July 2016

SpotLight: An Information Service for the Cloud

Xue Ouyang
University of Massachusetts Amherst

SPOTLIGHT:
AN INFORMATION SERVICE FOR THE CLOUD

A Masters Thesis Presented

by

XUE OUYANG

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL AND COMPUTER ENGINEERING

May 2016

Electrical and Computer Engineering

SPOTLIGHT:
AN INFORMATION SERVICE FOR THE CLOUD

A Masters Thesis Presented

by

XUE OUYANG

Approved as to style and content by:

David Irwin, Chair

Daniel Holcomb, Member

Michael Zink, Member

Christopher V. Hollot, Department Head
Electrical and Computer Engineering

ACKNOWLEDGMENTS

I would like to express my great appreciation to all those who provided me help and hope to finish my thesis. First of all, I would like to thank my advisor, Professor David Irwin, who gave me guidance for this project, stimulating ideas as well as available help and encouragement when needed. He is the source power, Spot Light for our project.

Then, thanks for my other committee members, Professor Daniel Holcomb and Professor Michael Zink, who helped and provided great advice for my thesis proposal and final presentation.

Also, I would express my gratitude to Professor Preshant Shenoy, who also provided some stimulative suggestions and help to coordinate our project. And I would like to appreciate AWS who provided educational funding to our school, and a special thanks goes to our school that provided us with full-access to all papers and other materials and proposal template to work on.

ABSTRACT

SPOTLIGHT:
AN INFORMATION SERVICE FOR THE CLOUD

MAY 2016

XUE OUYANG

B.E., WUHAN UNIVERSITY OF TECHNOLOGY, WUHAN, HUBEI, CHINA

M.S.E.C.E., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor David Irwin

Infrastructure-as-a-Service cloud platforms are incredibly complex: they rent hundreds of different types of servers across multiple geographical regions under a wide range of contract types that offer varying tradeoffs between risk and cost. Unfortunately, the internal dynamics of cloud platforms are opaque in several dimensions. For example, while the risk of servers not being available when requested is critical in optimizing these risk-cost tradeoffs, it is not typically made visible to users. Thus, inspired by prior work on Internet bandwidth probing, we propose actively probing cloud platforms to explicitly learn such information, where each "probe" is a request for a particular type of server. We model the relationships between different contracts types to develop a market-based probing policy, which leverages the insight that real-time prices in cloud spot markets loosely correlate with the supply (and availability) of fixed-price on-demand servers. That is, the higher the spot price for

a server, the more likely the corresponding fixed-price on-demand server is not available. We incorporate market-based probing into SpotLight, an information service that enables cloud applications to query this and other data, and use it to monitor the availability of more than 4500 distinct server types across 9 geographical regions in Amazon's Elastic Compute Cloud over a 3 month period. We analyze this data to reveal interesting observations about the platform's internal dynamics. We then show how SpotLight enables two recently proposed derivative cloud services to select a better mix of servers to host applications, which improves their availability from $\sim$70-90% to near 100% in practice.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

Companies are increasingly renting their computation and storage from Infrastructure-as-a-Service (IaaS) cloud platforms in lieu of maintaining their own private infrastructure. IaaS cloud platforms, such as Amazon's Elastic Compute Cloud (EC2) and Google Compute Engine (GCE), rent access to millions of servers in hundreds of data centers spread across the globe [10]. These platforms offer their customer's the illusion of infinite scalability on-demand. In reality, though, cloud resources are not infinite: if the demand for servers ever exceeds the available supply, then the platform cannot satisfy all requests. Since cloud platforms are undergoing rapid growth, staying ahead of customers' "demand curve" by continually increasing their supply of servers to satisfy rising demand is often challenging. In addition, cloud platforms are actually an aggregation of many smaller independent clusters of various server types spread across multiple data centers in different geographical regions, such that each server type in each data center of each region experiences its own daily, weekly, and seasonal fluctuations in demand. For example, EC2 customers may choose from up to 53 server types in 26 availability zones[1] across 9 regions.

Thus, despite their massive global scale, cloud platforms often experience periods where demand for a particular type of server (in some data center of some region) exceeds their supply or vice versa. Neither scenario is attractive. When demand exceeds supply, the platform must reject customer requests, which may cause customers

[1]Each availability zone is independent of others in the same geographical region, and consists of one or more data centers [10].

to lose trust in the platform, especially if it denies access to server resources at a critical business time, e.g., an unexpected spike in demand for a company's product or service. Likewise, when supply exceeds demand, the platform has idle resources that represents a partial waste of their massive capital investment in server infrastructure. Ideally, to maximize revenue, platforms would waste no resources by always operating at 100% utilization.

To address the issues above, cloud platforms are increasingly renting their servers under a wide range of *contracts* that specify different levels of availability and cost. In general, the stronger the availability guarantee the platform offers, the higher the cost. For example, for a relatively high cost, EC2 offers *reserved* "instances," i.e., virtual machines (VMs) with a specified allocation of CPU, memory, and I/O resources, which it guarantees will always be available—that is, EC2 guarantees the demand of reserved instances will never exceed their available supply. At the other end of the spectrum, for a relatively low cost, EC2 offers *spot* instances, which it may revoke at anytime and have no availability guarantee. Each contract serves a different purpose: reserved instances reassure wary customers that servers will be there when needed, while spot instances enable EC2 to increase utilization and gain some revenue from otherwise idle resources. Of course, most users rent *on-demand* instances, which cannot be revoked like spot instances, but have weaker availability characteristics than reserved instances—they are usually available but there is no guarantee.

While cloud platforms publish the prices and terms of their contracts, there is little else known about server availability in their underlying infrastructure. As a result, determining the actual value of each contract relative to the other is not possible, which makes it difficult to assess the mix of resources necessary to obtain a particular balance between the risk of servers not being available and their cost. For example, reserved instances incur a fixed cost over the length of their availability

guarantee. However, if a user does not fully utilize the reserved instance over its contract term, then its amortized per-hour cost may be much more than an on-demand instance. Determining whether the reserved instance is worth it requires knowing how frequently on-demand instances are unavailable—if their availability is near 100% then an on-demand instance may offer similar performance at a much lower cost.

To address the problem, we design SpotLight, a large-scale system for intelligently probing the cloud to learn about the availability of its underlying infrastructure. Although many cloud platforms now offer a diverse set of contracts, we focus on EC2 in this paper, since it remains the largest and most mature cloud platform. SpotLight is inspired by prior work on Internet bandwidth probing [14, 11, 5, 6]. Similar to the Internet, cloud platforms are large and complex "black boxes" that experience highly dynamic variations in resource availability over time. Thus, knowing the (current and historical) availability of server resources in the cloud is useful in selecting server types to optimize an application's performance and cost, just as knowing the availability of network resources, i.e., bandwidth, is useful in selecting routes to optimize network performance. Akin to launching probe packets into the network, SpotLight actively probes cloud *markets* by issuing "probe" requests for servers in various markets. Here, a market refers to a distinct server type offered under multiple contracts, i.e., reserved, on-demand, and spot. For EC2, each instance type in a particular availability zone of a geographical region represents a distinct market.

Each probe in SpotLight incurs a small cost, since it must pay for some minimum time on a requested server, if it is allocated. Thus, an explicit goal is to maximize the probability of detecting server unavailability for each probe. Our key insight is that EC2 likely offers the same physical pool of resources in each market under multiple different contract terms. As a result, there exist loose correlations between the availability and price of resources rented under different contracts in the same

market. For example, a spike in the price of spot instances in a market might correlate with the unavailability of on-demand instances in that market. SpotLight exploits these correlations to optimize when and where it probes. Our hypothesis is that, by intelligently probing for server availability, SpotLight provides useful information in selecting the server types and contracts to optimize application performance and cost. In evaluating our hypothesis, we make the following contributions.

• **Large-scale Availability Study.** We conduct the efirst large-scale study of availability in EC2 to quantify for different contracts in each market i) the availability information we can learn from each probe, ii) the frequency of server unavailability, and iii) the correlation between dynamic server prices and availability characteristics.

• **Adaptive Probing Policy.** We then use the data from our availability study to design a simple adaptive probing policy for SpotLight, which, given a probing budget, optimizes when and where to probe to maximize the probability of detecting server unavailability.

• **Implementation and Evaluation.** Finally, we implement a SpotLight prototype in `python`, deploy it on EC2, and use it to monitor the availability of over 4500 markets over multiple months. We evaluate its ability to detect and predict periods of unavailability, and then conduct a case study to quantify how SpotLight improves application performance. In particular, we show that a key assumption in recent work—that an application can always fail-over to on-demand servers if a spot instance is revoked—is not correct, since spot revocations often correlate with periods of unavailability in on-demand servers. As a result, the actual availability of such applications will be much lower than reported— roughly 75% instead of >99%. However, SpotLight can aid these applications in maintaining >99% availability by informing these applications of markets with a higher likelihood of availability.

CHAPTER 2

BACKGROUND AND MOTIVATION

While all cloud platforms now offer a complex set of server and contract options, we focus specifically on EC2 in this paper, since it remains the largest and most mature platform. In addition, EC2 is currently the only cloud platform that releases real-time spot price data that, as we show, provides some visibility into its internal availability characteristics. We first provide the details of EC2's contracts, as listed in Table 2.1, and then outline our probing approach, which exploits the relationships between servers offered under each contract.

2.1 Cloud Contracts

2.1.1 On-demand Servers

On-demand servers are the primary contract type offered by EC2, and other cloud platforms, where users may request servers at any time and, once allocated, pay a per unit-time price for their duration of use. Importantly, users, and not the platform, decide when to terminate on-demand servers. Thus, if they are willing to pay, users may run on-demand servers for as long as they wish.

While on-demand instances are simple for users to understand, they have a significant drawback for risk-averse users: they are not guaranteed to be available when requested. Put simply, the demand for EC2 servers may periodically exceed their supply, and, as a result, EC2 can and does often reject requests for on-demand servers by returning an `InsufficientInstanceCapacity` error code. This risk of rejection may prevent risk-averse users from employing on-demand cloud servers. For example, an

Contract Type	*Cost*	*Revocable*	*Availability*	*Obtainability*
On-demand	High	No	High	Not Guaranteed
Reserved	High	No	High	Guaranteed
Spot	Low	Yes	Variable	Not Guaranteed
Spot Blocks	Medium	No	Variable	Not Guaranteed

Table 2.1. Contract cost and characteristic tradeoffs

online service may decide that the risk of rejection at a critical business time is too costly to offset the potential savings from using cloud servers.

2.1.2 Reserved Servers

To cater to risk-averse users, EC2 also offers reserved servers, which it guarantees will be available when requested. Thus, if a user is not running a reserved server, and they request to start it, then EC2 guarantees it will not reject the request, i.e., it will have sufficient capacity to immediately start the server.[1] Currently, users pay a fixed cost for reserved servers for either a 1-year or 3-year term, regardless of whether or not the servers are running. While reserved servers cost 25-60% less than on-demand servers if they are fully utilized over their term, they require users to accurately estimate their long-term resource demands and eliminate much of the elasticity benefits of the cloud for variable workloads. Of course, if reserved servers are not fully utilized then they may cost significantly more than using equivalent on-demand servers only when necessary.

2.1.3 Spot Servers

Finally, EC2 offers spot servers, which, unlike on-demand and reserved servers, it may revoke at any time. Spot servers enable EC2 to gain revenue from otherwise idle resources without sacrificing the freedom to reclaim those resources for higher-

[1]EC2 may reject the initial request for a reservation. The guarantees above only apply to reservations EC2 has granted.

