Probabilistic Guarantees of Execution Duration for Amazon Spot Instances
University of California Technical Report Number 2016-05 [*]

Rich Wolski
Computer Science Department
University of California, Santa Barbara

John Brevik
Department of Mathematics
California State University, Long Beach

Ryan Chard
Department of Computer Science
Victoria University of Wellington

Kyle Chard
Computation Institute
University of Chicago

1 Abstract

In this paper we propose a methodology for implementing probabilistic guarantees of instance reliability in the Amazon Spot tier. Amazon offers "unreliable" virtual machine instances (ones, indeed, that may be terminated by Amazon in response to changing demand for its other cloud products) 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 unreliable EC2 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 from Amazon *post facto*, both randomly and using real-world workload traces. We also test the efficacy of the method experimentally by using it to launch instances in the Spot tier, 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.

2 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 [5]) 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. Typically, if the agreement is violated (the user receives a lower "quality of service"), he or she is entitled to some form of financial compensation. Thus the cloud computing model is one in which users can reason about the capabilities that their computations require, and will receive, in terms of SLAs and not the physical capabilities of specific resources.

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. In particular, vendors such as Amazon and Google [9] 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" [4], 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 [24, 13].

Amazon also offers the *same* instance types under a fixed reliability SLA at a fixed price. 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, market prices are often sig-

[*]This work is supported in part by NSF grants CNS-0905237, CNS-1218808, ACI-0751315

nificantly 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, users who wish to ensure that their instances will be reliable must submit large maximum bids. Indeed, 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. Moreover, while a large body of work has investigated bidding strategies for optimizing the use of the Spot tier [26, 14, 21, 25, 11, 20, 22, 10], 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 by Amazon.

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. From 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 price available from the Spot tier.

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.[3] 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 archival Amazon Spot tier pricing data.
- It demonstrates the effectiveness of DrAFTS by detailing the execution of both synthetic and "real-world" application workloads in the Amazon Spot tier using DrAFTS-determined bids.

These contributions indicate that it is possible to make effective predictions of bounds on future price fluctuations in the Amazon Spot tier.

3 Amazon Spot Instances

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

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

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. 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 form the request, Amazon will choose one (without regard for price).

Also, 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 48 different 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 making the request is willing to pay for the instance.[4] This price is not re-

[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/` for the remainder of this paper.

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

[3] The financial risk in the Spot tier can be quite substantial – `https://moz.com/devblog/amazon-ec2-spot-request-volatility-hits-1000hour/`.

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

vealed 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.

3.1 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 more deterministic schedule.

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.[5] 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 financial risk associated with the instance. That is, the user "risks" paying up to the maximum bid price for each hour the instance executes. Because typographical and human-understanding errors have occasionally led to excessive costs, Amazon limits maximum bids to be ten times the On-demand price (*cf.* Section 5) for an instance type.

[5]Note that when the instance is terminated due to a market price change, the user pays only for the hourly usage up to the start of the hour in which the instance is terminated.

3.2 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 2016. Specifically, we have gathered 2-hour histories every 15 minutes (to account for possible dropout) and removed duplicate entries. We further restrict the DRAFTS predictions discussed in Section 5 to the *Linux/UNIX* images which carry no software licensing fee.

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 (they may if the mapping of the AZ name to the resources happens to be the same for two users). 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 building DRAFTS to run as a general service would. That is, to build a general purpose DRAFTS service, it would be necessary to map the AZ names used by the service to the AZ names visible to each user.

4 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[6] in that they are for *continuous* availability durations.

That is, 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

[6]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.

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.

Note that probabilistically, as the maximum bid price approaches the current market price, the duration an instance will execute decreases and any bid at market price is immediately eligible for termination. Notice also that any bid price above the DRAFTS-predicted bid price will also meet the probability target (any probability greater than the target is acceptable) but with added financial risk. A DRAFTS-determined bid attempts to guarantee *at least* the specified probability of completion while, at the same time, minimizing the financial risk associated with each bid.

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 maximum 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 maximum bid fractionally and computes a series of time durations until the maximum 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.

