SpotLight: An Information Service for the Cloud

Xue Ouyang, David Irwin, and Prashant Shenoy
University of Massachusetts Amherst

Abstract—**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 along several dimensions. For example, while the risk of servers not being available when requested is critical in optimizing the cloud's 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.**

I. Introduction

With the rise in popularity of cloud computing, businesses are increasingly renting computation and storage from cloud providers in lieu of maintaining their own private infrastructure. Cloud platforms provide a number of benefits, including a pay-as-you-go billing model, the on-demand allocation of resources, and the illusion of near-infinite scalability. Cloud providers, such as Amazon's Elastic Compute Cloud (EC2), Microsoft Azure, and Google Compute Engine (GCE), now provide these services using a distributed infrastructure comprising millions of servers in hundreds of data centers spread across the globe [7]. Cloud platforms have evolved significantly since their early days. Early platforms exposed simple abstractions to their users—a choice of a small number of server types with a fixed per-hour price to lease each type. However, modern cloud platforms have evolved to offer a myriad of options for users to select from, including a plethora of server and storage types from a wide range of geographical locations. For example, EC2 users may choose from up to 53 server types in 26 availability zones[1] across 9 regions—a choice of more than 12,000 distinct options.

Users may also lease each server configuration under many different *contracts*, each of which expose a different set of cost and availability tradeoffs. Table I depicts several basic contract types for EC2 and their relative characteristics. At one end of the spectrum, reserved servers incur a, relatively high, fixed price, are non-revocable, and are guaranteed to be available upon request. At the other end of the spectrum lies spot servers, which are revocable and have a variable, but relatively low, cost and availability. Contracts (and prices) for on-demand servers, which are non-revocable but not guaranteed to be available, lie somewhere in between. Thus, today's platforms require users to make highly complex decisions in choosing from thousands of server configurations and multiple contracts.

Perhaps more importantly, modern cloud platforms are evolving into full-fledged *markets* where resource price and availability changes continuously based on supply and demand. Like any market, cloud markets experience phases of stability and volatility, which further complicates the decision of selecting which servers to run an application. In addition, the cloud market is *opaque* along several dimensions. For example, while server prices are typically published, server availability also varies in real-time based on market conditions, but is not always made visible to users. As a result, during a high demand period, a cloud platform may reject a user's request for an on-demand server due to a lack of available resources. Similarly, the platform may unilaterally revoke spot servers it has already allocated to satisfy requests for on-demand or reserved servers.

The complex decisions required when selecting cloud servers, as well as the dynamic and partially opaque nature of the cloud market, often leads users to make sub-optimal choices that increase their cost and reduce their application's performance. For example, conservative users may decide to lease on-demand, rather than spot, servers to avoid dealing with sudden revocations when running a workload of parallel batch jobs. However, during periods of market stability, these jobs may run on cheap spot servers at a small fraction of the cost of on-demand servers. Alternatively, price-conscious users might choose to run on cheap spot servers during a period of market volatility, causing them to incur higher costs due to the repeated loss of work from

[1]Each availability zone is physically independent of others in the same geographical region, and consists of one or more data centers [7].

Contract	*Cost*	*Revocable*	*Availability*
Reserved	Fixed	No	Guaranteed
On-demand	Fixed	No	High, but not guaranteed
Spot	Variable	Yes	Variable

Table I
BASIC CONTRACT TYPES IN EC2.

frequent server revocations. Making intelligent decisions is challenging, since it requires a careful analysis of historical trends, current conditions, and application characteristics to determine the best server configuration and contract. Given the complexity of such decisions, and the lack of complete information, users often resort to simple rules of thumb.