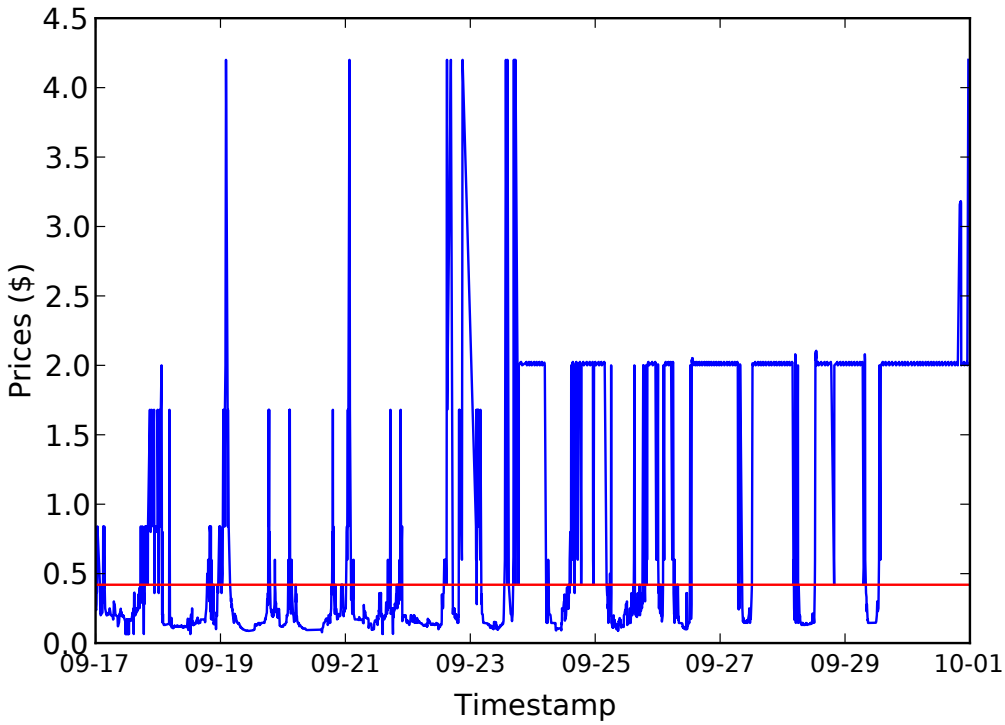


Figure 2.1. Spot prices vary dynamically and may exceed the on-demand price.

priority tasks, including requests for reserved and on-demand servers or for internal workloads. Without spot servers, to guarantee reserved servers are available when requested, EC2 would have to waste significant resources by maintaining a physically idle server for each unused reserved server. Thus, spot servers enable EC2 to lease unused reserved servers and prevent wasting those resources.

EC2 allocates spot servers using a market-based approach. Users request a spot server by specifying the maximum price they are willing to pay for it per unit time. EC2 then satisfies the request if the user's bid price is greater than the server's current *spot price*, which varies in real time. While allocated, users pay the variable spot price for the server and not their maximum bid price. The spot price is market-driven, and determined similar to a second-price auction [16] where the lowest winning bid dictates the spot price. However, if the spot price ever rises about the user's maximum bid price, EC2 immediately revokes the server after a brief warning [1].

EC2 operates a different spot market, with a different dynamic price, for each instance type and configuration of each availability zone in each region. As a result, there are currently ∼4500 spot markets with their own dynamic real-time spot price across multiple server configurations, e.g., Windows, Linux, and SUSE Linux with and without virtual networking, from the 53 instance types in the 26 availability zones over 9 regions in EC2. Importantly, the spot price for each market is public and published in real time. In addition, EC2 provides price data for each market over the past 3 months and pubic repositories are available that provide multiple years of price data. Thus, while spot servers come with no availability guarantees, users can analyze the historical data to estimate a spot server's availability for a given bid price, i.e., the percentage of time the spot price is below the bid price, based on its historical availability.

Figure 2.1 depicts the spot price at the end of September 2015 for the `c3.2xlarge` instance type in one availability zone of the U.S. East region. Note that the spot price periodically exceeds the on-demand price, indicated by the horizontal line, even though spot instances have weaker availability characteristics than on-demand instances. This is not an isolated incident: while not frequent, the spot price routinely rises above the corresponding on-demand price. In one documented case, the spot price rose to near $1000 per hour [2]. The rise was the result of a sudden spike in demand coupled with a high "convenience" bid made by a user to prevent revocations (under the assumption that the spot price would never rise above the on-demand price). As a result, EC2 introduced a maximum bid price for each spot instance—currently equal to 10× the price for the corresponding on-demand instance—to prevent customers from placing excessively high bids.

2.2 Active Probing Approach

EC2 offers numerous server types under the different contract types above. Optimizing the choice of contract type depends not only on a server's price, but also its availability. For example, if a user does not fully utilize a reserved server over its term, then its amortized per-hour cost may be much more than an on-demand server. Determining whether the reserved server is worth it requires knowing how frequently on-demand servers are unavailable—if their availability is near 100%, then an on-demand server may offer similar performance at a much lower cost. A similar decision exists when deciding whether to use a spot or on-demand server. *Unfortunately, the availability of on-demand servers is not made visible to users, which prevents users from making informed decisions.*

Thus, we propose to gather availability data for on-demand and spot servers by actively probing EC2 and exposing it to users, enabling them to make informed server and contract selection decisions. In this case, a probe is simply a request for an on-demand or spot server of a specific type (in a particular availability zone and region). If EC2 allocates a server for the request, then we record that the on-demand or spot servers are available. However, if EC2 does not allocate a server, but instead returns an `InsufficientInstanceCapacity` error code, then we record that server is not available. Of course, each probe may incur a cost, since there is a minimum charge—one hour of server time in EC2—for each allocated server. As a result, we cannot blindly flood EC2 with probes.

Instead, we adopt a market-based approach to probing that tracks the spot price of a server and triggers probes of the corresponding on-demand and spot servers when the spot price rises. *Our key insight is that EC2 likely divides the the same fixed pool of physical resources between the different contracts above.* As a result, the spot price in each spot market not only indicates the current price of spot servers, but also indirectly indicates the availability of on-demand and spot servers of the same type,

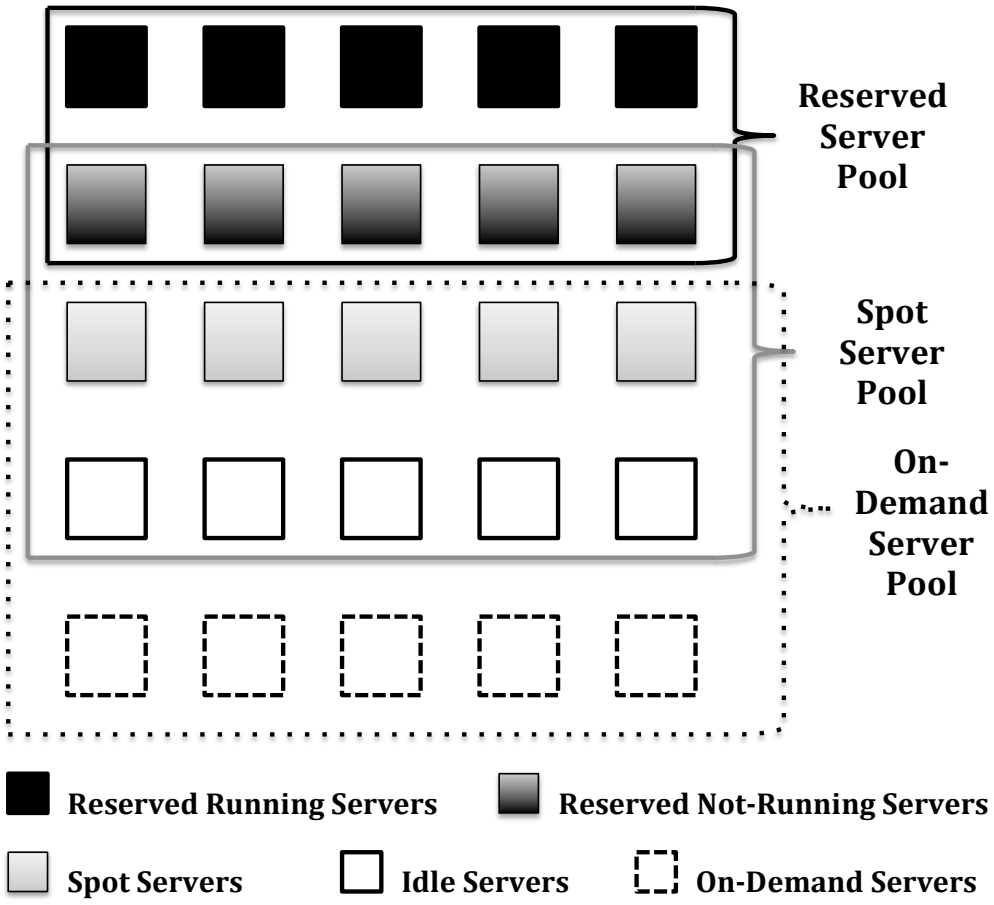


Figure 2.2. Relationship between reserved, on-demand, and spot servers hosted on the same pool of physical resources.

or, more precisely, in the same family, as we discuss in Chapter 3. For example, a sudden rise in the spot price may be the result of a surge in requests for on-demand (or reserved) servers, causing EC2 to revoke some spot servers and decrease the supply of resources available for spot servers. Of course, a sudden rise in the spot price could also result from a surge in requests for spot servers with high bids even if the supply is fixed (and there are no additional on-demand requests). Thus, there is only a partial relationship between the spot price dynamics and on-demand and spot availability.

Figure 2.2 depicts our model of the relationship between reserved, spot, and on-demand servers hosted on the same pool of physical resources. The total physical resources minus the resources allocated to running reserved and on-demand servers dictates the resources available for spot servers. In addition, there is an upper bound on the supply of resources available for on-demand servers equal to the total physical resources minus the number of reserved servers that have been granted regardless of whether they are running. Finally, there is also a lower bound on the supply of resources available for spot servers equal to the number of reserved servers that have been granted, but are not currently running. Thus, the allocation and deallocation of reserved and on-demand servers affects the supply of spot servers, and hence also the spot price. For example, a request for an unused reserved server reduces the pool of spot servers by one. Likewise, if there are no idle servers, new requests for on-demand servers are fulfilled by taking servers away from the spot pool. The reduction in supply of spot servers drives up the spot price by decreasing number of spot servers, and thus increasing the value of the lowest bid receiving resources, and hence the spot price.

CHAPTER 3

SPOTLIGHT DESIGN

We design SpotLight as a general information service for IaaS cloud platforms. SpotLight exports a query interface that enables applications or users to query information about the availability characteristics of different server types and contracts. SpotLight gathers the data for the query interface by both actively probing the platform, and by passively monitoring the spot price for each of the ∼4500 spot markets in EC2. In both cases, SpotLight logs the data in a database and defines an API that applications may use to query the database. While we focus on SpotLight's active probing in this paper, it also enables queries on historical spot price data across multiple markets. For example, an application might query SpotLight for the top ten server types with the longest mean-time-to-revocation for a bid price equal to the corresponding on-demand price over the past week. These servers would represent the most stable spot markets over the past week.

While SpotLight is a useful tool for users, we designed it for programmatic access by applications that use automated policies to continuously optimize server and contract selection based on changing cost and availability. In Chapter 6, we provide examples of how SpotLight benefits two such applications.

3.1 Market-Based Active Probing

Based on the relationship between the spot price and the availability of on-demand servers, we develop a cost-aware market-based probing policy. Our base policy triggers a probe whenever the spot price spikes above a certain threshold P_{spike} under the

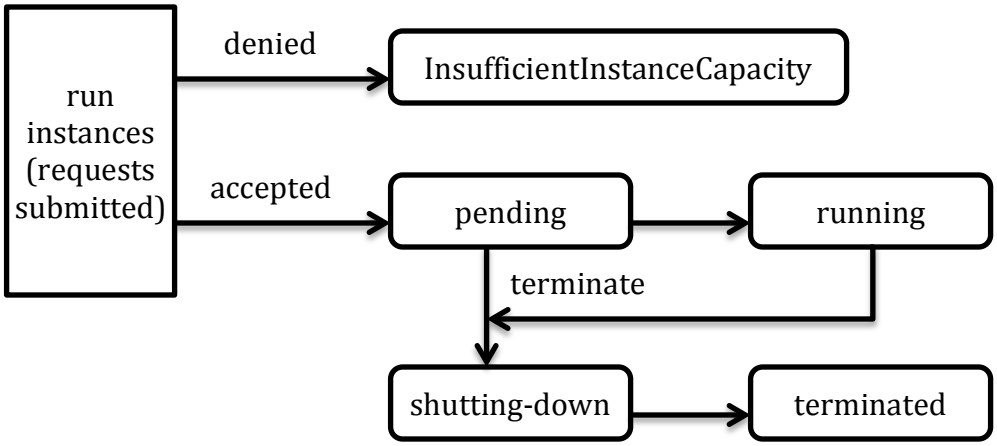


Figure 3.1. State machine for EC2 on-demand instances.

assumption that a spike in prices partially correlates with a decrease in the supply of resources available for spot servers. A probe is simply a request for a single on-demand server of the same type (in the same availability zone and region) as the spot market that experienced the spike. If the request is fulfilled, SpotLight logs the timestamp of the request, and then terminates the server. If the request returns an `InsufficientInstanceCapacity` error code, or any other error code, SpotLight logs both the timestamp of the request and the error code that caused it not to be fulfilled. Upon identifying that an on-demand server is not available, SpotLight continues to issue probes at a periodic interval until a probe request is fulfilled, and the server is available again.

