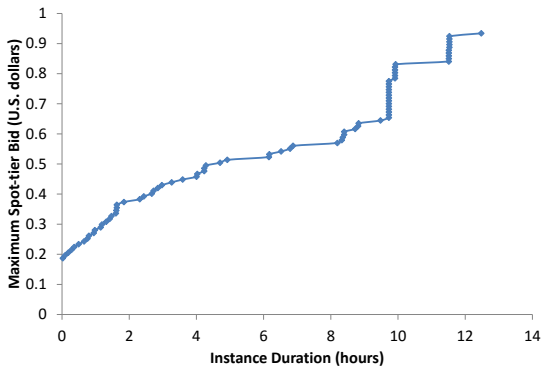

Figure 4: ***Bid-duration relationship, in the us-east-1a AZ for the "Linux/UNIX (Amazon VPC)" c3.4xlarge instance type at 10:16 AM PDT on April 18, 2016.***

To test DrAFTS we modified the provisioner to fetch the DrAFTS graph for each instance type from the DrAFTS service (*cf.* Section 3). Figure 4 shows an example of such a graph illustrating the relationship between predicted duration until an instance will be terminated due to market price and DrAFTS-predicted maximum bid cost in U.S. dollars. Using this graph (which is also supplied in a machine-readable format) a client of this service can determine what maximum bid to use to ensure a specific instance duration.

To evaluate performance we used a workload recorded from production usage of the platform. The workload included 8452 jobs over a 24-hour period. In order to reduce the execution time and cost of running experiments we used only the first 1000 jobs (requiring 366 instances). This represents a 3 hour and 20 minute period of submissions, for a total of approximately 8 hours of execution. To enable the workload to be "replayed" in the Spot tier at different times, we transformed the submission time of each job into a relative submission time offset by the time of the day the job was submitted.

We first explored using DrAFTS to select instance type and AZ for each job: using a required duration of one hour. This illustrates a baseline experiment in which accurate profiles might not be available. We configured DrAFTS with probability 0.99 for each candidate instance type and AZ and selected the one with the smallest maximum bid.

Method	Cost	Maximum Bid Cost
Original (80% On-demand)	$106.10	$176.98
DrAFTS Bid	$91.78	$98.60

Table 2: ***Comparison of Original Spot tier usage to selection using DrAFTS, February 28 and 29, 2016.***

Table 2 compares the results of one replay experiment to complete the workload which took place on April 28 and 29, 2016 using DrAFTS methods for selecting and pricing instances as well as the service's original bid determination method (80% of the On-demand price). The table shows the overall cost for all instances used and maximum (worst-case) cost that could have been incurred (*i.e.*, the risked cost) if the instances were charged at the bid value rather than the market price

The entire application run consisted of 366 instance requests to the Spot tier with each method. In both cases all 366 instances completed without being terminated by Amazon. This is consistent with the DrAFTS requested success probability of 0.99. While no termination differences were observed, DrAFTS both reduced the overall cost and the financial risk.

To further investigate the ability for DrAFTS to be used in an on-line setting we developed a simulator plugin to the provisioning service [9]. This simulator replicates AWS market conditions at a particular time and thus enables low cost experimentation under identical market conditions. The simulator is calibrated using observed distributions for AWS (*e.g.*, instance request time) and provisioner overheads (*e.g.*, job submission). Table 3 shows the results of running 35 simulated experiments when using the original method, DrAFTSbids based on 1 hour duration and when using profiles. The experiments were run on April 1 and 2, 2017 using the same 1000-job workload. It reports the average number of instances provisioned and terminated, as well as the average cost and risk across 35 experiments. The table shows that the DrAFTS methods again reduce costs. In this case, DrAFTS also significantly reduces the potential risk by more than a factor of 2. These results also demonstrate the (slight) advantage of using accurate job profiles to compute the DrAFTS bid. The difference in this case is small with respect to cost, as the workload contains few jobs that last longer than one hour. However, even in this "worst-case" scenario the risk is reduced by almost 7%. As expected the number of terminations increases as the profile-based method produces a "tighter" bid price (to meet a shorter durability guarantee).