4.1 Non-parametric Bounds Prediction

DRAFTS uses a non-parametric time series analysis method that we developed in prior work [18]. We originally designed this method for predicting the bounds (upper and lower) on variable latencies occurring in large-scale computing systems [19]. It is non-parametric and it automatically 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

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

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 online use. Details of this implementation as well as a fuller accounting of the statistical properties (including correction for autocorrelation) and detailed assumptions are available in [18, 19, 17].

4.2 DRAFTS Prediction Methodology

DRAFTS uses the time series method 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 indicates 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.

It then generates a series of durations from the series of predictions in which each element of the series is the duration over which the prediction would prevent a market-price termination. That is, for each upper bound prediction in the prediction history, DRAFTS computes the duration until that prediction is no longer sufficient to prevent Amazon from terminating an instance in the Spot tier due to a change in market price if the prediction were used as a maximum bid.

DRAFTS then uses the time series method again to predict a lower confidence bound (again with $c = 0.99$) on the 0.025 $(= 1 - .975)$ 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.

4.2.1 Correcting for Market Price Equality

The combination of the upper confidence bound on the 0.975 quantile of price and the lower confidence bound on the 0.025 quantile of duration is almost, but not quite, what is needed to create a statistical guarantee for the Spot tier. Recall that Amazon may or may not terminate an instance when its maximum bid is equal to the market price. The time series method assumes that the bound on the desired target quantile is contained in the observed time series. That is, it returns a value from the series that is statistically "guaranteed" (under the binomial assumptions described previously) to be greater than or equal to the desired quantile. In each case, the value returned is some previously occurring market price. Thus the initial prediction method is correct for the time until the market price *exceeds* the predicted maximum bid price. It does not, however, account for the possibility that Amazon could terminate an instance because the bid price is *equal* to the market price.

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.

4.3 Performance

For all experiments described in Section 5 except those in Subsection 5.4 each DRAFTS maximum bid required approximately 2 minutes to generate using server class machines. This time-to-solution is acceptable for research purposes and we believe it can be optimized substantially.

For the experiments described in Subsection 5.4 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 resulting service implementation of DRAFTS operates asynchronously. It periodically queries the Amazon price-history API and computes a set of maximum-bid predictions for each instance type and AZ.[7] To access the bids it generates, clients use a simple Representational State Transfer (REST) [8] API, via which they can request a set of DRAFTS maximum bids for a specific instance type and AZ. The service computes dura-

[7]This service is currently operational in prototype form; however, we have elided the URL for the purposes of blind submission.

tion 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 and therefore must be preconfigured with the AZ mapping for its clients. All application-driven experiments described in Subsection 5.4 use this asynchronous on-line web service to obtain bid values.

5 Results

We validate the effectiveness of the DRAFTS method for determining maximum bids in the Spot tier using three sets of experiments. In the first, we use backtesting across all combinations of Region, AZ, and instance types available from the AWS North American regions (*us-east-1*, *us-west-1*, and *us-west-2*). For these *post facto* experiments we examine both the extent to which DRAFTS correctly ensures durability and the degree to which it over bid the actual price (*i.e.* we compare to the optimal bids that a prescient oracle would use) in each case. The second experiment runs a series of identical instances over a fixed time period in the same Region and AZ using the DRAFTS-determined maximum bid. It records the fraction of those instances that were terminated by Amazon because the market price exceeded the maximum bid and compares this fraction to the DRAFTS guarantee we configured in each experiment. Finally, we incorporate DRAFTS bid predictions into a production genetics analysis platform that elastically provisions instances in the Spot tier to host its computation. We compare the effectiveness of using DRAFTS to compute a bid when applied to a workload derived from production execution traces.

5.1 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 stamp 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 stamp 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.