3.2 On-demand Probing

Based on the relationship between the spot price and the availability of on-demand servers, we will develop a cost-aware market-based probing policy. Our base policy triggers a probe whenever the spot price spikes above a certain threshold P_{spike} under the assumption a spike in prices loosely correlates with a decrease in the supply of resources available for spot servers. Recall that a probe is simply a request for a

single on-demand server of the same type (in the same availability zone and region) as the spot market that experienced a spike. If the request is fulfilled, SpotLight logs the timestamp of the request, and then terminates the server. If the request returns an `InsufficientInstanceCapacity` error code, or any other error code, depicted in Figure 3.1, SpotLight logs both the timestamp of the request and the error code that caused it not to be fulfilled. Upon identifying an on-demand server that is not available, SpotLight continues to issue probes at a periodic interval until a probe request is fulfilled.

SpotLight only uses spot price to trigger the initial probes. After discovering an unavailable on-demand server, it also issues probes to on-demand servers in the same family across all availability zones within the regions. We define a family as server types with the same prefix, such as m3.*, t2.*, or c4.*.

3.2.1 Probing within Family

SpotLight only uses spot prices to trigger the initial probes. After discovering an unavailable on-demand server, it then issues probes to on-demand servers in the same family. We define a family as server types with the same prefix, such as m3.*, t2.*, or c4.*. SpotLight issues probes to servers within the same family of the same availability zone, since different server types in the same family likely reside on the same pool of physical servers. In contrast, servers types in different families have characteristics, such as CPU capacity and number of cores, that vary across different physical servers, making it unlikely they share common physical resources. In addition, the size (in terms of both CPU cores and memory allocation) of server types within each family often differs by a factor of two, e.g., an `m3.2xlarge` is twice as large as an `m3.xlarge`, which is twice as large as an `m3.large`, etc. These sizes are likely chosen to simplify the allocation of server types within a family on the same physical resources, as bin-

packing server types with variable sizes complicates resource allocation, and often leads to server fragmentation and waste.

Since the spot price for each server type is a function of both the available supply and demand, the spot price for each server type within the same family may not spike at the same time even if there is a decrease in supply. For example, even if the supply shrinks, the set of bids and the demand may not be high enough to alter the spot price in some spot markets. Thus, a spike in the spot price for one server type that correlates with unavailability may also be associated with unavailability of on-demand servers of other types within the same family, even if the other server types did not experience a spot price spike.

3.2.2 Probing across Availability Zones

In response to detecting an unavailable on-demand server, SpotLight issues probes not only to servers within the same family of the same availability zone, but also to other availability zones. While availability zones in EC2 are designed to be physically independent, i.e., be located in different data centers at different locations in the same geographical area, their demand exhibits strong dependencies. For example, unless a user explicitly specifies a specific availability zone, EC2 has the freedom to fulfill a request from any availability zone. As a result, increases in aggregate demand for on-demand servers may be correlated across availability zones. Thus, detecting an unavailable on-demand server in one availability zone may indicate that servers in other availability zones are also unavailable (if the unavailability is due to a correlated increase in demand across all availability zones). SpotLight treats probes to related markets similarly to its initial probe: if any request returns an error code that causes not to be fulfilled, SpotLight logs the error code and continues to periodically probe the instance every δ time units until the probe request is fulfilled.

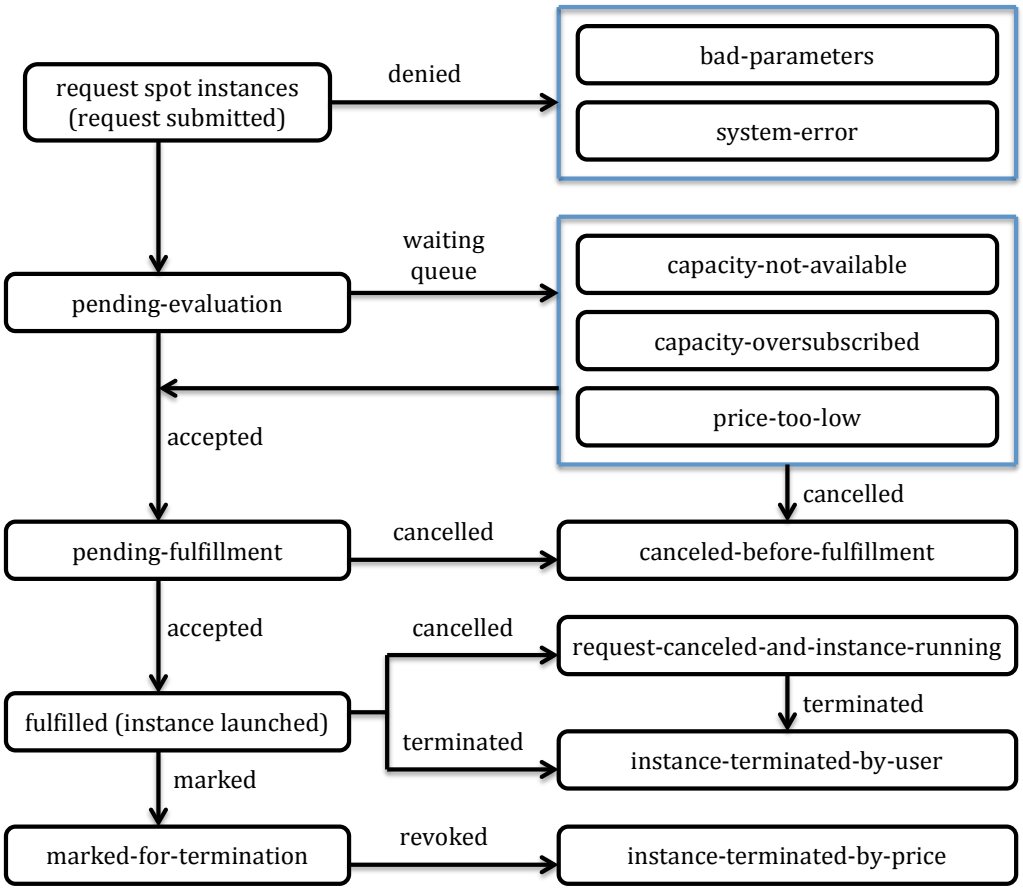


Figure 3.2. State machine for EC2 spot instance requests.

3.3 Spot Probing

SpotLight also includes the ability to probe the spot market directly by issuing probe requests for spot servers with different bid levels. EC2 may not fulfill a spot server request for multiple reasons, including due to a spot price being too low, capacity not being available, or capacity being oversubscribed, i.e., there being too many bids equal to the spot price to satisfy all of them. The primary difference between probing spot servers and probing on-demand servers is that a spot server probe requires SpotLight to specify a bid. SpotLight periodically issues probes for spot servers by setting the bid equal to the published spot price. As depicted in Figure 3.2, if the request returns the status code `capacity-not-available`, then,

just as above, SpotLight logs the timestamp and result of the request, and continues to issue the probe by bidding the spot price until the capacity becomes available. If the request returns status code `price-too-low` or `capacity-oversubscribed`, SpotLight continues to increase the bid by a small amount and re-issue the request until it determines the minimum bid price required to acquire the spot server.

Note that the price of spot instances is on average 10× less than the price of on-demand instances, so SpotLight simply issues these requests periodically, rather than only issuing them during times of price spikes. Our current prototype rate limits spot server probes by dividing its budget by the average historical spot price to determine the frequency it can issue probes over a given time window without exceeding its budget. As we discuss, since spot server probes rarely return a `capacity-not-available` error code, we do not include support for probing related spot servers within the same family.

3.4 Controlling Probing Costs

Each probe incurs a cost, since SpotLight must pay for one hour of server-time for each fulfilled probe request. The cost of such probes is likely to decrease in the future, as cloud platforms are adopting more fine-grained charging models. For example, Google Compute Engine charges only for the first 10 minutes if a serve is deactivated within its first 10 minutes. SpotLight may adjust its costs by simply changing its probing threshold T. A lower threshold issues more probes and incurs higher costs, while a higher threshold issues fewer probes and reduces costs. SpotLight may use historical spot price data for each market to determine a proper threshold for a given budget over some probing window, e.g., a month.

One problem with simply adjusting the threshold to control costs is that the probing budget may be too small to probe any but the rarest events, e.g., spikes >7× the on-demand price. As a result, SpotLight also enables users to set a sampling

ratio, such that it only probes a price spike greater than a threshold T with some probability p. By lowering p, we can also lower T and sample some fraction of less-volatile events. Currently, the cost of probes to related servers are deducted from the budget of the server that triggered the probes. We treat these related server probes as overhead, and do not consider the expected number of them when setting T and p. As a result, if we set T and p such that the budget should last for a specific duration based on historical data, we might deplete our budget before the time window is over. Of course, we could easily extend the scheme above to account for the expected cost of related server probes based on historical probing data.

Finally, SpotLight supports simple budgeting over a configurable time window, such that if it consumes its budget within the window, it simply stops probing until the next time window. In our current prototype, to maximize data collection, we set T equal to the on-demand price and sample every event.

CHAPTER 4

IMPLEMENTATION

We implemented a prototype of a information plane SpotLight in *python* based on AWS EC2 and one of its python API *Boto3*. This prototype is capable of monitoring and actively probing markets on EC2 all or specific *on-demand* and *spot* markets to learn market characteristics from probing requests. Users can use SpotLight to monitor and learn one or several or even all 4500 spot markets and more than 1000 on-demand markets (mainly for Linux, Windows, SUSE Linux) at the same time. SpotLight uses multiple concurrent market classes with their own identifiers *MarketID*, combined by availability zone (for spot market) or region (for on-demand market), product description and instance type, as well as market basic information and relationship between spot and on-demand markets. For each market, it will have its own probe manager to decide when and where to execute what kind of probe according to the probing algorithm indicated in Chapter 3. And there are 5 different probing functions, which are as following:

• **RequestOnDemand**. When the spot prices are exceeding the predefined threshold (1X on-demand price) that can be defined by SpotLight users, the spot market probe manager will make an on-demand instance request on its own on-demand market.

• **RequestInsufficiency**. When on-demand request denied with error code *InsufficientInstanceCapacity*, its market probe manager will i) make on-demand instance requests on that market periodically until the market is available again; ii) request its

related on-demand markets periodically within some time windows; iii) issue a spot instance request on the same market.

• **CheckCapacity**. To check whether any spot market's capacity is available or not, only issue one spot instance request with current spot price as bid price, if the capacity is not available, then the spot request will be held as *capacity-not-available* until it is available again. Also, when spot request held due to market unavailability, issue an on-demand instance request to verify the availability of on-demand market.

• **BidSpread**. Given any markets, the market probe manager will issue a Bid-Spread market process on that market to find actual bid price to get spot instance. Assume that we can find lower and upper bound by exponential, then use binary search to find actual price between spot price and upper bound. Usually, when the spot prices of that market are very stable, the bid prices to get spot instance are just the current spot price. However, when the spot prices with high volatility, the bid price can be higher than current spot price which will be shown in Figure 5.2 in Chapter 5. With Spotlight, with average 2-3 maximum 6 spot bid requests, we can find the intrinsic bid prices.

• **Revocation**. For selected markets by users with high volatility, market probe manager will issue spot instance requests on those markets when there were spot price spikes, to find whether the selected markets are easily to be revoked with spot price spikes.

The first two experiments are related to on-demand market probing and the later three are related to spot market probing. Also, when issuing requests, to maximize the probability to get instances from EC2 but only look at the obtainability of on-demand and spot markets, we removed all constraints including placement group, launch group, availability zone group constraints, schedule time for spot instances and VPC for both spot and on-demand instance requests. And all on-demand request will go through states listed in on-demand state machine as Figure 3.1 and all spot

instance requests will go through states listed in spot instance request state machine as Figure 3.2 with all states and status changes timestamps logged into database.

All requests are independent and concurrent at the same time and during the process of on-demand and spot instances requests, it would be needed to check the requests states changes continuously until new states updated. However, there are a lot of shared limited resources between those markets and each probe, for example, the number of running on-demand instance, the number of spot instance request per region, the number of API calls per minute and accessing database to record data for each market and probe and etc. Considering both the normal maximal number of on-demand instance for particular instance type and the maximum spot requests are 20, SpotLight uses multi-threading and it has the same limits for each experiment each region as AWS service limits. Hence, to manage limits and get requests states within one API call for each region, there is a manager and then market and request manager will get updated states from region manager, and also, to avoid conflict with accessing database, there will also be a database manager collecting data from all requests, all markets and all regions to insert into database. These hierarchical managers help SpotLight to maximize the utility of each API request to AWS server and minimize the conflicts on shared limited resource, hence requires high concurrency and scalability for the whole system.