Method	Avg. Instances	Avg. Cost	Avg. Max Bid Cost	Avg. Terminations
Original	226.4	$69.83	$219.69	0
DrAFTS (1-hr)	225.4	$66.39	$85.08	0.24
DrAFTS (profiles)	228.5	$66.36	$79.29	1.03

Table 3: ***Comparison of Original Spot tier usage to selection using DrAFTS, April 1–2, 2017. Results are reported as average values from 35 experiments with each method.***

This application example further supports our belief that DrAFTS is capable of implementing probabilistic guarantees of duration in the Spot tier. In both sets of experiments DrAFTS met its target probability and reduced both the overall cost and the financial risk.

4.4 Cost Optimization

The purpose of DrAFTS is to be able to predict the minimum bid that will ensure a user-specified level of durability. While DrAFTS does appear to be able to obtain a specific durability guarantee, it is also possible to use it to optimize the cost of reliable execution in AWS. Specifically, by setting the DrAFTS durability target probability to the same probability provided by the On-demand SLA (0.99) the two can be considered functionally equivalent in terms of providing a "guarantee" to the user.

Note that DrAFTS is "conservative" in that it uses the upper and lower bounds to indicate the *worst case* cost that will ensure the specified durability probability. In some cases, this worst-case bid is greater than the On-demand price for the same instance type

in the same AZ. In many cases, however, the worst-case DrAFTS bid is less than the On-demand price indicating that even if the user pays the maximum DrAFTS cost, it would still be less than the On-demand cost. Further, the actual cost in the Spot tier can be considerably less than the DrAFTS bid when the probability target is high (*e.g.* 0.99). One might ask how "tight" (close to optimal) are DrAFTS's predictions with respect to the market price. While this is heavily dependent on the variability of the particular series, we refer readers to our technical report in which we investigate tightness for all instance types and AZ pairs and show ratios of between 4.8 and 7.5 on average [28].

The relationship between DrAFTS and On-demand pricing suggests the following strategy for provisioning instances. When a user wishes to submit a request for an instance type in a specific AZ with a known maximum duration, she can consult DrAFTS for a maximum bid that will ensure 0.99 durability and compare that bid with the current On-demand price for the same instance type in the same AZ. If the DrAFTS bid is lower, she requests the instance with the DrAFTS bid. If it is equivalent or higher, she requests an On-demand instance.

In so doing, she ensures that regardless of which request she chooses to make, the resulting instance will have at least a 0.99 probability of running for its maximum duration. That is, from the AWS SLA, the availability probability is 0.99 for On-demand instances so she is assured of at least this availability. As mentioned, availability, as specified in the SLA is weaker than durability in that the availability SLA does not specify that the availability time be contiguous. For the purposes of cost optimization, however, we will consider availability equivalent to durability when the maximum duration is considerably smaller than the availability SLA period.

AZ	On-demand Cost	DrAFTS-based Strategy Cost	Savings
us-east-1b	$47277.1	$35983.2	23.89%
us-east-1c	$49883.2	$41298.2	17.21%
us-east-1d	$47756.6	$40536	15.12%
us-east-1e	$33777.2	$32657.2	3.32%
us-west-1a	$32876.9	$18362.1	44.15%
us-west-1b	$34303.2	$20452.4	40.38%
us-west-2a	$50367.1	$32670.2	35.14%
us-west-2b	$48659.4	$28505.1	41.42%
us-west-2c	$22326.6	$13094.3	41.35%

Table 4: Comparison of On-demand Cost to DrAFTS Cost for a durability guarantee of 0.99 backtested instances, October 1, 2016 through December 1, 2016.

