Managing Risk in a Derivative IaaS Cloud

Prateek Sharma, Stephen Lee, Tian Guo, David Irwin, and Prashant Shenoy

Abstract—Infrastructure-as-a-Service (IaaS) cloud platforms rent computing resources with different cost and availability tradeoffs. For example, users may acquire virtual machines (VMs) in the *spot market* that are cheap, but can be unilaterally terminated by the cloud operator. Because of this revocation risk, spot servers have been conventionally used for delay and risk tolerant batch jobs. In this paper, we develop risk mitigation policies which allow even interactive applications to run on spot servers.

Our System, SpotCheck is a derivative cloud platform, and provides the illusion of an IaaS platform that offers always-available VMs on demand for a cost near that of spot servers, and supports unmodified applications. SpotCheck's design combines virtualization-based mechanisms for fault-tolerance, and bidding and server selection policies for managing the risk and cost. We implement SpotCheck on EC2 and show that it i) provides nested VMs with 99.9989% availability, ii) achieves upto 2-5$\times$ cost savings compared to using on-demand VMs, and iii) eliminates any risk of losing VM state.

Index Terms—Distributed computing, Platform virtualization, System software, Virtual machine monitors

1 Introduction

Many enterprises, especially technology startup companies, rely in large part on Infrastructure-as-a-Service (IaaS) cloud platforms for their computing infrastructure [8]. Today's IaaS cloud platforms, which enable customers to rent computing resources on demand in the form of virtual machines (VMs), offer numerous benefits, including a pay-as-you-use pricing model, the ability to quickly scale capacity when necessary, and low costs due to their high degree of statistical multiplexing and massive economies of scale.

To meet the needs of a diverse set of customers, IaaS platforms rent VM servers under a variety of contract terms that differ in their cost and availability guarantees. The simplest type of contract is for an *on-demand* server, which a customer may request at any time and incurs a fixed cost per unit time of use. On-demand servers are *non-revocable*: customers may use these servers until they explicitly decide to relinquish them. In contrast, *spot* servers provide an entirely different type of contract for the same resources. Spot servers incur a *variable* cost per unit time of use, where the cost fluctuates continuously based on the spot market's instantaneous supply and demand. Unlike on-demand servers, spot servers are *revocable*: the cloud platform may reclaim them at any time. Typically, a customer specifies an upper limit on the price they are willing to pay for a server, and the platform reclaims the server whenever the server's spot price rises above the specified bid limit. Since spot servers incur a risk of unexpected resource loss, they offer weaker availability guarantees than on-demand servers and tend to be much cheaper.

This is an expanded and revised version of a preliminary paper that appeared at EuroSys 2015 [29].

This paper focuses on the design of a *derivative cloud platform*, which repackages and resells resources purchased from native IaaS platforms. Analogous to a financial derivative, a derivative cloud can offer resources to customers with different pricing models and availability guarantees not provided by native platforms using a mix of resources purchased under different contracts. The motivation for derivative clouds stems from the need to better support specialized use-cases that are not directly supported (or are complex for end-users to implement) by the server types and contracts that native platforms offer. Derivative clouds rent servers from native platforms, and then repackage and resell them under contract terms tailored to a specific class of user.

Nascent forms of derivative clouds already exist. PiCloud [5] offers a batch processing service on top of EC2 spot instances to reduce costs. Similarly, Heroku [4] offers a Platform-as-a-Service by repackaging and reselling IaaS resources as resource containers. As with PiCloud, Heroku constrains the user's programming model—in this case, to containerized applications.

In this paper, we design a derivative IaaS cloud platform, called SpotCheck, that intelligently uses a mix of spot and on-demand servers to provide high availability guarantees that approach those of on-demand servers at a low cost that is near that of spot servers. In doing so, SpotCheck must balance cost and risk. In this paper, we define, implement, and evaluate multiple risk mitigation strategies for spot instances. Unlike the examples above, SpotCheck does not constrain the programming model but instead offers unrestricted IaaS-like VMs to users, enabling them to execute any application. The simple, yet key, insight underlying SpotCheck is to host customer applications (within nested VMs) on spot servers whenever possible, and transparently migrate them to on-demand servers whenever the native IaaS platform revokes spot servers. SpotCheck offers customers numerous benefits compared to natively using spot servers. Most importantly, SpotCheck enables interactive applications, such as web services, to seamlessly run on revocable spot servers without sacrificing high availability, thereby lowering the cost of running these applications. We show that, in practice, SpotCheck provides nearly five nines of availability (99.9989%), which is likely adequate for all but the most mission critical applications.

SpotCheck raises many interesting systems design questions, including i) how do we transparently migrate a customer's application before a spot server terminates while minimizing performance degradation and downtime? ii) how do we manage multiple pools of servers with different costs and availability guarantees from native IaaS platforms and allocate (or re-sell) them to customers? iii) how do we minimize costs, while mitigating user risk, by renting the cheapest mix of servers that minimize spot server revocations, i.e., to yield the highest availability? In addressing these questions, we

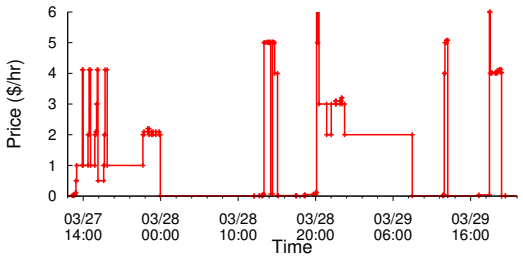


Fig. 1. Spot price of the `m1.small` server in EC2 fluctuates over time and can rise significantly above the on-demand price ($0.06 per hour) during price spikes. Note the y-axis is denominated in dollars, not cents.

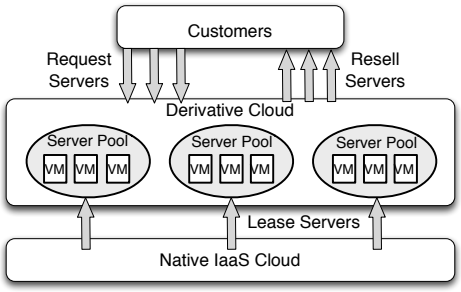


Fig. 2. A depiction of a derivative IaaS cloud platform.

make the following contributions.

Derivative Cloud Design. We demonstrate the feasibility of running disruption-*intolerant* applications, such as interactive multi-tier web applications, on spot servers, by migrating them i) to on-demand servers upon spot server revocation, and ii) back when spot servers become available again. SpotCheck requires live migrating applications from spot servers to on-demand servers within the bounded amount of time between the notification of a spot server revocation and its actual termination. SpotCheck combines several existing mechanisms to implement live bounded-time migrations, namely nested virtualization, live VM migration, bounded-time VM migration, and lazy VM restoration.

Intelligent Server Pool Management. We design server pool management algorithms that balance three competing goals: i) maximize availability, ii) reduce the risk of spot server revocation, and iii) minimize cost. To accomplish these goals, we present multiple risk management strategies for SpotCheck, including intelligently mapping customers to multiple pools of spot and on-demand servers of different types, and handling pool dynamics due to sudden revocations of spot servers or significant price changes.

Implementation and Evaluation. We implement SpotCheck on Amazon's Elastic Compute Cloud (EC2) and evaluate its migration mechanisms, pool management algorithms, and risk mitigation strategies. Our results demonstrate that SpotCheck achieves a cost that is nearly 5× less than equivalent on-demand servers, with nearly five 9's of availability (99.9989%), little performance degradation, and no risk of losing VM state.

2 Background and Overview

Our work assumes a native IaaS cloud platform, such as EC2, that rents servers to customers in the form of VMs, and offers a variety of server types that differ in their number of cores, memory allotment, network connectivity, and disk capacity. We also assume the native platform offers at least two types of contracts—on-demand and spot—such that it cannot revoke on-demand servers once it allocates them, but it can revoke spot servers. Finally, we assume on-demand servers incur a fixed cost per unit time of use, while the cost of spot servers varies continuously based on the market's supply and demand, as shown in Figure 1. [1]

Given the assumptions above, SpotCheck must manage pools of servers with different costs and availability values. While our work focuses on spot servers, largely as defined in EC2, such cost and availability tradeoffs arise in other scenarios. As one example, data centers that participate in demand response (DR) programs offered by electric utilities may have to periodically deactivate subsets of servers during periods of high electricity demand in the grid. While participation in DR programs significantly reduces electricity rates, it also reduces server availability.

Like the underlying native IaaS platform, SpotCheck offers the illusion of dedicated servers to its customers. In particular, SpotCheck offers its customers the equivalent of non-revocable on-demand servers, where only the user can make the decision to relinquish them. SpotCheck's goal is to provide server availability that is close to that of native on-demand servers for a cost that is near that of spot servers. To do so, SpotCheck uses low-cost spot servers whenever possible and "fails over" to high-cost on-demand servers, or other spot servers, whenever the native IaaS platform revokes spot servers. To maintain high availability, migrating from one type of native cloud server to another must be transparent to the end-user, which requires minimizing application performance degradation and server downtime. Section 7 quantifies how well SpotCheck achieves these goals.

SpotCheck supports multiple customers, each of which may rent an arbitrary number of servers. Since SpotCheck rents servers from a native IaaS cloud and repackages and resells their resources to its own customers, it must manage pools of spot and on-demand servers of different types and sizes, as depicted in Figure 2. Upon receiving a customer request for a new server, SpotCheck must decide which server pool should host the new instance. Upon revocation of one or more native servers from a spot pool, SpotCheck must migrate hosted customers to either an on-demand server pool or another spot pool. SpotCheck intelligently maps customers to pools to spread the risk of concurrent revocations across customers, which reduces the risk of a single customer experiencing a "revocation storm." In some sense, allocating customer requests to server pools is analogous to managing a financial portfolio where funds are spread across multiple asset classes to reduce volatility and market risk.

In addition to server pool management, SpotCheck's other key design element is its ability to seamlessly migrate VMs from one pool to another, e.g., from a spot pool to an on-demand pool upon a revocation, or from an on-demand pool to a spot pool when cheaper spot servers become available. To do this, we rely on the native IaaS platform to provide a small advance warning of spot server termination. SpotCheck then migrates its customers' VMs to native servers in other pools upon receiving a warning, and ensures that the migrations complete in the time between receiving the warning and the spot server actually terminating.