To address the problem, we design an information service for the cloud, called SpotLight, that uses a mix of passive and active probing based on a model of the cloud's underlying operation to reveal the market's hidden dynamics and enable better decision making. In addition to tracking and modeling historical prices, SpotLight uses active probing to infer and quantify server availability, which is not directly exposed by cloud platforms. While SpotLight may aid users in better understanding cloud dynamics, it is intended for programmatic decision-making by cloud cluster managers that implement policies to automatically optimize the choice of server type and contract based on market conditions and application resource usage characteristics. SpotLight is analogous to a stock information service for the cloud, which analyzes prices, volatility, and other characteristics of each stock (or server type and contract) in the stock (or cloud) market to aid investors (or users) in making intelligent decisions. Stock information services exist to "screen" and rate stocks, alleviating each individual investor from performing the tedious and complex tasks of gathering, tracking, and analyzing each stock's data themselves. SpotLight serves a similar purpose for applications that use cloud resources.

SpotLight is inspired by past work on *network tomography*, which uses active and passive probing from the network's edge to discover the network's internal state [8, 11]. Similarly, as a third-party service, SpotLight probes cloud platforms from the outside to infer their internal state, and its impact on price and availability. SpotLight's probing policy is market-based: it exploits correlations between real-time prices and availability to optimize when and where it probes—by probing more frequently during periods of volatility and less frequently during stable periods. Since some probes require allocating cloud servers that incur a monetary cost, SpotLight's probing policy is also cost-aware and adjusts its probing to satisfy a budget. Our hypothesis is that, by intelligently probing the cloud, SpotLight's information service enables applications to select server types and contracts that improve performance and cost. In evaluating our hypothesis, we make the following contributions.

Market-based Probing Policy. We design SpotLight's market-based probing policy by leveraging the insight that the real-time prices for spot servers partially correlate with the availability of on-demand and spot servers. Thus, SpotLight triggers probes of server availability when spot prices spike, which may indicate a decrease in the supply of available resources. After detecting an unavailable server, the probing policy fans out by probing related servers.

Large-scale Availability Study. We implement a SpotLight prototype in `python`, deploy it on EC2, and use it to monitor the availability of over 4500 server types over a three month period. Using SpotLight, we conduct the first large-scale study of availability on EC2. We then leverage our data to make a series of observations that reveal characteristics of the underlying EC2 cloud market that are either not well-known or have not been quantified in the past.

Implementation and Evaluation. Finally, we conduct case studies to quantify how SpotLight improves application performance. We show that a key assumption in recent work—that an application can always fail-over to on-demand servers if a spot server 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 is much lower than reported—roughly 70-90% instead of >99%. However, SpotLight can aid these applications in maintaining >99% availability by informing these applications of servers with higher availability.

II. 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. EC2 is also the only cloud platform that releases real-time spot price data, which, as we show, makes its internal operational dynamics partially visible. We first provide the details of EC2's contracts—listed in Table I—and then outline our probing approach, which exploits the relationships between servers offered under each contract.

A. Contract Types

On-demand Servers. On-demand servers are the primary contract type offered by cloud platforms, where users may request servers at any time and, once allocated, pay a per unit-time price while using them. Importantly, on-demand servers are not revocable: users, and not the platform, decide when to terminate them. Thus, once allocated, users may run on-demand servers for as long as they wish.

While on-demand servers are simple for users to understand, they have a significant drawback for risk-averse users: their availability is not guaranteed. Put simply, the demand for EC2 servers may periodically exceed their supply, and, as a result, EC2 can and often does reject requests for on-demand servers by returning an `InsufficientInstanceCapacity` error code. This risk of rejection may prevent risk-averse users from relying on on-demand cloud servers. For example, an online service may deem the risk of rejection at a critical business time too costly to offset the savings from using on-demand servers.

Reserved Servers. To cater to risk-averse users, EC2 also offers reserved servers, which are also not revocable, but are guaranteed to be available. That is, 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 always have sufficient capacity to immediately start the server.[2] 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. As a result, they eliminate much of the elasticity benefits of using the cloud for variable workloads. Of course, if reserved servers are not fully utilized then they may cost significantly more than using the equivalent on-demand servers only when necessary.

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 them for higher-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 [13] where the lowest winning bid dictates the spot price. However, if the spot price ever rises about the user's bid price, EC2 immediately revokes the server after a two minute warning.

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 EC2 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 public repositories provide multiple years of price data. Thus, while spot servers come with no availability guarantees, users can analyze the historical spot price 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.