Table 4 shows the cost savings obtained from using DrAFTS bids in this way for all of the backtested instances used to generate the results in Section 4.1. Column 1 of the table shows the AZ in which the instances are located, column 2 the total On-demand cost that would have been paid if each experimental instance in that AZ were launched as an On-demand instance, column 3 shows the cost if each instance is launched using either the DrAFTS bid in the Spot tier or the On-demand price in the On-demand tier based on whether the bid is lower than the price, and column 4 shows the percentage savings that this cost optimization strategy generates over the On-demand tier.

Notice that the savings varies considerably by AZ. Part of this variation is due to considerable volatility in Spot-tier prices for specific AZ and instance-type combinations. For example, the *c4.4xlarge* instance type in the *us-east-1e* AZ has a spot price that varied between $0.13 and $9.5—almost two orders of magnitude—during the test period. Because of this variation, DrAFTS had to choose a bid price that is relatively high compared to the On-demand price.

Alternatively, in the same experiment, the maximum bid determined by DrAFTS for a *m1.large* instance type in the *us-west-2c* AZ over the 300 experimental trials was $0.10 (the smallest bid was $0.02) and the On-demand price for an *m1.large* in the *us-west-2* Region was $0.175. In this case, the DrAFTS bid was below the On-demand price for each instance. Because the bid is the most that a user would need to spend (the actual cost of the instance was less), substantial savings are possible with the same level of durability as is prescribed in the On-demand SLA.

Table 5 shows the cost savings using backtesting over the same time period when DrAFTS uses a durability guarantee probability of 0.95 rather than 0.99. In this experiment, the success probability of 0.95 is correctly determined (graph omitted for space considerations) for all AZ and instance-type combinations. Notice that the savings (compared to the savings with a 0.99 probability target shown in Table 4) can be significantly higher. If the application could tolerate the premature termination of 5% of its Spot tier requests, the user could obtain substantial savings using DrAFTS.

AZ	On-demand Cost	DrAFTS-based Strategy Cost	Savings
us-east-1b	$280316	$164786	41.21%
us-east-1c	$292009	$178920	38.73%
us-east-1d	$273276	$170587	37.58%
us-east-1e	$202290	$181591	10.23%
us-west-1a	$198719	$52983.1	73.34%
us-west-1b	$196362	$63882.8	67.47%
us-west-2a	$309716	$159324	48.56%
us-west-2b	$303714	$136608	55.02%
us-west-2c	$67483.6	$21236	68.53%

Table 5: Comparison of On-demand Cost to DrAFTS Cost for a durability guarantee of 0.95 backtested instances, October 1, 2016 through December 1, 2016.

Thus DrAFTS is able to illuminate where the opportunities are to obtain substantial savings. While we have aggregated the savings data on a per-AZ basis for purposes of brevity in this paper, a use of DrAFTS can determine on an AZ-and-instance-type basis where the best cost optimization potential might be.

Finally, DrAFTS makes it possible for the user to determine the SLA (rather than Amazon) in order to best exploit this cost savings for a particular application. Specifically, if the user application can tolerate a lower success probability associated with each instance request, DrAFTS can use a tighter price bound (implying a greater probability of termination) and, thus, obtain greater savings.

5 RELATED WORK

There have been many efforts to model pricing histories and calculate bid prices that satisfy various requirements (*e.g.*, minimizing cost). In [30] the authors model spot prices and derive optimal bidding strategies for managing the trade off between producing higher bids to avoid revocations and lower bids to reduce costs. They focus on determining a bid price that ensures (probabilistically) that a job completes within a (fixed) duration. Our work is differentiated by its focus on providing durability, not cost optimization. Furthermore, the authors make assumptions regarding bid arrivals being independent and identically distributed, which our analysis shows is not true, and for which our method takes into account. Finally, DrAFTS does not require probability density functions or cumulative distribution functions.