3 SpotCheck Migration Strategies

We describe SpotCheck's migration strategies and mechanisms when migrating a VM from one native cloud server to another.

1. Spot price data is from either Amazon's publicly-available history of the spot price's past six months, or from a third-party spot price archive [20].

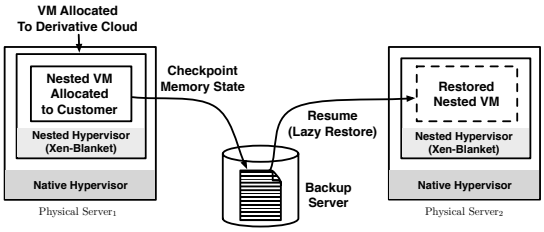


Fig. 3. SpotCheck bounded-time VM migration for moving nested VMs within an IaaS platform.

3.1 Nested Virtualization

SpotCheck rents VMs from native IaaS platforms that do not expose all of the functionality of the VM hypervisor. For example, EC2 allocates VMs to its customers, but does not expose control over VM placement or support VM migration to different physical servers. To address this limitation, SpotCheck uses *nested virtualization*, where a nested hypervisor runs atop a traditional VM, which itself runs on a conventional hypervisor [10], [35]. The nested hypervisor enables the creation of nested VMs on the host VM. Since the nested hypervisor does not need special support from the host VM, SpotCheck can install it on VMs rented from native IaaS platforms and use it to migrate nested VMs from one cloud server to another, as depicted in Figure 3.

Our SpotCheck prototype uses the XenBlanket nested hypervisor [35]. One benefit of using nested virtualization is that SpotCheck can create multiple nested VMs on a single host VM, allowing it to slice large native VMs into smaller nested VMs and allocate them to different customers, similar to how an IaaS platform slices a physical server into multiple VMs.

3.2 VM Migration

Since SpotCheck runs nested hypervisors on VM servers acquired from native IaaS platforms, it has the ability to migrate nested VMs from one server to another. SpotCheck leverages two VM migration mechanisms to implement its migration strategy: live migration and bounded-time VM migration.

Live VM migration enables SpotCheck to migrate a nested VM from one server to another, while incurring nearly zero downtime to a customer's resident applications. However, live-migration is requires time proportional to the size of the VM's memory and the page dirty (write) rate. As a result, live migrating a VM is not always feasible, since an IaaS platform may revoke a spot server at any time, while providing only a small warning period for the server to complete a graceful shutdown. Once the warning period ends, the IaaS platform forcibly terminates the VM. For example, EC2 provides a warning of 120 seconds before forcibly terminating a spot server [9]. Importantly, if the latency to live migrate a VM exceeds the warning period, then the spot server termination causes resident nested VMs to lose their memory state.

SpotCheck leverages an alternative migration approach, called bounded-time VM migration [30], [31], which provides a guaranteed upper bound on migration latency that is independent of a VM's memory size or the page dirty rate. Supporting bounded-time VM migration requires maintaining a partial checkpoint of a VM's memory state on an external disk by running a background process that continually flushes dirty memory pages to a backup server to ensure the size of the dirty pages does not exceed a specified threshold. This threshold is chosen such that any outstanding dirty pages can be safely committed upon a revocation within the time bound [30], [31]. The VM may then resume from the saved memory state on a different server, as depicted in Figure 3.

When migrating a nested VM from an on-demand server to a spot server, e.g., when a cheaper spot server becomes available, SpotCheck uses live migration regardless of the nested VM's memory size, since there is no constraint on the migration latency. When migrating a nested VM from a revoked spot server, bounded-time VM migration is usually necessary, since the migration must complete before the spot server terminates.

To support bounded-time VM migration, SpotCheck must manage a pool of backup servers that store the memory state of nested VMs on spot servers, and continuously receive and commit updates to this state. As we show in Section 7, each backup server is able to host tens of nested VMs without degrading their performance, which makes the incremental cost of using such additional backup servers small in practice.

3.3 Lazy VM Restoration

Bounded-time VM migration is a form of VM suspend-resume that saves, or suspends, the VM's memory state to a backup server within a bounded time period, and then resumes the VM on a new server. Resuming a VM requires restoring its memory state by reading it from the disk on the backup server into RAM on the new server. The VM cannot function during the restoration process, which causes downtime until the VM state is read completely into memory. Since the downtime of this traditional VM restoration is disruptive, SpotCheck employs lazy VM restoration [19], [22] to reduce the downtime to nearly zero. Lazy VM restoration involves reading a small number of initial VM memory pages—the *skeleton state*—from disk into RAM and then immediately resuming VM execution without any further waiting.

The remaining memory pages are fetched from the backup server on demand, akin to virtual memory paging, whenever the VM's execution reads or writes any of these missing pages. Lazy VM restoration substantially reduces the latency to resume VM execution at the expense of a small window of slightly degraded performance, due to any page faults that require reading memory pages on demand. Combining lazy VM restoration with bounded-time VM migration enables a new "live" variant of bounded-time VM migration that minimizes the downtime when migrating VMs within a bounded time period upon revocation.

3.4 Virtual Private Networks

While the migration mechanisms above minimize customers' downtime and performance degradation during migrations, maximizing transparency also requires that the IP address of customers' nested VMs migrate to the new host to prevent breaking any active network connections. In a traditional live migration, the VM emits an `arp` packet to inform network switches of its new location, enabling switches to forward subsequent packets to the new host and ensuring uninterrupted network connections for applications [15]. However, in SpotCheck, the underlying IaaS platform is unaware of the presence of nested VMs on the host VMs. SpotCheck currently employs a separate physical interface on the host VM to provide each nested VM its own IP address, in addition to the host's default interface and IP address. Thus, SpotCheck configures Network Address Translation (NAT) in the nested hypervisor to forward all network packets arriving at an IP address to its associated nested VM. IaaS platforms, such as EC2, make this feasible by supporting the creation of multiple interfaces and IP addresses on each host. However, since the IP address is associated with the host VM, the address does not automatically migrate with

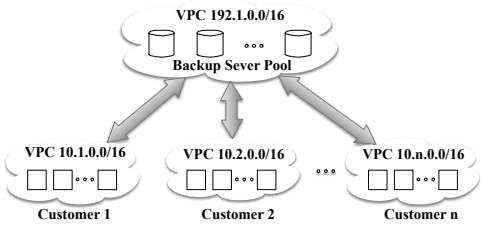


Fig. 4. SpotCheck creates separate VPCs for each customer. Customers can share a pool of backup servers by connecting to backup server VPC. In Amazon EC2, this is done through VPC peering.

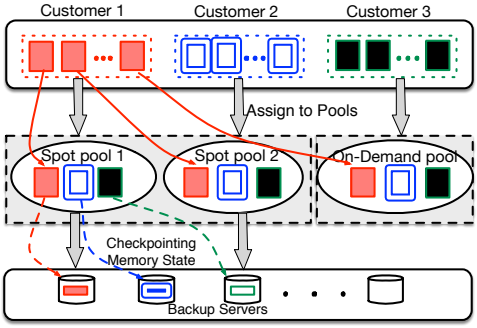


Fig. 5. SpotCheck's architecture using multiple pools.

the nested VM. Instead, SpotCheck must take additional steps to detach a nested VM's address from the host VM of the source and reattach it to the destination host. EC2 supports VPNs through its Virtual Private Cloud (VPC) feature, which creates logically isolated virtual networks and enables users to directly assign IP addresses to their VMs. This ensures the IP address of nested VMs remains unchanged after migration.

3.5 Putting it all together

SpotCheck combines nested virtualization, virtual private networks, VM migration, and lazy VM restoration to implement its migration strategies. Upon initial allocation, we assign a backup server to each nested VM on a spot server, which stores its memory state. Upon spot server revocation, SpotCheck migrates the nested VM to a new destination server via bounded-time VM migration.

The destination server is chosen by a higher-level server pool management algorithm. Once the VM's migration completes, SpotCheck uses VPC functions to deallocate the IP address on the source server, and then reallocate the IP address on the destination server and configure the nested hypervisor to forward packets to the new address. SpotCheck also must detach the VM's network-attached disk volume and reattach it to the destination server before the VM resumes operation. We discuss SpotCheck's treatment of storage more in Section 6. If SpotCheck employs bounded-time VM migration, it uses lazy VM restoration to minimize the migration downtime.

4 SpotCheck Architecture

SpotCheck's architecture maintains multiple pools of servers, as shown in Figure 5, where each pool contains multiple native VM servers of a particular type, specifying an allotment of CPU cores with specified performance, memory, network bandwidth, etc. For each server type, SpotCheck maintains separate spot and on-demand pools. SpotCheck exposes a user interface similar to that of a native IaaS platform, where customers may request and relinquish servers of different types. However, SpotCheck offers its customers the abstraction of non-revocable servers, despite often executing them on revocable spot servers.

SpotCheck maps its customers' nested VMs, which may be of multiple types, to different server pools, as illustrated in Figure 5. By diversifying its portfolio of spot pools and smartly placing customer VMs in these pools, SpotCheck is able to reduce the number of concurrent revocations. Policies for managing spot pools to are discussed in Section 5.1. In addition, SpotCheck also maintains a pool of backup servers, each capable of maintaining checkpoints of memory state for multiple nested VMs hosted on spot servers. Thus, SpotCheck assigns each native server from a *spot pool* to a distinct backup server, such that any nested VMs hosted on it write their dirty memory pages to their backup server in the background. SpotCheck does not assign native servers in the on-demand pool to a backup server, since they can live migrate any nested VMs hosted on them without any time constraints. Policies for mapping VMs to backup servers are discussed in Section 5.3.

Given the architecture above, we next describe the techniques and algorithms SpotCheck employs to manage server pools, handle pool dynamics, and mitigate the risk of revocation.

5 Managing Risks

This section describes the various risks encountered when running a derivative cloud on inherently volatile markets and presents multiple policies to manage these risks. SpotCheck's migration strategies provide system-level mechanisms that leverage spot servers by migrating applications away from spot servers upon revocation. However, running applications using these revocable spot servers requires managing multiple risks in order to reduce the number of revocation events, reduce the impact of revocation storms, and maintain the efficiency of restoration. SpotCheck manages potential risks in all three facets with a combination of policies that intelligently manages customer server pools, a backup server pool, and hot spare servers.