In Figure 1 we show the percentage of AZ-instance type combinations that achieved < 0.99, 0.99, or 1.0 success fraction. In this experiment, we tested all in-

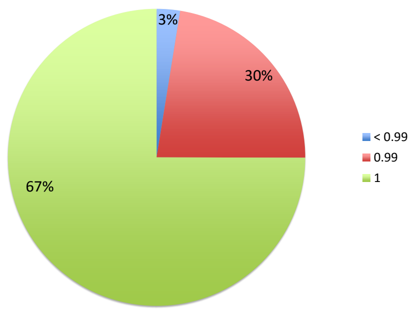

Figure 1: *Backtested* DRAFTS *correctness fractions for all instance types in us-east-1, us-west-1, and us-west-2 for the* 0.99 *quantile with* $c = 0.99$, *December 2015 through April 2016, using a sample size of* 300 *and a random instance duration of* 12 *hours.*

stance types available from the *us-east-1*, *us-west-1*, and *us-west-2* Regions.[8] Amazon reported 9 total AZs were available in these three Regions. There were 48 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 401.

For each combination we generated 300 Spot tier requests beginning at random times between December

[8]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*.

2015 and April 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 DRAFTS maximum bid and then determined whether that bid would have been sufficient to prevent Amazon from terminating the request if it had been made at the time selected in the past. The DRAFTS target probability was set to 0.99 with confidence bound $c = 0.99$.

From the figure, 3% of the combinations did not achieve either a 0.99 or 1.0 success fraction. Those that "failed" to achieve a success fraction of at least 0.99 all had a fraction of 0.98 except one, which had a fraction of 0.97. We believe that the methodology is in fact correct, despite our observation that not all of the experiments were able to generate a success fraction of 0.99 or higher (matching the target probability of 0.99 set for the experiment). In running the same experiment a second time, we observed that 5% of the combinations did not meet the DRAFTS probability target (again with all of the "failure" fractions 0.97 or 0.98). However (with four exceptions), the combinations that "failed" the first time and those that failed in the second set were different. We believe 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. Thus, depending on the random time stamps chosen, it is possible to see success fractions that are slightly below the probability target.

For the purposes of comparison, in Figure 2 we show the same statistics for the same sample when the *On-demand* price was used as a maximum bid instead of the DRAFTS-determined bid. 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 [3]. 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. This experiment 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 possible AZ and instance type combinations.

Further, many of the success fractions when the On-demand price was used as a maximum bid are substantially smaller than 0.99. Figure 3 shows the empirical cumulative distribution function (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.

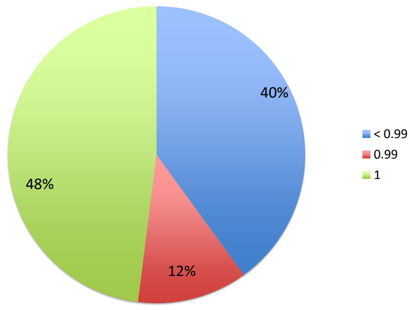

Figure 2: *Backtested correctness fractions for all instance types in us-east-1, us-west-1, and us-west-2 using the On-demand price as a maximum bid December 2015 through April 2016 with a sample size of* 300 *and a random instance duration of* 12 *hours.*

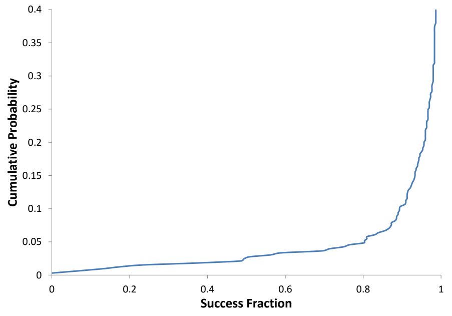

Figure 3: *CDF of Backtested* DRAFTS *correctness fractions less than* 0.99 *for all instance types in us-east-1, us-west-1, and us-west-2 using the On-demand price as a maximum bid December 2015 through April 2016 with a sample size of* 300 *and a random instance duration of* 12 *hours.*

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. Alternatively, the DRAFTS method does appear to provide a maximum bid that can ensure instance durability of at least 0.99 in the North American Regions.