Tang et al. [26] propose an optimal bidding strategy for instances in the Spot tier. Their approach uses a Constrained Markov Decision Process to minimize the expected cost of an instance, taking into account its checkpointing and restart delays. In [14] and [15] the authors hypothesize a parametric model for price histories (based on exponentials), which they fit using Expectation-maximization (E-M). The work described in [12] attempts to solve a cost minimization problem that is based on a Markovian state model. The authors estimate transition probabilities from the Spot price histories. The authors of [27] describe a neural network-based approach to predicting prices in the Spot tier. Their approach (based on a mixture of Gaussians and a Box-Jenkins time series methodology) generates one-step-ahead predictions (with a granularity of 1.3 hours) with good accuracy. However, they point out that predicting the market for longer time frames should be encouraged as future research. DrAFTS constitutes such research in that it combines time series predictions of the bounds on price (for the next 5 minute interval) with a duration prediction essentially providing predictive bounds for arbitrary future durations.

Our work differs from previous work in several ways. DrAFTS is focused on making equivalent the reliability guarantees available from Amazon instance services classes (*e.g.*, On-demand) that carry a reliability SLA with the durability that users can obtain from the Spot tier. It is not a method for determining a bidding strategy that minimizes expected instance cost. Also, the breadth of our study is wider in terms of the time period we observe and the combinations of Region, AZ, and instance type we test in our verification process. Finally, our method is non-parametric and adaptive. Thus, it includes both change-point detection and autocorrelation compensation features that parametric approaches do not.

The authors of [1] investigate, at some length, the market dynamics associated with the Amazon Spot tier. Their hypothesis is that pricing in the Spot tier is not driven solely by client demand (*i.e.*, Amazon introduces hidden externalities that affect pricing). We concur with the analysis presented in [1], motivating us to turn to the time series mechanism described previously as an efficient adaptive technique. Again, DrAFTS is only providing a statistical bound predicted price and thus need not recover the "true" underlying market dynamic completely.

Researchers have also examined the question of using "live migration" and checkpointing to avoid downtime when a web service is hosted in the Spot tier [13, 22, 25]. They suggest both a reactive strategy that sets the maximum bid price to the On-demand price and performs a migration when the Spot price nears the bid. They also investigate a proactive strategy that uses a constant factor (greater than 1.0) to set the maximum bid price. Our work complements this approach in that it attempts to determine the probability and duration until a termination may happen.

Finally, the authors of [23] present a unique transient guarantee that provides probabilistic assurances on revocation characteristics. The general idea of this approach is that spare capacity is offered by providers according to different classes of service. Each class includes probabilistic assurances with respect to the instances' mean time to revocation. This approach differs from our work, in that it takes a provider's perspective rather than a user's. Thus, the authors focus on characterizing the performance and value of users' use of transient servers while incorporating the overhead of fault-tolerance.

6 CONCLUSIONS AND FUTURE WORK

Our goal in developing DrAFTS has been to determine the extent to which it is possible to use on-line statistical forecasting to generate a probabilistic guarantee of instance durability when a cloud provider offers dynamically priced "Spot" resources. To verify our overall methodology, which is based on non-parametric forecasting of univariate time series bounds, as well as to investigate the practical feasibility of our approach, we have conducted a number of experiments with the Amazon Spot tier over a long period of time. Our results combine extensive "backtesting" of previous price histories, empirical tests that launch instances in the Spot tier and record the outcomes, an analysis of the "real-world" service usage of the Spot tier in terms of instance durability and costs, and analysis of the potential benefits available to users when using an adaptive provisioning strategy. These results indicate that DrAFTS is able to provide a probabilistic guarantee of durability in the Amazon Spot tier for large probabilities up to 0.99. This probabilistic guarantee compares favorably to the reliability guarantee offered by Amazon for its more expensive On-demand tier of service.

Our future work will analyze the degree to which the availability of DrAFTS predictions may affect the market they are serving. It is clear that widespread use of DrAFTS (if it were to occur) would change the pricing dynamics of the Amazon Spot tier. We wish to understand both whether the predictive capability is degraded if many market participants were to use DrAFTS to determine their bids and also whether the market, as a whole, will appear more or less stable than it is currently.