5.1 Server Pool Selection

When a customer requests an instance of a specific size, e.g. small server, SpotCheck must make trade-offs between cost, stability, and the frequency of concurrent revocations. That is, SpotCheck ideally would allocate stable server resources at cheap prices and avoid any customer losing significant (or all of their) spot servers at once. To satisfy such a requirement, SpotCheck makes a sequences of decisions by taking into account both a spot server's price history and a customer's existing spot allocation.

The first decision is which type of servers to request from the native IaaS cloud platform. In the simplest case, when a customer requests a new VM of a certain type, SpotCheck satisfies the request by allocating a native VM of the same type from the underlying IaaS platform, and then configures a nested VM within the native VM for use by the customer. Since nested virtualization supports the ability to run multiple nested VMs on a single host VM, SpotCheck also has the option of i) requesting a larger native VM than the one requested by the customer, ii) slicing it into smaller nested VMs of the requested type, and then iii) allocating one of the nested VMs to the customer. Slicing a native VM into smaller nested VMs is useful, since prices for spot servers of different types vary based on market-driven supply and demand. By default, SpotCheck supports the *slicing* mode which allows a wider selection of spot markets and minimizes cost by exploiting price differences between markets.

Presented with a set of spot markets to choose from, SpotCheck employs three different policies in choosing the spot server type.

The first strategy, referred to as *cheapest-first*, is a simple greedy policy that chooses the cheapest spot server, based on the current prices, to satisfy a request. We exploit the fact that the server size-to-price ratio is not uniform: a large server, say a `m3.large`, which is able to accommodate two medium VM servers of size `m3.medium` may be *cheaper* than buying two medium servers. Since the pricing of on-demand servers is roughly proportional to their resource allotment, such that a server with twice the CPU and RAM of another costs roughly twice as much, under ideal market conditions, the price of spot servers should also be roughly proportional to their resource allotment. However, we have observed that different server types experience different supply and demand conditions. In general, smaller servers appear to be more in demand than larger servers because their spot price tends to be closer to their on-demand price. As a result, larger servers are often cheaper, on a unit cost basis, than smaller server for substantial periods of time, which enables SpotCheck's greedy approach to exploit the opportunity for arbitrage. However, note that whenever SpotCheck slices a spot server into multiple nested VMs, it does incur additional risk, as a revocation requires migrating *all* of its resident nested VMs.

An alternative to the greedy cheapest-first strategy above is a conservative *stability-first* policy that allocates a native spot server (from the various possible choices) with the most stable prices. To increase availability, SpotCheck must reduce both the frequency of revocation events and the impact of each one, e.g., due to downtime. Allocating a spot server with a stable market price reduces the probability of a spot server revocation, which in turn increases availability.

Both cheapest-first and stability-first strategies do not consider the existing allocation of a customer's spot servers. Such strategies might be problematic when a customer's spot instances all belong to a single server pool, incurring concurrent revocations upon price spikes. A revocation event due to a price spike for a particular type of spot server can cause concurrent revocations within a single spot pool. However, different pools are *independent*, since spot prices of different server types fluctuate independently of one another and are uncorrelated, as seen in Figures 6(c) and (d). Hence, SpotCheck also supports a more sophisticated policy that bounds the maximum concurrent revocations per customer, defined as r, by distributing a customer's nested VMs across multiple pools. Revocation storms degrade nested VM performance and increase downtime by overloading backup servers, which must simultaneously broker the migration of every revoked nested VM. SpotCheck employs this policy to reduce the risk of a sudden price spike causing mass revocations of spot servers of a particular type at one location (or availability zone in EC2 parlance).

The key idea of this bounded greedy algorithm is to first identify the cost and stability ranges using cheapest-first and stability-first strategies, and then search for a specific spot server type that has cost and stability within the above ranges without violating concurrent threshold r. SpotCheck also favors the spot server type that incurs fewer concurrent revocations, i.e. smaller instances, as a tie breaker. This tie breaker is beneficial because it allows SpotCheck to maintain a reasonable amount of sliced servers, in case of customer shortage. Further, SpotCheck sets up a threshold of maximum number of concurrent revocations allowed, constraining the candidate server types.

5.2 Reducing Revocation Risks using Bidding

Once a spot market has been decided for a VM, SpotCheck must determine a bid price. Although SpotCheck has no control over the fluctuating price of spot servers, it does have the ability to determine a maximum bid price it is willing to pay for servers in each of its spot pools. Designing "optimal" bidding strategies in spot markets in various contexts is an active research area, and prior work has proposed a number of different policies [11], [20], [37]. Adapting these policies to SpotCheck's context may be possible. However, since our focus is on designing a derivative IaaS cloud, rather than bidding strategies, SpotCheck currently employs one of two simple policies: either place a single bid or bid at multiple different prices for a specific server type.

5.2.1 Single-level Bidding

With the single bid policy, SpotCheck picks a single bid price for every spot pool. The bid is chosen to minimize the expected cost of running on the spot instances and on-demand instances due to the revocation. A low bid implies a higher revocation rate and more time spent running on on-demand servers, and thus nullifies the lower average spot price. Similarly, a high bid price reduces revocations, but results in increased spot instance costs. In order to balance the tradeoff, SpotCheck finds a biding price b^* that is at the "knee" point of the revocation probability curve. Put simply, a "knee" point appears when the probability curve flattens out and can be found by calculating the local maxima of the curve.

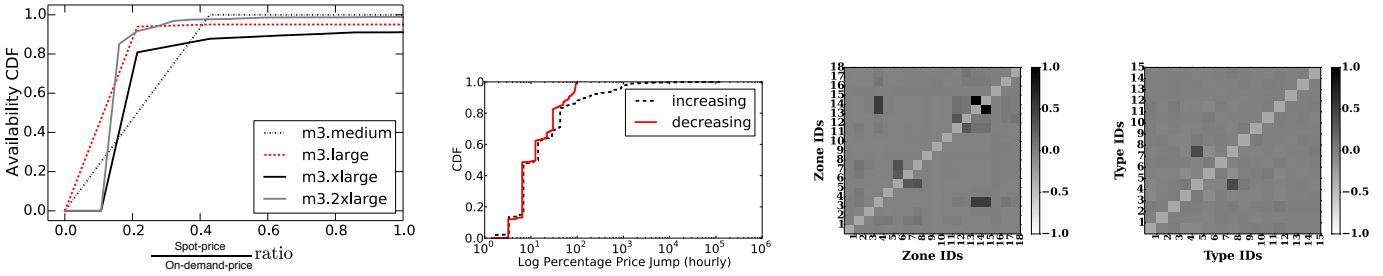
In the case of EC2's spot market, empirical data shows that the probability of revocation decreases with higher bid prices, but it flattens quickly, such that the "knee" of the curve, as depicted in Figure 6(a), is slightly lower than the on-demand price. Thus, simply bidding the on-demand price is an approximation of bidding an "optimal" value that is equal to the knee of this availability-bid curve. This implies that large price spikes are the norm, with spot prices frequently going from well below the on-demand price to well above it, as shown in Figure 6(b). Figure 6(a) also shows that the spot prices are extremely low on average compared to the equivalent prices for on-demand servers. This is likely due to the complexity of modifying applications to effectively use the spot market, which suppresses demand by limiting spot servers to a narrow class of batch applications.

The cost-optimal bidding level can be found with the following model. Given n customers, each with C_i servers, SpotCheck must provision a total of $V = \sum_i^n C_i$ nested VMs. Since SpotCheck maps these V nested VMs onto multiple pools, the total cost L of renting native servers from the IaaS platform is equal to the cost of the necessary spot servers plus the cost of the necessary on-demand servers plus the cost of any backup servers. Thus, the amortized cost per nested VM is L/V.

We represent the expected cost of running a spot server with a bid b as $E[c(b)]$ and it is:

$$E[c(b)] = (1-p) \cdot E[c_{spot}(b)] + p \cdot c_{od} + \varepsilon \quad (1)$$

where p denotes the probability of a revocation when it resides on a spot server, $E[c_{spot}(b)]$ denotes the average price of the spot server at a bid b, and c_{od} denotes the price of the equivalent on-demand server. We note that p is simply the probability of the spot price rising above the bid price, i.e., $p = P(c_{spot}(b) > bid)$, which is given by the cumulative distribution shown in Figure 6(a) that we derive empirically for different spot pools. Finally, the additional small constant cost, ε denotes the amortized cost to



(a) CDF of prices

(b) Price changes are large

(c) Correlations of prices between zones.

(d) Correlations between instance types.

Fig. 6. Price dynamics across EC2 spot markets from April to October 2014 for all `m3.*` types: the spot price distribution (a) has a long tail, (b) exhibits large price changes, and (c) is uncorrelated across locations and server types (d).

run the backup servers. A single backup server with cost c_b can be shared by multiple (N) VMs, yielding $\varepsilon = c_b/N$. SpotCheck's optimized backup server design can support upto 40 VMs, and thus the extra cost associated with backup servers is quite small.

The expected costs can be calculated for any bid level, and only requires availability and price information, both of which are obtained using the publicly available price traces published by Amazon. In order to find the optimum bid level b^* which minimizes $E[C(b)]$ in Equation 1, a simple numerical search using gradient descent is used to find the minima and the associated bid level. This operation is performed only once per spot market, and is re-run only upon significant price changes.

To compute a nested VM's availability, assume that the market price of a spot server changes once every T time units, such that the server will be revoked once every T/p time units, yielding a revocation rate of $R = p/T$. Here, we assume live migration does not result in significant downtime, while bounded-time VM migration incurs the downtime required to i) read sufficient memory state after a lazy restoration, ii) attach a networked disk volume to the new server, and iii) reassign the IP address to the new server. If D denotes the delay to perform these operations, the downtime experienced by the nested VM is $D \cdot R$ per unit time, i.e., $D \cdot p/T$.

Thus, our expected cost equation above allows us to analyze different pool management and bidding policies. This expected cost includes the cost of running the nested VM on either a spot or on-demand server, and the cost of any backup servers. We also assume that nested VMs use an associated EBS volume in EC2 to provide persistent network-attached storage. However, we do not include storage costs, since they are negligible at the backup server, and thus the same when using SpotCheck or the native IaaS platform. Similarly, our analysis does not include costs associated with external network traffic, since these costs are the same when using SpotCheck or the native IaaS platform. Note that there is no cost in EC2 associated with the network traffic between nested VMs and their backup server, since network traffic between EC2 servers incurs no charge.