To compute market obtainability and availability, SpotLight will first require some data from above experiments, and these data will help it to decide when and where to probe to learn markets obtainability. After gathering data and actively analysis, SpotLight can help users to learn markets by actively efficiently probing or help systems like SpotCheck or SpotOn to improve their performance by choosing market with high obtainability and low volatility.

CHAPTER 5

ANALYSIS AND OBSERVATIONS

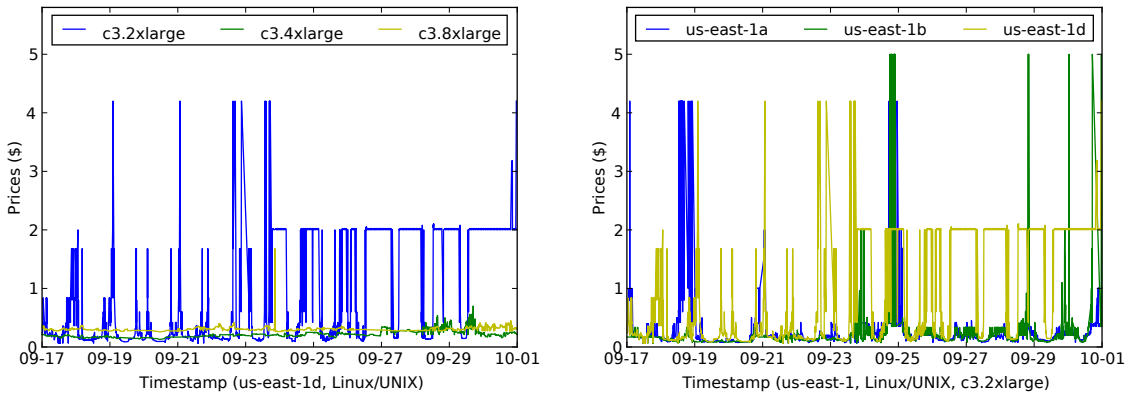
We deployed SpotLight on EC2 and used it to conduct the first large-scale availability study of EC2 over a three month period. Below, we make a series of observations about the underlying operation of EC2 from our data that are either not well-known or have not been quantified in the past. However, note that EC2 is a dynamic, constantly changing environment, so the absolute numbers below can and will change based on changes in EC2's supply and demand over time. Our observations are independent of the absolute numbers and intended to provide new insight into EC2's dynamics that may affect application performance. In practice, SpotLight would run continuously, enabling it to track these changes in EC2's operation to inform application resource allocation decisions.

5.1 Spot Markets

What we did first is to analyze spot prices and we get the following observations. The current spot market is inefficient, and there is ample opportunity for arbitrage and the actual intrinsic bid price could be much higher than the published spot prices.

5.1.1 Inefficient Spot Markets

We all know that there could be different spot price of the same instance type for different availability zones, in an efficient market,the price for each instance in each availability zone would be close to the other. Also, in an efficient market, the price of spot instances would be much closer to the price of on-demand instances (much



(a) Instance Types (b) Availability Zones

Figure 5.1. The difference in spot price between different servers in the same c3.* family and availability zone (us-east-1d) (a) and the spot price for the c3.2xlarge server type across multiple availability zones (b).

closer than the current 10X decrease). But the reality is that the spot price would be periodically exceed and be 5 times or more than the on-demand price, as depicted in Figure 2.1, also, spot prices for same instance type in different availability zones vary. For example, as Figure 5.1(b) depicted, even though these three markets are in the same region, the spot price for us-east-1d sometimes could be 5-6 times of spot price in the other two zones. Of course, there exist periods where spot prices are very close to each other between us-east-1a and us-east-1b. The higher spot prices, on the other hand, indicate some correlation between supply and demand. Different availability zones are different resource pools for the same instance types in the same region, hence, the difference of spot prices also shows that each resource pool has its own supply and demand, resulting in different spot prices changes. The higher spot prices than others might indicate higher demand in that particular availability zones.

In an efficient market, there is no potential for "arbitrage" such that you can buy larger servers for lower prices than smaller servers.Also, in the specific availability zone, the spot prices for the same family type and even the same prefix instance types, the larger instance type like *.8xlarge shall be more expensive than *.4xlarge instances

that would be more expensive than *.2xlarge instances, but from Figure 5.1(a), it is shown that the spot prices for c3.2xlarge sometimes could even be much more higher than c3.4xlarge and c3.8xlarge. Hence, there is a potential for "arbitrage" where some users can buy c3.8xlarge instances with very low price and split it into 4 instances of c3.2xlarge or 2 instances of c3.4xlarge and resell them.

5.1.2 Intrinsic Price for Spot Markets

Even though there are published spot price for all markets in each account, the actual bid prices to get the spot instance sometimes might not be just the same as "published" spot prices because the EC2 servers need some system time to propagate the actual spot price; also the time delay from a new spot price to be shown in the history is about 20 to 40 seconds in average. On one hand, because of the demand and supply might not be met all the time and the always changing demands from users increase the probability of requiring higher bid price to actually get the spot instance. As shown in Figure 5.2, the actual bid price to get spot instance is not always equal to the spot price but higher than the spot price because of the urgent demand at that timestamp.

On the other hand, for most EC2 user, when they are using any instances, what they want is to getting and holding their instances until their job finished and most of these jobs might need several hour, but in order to hold the spot instances for that long and not interrupted by the revocation of spot prices, the bid price should be at least higher than maximal spot prices during the next several hours, which will presumably be much higher than the current spot prices and as shown in Figure 5.3, the least bid price for different holding hours will depend on the maximum during next several hours. Therefore, when spot instances are needed, to finish the jobs before revocation, the least bid prices are much higher than the current spot prices.

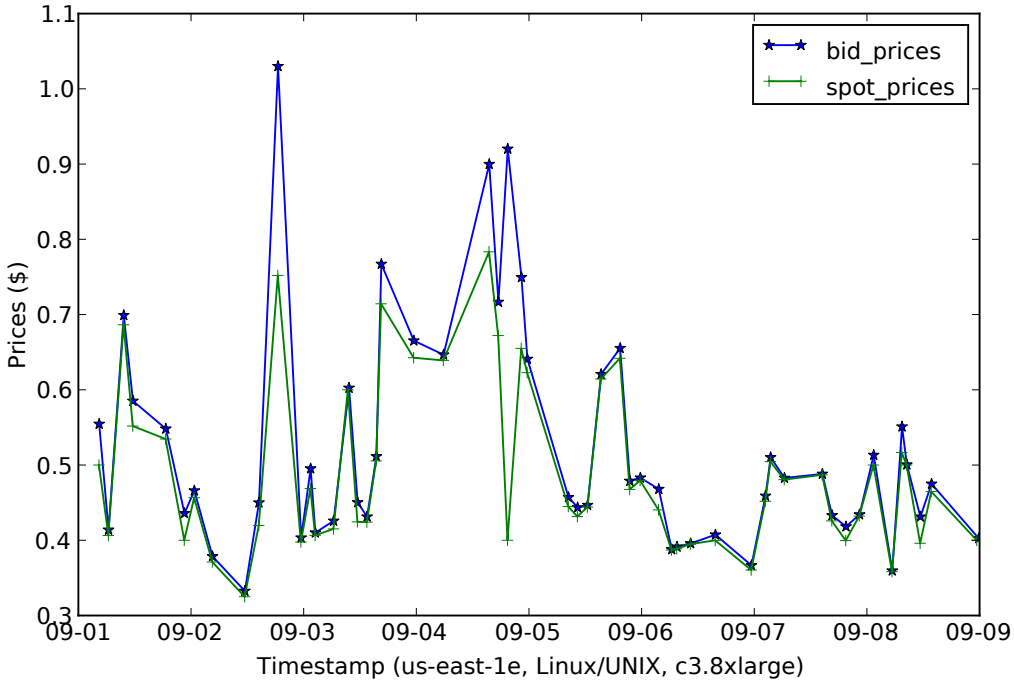


Figure 5.2. Intrinsic price to get spot instances

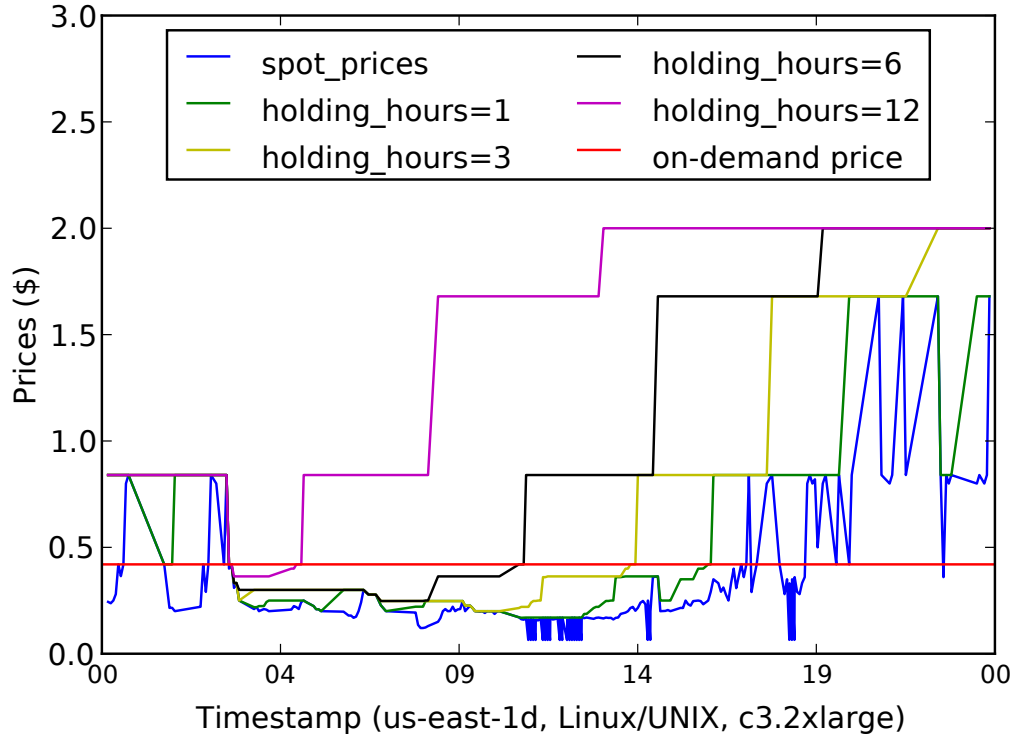


Figure 5.3. Least price to hold spot instances for several hours

5.2 On-demand Unavailability

Cloud platforms offer their customer's the illusion of infinite scalability on-demand. However, in reality, cloud resources are not infinite: if the demand for servers ever exceeds the available supply, then the platform cannot satisfy all requests. SpotLight's probing algorithm collects data that quantifies this unavailability for on-demand servers in EC2. Here, we set the probing threshold to be equal to the corresponding on-demand price for each market and set the sampling probability to 100% to capture all samples. We are particularly interested in these time periods where the spot price exceeds the on-demand price, since the use of spot servers during these periods seems particularly counter-intuitive, as users could presumably use on-demand servers with stronger availability characteristics for a lower price. That is, unless EC2 is rejecting new requests for on-demand instances due to an insufficient supply.

For on-demand servers, we have the following observations. First, on-demand servers are not always available, and these periods of unavailability are often correlated with either spikes in the spot price or the unavailability of a server in the same family (within and across availability zones). Then, while cloud platform's are global, the availability characteristics of different servers are local and highly dynamic. That is, the characteristics of one server type in one region and availability zone may be drastically different than another server type in another region and availability zone.