SOFTWARE AVAILABILITY

The DrAFTS service is available at: http://predictspotprice.cs.ucsb.edu. Spot price histories are available at: http://www.cs.ucsb.edu/~rich/drafts-sc-2017-price-data/. The application simulator is available at: https://github.com/globus-labs/SCRIMP.

ACKNOWLEDGEMENTS

This research was supported in part by NSF grant ACI-1541215: The Aristotle Cloud Federation.

REFERENCES

[1] Orna Agmon Ben-Yehuda, Muli Ben-Yehuda, Assaf Schuster, and Dan Tsafrir. 2013. Deconstructing Amazon EC2 spot instance pricing. *ACM Transactions on Economics and Computation* 1, 3 (2013), 16.

[2] Amazon Web Services. 2016. Amazon EC2 Service Level Agreement. (2016). https://aws.amazon.com/ec2/sla/ accessed August 2016.

[3] Amazon Web Services. 2016. Amazon EC2 Spot Instances. (2016). http://aws.amazon.com/ec2/purchasing-options/spot-instances/ accessed April 2016.

[4] Amazon Web Services. 2016. Elastic Compute Cloud. (2016). https://aws.amazon.com/ec2/ accessed April 2016.

[5] Aristotle Cloud Federation. 2016. Aristotle Cloud Federation. (2016). https://federatedcloud.org accessed August 2016.

[6] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. 2015. *Time series analysis: forecasting and control*. John Wiley & Sons.

[7] Ryan Chard, Kyle Chard, Kris Bubendorfer, Lukasz Lacinski, Ravi Madduri, and Ian Foster. 2015. Cost-Aware Cloud Provisioning. In *Proceedings of the 11th IEEE International Conference on e-Science*. 136–144. DOI: https://doi.org/10.1109/eScience.2015.67

[8] Ryan Chard, Kyle Chard, Bryan Ng, Kris Bubendorfer, Alex Rodriguez, Ravi Madduri, and Ian Foster. 2016. An Automated Tool Profiling Service for the Cloud. In *Proceedings of the 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. 223–232. DOI: https://doi.org/10.1109/CCGrid.2016.57

[9] Ryan Chard, Kyle Chard, Rich Wolski, Ravi Madduri, Bryan Ng, Kris Bubendorfer, and Ian Foster. 2017. Cost-Aware Cloud Profiling, Prediction, and Provisioning as a Service. *IEEE Cloud Computing Magazine* PP (2017).

[10] Roy T Fielding and Richard N Taylor. 2002. Principled design of the modern Web architecture. *ACM Transactions on Internet Technology (TOIT)* 2, 2 (2002), 115–150.

[11] Google Cloud Platform. 2016. Google Preemptable Virtual Machines. (2016). https://cloud.google.com/preemptible-vms/ accessed April 2016.

[12] Weichao Guo, Kang Chen, Yongwei Wu, and Weimin Zheng. 2015. Bidding for Highly Available Services with Low Price in Spot Instance Market. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*. ACM, 191–202.

[13] Xin He, Prashant Shenoy, Ramesh Sitaraman, and David Irwin. 2015. Cutting the Cost of Hosting Online Services Using Cloud Spot Markets. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*. ACM.

[14] Bahman Javadi, Ruppa K Thulasiram, and Rajkumar Buyya. 2011. Statistical modeling of spot instance prices in public cloud environments. In *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*. IEEE, 219–228.

[15] Bahman Javadi, Ruppa K Thulasiram, and Rajkumar Buyya. 2013. Characterizing spot price dynamics in public cloud environments. *Future Generation Computer Systems* 29, 4 (2013), 988–999.

[16] Bogumił Kamiński and Przemysław Szufel. 2015. On optimization of simulation execution on Amazon EC2 spot market. *Simulation Modelling Practice and Theory* 58 (2015), 172–187.