5.2.2 Multi-level Bidding

Alternatively, SpotCheck also supports multi-level bidding within a spot market. The goal of this bidding strategy is to reduce the number of concurrent revocations and thus mitigate the occurrence of revocation storms. Bidding at multiple levels means that a price increase does not necessarily affect *all* the servers in a market. We use a simple, two-level bidding strategy wherein we have a *low* and a *high* bid. Servers are randomly placed either in the low bid pool or the high bid pool. A spot price increase is going to affect the low-bid servers first and cause them to be revoked; it only affects the high-bid servers when the price crosses the high bid, which may happen after a small delay as the price ramps up, or may not happen at all. Of course, a sudden increase in price above the high-bid mark will cause all the servers to be revoked simultaneously. If the gap between revocation of the low and high bid servers is large enough, then the impact of the revocation storm is reduced, because the backup server will have to bear the brunt of only half the number of concurrent migrations. Additionally, requesting a smaller number of on-demand servers may also reduce the latency of server acquisition [25].

Our two-level bidding strategy is as follows. The low-bid is set to the on-demand price (as before), and then we use a numerical search approach to find the high-bid. Servers are equally and randomly distributed among the two bid levels. Just like in single level bidding, there is a tradeoff between the bid and the cost. A bid higher than the on-demand price means that we are ready to pay that price, and thus bidding too high is not cost optimal.

Therefore, when choosing the bid levels, we seek to minimize the i) revocation storm size and ii) expected cost. In single level bidding, a revocation storm affects all n of the servers in that market. With two-level bidding, some storms affect only $n/2$ servers, and thus their impact is said to be mitigated. We thus use the *fraction of revocation storms mitigated*, f_r as a metric. A storm is mitigated if the gap between revocation of high and low bid servers is at least t_h. Once the t_h threshold is crossed, the high and low bid revocations will not be simultaneous because the backup server will have finished lazily restoring the VMs. Based on experimental analysis, we set $t_h = 10$ minutes.

Since the low-bid is fixed (equal to on-demand price), we use a simple numerical search for the high-bid which maximizes the fraction of revocation storms mitigated, f_r, such that the increase in cost stays under a threshold. Thus, we have the constraint: $E[c_r] \leq \alpha \cdot E[C]$, where $E[C]$ is the expected cost for the single-level bidding policy. Expectations for both f_r, c_r are obtained by using historical price traces, and we use an $\alpha = 0.2$, i.e., we limit the increase in cost to 20%. The upper bound on the search for the high-bid is enforced by Amazon, which limits the maximum bid to be 10 times the on-demand price.

5.3 Reducing Concurrent Revocations with Backup Servers

After requesting spot servers from the native IaaS platform, SpotCheck must assign each nested VM within a spot pool to a distinct backup server. SpotCheck also distributes nested VMs in a spot pool across multiple backup servers. The task of assignning VMs to backup servers is analogous to VM placement and server consolidation [27] where the goal is to pack VMs onto a minimum

number of physical servers.

Since each spot pool is subject to concurrent revocations, spreading one pool's VMs across different backup servers reduces the probability of any one backup server experiencing a large number of concurrent revocations. The approach also spreads the read and write load due to supporting bounded-time VM migration across multiple backup servers. To this end, SpotCheck employs a round-robin policy to map the nested VMs within each pool across the set of backup servers. With the **round-robin policy**, SpotCheck simply assigns each nested VM to the next available backup server[2]. If any backup server becomes fully utilized, SpotCheck provisions a native VM from the IaaS platform to serve as a new backup server, and adds it to the backup server pool. A backup server in SpotCheck can host multiple($N = 40$) VMs, and may not always be fully utilized. The under-utilization can occur because VM arrivals and lifetimes are dynamic, and SpotCheck does not have apriori knowledge about VM creation/termination requests which it can use to provision the minimum number of backup servers.

SpotCheck's round-robin policy might lead to unbalanced backup servers in terms of concurrent revocations. The optimal mapping from spot servers to backup servers to minimize the maximum number of concurrent revocations can be formulated as an integer linear program (ILP). Let N spot servers be belong to different spot pools (S), and p_{is} denotes the mapping between servers and server-pools. The backup servers are denoted by M, and their capacity is denoted by U. Our goal is to find such an optimal mapping X for every spot server, where x_{ij} denotes assignment of server i to backup j. We can then represent the number of concurrent revocations cr_j of j^{th} backup server: $cr_j = \underset{s \in S}{\operatorname{argmax}} \sum_{i=0}^{m} p_{si} x_{ij} w_i$. Intuitively, cr_j is defined by the largest server pool hosted. We define revocation storm severity to be the maximum concurrent revocations on any backup server $cr = \max_j cr_j$. Our objective is to minimize cr with the following constraints:

$$\sum_{i \in N} w_i x_{ij} \leq U \qquad \forall j \in M \tag{2}$$

$$\sum_{j \in M} x_{ij} = 1 \qquad \forall i \in N \tag{3}$$

$$x_{ij} \in \{0,1\} \qquad \forall i \in N, j \in M$$

Constraint 2 ensures no backup servers will be overloaded and the other constraints make sure all spot servers are assigned to only one backup server. This ILP can be solved by an off-the-shelf solver like CPLEX. However, this ILP formulation requires remapping of VMs to backup servers periodically, and is not feasible in the current SpotCheck implementation. We develop an online version of this backup assignment which doesn't require remappings below.

Online greedy backup assignment policy. The round-robin policy discussed earlier does not try to minimize the number of concurrent revocations, and the ILP formulation is an offline approach. We have developed an online policy (called *online-greedy*) which seeks to minimize the number of concurrent revocations, and can be run dynamically as VMs are launched.

The online-greedy policy runs after a VM has been assigned to a server pool i. It places the VM into a backup server which has the least number of VMs from pool i, and which still has capacity available to host one more VM. Since the number of concurrent revocations is simply the number of VMs from the same pool, by picking the backup server with the smallest number of VMs from that pool, the backup servers are not overloaded with VMs from the same pool. Thus, the online-greedy policy seeks to equalize the number of VMs from each pool across all the backup servers.

5.4 Reducing Downtime with Hot Spares

When the market price rises above the bid price, the IaaS platform revokes all servers within a pool at the same time, resulting in a revocation storm. A simple approach to handling concurrent revocations is to request an equivalent number of on-demand servers from the IaaS platform and migrate each nested VM to a new on-demand server. However, requesting new servers in a lazy fashion when necessary is only feasible if the latency to obtain them is smaller than the warning period granted to a revoked server. Note that there is never a risk of losing nested VM state, since the backup server stores it even if there is not a destination server available to execute the nested VM. For example, empirical studies have shown that it takes up to 90 seconds to start up a new on-demand server in EC2 [25], while the warning period for a spot server is two minutes, which leaves only 30 seconds to migrate the spot servers state to the new server. If the allocation latency were to exceed the warning time, such a lazy strategy is not possible due to the risk of significant VM downtime. To handle this scenario, SpotCheck proactively acquires a pool of hot spares, servers that are ready to receive nested VMs from revoked spot servers immediately without waiting for a new server to come online. While reducing the risk of downtime, hot spares inevitably increase SpotCheck's overhead cost. Therefore, it is important to only maintain a necessary amount of hot spare servers.

SpotCheck's hot-spare policy seeks to ensure that a small fraction of VMs affected by a revocation storm have a stand-by on-demand server. If the expected maximum number of simultaneous revocations is $E[R_M]$, then we deploy $\beta \cdot E[R_M]$ hot spares. Thus the cost of the hot spares is proportional to the simultaneous revocations, which in turn is a result of pool management and bidding policy. For example, we can set $\beta = 0.1$, which means that 10% of VMs migrating face minimal downtime, while the rest could potentially be affected due to the delay in acquiring on-demand servers from the native IaaS. The hot-spare pool is replenished after the hot spares are used up during migrations.

An alternative approach to using dedicated hot spares is to use existing servers in other stable pools as staging servers. This approach is attractive if these existing servers are not fully utilized by the nested VMs. Here, the staging servers only run the nested VMs from a revoked spot server temporarily, while SpotCheck makes concurrent requests for new on-demand or spot servers to serve as the final destination. This strategy doubles the number of migrations and the associated overhead, but it also enables the system to reduce risk without increasing its costs. Hot spares and staging servers may also serve as a temporary haven for displaced spot VMs, in the rare case when requests for on-demand servers fail because they are unavailable from the IaaS platform[3].

5.5 Providing Security Isolation using VPCs

By default, SpotCheck VMs share the cloud servers (using nested virtualization) and the backup servers with different VMs belonging to other customers. This sharing may reduce both the performance and security isolation among VMs. To provide improved isolation,

2. A backup server is not necessary for running stateless services e.g., a single web server that is part of a tier of replicated web servers, since these services are designed to tolerate failures. However, as with any IaaS platform, SpotCheck does not make any assumptions about applications that run on it, and may incur slightly higher costs than necessary for stateless services.

3. IaaS platforms attempt to provision resources to stay ahead of the demand curve, but they may run out of on-demand servers if demand exceeds supply.

SpotCheck also provides a *Private Cloud* mode, which removes sharing of cloud servers and backup servers between different customers.

In *private cloud* mode, SpotCheck does not place different customers' VMs on the same cloud server, but instead provides a dedicated VPC to each customer to improve network isolation. More importantly, VMs which run in this mode have their own dedicated backup servers. All the bidding, pool management, and other policies are still applicable in this mode, and the VMs among different VPCs do not interact in any way. The key difference is the non-sharing of backup servers, which prevents VMs from one user from interfering with other users' VMs. Multiple VMs sharing a backup server amortizes the backup server cost among them, and in the private cloud mode, the number of VMs run by a customer may not be large enough to completely pack the backup servers. This under-utilization backup servers increases the cost of running in private cloud mode if the number of VMs is small. Thus, the private cloud mode provides increased isolation, at a potentially higher cost, which is a function of the number of VMs that a customer is running in this mode. While we use a relatively large and powerful backup server (`m3.xlarge`) which can service up to 40 VMs, it may be excessive for smaller private clouds. If the number of customer VMs is significantly less than 40, SpotCheck automatically chooses progressively smaller and cheaper backup servers. For example, 20 VMs can be serviced by the `m3.large` server type, at half the cost of the extra large server. Note that the ratio of computing and storage resources on the backup servers to the multiplexing factor remains the same, and the performance of VMs in the private cloud mode is unaffected. This allows the private cloud mode to be cost feasible even at small sizes.