5.2 Tightness

While DRAFTS does appear to be able to obtain a specific durability guarantee probabilistically, it does so "conservatively" by using the upper confidence bound on the quantile estimate from the price series and the lower confidence bound from the quantile estimate from the duration series. In addition to being correct in terms of the probabilistic guarantee, the DrAFTS methodology should also be "tight" in the sense that it should be as close to the market price as possible to minimize the difference between the maximum bid and the price paid. That is, because a user "risks" paying an hourly price up to the maximum bid for each instance the best bid is one that both assures the target durability probability and minimizes the "cost uncertainty" which is the difference between the actual cost and the maximum possible cost. We term this cost uncertainty the *uncertainty risk* associated with the use of the Spot tier.

Table 1 shows the tightness of the DrAFTS bids relative to the actual cost that would have been paid had we launched all instances of each instance type in each AZ. In the table, the first column is the AZ name as it appeared to our test account. The second column shows the dollar cost that would have been paid in the AZ for the successful instances (ones that outlived their bids) had they been requested from the Spot tier. The third column shows the total DrAFTS bid value for the AZ shown in column one. This value represents the amount of uncertainty risk a user would have taken on to complete the workload in that AZ. The fourth column shows the ratio of the total DrAFTS uncertainty risk to the total Spot tier cost (ratio of column three to column two). The fifth column shows the total On-demand price the user would have paid had she decided to use an On-demand instance to assure durability, and the last column shows the ratio of the total On-demand cost to the total cost incurred in the Spot tier.

The results in Table 1 illustrate the relationship between risk and cost for a 0.99 durability guarantee. For all but two of the AZs, the ratio of DrAFTS total uncertainty risk to the actual Spot tier cost is higher than the ratio of the On-demand cost to the Spot tier cost. Assuming DrAFTS is not behaving pathologically, this comparison indicates that prices in the Spot tier can fluctuate somewhat dramatically. For example, in *us-west-1b* DrAFTS had to recommend a bid that was, on the average, 7.5 times the Spot tier cost to account for the possibility that the prices might "spike" to this degree. However in *us-east-1b* and *us-west-2a* the DrAFTS bids risk less than what a user would need to pay in the On-demand tier to obtain the same durability level.

While it might appear that DrAFTS fails to obtain tight bounds relative to the On-demand cost, recall from Figure 2 that the On-demand price (when used as a maximum bid in the Spot tier) fails to ensure a 0.99 durability level in almost 40% of the AZ-instance type combinations. That is, in approximately 40% of the tested combinations, a bid above the On-demand cost is necessary to ensure a 0.99 success probability. In the Spot tier, then, it seems that users must risk paying more than the On-demand price to get an equivalent durability guarantee while the expected cost should be less (except for *us-east-1b* and *us-west-2a* where the DrAFTS uncertainty risk is lower than the On-demand cost). This variability suggests that it is possible to choose an AZ and instance type pair to minimize cost subject to the guarantee that DrAFTS can provide. We explore this possibility further in the next two sections.

5.3 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.

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

AZ	Spot tier Cost	DRAFTS Risk	Ratio	On-demand Cost	Ratio
us-east-1b	$8007.4	$46070.4	5.8	$47804.5	6.0
us-east-1c	$21722.5	$83483.8	3.8	$54240.9	2.5
us-east-1d	$10747.2	$79800.1	7.4	$57188.1	5.3
us-east-1e	$2038.4	$9815.0	4.8	$4161.2	2.0
us-west-1a	$9488.3	$69297.7	7.3	$39295.7	4.1
us-west-1b	$9495.9	$71441.0	7.5	$45145.4	4.8
us-west-2a	$9656.2	$53974.6	5.6	$57353.5	5.9
us-west-2b	$15168.3	$102984.0	6.8	$56986.2	3.8
us-west-2c	$14772.9	$85521.3	5.8	$57579.0	3.9

Table 1: *Comparison of* DRAFTS *Uncertainty Risk to the On-demand cost for all backtested instances.*

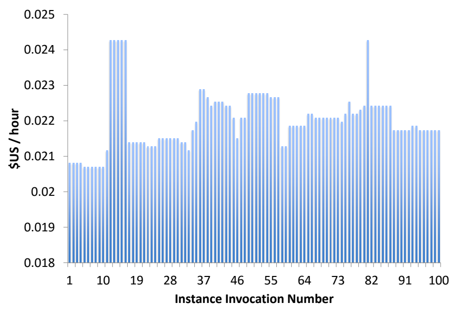

Figure 4: DRAFTS *Maximum Bids, p=0.95, 100 experimental instance launches of type c4.large in the us-east-1 Region, November 15 through November 22, 2015.*