[2]EC2 may reject the initial request for a reservation. The guarantee above only applies to reservations EC2 has already granted to a user.

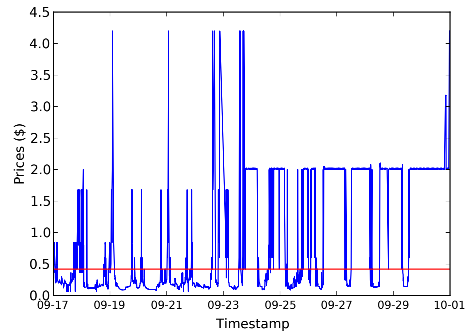


Figure 1. Spot prices are dynamic and may exceed the on-demand price.

Figure 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 much 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 [1]. The rise was the result of a sudden spike in demand coupled with a high "convenience" bid made by users to prevent revocations (under the rational assumption that the spot price should 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.

B. Active Probing Approach

EC2 offers numerous server types under the different contract types above. Optimizing a user's 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 server or an on-demand server. *Unfortunately, the availability of on-demand and spot servers is not made visible to users, which prevents them 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

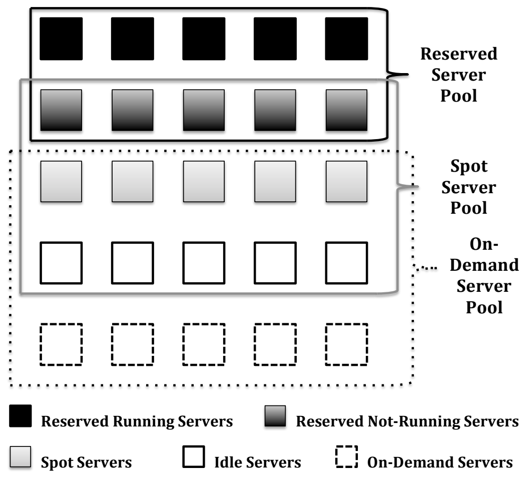


Figure 2. Relationship between reserved, on-demand, and spot servers hosted on the same pool of physical resources.

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 server is available. However, if EC2 does not allocate a server, but instead returns an `InsufficientInstanceCapacity` error code, then we record that the 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 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, or, more precisely, in the same family, as we discuss in §III. For example, a sudden rise in the spot price may be the result of a surge in requests for on-demand (or reserved) servers, which causes EC2 to revoke some spot servers to satisfy the requests and, hence, decreases 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 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 the number of spot servers, and thus increasing the value of the lowest bid receiving resources, and hence the spot price.

III. 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 EC2's ~4500 spot markets. In both cases, SpotLight logs the data in a database and defines an API that applications may use to query it. 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 useful for users, we designed it for programmatic access by applications using automated policies to continuously optimize server and contract selection based on changing cost and availability. In §VI, we provide examples of how SpotLight benefits two such applications.

A. 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 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 an on-demand server is not available, SpotLight continues issuing probes each interval until a request is fulfilled, and the server is available.

Probing within a Server 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 an even factor, 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 servers with variable sizes complicates resource allocation, leading 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 overall 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 on-demand unavailability may also be associated with the 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.

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 servers within the same family in 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, we have found that their demand exhibits strong dependencies. For example, unless a user explicitly specifies a particular availability zone in their request, 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 it not to be fulfilled, SpotLight logs the error code and continues to periodically probe the server every δ time until the request is fulfilled.

B. Controlling Probing Costs

Each probe incurs a cost, since SpotLight must pay for at least one hour of server-time—EC2's minimum granularity for billing—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 server is deactivated within its first 10 minutes, while Microsoft Azure charges on a per-minute basis. SpotLight may adjust its costs by simply changing its probing price threshold T_{price} that triggers a probe when the spot price rises above the threshold. 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.

However, 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., spot price spikes $>7\times$ the on-demand price. Thus, SpotLight also enables users to set a sampling ratio, such that it only probes a price spike greater than a threshold T_{price} with some probability p. By lowering p, we can also lower T_{price} and sample some fraction of less-volatile events. We set T_{price} and p based on historical data to ensure the budget lasts for a specific time window. 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_{price} equal to the on-demand price and sample every event. 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_{price} and p. Of course, we could easily extend the scheme above to account for the expected cost of related server probes based on historical probing data.