While the above private cloud mode provides isolation, it can also result in higher costs for customers with low VM requirements. To address this, SpotCheck also offers a *VPC-only* mode, which provides VPCs to customers but shares backup servers among VPCs. The VPCs provide network isolation, and the shared backup servers remove the cost overhead. Thus, foregoing the backup server isolation results in lower costs. Backup servers can be shared among VPCs by EC2's VPC-peering mechanism.

5.6 Arbitrage Risks

One caveat in our analysis is that we do not consider the second-order effects of our system on spot prices and availability. While it is certainly possible that widespread use of SpotCheck may perturb the spot market and affect prices, our analysis assumes that the market is large enough to absorb these changes. Regardless, our work demonstrates that a substantial opportunity for arbitrage exists between the spot and on-demand markets. Consumers have a strong incentive to exploit this arbitrage opportunity until it no longer exists. SpotCheck also benefits EC2, since it should raise the demand and price for spot servers by opening them up to a wider range of applications.

The increasing popularity and demand of derivative clouds might also incentivize IaaS platforms to increase their pool of spot servers. However, our analysis assumes that on-demand servers of some type will always be available. While on-demand servers of a particular type may become unavailable, we assume the market is large enough, so that on-demand servers of some type are always available somewhere. As we discuss, SpotCheck's pool management strategies operate across multiple markets by permitting the unrestricted choice of server types and availability zones (within a region). These strategies protect against the rare event where one type of on-demand server becomes unavailable.

Of course, regardless of the risk mitigation strategies above, SpotCheck cannot provide higher availability than the underlying IaaS platform. For example, if the IaaS platform fails or becomes disconnected, as occasionally happens to EC2 [16], SpotCheck would also fail. Since we do not have access to long-term availability data for EC2 or other IaaS platforms, in our experiments, the term "availability" refers to relative availability with respect to the underlying IaaS platform, which we assume is 100% available.

6 SpotCheck Implementation

We implemented a prototype of SpotCheck on EC2 that is capable of exercising the different policy options from the previous section, allowing us to experiment with the cost-availability tradeoffs from using different policies. SpotCheck provides a similar interface as EC2 for managing virtualized cloud servers, although the servers are provisioned in the form of nested VMs.

SpotCheck Controller. SpotCheck's main component is the centralized controller which is implemented in `python`. It interfaces between customers and the underlying native IaaS platform. It runs on a dedicated server and maintains a global and consistent view of SpotCheck's state, e.g., the information about all of its provisioned spot and on-demand servers and all of its customers' nested VMs and their location. We do not include controller costs in our estimates, since we expect them to be negligible, as they are amortized across all the VMs.

Customers interact with SpotCheck's controller via an API that is similar to the management API EC2 provides for controlling VMs. Internally, the controller uses the EC2 REST APIs to issue requests to EC2. The controller monitors SpotCheck's state by tracking the cloud server each nested VM runs on, the IP address associated with the nested VM, and the customer's access credentials, and stores this information in a database.

The controller also implements the various pool management strategies from the previous section, e.g., by determining the bids for spot instances and triggering nested VM migrations from one server pool to another. Finally, the controller monitors the nested VMs, the mapping of nested VMs to backup servers, and the current spot price in each spot pool. Our prototype implementation uses the XenBlanket [35] nested hypervisor running on a modified version of Xen 4.1.1. The driver domain (`dom-0`) runs Linux 3.1.2 with modifications for supporting XenBlanket. XenBlanket is compatible with all EC2 instance types that support hardware virtual machines (HVM). VMs use network-attached EBS volumes to store the root disk and any persistent state. The nested hypervisor consumes minimal CPU and memory (< 200 MB) resources, and the resultant nested VMs have slightly reduced memory size compared to the native IaaS VM. We account for any performance degradation due to the overhead of nested virtualization as well as the reduced resource availability for the nested VMs when computing the cost of the nested VMs in Section 7.2.

To implement SpotCheck, we modified XenBlanket to support bounded-time VM migration in addition to live migration. We adapt the bounded-time VM migration implemented in Yank [30] for use with nested virtualization and implement additional optimizations to reduce downtime during migration. In particular, the continuous checkpoints due to bounded-time VM migration guarantee that during the last checkpoint the nested VM is able to transfer the stale state within the warning time. SpotCheck configures nested VMs running on spot servers to use bounded-time VM migration, while those mapped to an on-demand pool use live migration.

Nested VMs mapped to a spot server pool are also mapped to a backup server, which must process a write-intensive workload during normal operation and must process a workload that includes a mix of reads and writes during revocation events, e.g., to read the memory state of a revoked nested VM and migrate it.

During revocations, the backup server prepares for nested VM restoration by loading images into memory. In addition, we also implement bandwidth throttling using `tc` on a per-VM basis to limit the network bandwidth used for each migration/restoration operation, and to avoid affecting nested VMs that are not migrating. Thus, we optimize our backup server implementation for the common case of efficiently handling a large number of concurrent revocations without degrading performance for long durations. Our SpotCheck prototype uses the `m3.xlarge` type as backup servers, since they currently offer the best price/performance ratio for our workload mix. Our prototype uses a combination of SSDs and EBS volumes to store the memory images.

Lazy restoration requires transferring the "skeleton" state of a VM, comprising the vCPU state, all associated VM page tables, and other hardware state maintained by the hypervisor, to the destination host and immediately beginning execution. This skeleton state is small, typically around 5MB, and is dominated by the the size of the page tables. The skeleton state represents the minimum amount of state sufficient for the hypervisor on the destination host to create the domain for VM and begin executing instructions. The missing memory pages, which reside on the backup server's disk, are mapped to the domain's memory when available and the VM resumes execution. A background process concurrently pre-fetches the remaining unrestored pages for faster restoration.

We conducted extensive measurements on EC2 to profile the latency of SpotCheck's various operations. Table 1 shows the results for one particular server type, the `m3.medium`. Our measurements show that EC2 provides an opportunity to gracefully shutdown the VM, by issuing a shutdown command, before forcibly terminating the VM two minutes after issuing the shutdown. Thus, we replace the default shutdown script with our own script, which EC2 invokes upon revocation to notify SpotCheck of the two minute warning. However, as we mention previously, as of January 2015 [9], EC2 now provides an explicit two minute notification of shutdown through the EC2 management interface.

When employed natively our live bounded-time VM migration incurs a brief millisecond-scale downtime similar to that of a post-copy live migration. However, Table 1 shows that EC2's operations also contribute to downtime. In particular, SpotCheck can only detach a VM's EBS volumes and its network interface after the VM is paused, and it can only reattach them after the VM is resumed. From Table 1, these operations (in bold) cause an average downtime of 22.65 seconds. While significant, this downtime is not fundamental to SpotCheck: EC2 and other IaaS platforms could likely significantly reduce the latency of these operations, which would further improve the performance and availability we report in Section 6. Even now, this $\sim$23 second downtime is not long enough to break TCP connections, which generally requires a timeout of greater than one minute.

Finally, SpotCheck's implementation builds on our prior work on Yank [30] by including the performance optimizations above. In particular, these optimizations enable i) SpotCheck's backup servers to support a much larger number of VMs and ii) lazy on-demand fetching of VM memory pages to drastically reduce restoration time, e.g., to $<$0.1 seconds. We quantify the impact of these optimizations on cost, performance, and availability.

TABLE 1
Latency for various SpotCheck operations on EC2 for the `m3.medium` server type across 20 measurements over a one week period.

	Median(sec)	Mean(sec)	Max(sec)	Min(sec)
Start spot instance	227	224	409	100
Start on-demand instance	61	62	86	47
Terminate instance	135	136	147	133
Unmount and detach EBS	10.3	10.3	11.3	9.6
Attach and mount EBS	5	5.1	9.3	4.4
Attach Network interface	3	3.75	14	1
Detach Network interface	2	3.5	12	1

7 Evaluation

Our evaluation consists of a mix of end-to-end experiments and simulations. For our end-to-end experiments, we quantify SpotCheck's performance under different scenarios using a combination of EC2 servers and our own local servers. For our simulations, we combine performance measurements from our end-to-end experiments with historical spot pricing data on EC2 to estimate SpotCheck's cost savings and availability at scale over a long period. We run all the microbenchmark experiments in a single EC2 availability zone, while our simulations include cross-availability zone experiments within a single region.

XenBlanket requires servers with have HVM capabilities, and we primarily use `m3.*` server types. In particular, we use `m3.xlarge` server types for our backup servers, and, by default, host nested VMs on `m3.medium` server types. We evaluate SpotCheck using two well-known benchmarks for interactive multi-tier web applications: TPC-W [3] and SPECjbb2005 [2]. We are primarily interested in memory-intensive workloads, since the continuous checkpointing of memory pages imposes the most performance overhead for these workloads.

TPC-W simulates an interactive web application. We use Apache Tomcat (v6.26) as the application server and MySQL (v5.0.96) as the database. We configure clients to perform the "ordering workload" in our experiments.

SPECjbb is a server-side benchmark that is generally more memory-intensive than TPC-W. The benchmark emulates a three-tier web application, and particularly stresses the middle application server tier when executing the test suite.

All nested VMs run the same benchmark with the same 30 second time bound for bounded-time migration, which we choose conservatively to be significantly lower than the two minute warning provided by EC2. Thus, our cost and availability results are worse than possible if using a more liberal time bound closer to the two minute warning time. In our experiments, we compare SpotCheck against i) Xen's pre-copy live migration, ii) an unoptimized bounded-time VM migration that fully restores a nested VM before starting it (akin to Yank [30]), (iii) SpotCheck's optimized Full restore, iv) an unoptimized bounded-time VM migration that uses lazy restoration, and finally v) SpotCheck's optimized bounded-time VM migration with lazy restoration.