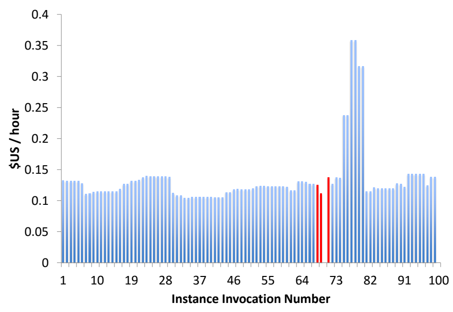

Figure 5: DRAFTS *Maximum Bids, p=0.95, 100 experimental instance launches of type c3.2xlarge in the us-west-1 Region, January 7 through January 14, 2016.*

seconds and a standard deviation of 687 seconds.

Figure 4 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 often 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 5 we show similar results for a second experiment where backtesting showed that DRAFTS would be less conservative at the 0.95 level. In this experiment, we used the same scripted experimental methodology to launch *c3.2xlarge* instances in the *us-west-1* Region. This 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 assumption that the failure percentage shown in Figure 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.

5.4 Application-Driven Experiments

As another test of DRAFTS efficacy we integrated it with a for-fee "Software-as-a-Service" for genome analysis [16, 15]. This service is used by more than 300 researchers and has consumed more than half a million Amazon instance hours over the past year. The service enables users to define and execute workflows composed of various genome analysis applications. Executed workflows are decomposed into individual jobs, which are then queued for execution. The service implementation includes a *provisioner* that monitors the job queue and provisions instances in the Spot tier to execute individual applications [6]. While the service exploits computational profiles – descriptions of the requirements of a particular application (*e.g.*, CPU and memory requirements) – to select suitable instance types [7] this information (which includes an execution time estimate) is not used in the generation of a bid. Indeed, before the work described herein, the developers of this service had no reliable way to use the execution times to influence their bid determinations.

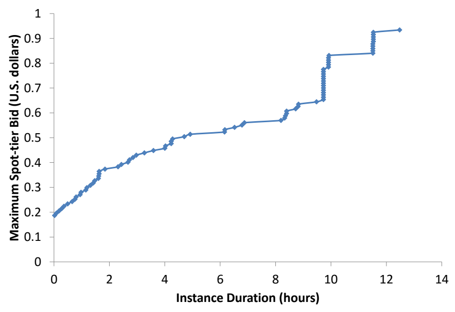

Figure 6: *Bid-duration relationship, in the us-east-1a AZ for the c3.4xlarge instance type at 10:16 AM PDT on April 18, 2016.*

Instance Type	Core Count (vCPU)
c3.2xlarge	8
c3.4xlarge	16
c3.8xlarge	32
g2.2xlarge	8
g2.8xlarge	32
m3.2xlarge	8
r3.2xlarge	8
r3.4xlarge	16
r3.8xlarge	32

Table 2: *Instance types considered by the genomics provisioner in the us-east-1 Region.*

When the work that is assigned to an instance finishes, the provisioner either terminates the instance or assigns new work to it. However it makes this decision shortly before the instance hour finishes. The instances used by this service are "Linux/UNIX (Amazon VPC)" which are priced separately from the standard "Linux/UNIX" variety we tested in the previous experiments. Because we had not used the "Linux/UNIX (Amazon VPC)" product type in the development of DRAFTS we believe it to be a useful test of the method's wider applicability.

To test DRAFTS with the genome analysis service, its provisioner was altered to use the DRAFTS service (*cf.* Section 4) to fetch the DRAFTS graph for each instance type under consideration in the *us-east-1* region. For example, Figure 6 shows the relationship between predicted duration until an instance will be terminated due to market price (on the *x*-axis) and DRAFTS-predicted maximum bid cost in U.S. dollars (on the *y*-axis) for the *c3.4xlarge* instance type in the *us-east-1a* AZ and the "Linux/UNIX (Amazon VPC)" product. The units along the *x*-axis are hours. 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.