C. Probing Spot Servers

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 are 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. 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 increases the bid by a small amount and re-issues the request until it finds 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 `capacity-not-available`, we do not support probing related spot servers within the same family.

IV. Implementation

We implemented a prototype of SpotLight in `python` and deployed it on EC2. Our prototype interacts with EC2 via its Boto3 `python` API, and concurrently monitors all ∼4500 spot markets and probes the corresponding spot and on-demand servers based on the market-based probing policy described in the previous section. When issuing probe requests, our prototype removes all additional EC2 constraints that might cause a request rejection, including placement group, launch group, and availability zone group constraints, as well as network-related VPC constraints for spot and on-demand instance requests. To avoid rejected requests, SpotLight also imposes the same limits on its probing as EC2 imposes on its requests, including a strict limit on the number of running on-demand servers, the number of spot server requests per region, and the maximum number of API calls per minute. SpotLight logs the entire timeline for each request in its database by recording a timestamp for each state transition over a request's lifetime.

V. Data 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. 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 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 allocation decisions.

> **Observation #1:** The current spot market is inefficient, and there is ample opportunity for arbitrage.

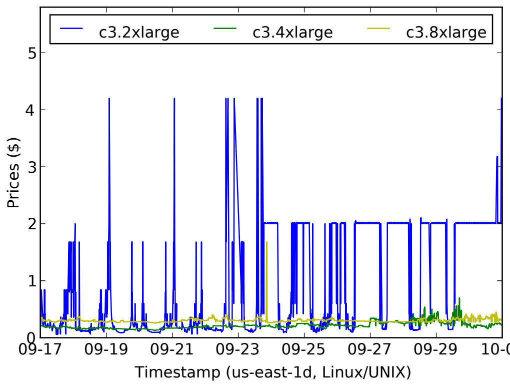
Our first observation is that the spot market is not an "efficient" market in that there is ample opportunity for arbitrage. For example, larger servers within the same family are often available for lower prices than smaller servers. Thus, users must consider a wide range of servers and contracts when deciding the optimal server types to select. Such inefficiency also opens up arbitrage opportunities where users may purchase large servers and use resource isolation mechanisms, such as nested virtualization, to carve them up and re-sell smaller servers to users for a significantly lower price than EC2. Figure 3(a) demonstrates this point by showing how the spot price for the c3.2xlarge, in this case, is often much higher than the c3.4xlarge or c3.8xlarge, which are 2× and 4× as large. These opportunities for arbitrage also exist across availability zones, where the spot price of the same server in different availability zones may vary significantly. Figure 3(b) demonstrates this point by showing how the spot price in the US-east-1d availability zone is often 5-6× more than in the other two availability zones.

An efficient market would not exhibit wide price disparities for similar resources within each family of servers or across availability zones. This observation reinforces the importance of making intelligent decisions about which server types and contracts to use, and implies that users should consider many server types when making allocation decisions. It also motivates automated systems to exploit arbitrage opportunities, as discussed in the next section.

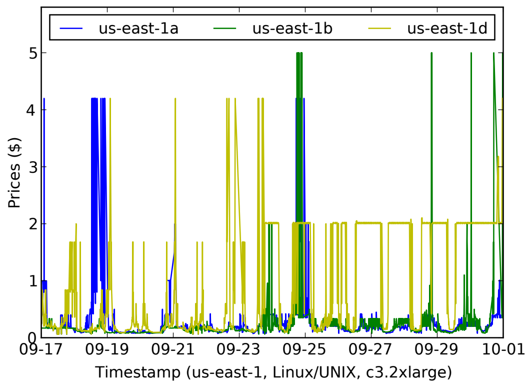
> **Observation #2:** On-demand servers are not always available, and these periods of unavailability are often correlated with either spikes in the spot price or the unavailability of a server in the same family (within and across availability zones).

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 much stronger availability characteristics for a lower price. That is, unless EC2 is rejecting new requests for on-demand instances due to an insufficient supply.