7.1 End-to-End Experiments

SpotCheck uses a backup server to checkpoint VM state and support bounded-time VM migration. SpotCheck's cost overhead is primarily a function of the number of VMs each backup server multiplexes: the more VMs it multiplexes on a backup server, the lower its cost. Figure 7 shows the effect on nested VM performance for SpecJBB and TPC-W as the load on the backup server increases. First, we evaluate the overhead of continuously checkpointing memory and sending it over the network to the backup server. The "0" and "1" columns in Figure 7 represent performance difference between no checkpointing and checkpointing using a dedicated

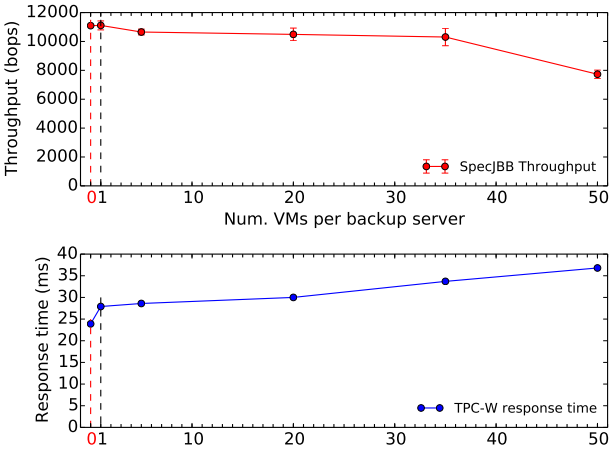
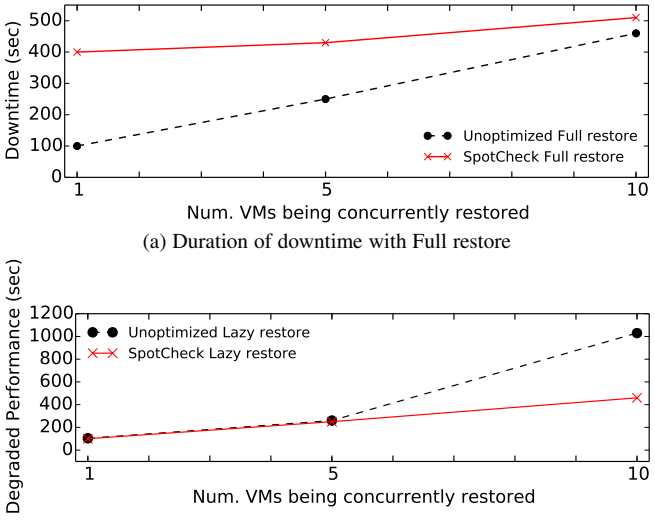


Fig. 7. Effect on performance as the number of nested VMs backing up to a single backup server increases.

backup server, respectively. By simply turning checkpointing on and using a dedicated backup server, we see that TPC-W experiences a 15% increase in response time, while SpecJBB experiences no noticeable performance degradation during normal operation. With an increasing number of nested VMs all backing up to a single server, saturation of the disk and network bandwidth on the backup server leads to a decrease in nested VM performance after 35 VMs, where SpecJBB throughput decreases and TPC-W response time increases significantly, e.g., by roughly 30% each. Note that the nested VM incurs this performance degradation as long as it is running on a spot server. Thus, to ensure minimal performance degradation during normal operation, SpotCheck assigns at most 35-40 VMs per backup server. As a result, SpotCheck's cost overhead for backing up each nested VM is roughly $1/40 = 2.5\%$ of the price of a backup server. For our `m3.xlarge` backup server, which costs $0.28 per hour in the East region of EC2, the amortized cost per-VM across 40 nested VMs is $0.007 or less than one cent per VM.

In addition to performance during normal operation, spot server revocations and the resulting nested VM migrations and restorations impose additional load on the backup server. Figure 8 shows the length of the period of downtime or performance degradation when migrating nested VMs via the backup server. In this case, we compare migrations that utilize lazy restoration with those that use a simple stop-and-copy migration. A stop-and-copy approach results in high *downtime*, whereas a lazy restore approach results in much less downtime but some *performance degradation* when memory pages must be fetched on-demand across the network on their first access. Since lazy restore incurs less downtime, it reduces the effect of migrations on interactive applications. Figure 8 shows that when concurrently restoring 1 and 5 nested VMs the time required to complete the migration is similar for both lazy restoration and stop-and-copy migration, which results in performance degradation or downtime, respectively, over the time window.

However, when executing 10 concurrent restorations, the length of the lazy restoration is much longer than that of the stop-and-copy migration. This occurs because lazy restoration uses random reads that benefit less from prefetching and caching optimizations than a stop-and-copy migration, which uses sequential reads. This motivates SpotCheck's lazy restoration optimization that uses the `fadvise` system call to inform the kernel how SpotCheck will use the VM memory images stored on disk, e.g., to expect references in random order in the near future. The optimization results in a



(a) Duration of downtime with Full restore

(b) Duration of degraded performance with Lazy restore

Fig. 8. Duration of downtime during a traditional VM restore, and performance degradation during a lazy restore.

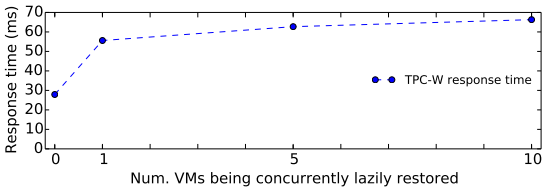


Fig. 9. Effect of lazy restoration on VM performance.

significant decrease in the restoration time for lazy restore. Thus, SpotCheck's optimizations significantly reduce the length of the period of performance degradation during lazy restorations. Of course, SpotCheck also assigns VMs to backup servers to reduce the number of revocation storms that cause concurrent migrations.

In addition to the time to complete a migration, SpotCheck also attempts to mitigate the magnitude of performance degradation during a migration and lazy VM restoration. During the lazy restoration phase the VM experiences some performance degradation, which may impact latency-sensitive applications, such as TPC-W. Since the first access to each page results in a fault that must be serviced over the network, lazy restoration may cause a temporary increase in application response time. Figure 9 shows TPC-W's average response time as a function of the number of nested VMs being concurrently restored, where zero represents normal operation. The graph shows that when restoring a single VM the response time increases from 29ms to 60ms for the period of the restoration. Additional concurrent restorations do not significantly degrade performance, since SpotCheck partitions the available bandwidth equally among nested VMs to ensure restoring one VM does not negatively affect the performance of VMs using the same backup server. Note that SpotCheck's policies attempt to minimize the number of evictions and migrations via pool management, and thus the performance degradation of applications *during the migration process* is a rare event. Even so, our evaluation above shows that application performance is not adversely affected even when the policies cannot prevent migrations.

Result: *SpotCheck executes nested VMs with little performance degradation and cost overhead during normal operation using a high VM-to-backup ratio and migrates/restores them with only a brief period of performance degradation.*

TABLE 2
SpotCheck's customer-to-pool mapping policies.

Policy	Description
1P-M	VMs mapped to a single m3.medium pool
2P-ML	VMs equally distributed between two pools : `m3.medium` and `m3.large`.
4P-ED	VMs equally distributed to four pools consisting of four `m3` server types
4P-COST	VMs distributed based on past prices. The lower the cost of the pool over a period, the higher the probability of mapping a VM into that pool.
4P-ST	VMs distributed based on number of past migrations. The fewer the number of migrations over a period, the higher the probability of mapping a VM into that pool.

7.2 SpotCheck Policies and Cost Analysis

As we discuss in Section 4, SpotCheck may employ a variety of bidding and VM assignment policies that tradeoff performance and risk. Here, we evaluate SpotCheck's cost using various bidding policies based on the EC2 spot price history from April 2014 to October 2014. In particular, Table 2 describes the policies we use to assign VMs to spot pools. The simplest policy is to place all VMs on servers from a single spot market (1P-M); this policy minimizes costs if SpotCheck selects the lowest price pool, but increases risk, since it may need to concurrently migrate all VMs if a price spike occurs. We examine two policies (2P-ML and 4P-ED) that distribute VMs across servers from different spot markets to reduce risk, albeit at a potentially higher cost. We also examine two policies (4P-COST and 4P-ST) that probabilistically select pools based on their weighted historical cost, or their weighted historical price volatility. The former lowers cost, while the latter reduces performance degradation from frequent migrations.

Figure 10 shows SpotCheck's average cost per hour when using each policy. As expected, the average cost for running a nested VM using live migration, i.e., without a backup server, is less than the average cost using SpotCheck, since live migration does not require a backup server. Of course, using only live migration is not practical, since, without a backup server, SpotCheck risks losing VMs before a live migration completes. In this case, 1P-M has the lowest average cost, since SpotCheck maps VMs to the lowest priced spot pool. Distributing VMs across two (2P-ML) and then four (4P-ED) pools marginally increases costs. The two policies that probabilistically select pools based on either their historical cost or volatility have roughly the same cost as the policy that distributes across all pools. Note that the average cost SpotCheck incurs for the equivalent of an `m3.medium` server type is ∼$0.015 per hour, while the cost of an `m3.medium` on-demand server type is $0.07, or a savings of nearly 5×.

SpotCheck runs VMs using nested virtualization, which can have significant performance impact. Xen-blanket's performance overhead (relative to single-level virtualization) is highly workload dependent [18], [35], and ranges from 0-68% depending on whether the workload is CPU or I/O bound. Thus, the performance of some SpotCheck VMs is reduced to about half of the native cloud performance. In case the VM's performance is adversely affected by nested virtualization, we normalize the cost savings relative to the performance. Thus for a VM whose performance reduces by 50%, the cost savings are also decreased by 50%—this still yields *worst-case* cost savings of about 2.5× compared to on-demand instances. We note that nested hypervisors are still nascent and performance optimizations can reduce these overheads.

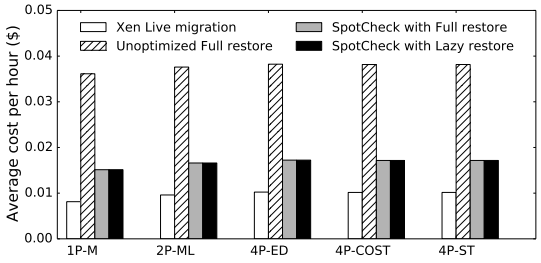


Fig. 10. Average cost per VM under various policies.

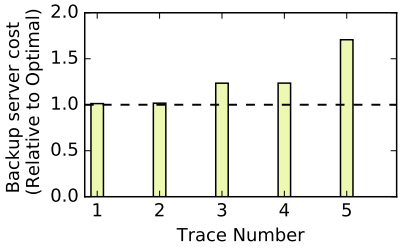


Fig. 11. Cost of backup servers relative to optimal packing for the Eucalyptus cloud traces.