[17] Ravi Madduri, Kyle Chard, Ryan Chard, Lukasz Lacinski, Alex Rodriguez, Dinanath Sulakhe, David Kelly, Utpal Dave, and Ian Foster. 2015. The Globus Galaxies platform: delivering science gateways as a service. *Concurrency and Computation: Practice and Experience* 27, 16 (2015), 4344–4360. DOI: https://doi.org/10.1002/cpe.3486 CPE-15-0040.

[18] Ravi K. Madduri, Dinanath Sulakhe, Lukasz Lacinski, Bo Liu, Alex Rodriguez, Kyle Chard, Utpal J. Dave, and Ian T. Foster. 2014. Experiences building Globus Genomics: a next-generation sequencing analysis service using Galaxy, Globus, and Amazon Web Services. *Concurrency and Computation: Practice and Experience* 26, 13 (2014), 2266–2279. DOI: https://doi.org/10.1002/cpe.3274 CPE-13-0338.R2.

[19] D. Nurmi, J. Brevik, and R. Wolski. 2005. Modeling Machine Availability in Enterprise and Wide-area Distributed Computing Environments. In *Proceedings of Europar 2005*.

[20] Daniel Nurmi, John Brevik, and Rich Wolski. 2008. QBETS: Queue bounds estimation from time series. In *Job Scheduling Strategies for Parallel Processing*. Springer, 76–101.

[21] Daniel Nurmi, Rich Wolski, and John Brevik. 2008. Probabilistic advanced reservations for batch-scheduled parallel machines. In *Proceedings of the 13th ACM SIGPLAN symposium on principles and practice of parallel programming*. ACM, 289–290.

[22] Prateek Sharma, Stephen Lee, Tian Guo, David Irwin, and Prashant Shenoy. 2015. Spotcheck: Designing a derivative iaas cloud on the spot market. In *Proceedings of the Tenth European Conference on Computer Systems*. ACM, 16.

[23] Supreeth Shastri, Amr Rizk, and David Irwin. 2016. Transient Guarantees: Maximizing the Value of Idle Cloud Capacity. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '16)*. IEEE Press, Piscataway, NJ, USA, Article 85, 11 pages. http://dl.acm.org/citation.cfm?id=3014904.3015019

[24] Yang Song, Murtaza Zafer, and Kang-Won Lee. 2012. Optimal bidding in spot instance market. In *INFOCOM, 2012 Proceedings IEEE*. IEEE, 190–198.

[25] Supreeth Subramanya, Tian Guo, Prateek Sharma, David Irwin, and Prashant Shenoy. 2015. SpotOn: a batch computing service for the spot market. In *Proceedings of the Sixth ACM Symposium on Cloud Computing*. ACM, 329–341.

[26] ShaoJie Tang, Jing Yuan, and Xiang-Yang Li. 2012. Towards optimal bidding strategy for Amazon EC2 cloud spot instance. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*. IEEE, 91–98.

[27] Robert M Wallace, Volodymyr Turchenko, Mehdi Sheikhalishahi, Iryna Turchenko, Vladyslav Shults, José Luis Vazquez-Poletti, and Lucio Grandinetti. 2013. Applications of neural-based spot market prediction for cloud computing. In *Intelligent Data Acquisition and Advanced Computing Systems (IDAACS), 2013 IEEE 7th International Conference on*, Vol. 2. IEEE, 710–716.

[28] Rich Wolski and John Brevik. 2016. *Probabilistic Guarantees of Execution Duration for Amazon Spot Instances*. Technical Report 2016-05. University of California, Santa Barbara. https://www.cs.ucsb.edu/research/tech-reports/2016-05.

[29] Murtaza Zafer, Yang Song, and Kang-Won Lee. 2012. Optimal bids for spot vms in a cloud for deadline constrained jobs. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*. IEEE, 75–82.

[30] Liang Zheng, Carlee Joe-Wong, Chee Wei Tan, Mung Chiang, and Xinyu Wang. 2015. How to Bid the Cloud. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. ACM, 71–84.