Figure 4 plots the probability over our monitoring period that EC2 rejects a probe request for an on-demand server, e.g., by returning an `InsufficientInstanceCapacity` error code, as a function of the size of the spike in the corresponding



(a) Server Types



(b) Availability Zones

Figure 3. 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 (a).

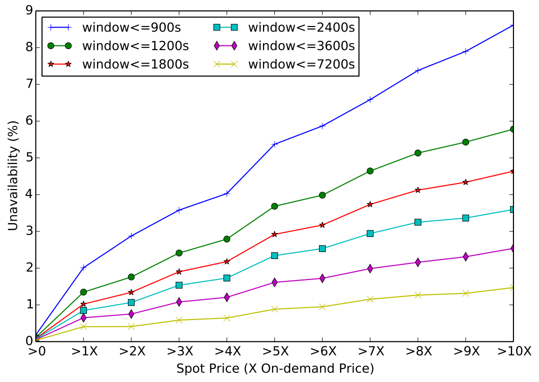


Figure 4. Probability that an on-demand server is unavailable as a function of the size of a spot price spike in the corresponding spot market.

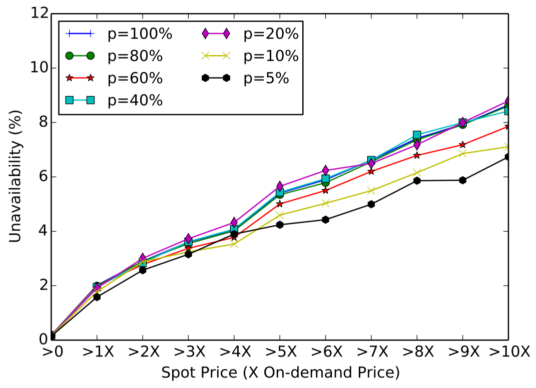


Figure 5. Probability that an on-demand server is unavailable as a function of different probing thresholds T_{price} for different sampling probabilities.

spot price. 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 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 $\sim$4500 availability zones and instance types from all regions. The graph shows a clear trend: as the size of a spot price spike 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 $<1\times$ the on-demand price) to near 10% (for spikes $>10\times$).

The 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. 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.

Figure 5 then shows the results when SpotLight adjusts its sampling probability p from Section III-B as a function of its probing threshold T_{price} on the x-axis. This graph uses the window $\leq$ 900s from Figure 4 for p=100%, such that SpotLight probes every price spike above the threshold on the x-axis. The other lines indicate results when SpotLight only probes a random sample of the price spikes as indicated by p. The graph shows that even low sampling probabilities, where SpotLight only samples 5-20% of the price spikes, yield similar results as probing all price spikes. The cost reduction is equivalent to the sampling probability p, such that a 5% sampling probability yields a cost that is 5% of the maximum costs when probing every sample. As a result, SpotLight can reduce costs significantly by only probing a small fraction of price spikes at any given threshold.

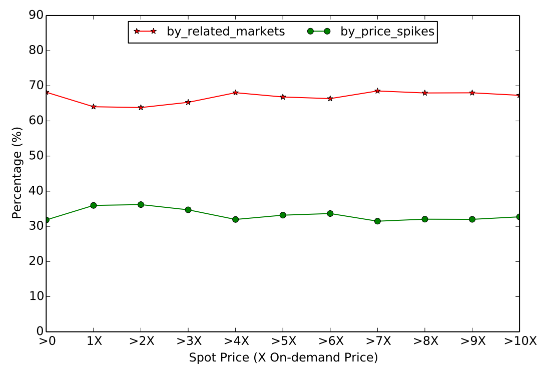


Figure 6. The percentage of rejected probes triggered by spot price spikes versus those triggered by probes of related markets in the same family.

In addition to detecting a rejected probe due to a spot price spike, SpotLight also issues probe requests to all servers within the same family across each availability zone. Figure 6 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. The graph shows that the relationship is constant regardless of the spot price. 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 7 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, 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 availability zones, resulting in a higher probability of concurrently exhausting resources.