The cost of SpotCheck VMs also depends on the utilization of the backup servers, since the backup server costs are shared by all the VMs. Due to the dynamic arrival and lifetimes of VMs, SpotCheck's *online* backup server policy may leave backup servers under-utilized, and thus increase effective costs. We evaluate the costs of backup servers using the Eucalyptus cloud workload trace [7]. Figure 11 shows the backup server costs of SpotCheck's backup-server allocation policy relative to the optimal bin-packing policy which minimizes the number of backup servers. We can see from Figure 11 that the increase in backup-server costs (compared to the optimal) ranges from 2% to 65% (for the short trace #5). This translates to a per-VM cost increase of 1-33% compared to the full utilization scenario. Taking under-utilization of backup servers into account, the worst-case cost savings for SpotCheck is still more than 2× compared to the on-demand instances.

While reducing cost is important, maximizing nested VM availability and performance by minimizing the number of migrations is also important. Here, we evaluate the unavailability of VMs due to spot server revocations. For these experiments, we assume a period of performance degradation due to detaching and reattaching EBS volumes, network reconfiguration, and migration. We seed our simulation with measurements from Table 1 and the microbenchmarks from the previous section. In particular, we assume a downtime of 23 seconds per migration due to the latency of EC2 operations. Based on these values and the spot price history, Figure 12 shows nested VM unavailability as a percentage over the six month period from April to October for each of our policies. As above, we see that live migration has the lowest unavailability, since it incurs almost no downtime, but is not practical, since it risks losing VM state. We also examine both an unoptimized version of bounded-time VM migration requiring a full restoration before resuming (akin to Yank) and our optimized version that also requires a full restoration. The graph shows that the optimizations in Section 5 increase the availability. The graph also shows that, even without lazy restoration, SpotCheck's unavailability is below 0.25% in all cases, or an availability of 99.75%.

However, we see that using lazy restore brings SpotCheck's unavailability close to that of live migration. Since the `m3.medium`

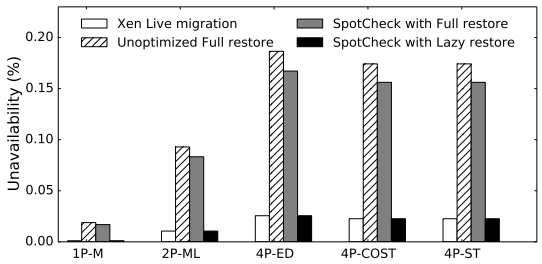


Fig. 12. Unavailability for live migration and SpotCheck (with and without optimizations and lazy restore).

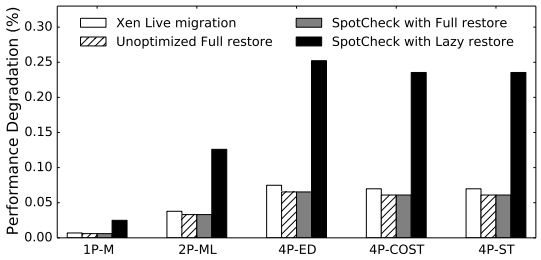


Fig. 13. Performance degradation during migration.

spot prices over our six month period are highly stable, the 1P-M policy results in the highest availability of 99.9989%, as well as the lowest cost from above. This level of availability is roughly 10× that of directly using spot servers, which, as Figure 6(a) shows, have an availability between 90% and 99%. The other policies exhibit slightly lower availability ranging from 99.91% for 2P-ML to 99.8% for 4P-ED. In addition to availability, performance degradation is also important. Figure 13 plots the percentage of time over the six month period a nested VM experiences performance degradation due to a migration and restoration. The graph shows that, while SpotCheck with lazy restoration has the most availability, it has the longest period of performance degradation. However, for the single pool 1P-M policy, the percentage of time the nested VM operates under degraded performance is only 0.02%, while the maximum length of performance degradation (for 4P-ED) is only 0.25%. For perspective, over a six month period, SpotCheck using the 1P-M policy has only 2.85 combined minutes of degraded performance due to migrations and restorations.

Result: *SpotCheck achieves nearly 2-5× savings compared to using an equivalent on-demand server from an IaaS platform, while providing 99.9989% availability with migration-related performance degradation only* 0.02% *of the time.*

The cost-risk tradeoff between choosing a *single pool* versus *two pools* versus *four pools* is not obvious. While, in the experiments above, 1P-M provides the lowest cost and the highest availability, the risk of SpotCheck having to concurrently migrate all nested VMs at one time is high, since all VMs mapped to a backup server are from a single pool. For the six month period we chose, the spot price in the `m3.medium` pool rarely rises above the on-demand price, which triggers the migrations and accounts for its high availability. The other policies mitigate this risk by increasing the number of pools by distributing the VMs across these pools. Since the price spikes in these pools are not correlated, the risk of losing all VMs at once is much lower. Table 3 shows the probability of concurrent revocations of various sizes as a factor of the total number of VMs N. We note that the probability of all N VMs migrating in a single pool scenario is higher compared to the two-pool scenario and nearly non-existent in the case of the four-pool policy. Also, by distributing VMs across pools, SpotCheck increases the overall frequency of migration, but reduces the number of mass migrations.

Result: *Distributing nested VMs mapped to each backup server across pools lowers the risk of large concurrent migrations. For example, comparing 1P-M to 4P-ED, the average VM cost in 4P-ED increases by $0.002 and the availability reduces by 0.15%, but the approach avoids all mass revocations.*

Pool selection policy comparison. Our results demonstrate that each of SpotCheck's server pool selection policies provide similar cost savings (Figure 10) and availability (Figure 12). Performance degradation is lowest for single-pool policy (1P-M), but negligible even for the worst-performing policy (4P-ED as shown in Figure 13), while the four-pool policies drastically reduce the risk of mass migration events (from Table 3).

TABLE 3
Probability of the maximum number of concurrent revocations for different pools. N is the number of VMs.

	Max. num. of concurrent revocations			
	N/4	N/2	3N/4	N
1-Pool	0	0	0	1.74×10^{-4}
2-Pool	0	3.75×10^{-3}	0	2.25×10^{-5}
4-Pool	7.4×10^{-3}	7.71×10^{-5}	1.92×10^{-5}	0

7.3 Comparison of Risk Mitigation Policies

Two-level Bidding: To evaluate the impact of two-level bidding, we use pick two bids and observe the impact on the revocations and the expected cost. For two-level bidding, the low-bid is the on-demand price. Thus we keep the low-bid equal to the on-demand price and vary the high-bid. We are interested in comparing with the single-level bidding policy, and keep all parameters such as the workload and other policies constant. The impact of two-level bidding is shown in Figure 14, which shows the decrease in revocation storms and increase in cost vs. the high-bid. As the high-bid increases, the fraction of revocation storms which are mitigated (only low-bid servers affected) increases upto a limit, after which it starts to flatten out. Correspondingly, the cost also increases because of the higher bids. For the `m1.2xlarge` instance, the two-level bidding strategy can mitigate almost 60% of revocation storms with a 20% increase in cost. That is, whenever a revocation event occurs, it will only affect half of the servers 60% of the time. Thus, two-level bidding can be an effective strategy to increase the number of effective pools and mitigate revocation storms.

Backup Selection: The backup server selection policies determine the load on the backup server during revocations, for which we measure the number of concurrent revocations faced by each backup server. During a revocation, the backup server is faced with increased checkpointing frequency and must provide pages to the lazy restoration process. An overloaded backup server servicing a large number of lazy restorations is detrimental to smooth migrations. Accordingly, we compare the different backup selection policies in terms of the number of concurrent revocations in Table 4. For different pool management policies, the impact of backup selection varies, because the number of pools determines the "spread" of VMs. We compare the online-greedy policy with the default round-robin policy. When using a single pool, there is a slight reduction in the number of concurrent revocations with the online-greedy policy (2%), whereas the reduction is 18% with 4 pools. Thus, the online-greedy backup selection policy reduces the concurrent revocation load on the backup servers.

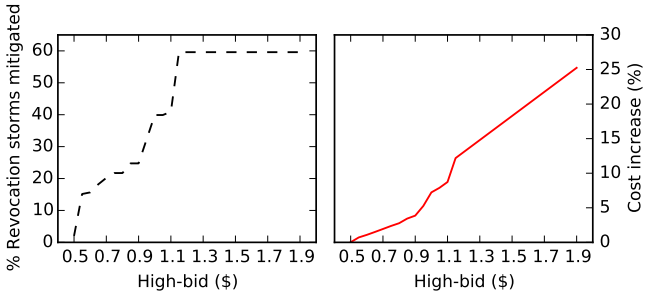


Fig. 14. Two-level bidding reduces impact of revocation storms because only half the servers are affected. As the high-bid is increased, the percentage of storms mitigated increases upto a limit, and so does the cost (compared to single-level bidding).

TABLE 4
Reduction in max number of concurrent revocations with the online greedy backup server selection, when compared to round-robin.

Num. Spot Pools	Reduction in concurrent revocations
1 Pool	2%
2 Pool	13.7%
4 Pool	18.3%

Hot spares: Hot spares are readily available, already running on-demand servers used to migrate VMs upon a revocation. Hot spares reduce the downtime during migration, but incur an additional cost, which is shown in Table 5. For different policies, the number of simultaneous revocations (size of revocation storm) affects the hot spare cost, and is lower when the number of markets is larger, and also when two-level bidding is employed. With a single pool, having 10% servers as hot spares results in a 50% increase in expected cost, whereas the overhead of hot spares is only 15% when using 2 pools and 2-level bidding.

VPC: The cost of running in the private cloud mode is higher than the default shared mode of operation because the backup server cost is not shared by a larger number of VMs. The cost of running VMs in the private cloud mode is shown in Figure 15. As the private cluster size increases, the cost decreases because the multiplexing of backup servers increases. SpotCheck is able to select smaller backup servers for smaller number of VMs, and cost of running 5 VMs is 40% higher per VM when compared to the default shared-everything mode.