5.2.1 Global On-demand Unavailability

Figure 5.4 plots the probability over our monitoring period that EC2 rejects a probe request for an on-demand server as a function of the size of the spike in the corresponding spot price, for example, by returning an `InsufficientInstanceCapacity` error code. Note that the x-axis is in multiples of the on-demand price, where $k\times$ represents k times the on-demand price. Each line represents a time window over which we cluster together short periods of unavailability. That is, if the window is

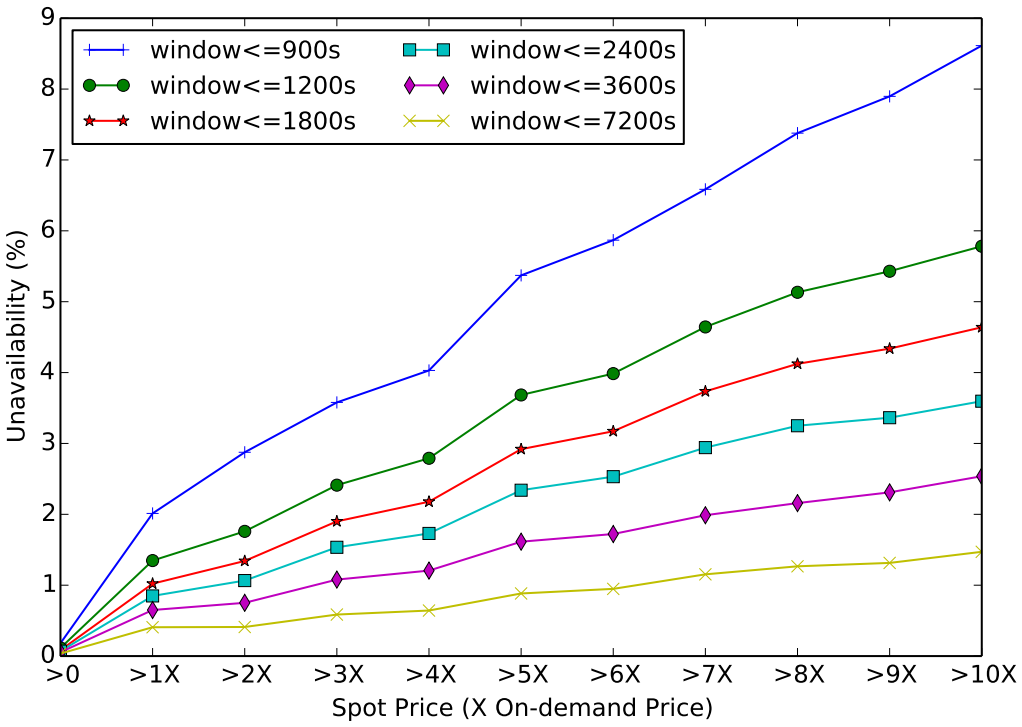


Figure 5.4. Probability that an on-demand server is unavailable as a function of the size of a spot price spike in the corresponding spot market.

one hour, and there are multiple spikes within the hour correlated with unavailability, we only count the first spike within the hour that correlates with a rejected probe request. This graph aggregates data from EC2's global market including all ∼4500 availability zones and instance types from all regions. The graph shows a clear trend: as the size of spot price spikes increases, the probability of on-demand instances (of the same type in the same availability zone of the same region) being unavailable increases from near 0% (for spikes less than 1× the on-demand price) to near 10% (for spikes greater than 10×).

This graph confirms the relationship between the spot price and the unavailability of on-demand servers. The rise in prices may be due to multiple reasons. For example, users may realize on-demand servers are not available and switch to requesting spot servers, thereby increasing their demand and the spot price. Alternatively, the price

increase may also result from a spot server shortage, since unavailable on-demand servers may indicate there are no idle resources left for the spot pool. The graph above demonstrates that there is some correlation between spot price spikes (of varying sizes) and the probability that on-demand servers are not available. However, the correlation is only probabilistic—in many cases, spot price spikes are not correlated with rejected probes. In many cases, a spike in the spot price may simply be due to an increase in demand for spot servers that is independent of the supply and demand of on-demand servers.

We note that, while correlation does not necessarily imply causation, the probability of a rejected on-demand request rises as we extend our time window after the spike (and not vice versa). This behavior may stem from EC2 satisfying a sudden increase in demand for on-demand instances (or a decrease in their supply due to failure) by revoking spot instances. Since spot instances take up to 2 minutes to shutdown, there is some delay in shifting resources from the spot pool to the on-demand pool. However, if the idle resources combined with the spot instances are not enough to satisfy an increase in on-demand requests, then EC2 must start rejecting requests. Of course, in many cases, EC2 will be able to satisfy on-demand requests using a combination of idle resources and by revoking spot instances. In addition, a spike in the spot price may also result from an increase in the demand for spot instances. However, our data indicates that, the higher the spot price spike, the more likely EC2 is experiencing these extreme supply shortages.

5.2.2 Local On-demand Unavailability

Basically, there are the same 9 regions for all AWS accounts but the number of availability zones in regions and the actual mapping to physical availability zones could be different for different accounts. This region is also a main region, where

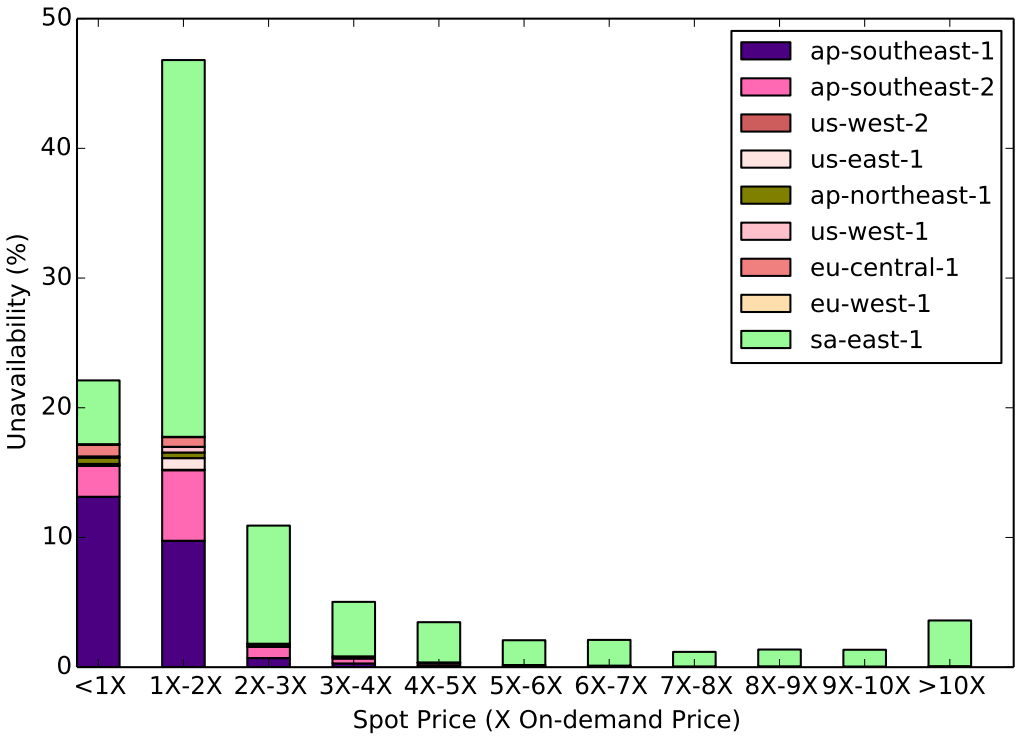


Figure 5.5. The percentage of rejected probes in each region as a function of the size of the spot price spike.

there are about 1000 markets in this region out of 4500 markets in 9 regions, hence the obtainability for us-east is very important.

While the discussion above relates to the global market, EC2 and other cloud platforms are actually an aggregation of many smaller independent clusters that have their own supply and experience their own local demand. Some pools are well-provisioned, while others may be under-provisioned. Figure 5.5 plots the fraction of rejected on-demand probe requests that occurred in each region using the same data from Figure 5.4 as a function of the size of the spot price spike that triggered the rejection. The figure shows that a few regions dominate the number of rejected on-demand probes, and thus appear to be under-provisioned, particularly in the sa-east-1 (South America), ap-southeast-1 (Singapore), and ap-southeast-2 (Australia) regions. By contrast, EC2's largest region (by a wide margin)—U.S. East—-experiences many

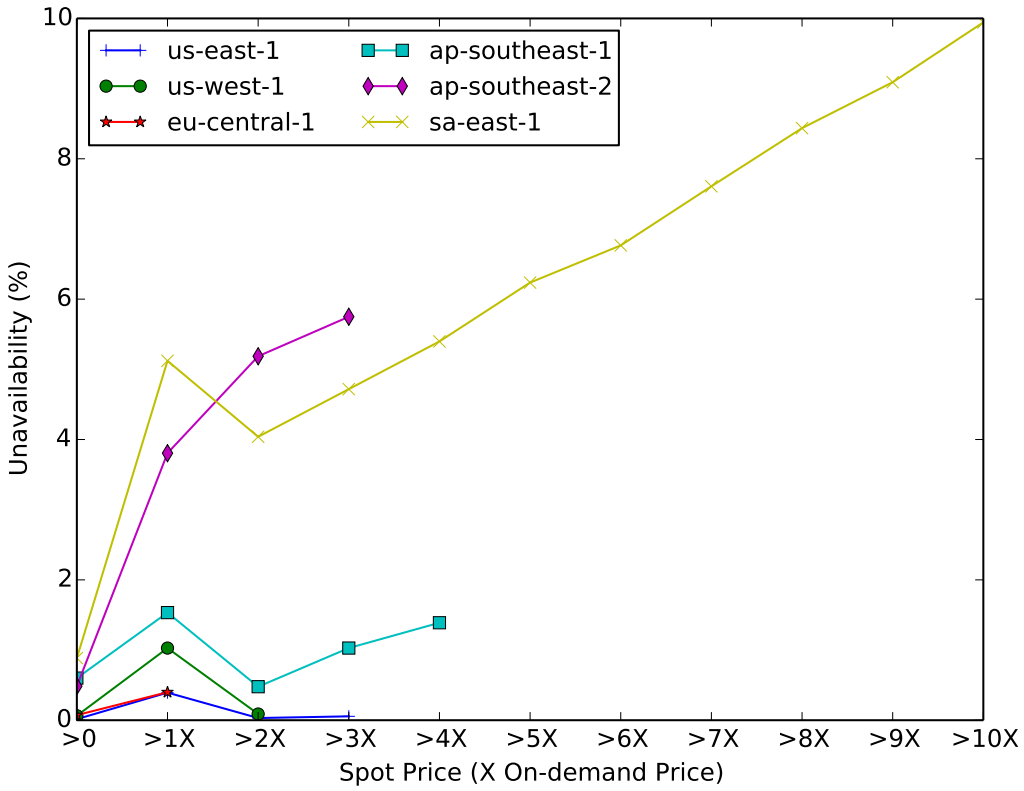


Figure 5.6. The probability of detecting an unavailable on-demand server in different regions as a function of the spot price spike.

fewer rejected probes. The graph also shows the total number of rejected probes decreases substantially as the size of the spot price spikes increases. Thus, there are nearly zero events where the spot price spikes beyond 4× the on-demand price outside of sa-east-1, which is located in Brazil.

Figure 5.6 shows the probability of a rejected probe request as a function of the size of the spot price spike for multiple different regions. In this case, the window size, as defined for Figure 5.4, is 900 seconds (or 15 minutes). The figure shows that some regions have much higher likelihood of experiencing periods of unavailability than others. For example, the us-east-1 region, which is EC2's largest, appears well-provisioned with a probability of unavailability less than 1%. Note that the drop off in lines for some regions are due to a lack of spot price spikes at those levels. The us-

east-1 region experiences very few spot price spikes greater than 2× the on-demand price, and the spot price spikes it does experience are likely the result of an increase in demand, rather a supply shortage. In contrast, sa-east-1 (Brazil), ap-southeast-1 (Singapore), and ap-southeast-2 (Australia) appear under-provisioned and have much higher probability of having unavailable on-demand servers.

The results above demonstrate that users should make decisions on a per-region basis. For example, a reserved server in Brazil is worth more than in the U.S. east, since on-demand servers in the U.S. East are rarely unavailable, while on-demand servers in Brazil are often unavailable. The data also shows the challenge in modeling EC2 spot markets, where each individual pool of servers in each region may different supplies and experience very different demand signals.

5.2.3 On-demand Unavailability for Related Markets

In addition to detecting a rejected probe due to a spot price spike, SpotLight also issues probe requests to all server's within the same family across each availability zone.

Figure 5.7 plots the percentage of rejected probes based on spot price spikes versus those from probing the related markets in the same family across all availability zones. The figure shows that the percentage of rejected probe requests due to probing server types within the same family (after detecting a rejected request due to a price spike) at ∼70% is greater than the percentage of rejected requests SpotLight receives due to price spikes at ∼30%. That is, for each rejected probe triggered by a rise in the spot price, SpotLight on average detects two servers within the same family (in some availability zone) that are also unavailable. As the graph shows, this relationship is generally constant regardless of the spot price level. Thus, the unavailability of an on-demand server of one type indicates a higher probability of unavailability for on-demand servers of other types within the same family.

Figure 5.7. The percentage of rejected probes triggered by spot price spikes versus those triggered by probes of related markets in the same family.