Finally, Figure 8 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 >83% of the unavailability periods last <1 hour, but there is a non-trivial fraction (∼17%) that last multiple hours with 5% lasting >10 hours. Such long unavailability periods can significantly impact application performance.

Observation #3: While cloud platforms are global, the availability characteristics of different servers are local and highly dynamic. The characteristics of one server type in one region and availability zone are different than other server types in other regions and availability zones.

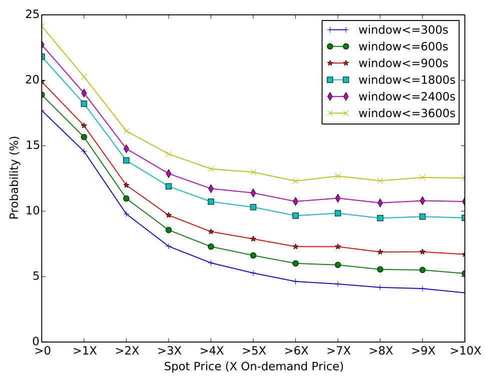


Figure 7. After detecting an unavailable server, the probability at least one related on-demand server in another availability zone is unavailable.

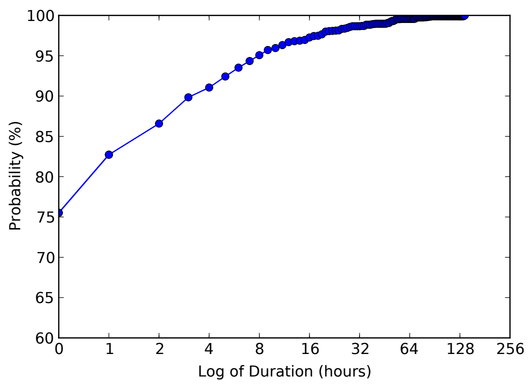


Figure 8. CDF of the duration of unavailability for on-demand servers.

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 9 plots the fraction of rejected on-demand probe requests that occurred in each region using the same data from Figure 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 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 10 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 4, is 900 seconds (or 15 minutes). The figure shows that some regions have a much higher likelihood of expe-

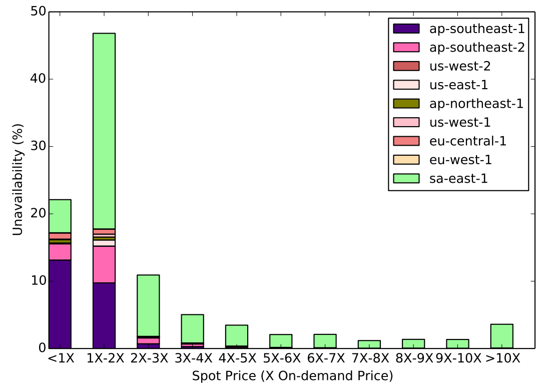


Figure 9. The percentage of rejected probes in each region as a function of the size of the spot price spike.

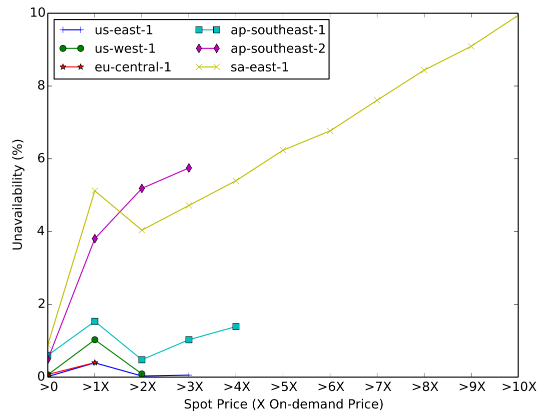


Figure 10. The probability of detecting an unavailable on-demand server in different regions as a function of the spot price spike.

riencing 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 than a supply shortage. In contrast, sa-east-1 (Brazil), ap-southeast-1 (Singapore), and ap-southeast-2 (Australia) appear highly under-provisioned and thus 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 of modeling EC2 spot markets, where each pool of servers in each region may experience different supply and demand signals.