The genome analysis service is preconfigured to use only a subset of instance types – those that are suitable for executing a broad range of applications required by its users. Table 2 shows the subset of instance types used in the *us-east-1* Region. Also, somewhat curiously, Amazon reports five AZs in *us-east-1* as being available for instance launches when the product type is "Linux/UNIX (Amazon VPC)". Recall that for not VPC instance types, only four AZs are available (*cf.* Subsection 5.1). Further, we mapped the AZ names visible to the software service account responsible for making Spot tier requests to the AZ names visible to the DRAFTS system.

The provisioning mechanism chooses which instance type to use based on application requirements. All of the instance types in the restricted set support 8, 16, or 32 cores (or "virtual CPUs" which is abbreviated as vCPU in the Amazon documentation). When an application is executed as part of a workflow, the application is mapped to a specific core count necessary for execution. The service chooses from among all instance types that can support the request and selects the AZ and instance type with the lowest current market price. Without DRAFTS predictions the provisioner uses a static maximum bid value (in this case 80% of the On-demand price for that instance type) in its Spot tier request.

We employ a representative workload to evaluate the use of DRAFTS in a real-world setting. The workload is derived from recording execution traces of five separate instances of the genome analysis service. To ensure a diverse workload we selected the busiest day from each instance of the service. Each job's submission time, execution time, and instance requirements (*e.g.*, required number of CPUs and estimated duration) are included in the workload. The workload includes 8452 jobs over a 24-hour period. In order to reduce the execution time and cost of running experiments on a commercial cloud we have reduced the workload to include only the first 1000 jobs in this test. 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 have transformed the submission time of each job into a relative submission time offset by the time of the day the job was submitted.

DRAFTS assumes that its users can specify the duration for which the probabilistic guarantee is required. Rather than changing the workflow scheduling algorithm (which does not take into account execution time) to include estimated instance durations, we simply assumed that the instance would be needed for no more than one hour when it was launched. Indeed, observing previous executions of the service for different user workloads shows that many of the instances are terminated after only a single hour by the provisioner. Technically, the DRAFTS predictions should only be valid for these one-hour instances; however, we also tested whether the one-hour maximum bid would remain "good" for those instances that exceeded one hour in duration (the longest running job has a duration of approximately 8 hours). The results of this test are not substantially different from the results we present here. As part of our future work we plan to consider how to make use of estimates of future instance lifetime.

To use DRAFTS to select which instance type and AZ, we experimented with two different approaches. The first computed the DRAFTS bid required to assure a duration of one hour with probability 0.99 for each candidate instance type and AZ and selected the one with the smallest maximum bid. The second used the average price since the last observed change point in the price series for each instance type and AZ combination. It then selected the lowest average price (but used the DRAFTS-determined maximum bid for that selection).

Table 3 compares the results of one replay experiment to complete the workload which took place on February 28 and 29, 2016 using two different DRAFTS methods for selecting and pricing instances. The entire application run consisted of 366 instance requests to the Spot tier. Using the service's original bid determination method (80% of the On-demand price) all 366 instances would have completed without being terminated by Amazon. The first row of the table shows the overall cost for all instances that the original method incurred, and the total uncertainty risk associated with its bidding strategy. The risk is computed as the difference between the maximum bid and the actual price paid for each instance. In the second row, we show the values the experiment generated when it used the lowest DRAFTS bid to select each AZ-instance type combination (this experiment used DRAFTS to determine the bid in for each instance). Finally, the third row shows the result that would have resulted from using the lowest average price instead of the lowest DRAFTS bid as the selector.

Method	Cost	Uncertainty Risk
Original	$106.10	$70.88
DRAFTS Bid	$91.78	$6.82
DRAFTS Avg.	$92.37	$7.38

Table 3: *Comparison of Original Spot tier usage to two different* DRAFTS *methods for selecting and pricing instances, February 28 and 29, 2016.*

This example further supports our belief that DRAFTS is capable of implementing probabilistic guarantees of duration in the Spot tier. All instances launched with a DRAFTS-determined maximum completed successfully. With a success probability of 0.99 and the conservative nature of DRAFTS predictions, these results are consistent with the previous experiments. Secondarily, in this example, DRAFTS would have both reduced the overall cost and the financial risk. Indeed, the risk is so low that it indicates the DRAFTS bids for the specific instance types and AZs in this experiment were much tighter than for the overall population of Region, AZ, and instance type combinations (*cf.* Table 1).