When there is no availability zone specified in user's requests, AWS will automatically allocate an availability zone for the users according to its supply and demand in different availability zones. So when the demand exceed the supply in one of the availability zones, AWS probably allocate new requests that did not specify availability zones to some other availability zones where there is sufficient capacity, which will increase demands on those availability zones and increasing the probability of being insufficient instance capacity in those availability zones.

Figure 5.8 shows that, after detecting an unavailable on-demand server, the probability of a related on-demand server in another availability zone being unavailable decreases as the spot price increases. This trend may result from imbalances in supply and demand across availability zones. When spot prices spike in one market, related markets in other availability zones not experiencing a spike likely have ample resources. In contrast, when spot prices are low, demand is likely more balanced across

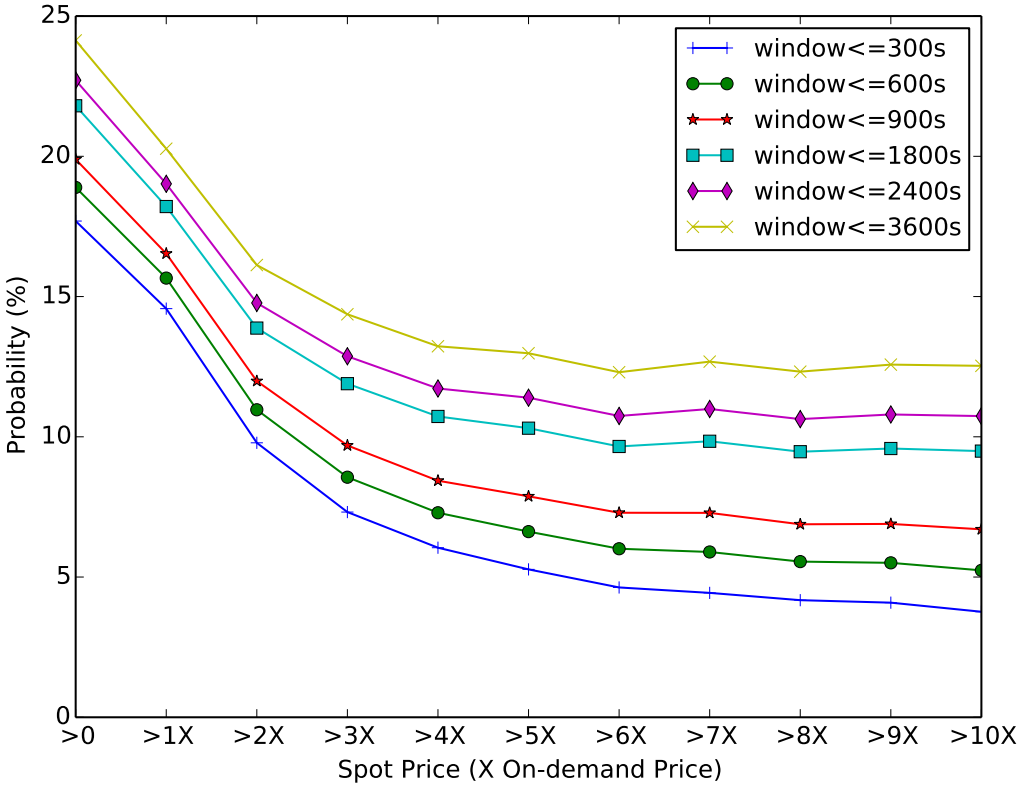


Figure 5.8. After detecting an unavailable server, probability at least one related on-demand server in another availability zone is unavailable.

availability zones, resulting in a higher probability of them concurrently running out of resources.

As depicted in Figure 5.8, when the spot price be higher, the correlation between availability zones became smaller. As the size of spot price spikes increase, the probability of having on-demand insufficiency within 1 hour decrease from near 24% to 12.5%. And as the time duration decrease, no doubt that the probability of having on-demand insufficiency in other availability zones decrease because the number of reachable on-demand insufficiency within the same time range will decrease. This might indicate the smaller spot prices, the on-demand insufficiency could be more related to the actual demand for on-demand instances across that region for some availability zones; while the higher spot prices, the on-demand insufficiency could be more related to the actual demand both for on-demand and on spot instances in that particular local availability zone.

5.2.4 On-demand Unavailability Duration

Finally, Figure 5.9 shows the cumulative distribution function (on a log scale) for the duration of each period of on-demand unavailability across the global market. The graphs show that more than 83% of the unavailability periods last <1 hour, but there is a non-trivial fraction ($\sim$ 17%) that last multiple hours with 5% lasting >10 hours. Such long periods of unavailability can have significant impacts on application performance.

5.3 Spot Unavailability

SpotLight also probes for the availability of spot servers. We have the following observation for spot markets. The availability of spot servers follows the opposite trend as on-demand servers: their availability (for a bid price much greater than the spot price) decreases as the spot price decreases. Overall, the availability of spot

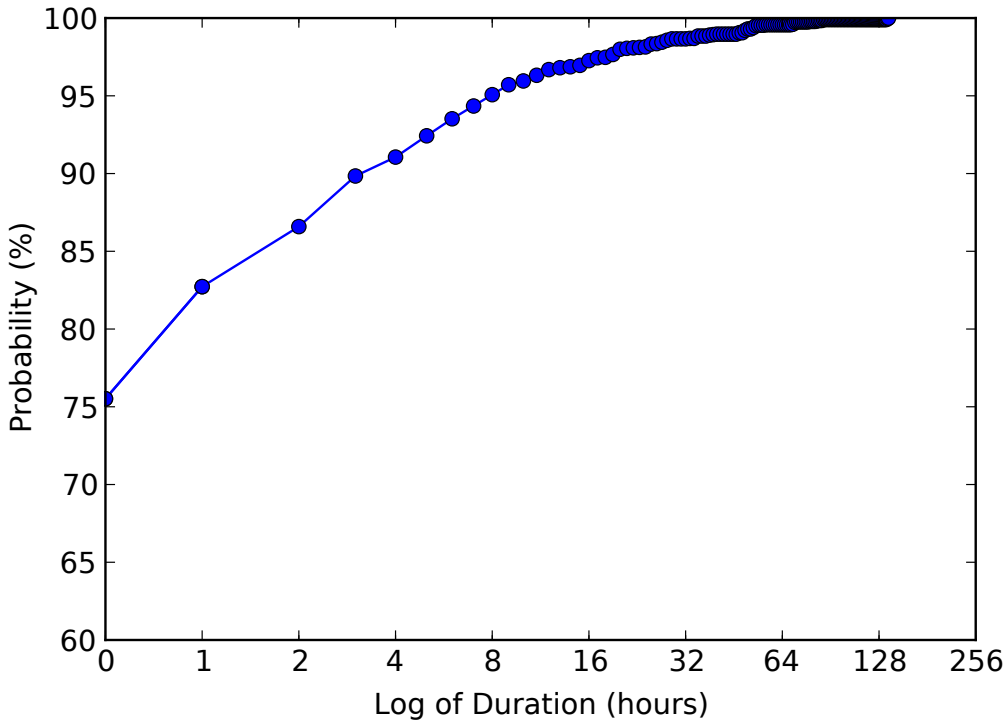


Figure 5.9. CDF of the duration of unavailability periods for on-demand servers.

servers is much higher than on-demand servers, likely because their unavailability is reflected as a rise in the spot price.

While the availability of on-demand servers decreases as the spot price increases, the availability of spot servers moves in the opposite direction: it decreases as the spot price decreases. Figure 5.10 shows this trend for different regions. The graph shows that as the spot price increases along the x-axis, the probability of a spot market probe (with a bid price much higher than the spot price) being rejected, i.e., by returning the `capacity-not-available` error code, decreases. Note that the line for sa-east-1 drops at low price levels because it never experiences such low prices. This trend likely results from the fact that EC2 has no incentive to sell spot servers below the cost of the energy it takes to operate them. Overall, however, over our monitoring period, the spot market rarely rejected probe requests; that is, prices were rarely low enough to trigger the conditions above. We found that, compared to

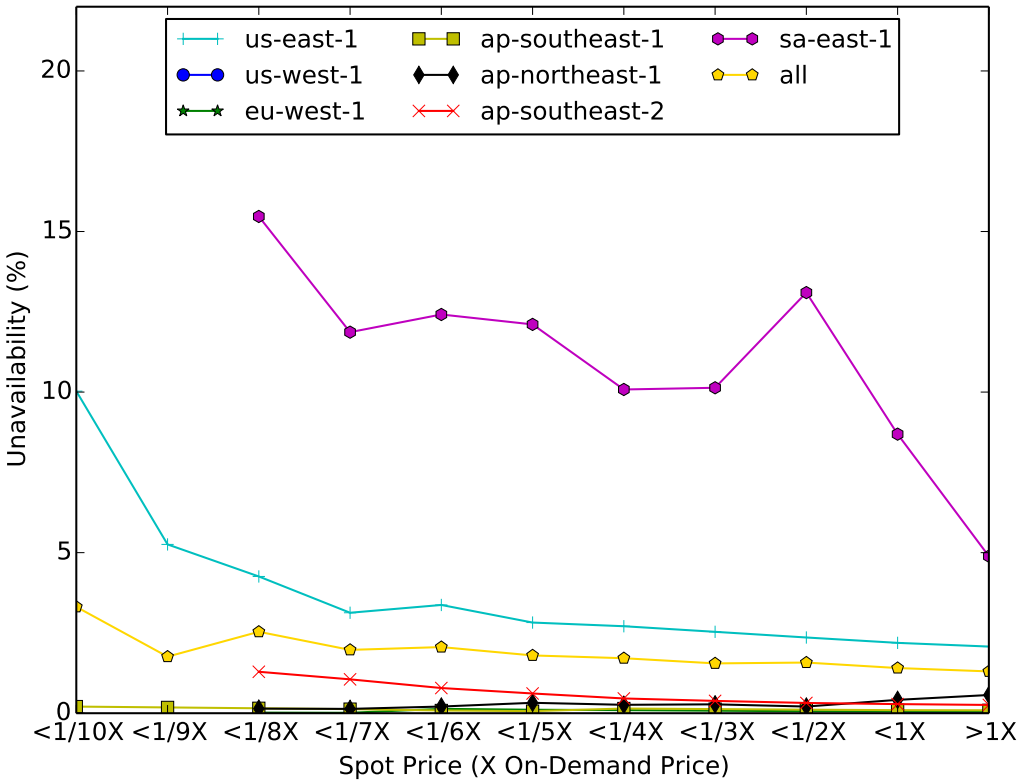


Figure 5.10. Probability of having capacity-not-available for all spot markets and all regions.

on-demand servers, the spot market's availability is visible and its capacity is always nearly always available to users for the right price. One advantage of the spot market is that, while the spot price is unpredictable and fluctuates, users may use the price to estimate availability, which is in contrast to on-demand servers, where users have no visible availability signal.

The x-axis in Figure 5.10 and Figure 5.11 that shows the times of on-demand prices over spot prices is different from previous figures that shows the times of spot prices over on-demand prices. That is to say the higher x-axis label in these two graph indicates the smaller size of spot prices.