8 Related Work

Designing Derivative Clouds. Prior work on interclouds [12] and superclouds [23], [36] propose managing resources across multiple IaaS platforms by using nested virtualization [10], [35], [41] to provide a common homogeneous platform. While SpotCheck also leverages nested virtualization, it focuses on exploiting it to transparently reduce the cost and manage the risk of using revocable spot servers on behalf of a large customer base. Our current prototype does not support storage migration or inter-cloud operation; these functions are the subject of future work. Cloud Service Brokers [28], such as RightScale [6], offer tools that aid users in aggregating and integrating resources from multiple IaaS platforms, but without abstracting the underlying resources like SpotCheck. PiCloud [5] abstracts spot and on-demand servers rented from IaaS platforms by exposing an interface to consumers that allows them to submit batch jobs. In contrast, SpotCheck provides the abstraction of a complete IaaS platform that supports any application. Finally, SpotCheck builds on a long history of

TABLE 5
Percentage increase in cost due to hot spares

Policy	Price Increase
1 Pool	50%
2 Pool	25%
1 Pool 2-level bidding	30%
2 Pool 2-level bidding	15%

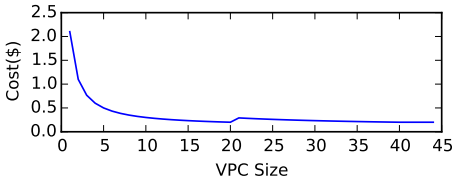


Fig. 15. Cost of running in private cloud mode for different sizes.

research in market-based resource allocation [13], which envisions systems with a fluid mapping of software to hardware that enable computation and data to flow wherever prices are lowest.

Spot Market Bidding Policies. Prior work on optimizing bidding policies for EC2 spot instances are either based on analyses of spot price history [11], [20], [37] or include varying assumptions about application workload, e.g., job lengths, deadlines [26], [32], [33], [39], [40], which primarily focus on batch applications. By contrast, SpotCheck's bidding strategy focuses on reducing the probability of mass revocations due to spot price spikes, which, as we discuss, may significantly degrade nested VM performance.

Virtualization Mechanisms. Prior work handles the sudden revocation of spot servers either by checkpointing application state at coarse intervals [21], [34], [38] or eliminating the use of local storage [14], [24]. In some cases, application modifications are necessary to eliminate the use of local storage for storing intermediate state, e.g., MapReduce [14], [24]. SpotCheck adapts a recently proposed bounded-time VM migration mechanism [30], [31],which is based on Remus [17] and similar to microcheckpointing [1], to aggressively checkpoint memory state and migrate nested VMs away from spot servers upon revocation. Our lazy restore technique is similar to migration mechanisms, such as post-copy live migration [19] and SnowFlock [22].

9 Conclusion

SpotCheck is a derivative IaaS cloud that offers low-cost, high-availability servers using cheap but volatile spot servers from the native IaaS platform. In this paper, we showed that revocation risk of using spot servers can be reduced by using policies for bidding, server-selection, backup-servers, and hot-spares. By using a combination of these risk mitigation policies, along with novel virtualization-based bounded time live-migration mechanism, SpotCheck is able to provide more than four 9's availability to its customers, which is more than 10× that provided by the native spot servers. At the same time, SpotCheck's VMs cost 2-5× less than the equivalent on-demand servers.

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

References

[1] QEMU Microcheckpointing. http://wiki.qemu.org/Features/MicroCheckpointing.
[2] SPECjbb2005. https://www.spec.org/jbb2005/.
[3] TPC-W Benchmark. http://jmob.ow2.org/tpcw.html.
[4] Heroku. http://www.heroku.com, May 1st 2014.
[5] PiCloud. http://www.multyvac.com, May 1st 2014.
[6] RightScale. http://rightscale.com, May 1st 2014.

[7] Eucalyptus workload traces. https://www.cs.ucsb.edu/~rich/workload/, 2015.

[8] AWS Case Study: Netflix. http://aws.amazon.com/solutions/case-studies/netflix.

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

[10] M. Ben-Yehuda, M. Day, Z. Dubitzky, M. Factor, N. Har'El, A. Gordon, A. Liguori, O. Wasserman, and B. Yassour. The Turtles Project: Design and Implementation of Nested Virtualization. In *OSDI*, October 2010.

[11] O. Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir. Deconstructing Amazon EC2 Spot Instance Pricing. In *CloudCom*, 2011.

[12] D. Bernstein, E. Ludvigson, K. Sankar, S. Diamond, and M. Morrow. Blueprint for the Intercloud - Protocols and Formats for Cloud Computing Interoperability. In *ICIW*, 2009.

[13] A. Bestavros and O. Krieger. Toward an Open Cloud Marketplace: Vision and First Steps. *IEEE Internet Computing*, 18(1), January/February 2014.

[14] N. Chohan, C. Castillo, M. Spreitzer, M. Steinder, A. Tantawi, and C. Krintz. See Spot Run: Using Spot Instances for MapReduce Workflows. In *HotCloud*, June 2010.

[15] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live Migration of Virtual Machines. In *NSDI*, 2005.

[16] J. Clark. Amazon Cloud Goes Down in Northern Virginia. The Register, September 13th 2013.

[17] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield. Remus: High Availability via Asynchronous Virtual Machine Replication. In *NSDI*, April 2008.

[18] X. He, P. Shenoy, R. Sitaraman, and D. Irwin. Cutting the cost of hosting online services using cloud spot markets. In *High-Performance Parallel and Distributed Computing*. ACM, 2015.

[19] M. R. Hines, U. Deshpande, and K. Gopalan. Post-copy Live Migration of Virtual Machines. *SIGOPS Operating Systems Review*, 43(3), July 2009.

[20] B. Javadi, R. Thulasiram, and R. Buyya. Statistical Modeling of Spot Instance Prices in Public Cloud Environments. In *UCC*, December 2011.

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

[22] H. A. Lagar-Cavilla, J. A. Whitney, A. M. Scannell, P. Patchin, S. M. Rumble, E. De Lara, M. Brudno, and M. Satyanarayanan. SnowFlock: Rapid Virtual Machine Cloning for Cloud Computing. In *EuroSys*, 2009.

[23] C. Liu and Y. Mao. Inception: Towards a Nested Cloud Architecture. In *HotCloud*, June 2013.

[24] H. Liu. Cutting MapReduce Cost with Spot Market. In *HotCloud*, 2011.

[25] M. Mao and M. Humphrey. A Performance Study on VM Startup Time in the Cloud. In *CLOUD*, June 2012.

[26] M. Mattess, C. Vecchiola, and R. Buyya. Managing Peak Loads by Leasing Cloud Infrastructure Services from a Spot Market. In *HPCC*, September 2010.

[27] K. Mills, J. Filliben, and C. Dabrowski. Comparing vm-placement algorithms for on-demand clouds. In *CLOUDCOM*. IEEE, 2011.

[28] D. Plummer. Cloud Services Brokerage: A Must-Have for Most Organizations. Forbes, March 22nd 2012.

[29] P. Sharma, S. Lee, T. Guo, D. Irwin, and P. Shenoy. Spotcheck: Designing a Derivative IaaS Cloud on the Spot Market. In *EuroSys*, 2015.

[30] R. Singh, D. Irwin, P. Shenoy, and K. Ramakrishnan. Yank: Enabling Green Data Centers to Pull the Plug. In *NSDI*, April 2013.

[31] R. Singh, P. Sharma, D. Irwin, P. Shenoy, and K. Ramakrishnan. Here Today, Gone Tomorrow: Exploiting Transient Servers in Data Centers. *IEEE Internet Computing*, 18(4), July/August 2014.

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

[33] S. Tang, J. Yuan, and X. Li. Towards Optimal Bidding Strategy for Amazon EC2 Cloud Spot Instance. In *CLOUD*, June 2012.

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

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

[36] D. Williams, H. Jamjoom, and H. Weatherspoon. Plug into the Supercloud. *IEEE Internet Computing*, 17(2), 2013.

[37] H. Xu and B. Li. A Study of Pricing for Cloud Resources. *Performance Evaluation Review*, 40(4), March 2013.

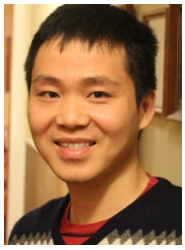
[38] S. Yi, D. Kondo, and A. Andrzejak. Reducing Costs of Spot Instances via Checkpointing in the Amazon Elastic Compute Cloud. In *CLOUD*, 2010.

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

[40] S. Zaman and D. Grosu. Efficient Bidding for Virtual Machine Instances in Clouds. In *CLOUD*, July 2011.

[41] F. Zhang, J. Chen, H. Chen, and B. Zang. CloudVisor: Retrofitting Protection of Virtual Machines in Multi-tenant Cloud with Nested Virtualization. In *SOSP*, October 2011.

Prateek Sharma is a PhD student in the College of Information and Computer Sciences at the University of Massachusetts Amherst. His current research focuses on Cloud Computing, Operating Systems, and Virtualization. Sharma received his masters degree in computer science from Indian Institute of Technology, Bombay. Contact him at prateeks@cs.umass.edu.



Stephen Lee is a PhD student in the College of Information and Computer Sciences at the University of Massachusetts Amherst. His research interests include green computing and sustainability in smart buildings. Lee received a masters degree in computer from Chennai Mathematical Institute, India. Contact him at stephenlee@cs.umass.edu.

Tian Guo is an Assistant Research Professor in the Computer Science Department at Worcester Polytechnic Institute. She received her Ph.D. and M.S. in Computer Science from the University of Massachusetts Amherst in 2013 and 2016, respectively, and her B.E. in Software Engineering from Nanjing University in 2010. Her research interests include distributed systems, cloud computing and mobile computing. Contact her at tian@cs.wpi.edu.

David Irwin is an Assistant Professor in the Department of Electrical and Computer Engineering at the University of Massachusetts Amherst. He received his Ph.D. and M.S. in Computer Science from Duke University in 2007 and 2005, respectively, and his B.S. in Computer Science and Mathematics from Vanderbilt University in 2001. His research interests are broadly in experimental computing systems with a particular emphasis on sustainability.

Prashant Shenoy received the B.Tech degree in computer science and engineering from the Indian Institute of Technology, Bombay, in 1993, and the M.S and Ph.D. degrees in computer science from the University of Texas, Austin, in 1994 and 1998, respectively. He is currently a Professor of Computer Science at the University of Massachusetts. His current research focuses on cloud computing and green computing. He is a distinguished member of the ACM and a Fellow of the IEEE.