6 Related Work

In [11, 20, 22] the authors examine the question of using "live migration" and checkpointing to avoid downtime when a web service is hosted in the Spot tier. Using nested hypervisors, they describe a scheduler that can migrate a running web service between Spot instances and to do so without incurring an outage, their scheduler must predict when a Spot instance will be terminated in the future. They suggest both a reactive strategy that sets the maximum bid price to that of 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 provide a way to determine the probability and duration until a termination may happen. The authors show that combining On-demand and Spot instances lowers the cost associated with hosting a long-running web service. Our work can be used to augment the proactive approach they describe.

The work described in [10] postulates the use of Paxos (a distributed consensus algorithm) to manage replicated application state across Spot instances. It then attempts to solve a cost minimization problem that is based on a Markovian state model. The authors estimate transition probabilities directly from the Spot price histories.

Our work differs from this work in several ways. First, we focus exclusively on predicting the time until Spot instance termination as a function of the probability target provided to the DRAFTS method. Using the time series bound predictor our technique also takes into account the effects of autocorrelation in each Spot price history. However, because it provides a bound on duration, it may be possible to use DRAFTS as a method of estimating the Spot instance failure probabilities that their methodology requires.

The authors of [24] 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) for the spot market that are quite accurate. 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 durations into the future. The length of the prediction interval is determined by the probability of the bounds being too high.

In [12] and [13] the authors hypothesize a parametric model for price histories (based on exponentials) that they fit using Expectation-maximization (E-M). They also restrict their investigation to a few of the popular instance types available at the time. Our work is different in that our method is non-parametric and adaptive. Because DRAFTS does not require the complete distribution for each price history, it does not require the solution to a non-linear optimization problem (*e.g.* the use of the E-M algorithm). It can also be implemented using an incremental state update making it efficient enough for use in an on-line forecasting context. Further, it includes both change-point detection and autocorrelation compensation features that this previous work does not include in their parametric approach.

In [23] Tang *et al.* 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. Our work differs from this work in several ways. First, DRAFTS is focused on equivocating 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, DRAFTS generates a single maximum bid prediction for a given instance. The Tang methodology returns a probablistic strategy that is used to choose (randomly) between competing deterministic strategies.

Finally, the authors of [2] 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 [2], 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. The efficiency of the method combined with its non-parametric nature makes it possible to adapt to any introduced externalities "fast enough" to make on-line prediction possible.

7 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 offers dynamically priced "spot" resources. To this end, we have developed an implementation of a system for predicting the bid value that will ensure a specified level of durability with a specified probability.

To verify the 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. Our results combine extensive "backtesting" of previous price histories, empirical tests that launch instances in the Spot tier and record the outcomes, and an analysis of the "real world" service usage of the Spot tier in terms of instance durability and costs. 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 guarantee offered by Amazon for its more expensive On-demand tier of service where a durability SLA is available.

Our future work will examine the degree to which DRAFTS can be applied to other public cloud platforms (*e.g.* Preemptable Virtual Machines in Google Compute Engine). We also plan to 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. Finally, we plan an on-line service for implementing DRAFTS as a free service offering. We have experimented with several implementation strategies that could turn our current prototype into a generally applicable service and believe that it is now possible to do so.

References

[1] Agmon Ben-Yehuda, O., Ben-Yehuda, M., Schuster, A., and Tsafrir, D. Deconstructing Amazon EC2 spot instance pricing. *ACM Transactions on Economics and Computation 1*, 3 (2013), 16.

[2] Agmon Ben-Yehuda, O., Ben-Yehuda, M., Schuster, A., and Tsafrir, D. Deconstructing amazon ec2 spot instance pricing. *ACM Transactions on Economics and Computation 1*, 3 (2013), 16.