Figure 5.10 shows that as the size of spot price spikes increases, the probability of having spot insufficiency decreases from near 10% to near 1% for region us-east-1 and

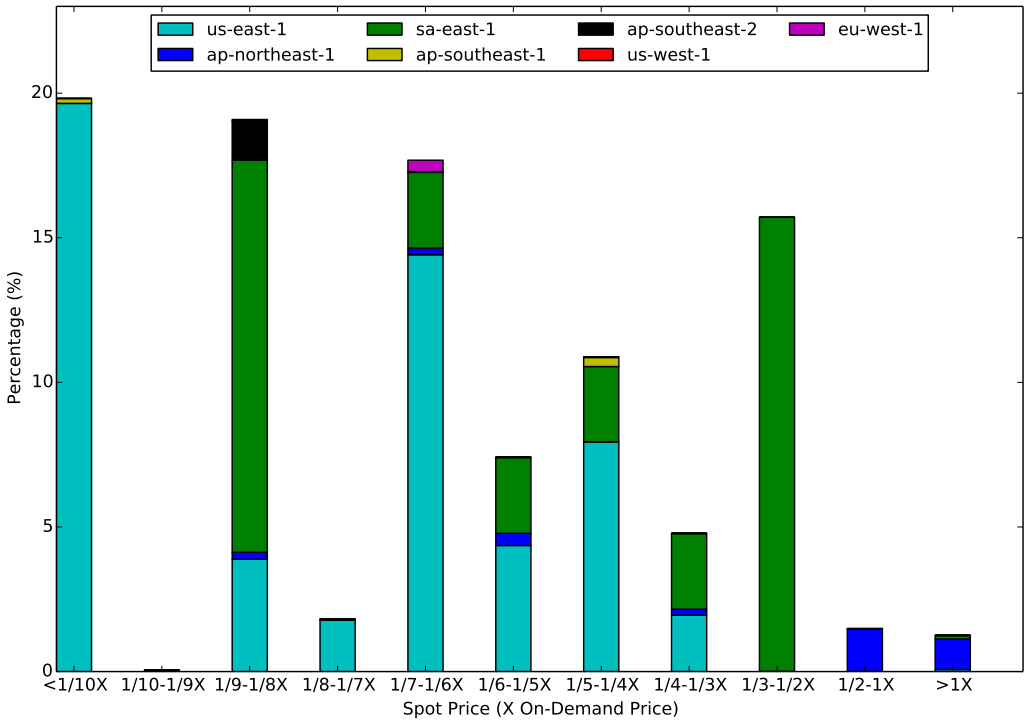


Figure 5.11. Spot insufficiency distribution for all regions.

lower for other regions except that sa-east-1 is not very stable and the probability vary between thresholds. As our data indicated, the lower spot price, the higher probability of having capacity not available for spot instance requests. Spot markets are used to bid on spare instances on EC2 to earn money for those spare VMs. Since running instances has their own minimal cost for the energy and etc, when the spot price is too low, the spot price can not cover the minimal cost, then EC2 might prefer to set spot unavailability rather than selling those instances.

And as depicted in Figure 5.11, most spot unavailability (almost 98%) are happened when spot price is below on-demand price, which indicates that when spot price is lower than on-demand price and on-demand or reserved instances are needed, the spot instance pool might be minimized to provide more on-demand instances. Also, most spot insufficiency are from us-east-1 and sa-east-1, and in us-east-1, this

might be caused by the real supply and demand in this region, as for sa-east-1, this might be caused by its volatility.

As depicted in Figure 5.10, for the global spot insufficiency, the probability is increasing as the size of spot prices decrease except for one data point when on-demand prices are above 9× of spot prices. When look at the distribution of spot insufficiency, there are few data when on-demand prices are between 9× and 10× of spot prices compared to other data points. Also, depicted by these two graphs, the spot market rarely returns capacity-not-available, which is likely because it almost always has some capacity available for the right price. Its availability/obtainability is more visible (via the spot price) than on-demand instances. As a result, it can discourage users from making requests when capacity is scarce (or unavailable) by raising the price.

5.4 Relationship between On-demand and Spot Unavailability

To compare the unavailability between on-demand and spot, we also issue spot request at the same market when on-demand is not available for that market and issue on-demand request when spot is not available for that market.

In Figure 5.12, the labels indicates the probability of having at least one of the related markets that is also unavailable after detected market unavailability in the first market, for example, *od-od* shows the probability that at least having one related on-demand server for all availability zones is unavailable within time window after on-demand unavailability detected; *od-spot* shows the probability that at least having one related spot server for all availability zones is unavailable within time window after on-demand unavailability detected.

As depicted in Figure 5.12, the probability (17.6%) of having related markets unavailable during the time window (1 hour) for on-demand is higher than the prob-

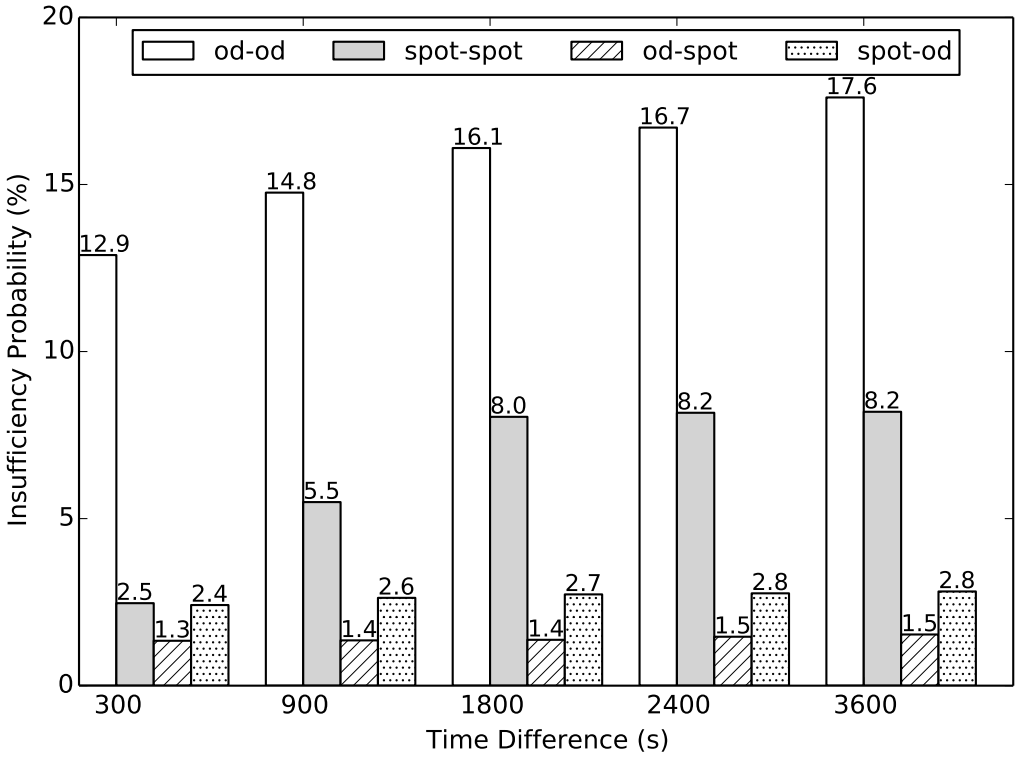


Figure 5.12. On-demand and spot unavailability comparison.

ability (8.2%) of having related markets unavailable during the time window (1 hour) for spot markets, which means that the unavailability relationship between related markets is stronger than the unavailability relationship between spot. Since the unavailability can be caused by a lot of reasons, the increasing demand on specific market or specific availability zone or specific instance type could be one of them especially for the spot server. This would result in the difference in the related markets availability for on-demand and spot servers.

Also, the probability of both having on-demand and spot unavailability during the times window is much lower, which all of them are below 3%. This means that it is rare to have both market to be not available. Even though according to the figure 2.2, there are two main overlapped servers,which are *idle* and *spot* servers in on-demand and spot server pools, there are still specific servers for each pools, for

example, *reserved-not-running* servers for spot pool and *on-demand servers* for on-demand pool. When the demand for on-demand pool increases and the on-demand pool is not available, the spot pool could also be available if existed the not-running reserved servers. Also, on the other hand, when the spot pool is not available, the on-demand pool could also be available if existed not-running on-demand servers.

CHAPTER 6

CASE STUDIES

We conduct two case studies to demonstrate how to leverage the information gathered by SpotLight in the previous section to improve application performance. We examine two applications from prior work—SpotCheck [12] and SpotOn [15]—that minimize the cost of running interactive and batch workloads by opportunistically running on spot servers.

6.1 SpotCheck

SpotCheck is a derivative cloud platform that rents spot servers from EC2, partitions them up using nested virtualization [18], and re-sells the nested VMs to users under a new contract that guarantees high availability, e.g., 99.99989% available. SpotCheck's goal is to support interactive applications that require high availability using spot servers.

SpotCheck achieves this high availability by leveraging a live bounded-time VM migration mechanism that enables it to simply migrate nested VMs to on-demand VMs if spot servers are ever revoked, e.g., the spot price ever rises above the on-demand price. The mechanism asynchronously copies VM memory state to a backup server such that at all times the VM has a bounded amount of outstanding state that must be copied to complete a migration in the event of a revocation. SpotCheck then leverages the two-minute revocation warning provided by EC2 by ensuring it can always live copy a VM's outstanding memory state before a spot VM is revoked. Essentially, SpotCheck runs on spot servers when the spot price is below the on-

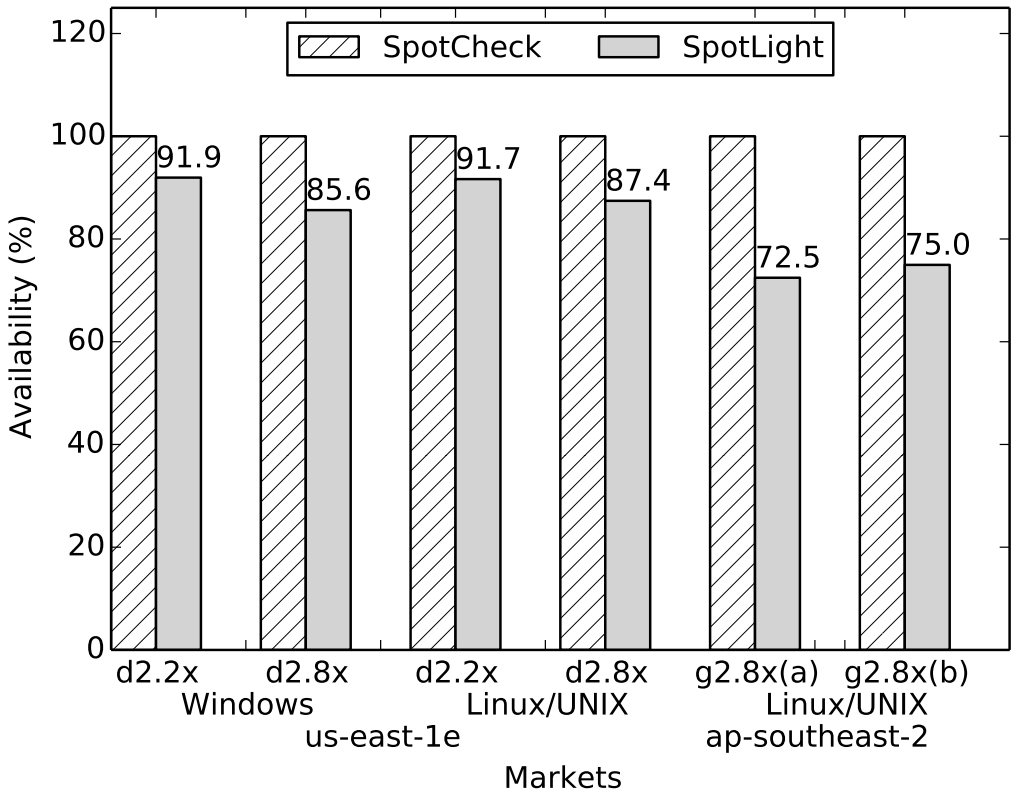


Figure 6.1. The availability of SpotCheck in practice based on the availability data for on-demand servers gathered by SpotLight

demand price, and then migrates to on-demand servers whenever the spot price rises above the on-demand price. The availability is not 100% only due to the small time nested VM's must be paused to transfer the last bits of memory state. Our prior work shows that by highly multiplexing the backup server and amortizing its cost among many VMs, SpotCheck is able to provide the availability of on-demand servers for a cost near that of spot servers.

However, SpotCheck makes a key assumption that on-demand servers are always available as a fallback whenever a spot server is revoked. Critically, in SpotCheck, spot servers are revoked as the result of spikes in the spot price above the on-demand price. Unfortunately, SpotLight shows that these are exactly the times when on-demand servers are least likely to be available. Figure 6.1 shows the actual availability

for different server types in EC2's U.S. east region, which is the most well-provisioned region in EC2. The graph shows that, rather than four 9's of availability, in practice, SpotCheck would only provide between 85.6% and 93.1% availability for these server types due to on-demand servers not being available when spot servers are revoked. The results for other regions are much worse. For example, in the Asia/Pacific Southeast region, SpotCheck's availability is as low as 73%.

However, SpotCheck can use SpotLight's data to improve its selection of on-demand servers and increase its availability back to near 100%. Namely, rather than falling back to on-demand servers of the same type, SpotCheck can select on-demand servers from a different uncorrelated server family that is not experiencing spot price spikes or unavailability. As the graph indicates, by selecting markets that are independent, i.e., hosted on different physical servers, SpotCheck's preserves its assumption that the on-demand servers it falls back on are available when its spot servers are revoked, and its availability increases back to its original value near 100%.