Observation #4: The availability of spot servers follows exactly 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 servers is much higher than on-demand servers, likely because their unavailability is reflected as a rise in the spot price, which dampens demand.

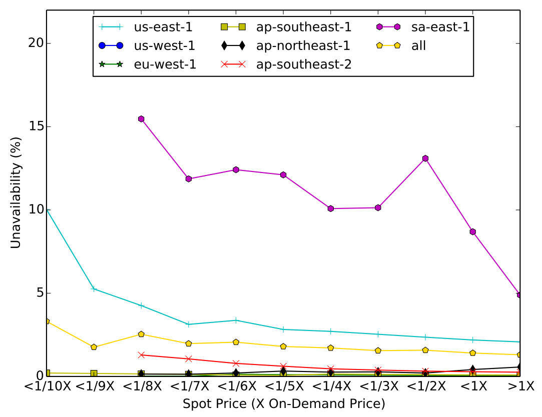


Figure 11. The probability of a spot server being unavailable, regardless of the bid price, as a function of the spot price.

SpotLight also probes for the availability of spot servers. While the availability of on-demand servers decreases as the spot price increases, the availability of spot servers moves in exactly the opposite direction: it decreases as the spot price decreases. Figure 11 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. When the spot price is low, 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. Interestingly, while us-east-1 has the highest availability for on-demand servers, it has one of the lowest availabilities for spot servers (next to sa-east-1 in Brazil). This may result from higher operating costs in us-east-1, e.g., to purchase electricity, than in other regions. In addition, when the spot price is high, EC2 has an incentive to allocate spot servers to users if they are will to pay a much higher price for them than on-demand servers. The data reflects this trend, as the unavailability of spot servers decreases as the spot price rises—-as in any market, the more users are willing to pay, the less likely EC2 is to reject them.

VI. 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 systems from prior work—SpotCheck [9] and SpotOn [12]—that minimize the cost of running interactive and batch workloads by opportunistically using spot servers.

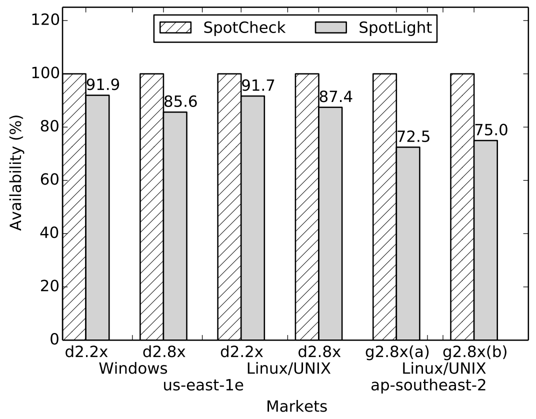


Figure 12. The availability of nested VMs in SpotCheck in practice based on the availability data for on-demand servers gathered by SpotLight

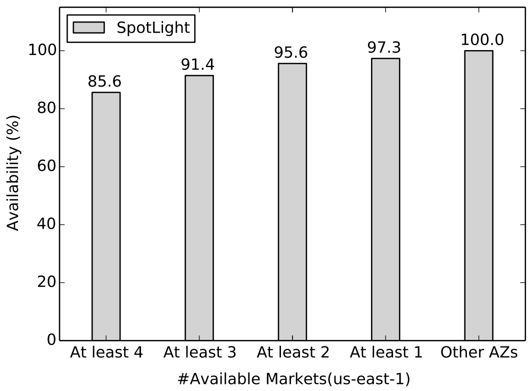


Figure 13. Availability of SpotCheck under SpotLight when considering the 4 markets in us-east-1e listed in Figure 12

A. *SpotCheck*

SpotCheck is a derivative cloud platform that rents spot servers from EC2, partitions them up using nested virtualization [15], 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 to ensure it can always 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-demand price, and then migrates to on-demand servers when 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 during a live migration 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 12 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. Figure 13 shows the correlated availability of the 4 markets from Figure 12: the x-axis represents the fraction of time that an on-demand server from at least one of the markets is available. The graph shows that at least one of the 4 markets is available 97.3% of the time. Thus, by selecting from among only these 4 markets, SpotCheck could maintain 97.3% availability. Further, if considering these markets in other availability zones, the availability is 100% (at least one server is always available). Thus, by selecting independent markets, i.e., hosted on different physical servers, SpotCheck's preserves its assumption that on-demand servers (in some market) are available when its spot servers are revoked, and its availability increases back to its original value near 100%.