[3] Amazon Web Services. Amazon ec2 service level agreement, 2016. `https://aws.amazon.com/ec2/sla/` accessed August 2016.

[4] Amazon Web Services. Amazon ec2 spot instances, 2016. `http://aws.amazon.com/ec2/purchasing-options/spot-instances/` accessed April 2016.

[5] Amazon Web Services. Elastic compute cloud, 2016. `https://aws.amazon.com/ec2/` accessed April 2016.

[6] Chard, R., Chard, K., Bubendorfer, K., Lacinski, L., Madduri, R., and Foster, I. Cost-aware cloud provisioning. In *Proceedings of the 11th IEEE International Conference on e-Science* (Aug 2015), pp. 136–144.

[7] Chard, R., Chard, K., Ng, B., Bubendorfer, K., Rodriguez, A., Madduri, R., and Foster, I. An automated tool profiling service for the cloud. In *Proceedings of the 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)* (May 2016), pp. 223–232.

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

[9] Google Cloud Platform. Google preemptable virtual machines, 2016. `https://cloud.google.com/preemptible-vms/` accessed April 2016.

[10] Guo, W., Chen, K., Wu, Y., and Zheng, W. 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* (2015), ACM, pp. 191–202.

[11] He, X., Shenoy, P., Sitaraman, R., and Irwin, D. 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* (2015), ACM.

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

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

[14] Kamiński, B., and Szufel, P. On optimization of simulation execution on amazon ec2 spot market. *Simulation Modelling Practice and Theory 58* (2015), 172–187.

[15] Madduri, R., Chard, K., Chard, R., Lacinski, L., Rodriguez, A., Sulakhe, D., Kelly, D., Dave, U., and Foster, I. The globus galaxies platform: delivering science gateways as a service. *Concurrency and Computation: Practice and Experience 27*, 16 (2015), 4344–4360. CPE-15-0040.

[16] Madduri, R. K., Sulakhe, D., Lacinski, L., Liu, B., Rodriguez, A., Chard, K., Dave, U. J., and Foster, I. T. 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. CPE-13-0338.R2.

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

[18] Nurmi, D., Brevik, J., and Wolski, R. Qbets: Queue bounds estimation from time series. In *Job Scheduling Strategies for Parallel Processing* (2008), Springer, pp. 76–101.

[19] Nurmi, D., Wolski, R., and Brevik, J. Probabilistic advanced reservations for batch-scheduled parallel machines. In *Proceedings of the 13th ACM SIGPLAN symposium on principles and practice of parallel programming* (2008), ACM, pp. 289–290.

[20] Sharma, P., Lee, S., Guo, T., Irwin, D., and Shenoy, P. Spotcheck: Designing a derivative iaas cloud on the spot market. In *Proceedings of the Tenth European Conference on Computer Systems* (2015), ACM, p. 16.

[21] Song, Y., Zafer, M., and Lee, K.-W. Optimal bidding in spot instance market. In *INFOCOM, 2012 Proceedings IEEE* (2012), IEEE, pp. 190–198.

[22] Subramanya, S., Guo, T., Sharma, P., Irwin, D., and Shenoy, P. Spoton: a batch computing service for the spot market. In *Proceedings of the Sixth ACM Symposium on Cloud Computing* (2015), ACM, pp. 329–341.

[23] Tang, S., Yuan, J., and Li, X.-Y. Towards optimal bidding strategy for amazon ec2 cloud spot instance. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on* (2012), IEEE, pp. 91–98.

[24] Wallace, R. M., Turchenko, V., Sheikhalishahi, M., Turchenko, I., Shults, V., Vazquez-Poletti, J. L., and Grandinetti, L. 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* (2013), vol. 2, IEEE, pp. 710–716.

[25] Zafer, M., Song, Y., and Lee, K.-W. Optimal bids for spot vms in a cloud for deadline constrained jobs. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on* (2012), IEEE, pp. 75–82.

[26] Zheng, L., Joe-Wong, C., Tan, C. W., Chiang, M., and Wang, X. How to bid the cloud. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication* (2015), ACM, pp. 71–84.