6.2 SpotOn

SpotOn is similar to SpotCheck except that it focuses on optimizing the performance of batch applications running on spot servers. SpotOn's goal is to leverage spot servers to minimize the cost of running batch applications at near the performance level of on-demand servers. Since SpotOn focuses narrowly on batch applications, it is capable of leveraging a much wider range of fault-tolerance mechanisms than SpotCheck. In particular, to ensure progress despite revocations, SpotOn either replicates a batch job across multiple spot servers or periodically checkpoints a batch job running on a spot server. With replication, if all spot servers hosting replicas of a job are revoked, SpotOn restarts the job on an on-demand server to ensure its completion. Likewise, with checkpointing, if the spot server is revoked, SpotOn restarts a job from its last checkpoint on the corresponding on-demand server.

Thus, similar to SpotCheck, SpotOn leverages spot servers to minimize the cost of running the job, while maintaining performance near that of on-demand servers. SpotOn selects the server type to run a job by determining the fault-tolerance mechanism—replication or checkpointing—and spot market with the lowest expected cost. To do so, SpotOn simply brute force computes the expected cost per unit-time for each market to run a job until it either completes or is revoked on each market, and then runs the job on the server with the lowest expected cost. For example, the expression below shows the expected cost when checkpointing a job for a spot market k: the numerator is the overall cost while the denominator is the running time of the application modulo overhead.

$$\frac{[(1-P_k)*T + P_k*E(Z_k)]*spot-price}{(1-P_k)*T + P_k*(E(Z_k)-T_L) - (E(Z_k)/\tau)*T_c} \tag{6.1}$$

In this formulation, T_c represents the time to checkpoint a job based on its memory footprint, τ is the checkpointing frequency, P_k is the probability a job is revoked before it completes, $E[Z_k]$ represents the expected time to revocation, and T is the remaining running time of the job. The numerator represents the expected cost of running a job until it is either revoked or finishes, while the denominator represents the expected time to run a job until it is either revoked or finishes, while considering the overhead related to checkpointing and the loss of work from revocation. SpotOn computes such an expected cost for checkpointing and replication for all markets and then chooses the lowest expected cost to run a job. See prior work for more details [15].

Of course, as with SpotCheck, SpotOn implicitly assumes that on-demand servers are always available. Thus, it does not consider the probability that on-demand servers will not be available when considering which servers to run a job on. In this case, the lack of available on-demand servers will decrease performance by increasing job running time. Since SpotOn currently does not consider this option, it may choose a spot market that has a minimum expected cost but has on-demand servers that are

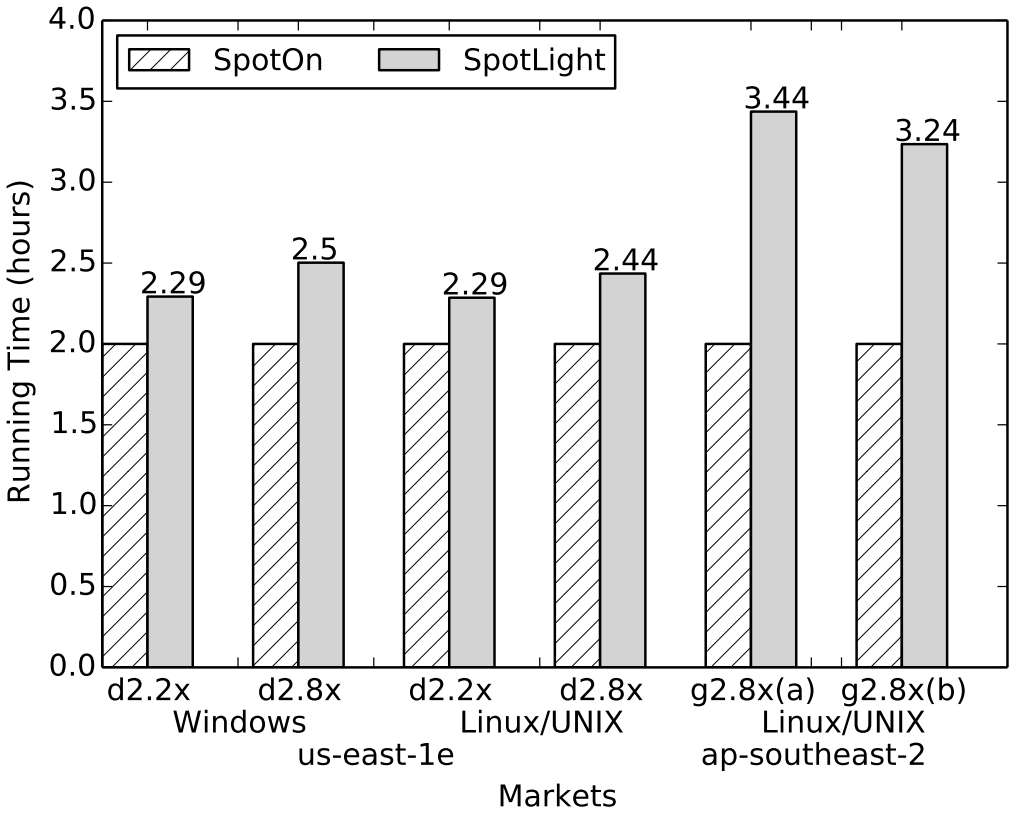


Figure 6.2. Average running time of SpotOn in practice based on the availability data for on-demand servers gathered by SpotLight

much more likely to be unavailable. As a result, the overall performance of the job may be significantly worse than a spot market with only a slightly higher expected cost, but highly available on-demand servers.

Figure 6.2 plots the completion time for a representative job with a running time of one hour and a memory footprint of 8GB, which takes approximately six minutes to checkpoint, for the same spot markets as in Figure 6.1. For this experiment, we plot the expected completion time for 100 trials where the job is started at a random time both assuming on-demand servers are always available after a revocation, and using the on-demand availability data collected by SpotLight. The figure shows that the job's running time increases by 15-72% relative to the system that that assumes on-demand servers are always available. Similar to SpotCheck, the performance in

ap-southeast-2 is significantly worse than the other regions. As before, SpotLight can reduce the running time back to near two hours by enabling SpotOn to select different server types with uncorrelated availability.

CHAPTER 7

RELATED WORK

There is significant prior work on optimizing the use of spot and on-demand servers on EC2 to minimize the cost of executing a workload. Much of this work models spot price dynamics and then examines various bidding policies [13, 20, 21, 4, 3] to determine the optimal bidding strategy to minimize costs, while ensuring an application either finishes within some deadline or maintains a specific availability target (with high probability). In this case, when the bid price is below the spot price, applications simply wait until the spot price falls before resuming execution, which lengthens the running time of batch jobs and decreases the availability of online services.

Another class of work not only optimizes bidding, but also employs fault-tolerance mechanisms, such as checkpointing, to ensure that jobs can efficiently resume after a revocation [17, 7, 19, 8]. Checkpointing introduces a tradeoff: checkpointing too frequently incurs an overhead that reduces performance but decreases the work lost on each revocation, while checkpointing too rarely causes applications to lose more work on each revocation. Thus, determining the optimal checkpointing frequency reduces costs by maximizing the useful computation spot servers are able to perform. Other work looks at employing a mix of spot and on-demand servers to either hedge against the chance of revocation [9], use on-demand servers as backups for spot servers [12], or as a fallback if spot servers are revoked (or are too volatile to satisfy performance requirements) [15].

Our work differs from this prior research in that we focus, not on optimizing a particular application for spot or on-demand servers, but instead on revealing the correlation between spot prices and the internal availability dynamics of the cloud. We model the relationship between reserved, spot, and on-demand servers and show how SpotLight partially exposes these relationships to users by actively probing the cloud based on the spot price. As we demonstrate in our case studies, much of the work above that mixes spot and on-demand servers assumes that on-demand servers are always available, which we show is not true in practice. In fact, we show that on-demand servers are least available when users of spot servers need the most: when spot prices spike. By quantifying the availability of spot and on-demand servers, SpotLight also enables users to better assess their value when selecting whether to use reserved, spot, or on-demand servers.

Finally, much of the work above is either based on or evaluated on data from just a few spot markets, while our work is based on three months of data from nearly all of the $\sim$4500 spot markets and server types in EC2. We know of no prior system that has either operated at or collected data at the scale of SpotLight. The prior systems above that were actually deployed were done so only to conduct a small set of experiments on EC2. In contrast, SpotLight continuously ran for three months at such a massive scale that any bug in its probing algorithm could incur significant usage costs on EC2.

CHAPTER 8

CONCLUSION

We present SpotLight, an information service for the cloud. SpotLight's key contribution is an active market-based probing policy that reveals the internal cloud dynamics related to the availability of servers. Understanding server availability under different contract types is critical in selecting the optimal server and contract type for applications that minimizes cost subject to a user's risk tolerance. We deployed SpotLight on EC2 and collected availability data from all ∼4500 server types across all regions and availability zones over a three month period. We then demonstrate how SpotLight's information can improve two applications—SpotCheck and SpotOn—that implicitly assume on-demand servers are always available. We show that, in practice, since this assumption is not true, these applications would perform much worse than expected. However, by using SpotLight's data, they can maintain their performance by selecting server types that are likely to be available with high probability.

BIBLIOGRAPHY

[1] Barr, J. New - EC2 Spot Instance Termination Notices. https://aws.amazon.com/blogs/aws/new-ec2-spot-instance-termination-notices/, January 6th 2015.

[2] Engineering, Moz. Amazon EC2 Spot Request Volatility Hits $1000/hour. https://moz.com/devblog/amazon-ec2-spot-request-volatility-hits-1000hour/, September 27th 2011.

[3] Guo, W., Chen, K., Wu, Y., and Zheng, W. Bidding for Highly Available Services with Low Price in Spot Instance Market. In *HPDC* (June 2015).

[4] He, X., Sitaraman, R., Shenoy, P., and Irwin, D. Cutting the Cost of Hosting Online Internet Services using Cloud Spot Markets. In *HPDC* (June 2015).

[5] Hu, N., and Steenkiste, P. Evaluation and Characterization of Available Bandwidth Probing Techniques. *JSAC 21*, 6 (August 2003).

[6] Jain, M., and Dovrolis, C. Pathload: A Measurement Tool for End-to-End Available Bandwidth. In *PAM* (March 2002).

[7] Khatua, S., and Mukherjee, N. Application-centric Resource Provisioning for Amazon EC2 Spot Instances. In *EuroPar* (August 2013).

[8] Marathe, A., Harris, R., Lowenthal, D., de Supinski, B., Rountree, B., and Schulz, M. Exploiting Redundancy for Cost-effective, Time-constrained Execution of HPC Applications. In *HPDC* (June 2014).

[9] Menache, I., Shamir, O., and Jain, N. On-demand, Spot, or Both: Dynamic Resource Allocation for Executing Batch Jobs in the Cloud. In *ICAC* (2014).

[10] Morgan, T. A Rare Peek Into the Massive Scale of AWS. EnerpriseTech, November 14th, 2014 2014.

[11] Prasad, R., Dovrolis, C., Murray, M., and k. claffy. Bandwidth Estimation: Metrics, Measurement Techniques, and Tools. *IEEE Network 17*, 6 (November 2003).

[12] Sharma, Prateek, Lee, Stephen, Guo, Tian, Irwin, David, and Shenoy, Prashant. SpotCheck: Designing a Derivative IaaS Cloud on the Spot Market. In *EuroSys* (April 2015).

[13] Song, Y., Zafer, M., and Lee, K. Optimal Bidding in Spot Instance Market. In *Infocom* (March 2012).

[14] Strauss, J., Katabi, D., and Kaashoek, F. A Measurement Study of Available Bandwidth Estimation Tools. In *IMC* (October 2003).

[15] Subramanya, S., Guo, T., Sharma, P., Irwin, D., and Shenoy, P. SpotOn: A Batch Computing Service for the Spot Market. In *SOCC* (August 2015).

[16] Vickrey, W. Counterspeculation, Auctions, and Competitive Sealed Tenders. *Journal of Finance 16*, 1 (1961).

[17] Voorsluys, W., and Buyya, R. Reliable Provisioning of Spot Instances for Compute-Intensive Applications. In *AINA* (2012).

[18] Williams, D., Jamjoom, H., and Weatherspoon, H. The Xen-Blanket: Virtualize Once, Run Everywhere. In *EuroSys* (2012).

[19] Yi, S., Kondo, D., and Andrzejak, A. Reducing Costs of Spot Instances via Checkpointing in the Amazon Elastic Compute Cloud. In *CLOUD* (July 2010).

[20] Zafer, M., Song, Y., and Lee, K. Optimal Bids for Spot VMs in a Cloud for Deadline Constrained Jobs. In *CLOUD* (2012).

[21] Zheng, L., Joe-Wong, C., Tan, C. Wei, Chiang, M., and Wang, X. How to Bid the Cloud. In *SIGCOMM* (August 2015).