B. *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 jobs at near the performance of on-demand servers. Since SpotOn focuses narrowly on batch jobs, 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 it. 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.

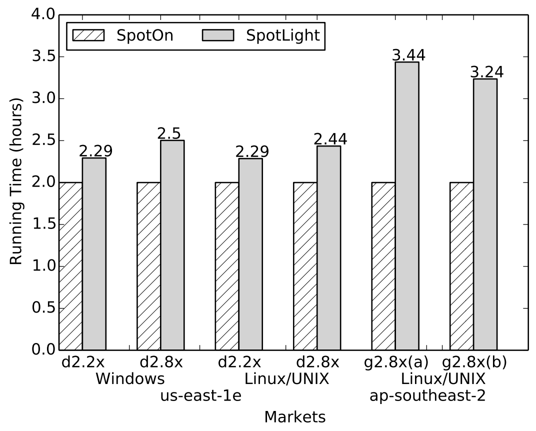


Figure 14. Average job running time using SpotOn in practice based on the availability data for on-demand servers gathered by SpotLight

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. See prior work for the closed-form equations that compute expected cost based on the time to checkpoint a job, checkpointing frequency, replication degree, revocation probability, and job running time [12]. SpotOn computes the expected cost for checkpointing and replication for all markets and then chooses the lowest expected cost to run a job.

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 are not 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 does not consider this option, it may choose a spot market that has a minimum expected cost but has on-demand servers that are 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 14 plots the completion time for a representative job with a running time of two hours and a memory footprint of 8GB, which takes approximately six minutes to checkpoint, for the same spot markets as in Figure 12. 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 a system 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.

VII. 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 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) [2, 3, 10, 17, 18]. In this case, when the bid price is below the spot price, applications simply wait until the spot price falls below the bid price again 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 [4, 5, 14, 16]. 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 is an important problem. Other work looks at employing a mix of spot and on-demand servers to either hedge against the chance of revocation [6], use on-demand servers as backups for spot servers [9], or as a fallback if spot servers are revoked (or are too volatile to satisfy performance requirements) [12].

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 EC2. 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 the applications above need spot servers the most: when spot prices spike. By quantifying the availability of spot and on-demand servers, SpotLight enables users to assess the true value of reserved, spot, and on-demand servers. Note that

EC2 continues to increase the diversity of their contract types. For example, during our evaluation, they released spot block contracts, which enable users to bid on servers for fixed blocks of time between one and six hours, such that EC2 promises to revoke only at the end of the time block. We plan to incorporate probing the availability of spot blocks into SpotLight as part of future work.

Of course, our work is based on EC2's current operational dynamics. If EC2 were to change how it sets the spot price, e.g., if it were not market-based, or eliminate spot prices entirely then the correlations we identify might no longer hold. For example, Google Compute Engine offers preemptible instances, which are akin to spot instances, in that they may be revoked, but are sold for a fixed price per unit-time. As a result, Google reveals no information, similar to the spot price, that reveals their internal operational dynamics. The only way to discover these internal dynamics is to flood the system with probe requests, which is likely too expensive for any single entity (although such information could potentially be crowd-sourced from users). While EC2 could also block SpotLight's probe requests, it might be difficult to distinguish them from normal requests of a periodic service (as probes appear as requests for instances).

Finally, much of the work above is based on (or evaluated using) 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 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 could incur significant EC2 usage costs.

VIII. 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 deploy SpotLight on EC2 and collect availability data from all $\sim$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.

Acknowledgements. This work is supported by NSF grants #1422245 and #1229059.

References

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

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

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

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

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

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

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

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

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

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

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

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

[13] W. Vickrey. Counterspeculation, Auctions, and Competitive Sealed Tenders. *Journal of Finance*, 1961.

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

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

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

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

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