SpotCheck: Designing a Derivative IaaS Cloud on the Spot Market

Prateek Sharma Stephen Lee Tian Guo David Irwin Prashant Shenoy
University of Massachusetts Amherst
{prateeks,stephenlee,tian,shenoy}@cs.umass.edu irwin@ecs.umass.edu

Abstract

Infrastructure-as-a-Service (IaaS) cloud platforms rent resources, in the form of virtual machines (VMs), under a variety of contract terms that offer different levels of risk and cost. For example, users may acquire VMs in the *spot market* that are often cheap but entail significant risk, since their price varies over time based on market supply and demand and they may terminate at any time if the price rises too high. Currently, users must manage all the risks associated with using spot servers. As a result, conventional wisdom holds that spot servers are only appropriate for delay-tolerant batch applications. In this paper, we propose a derivative cloud platform, called SpotCheck, that transparently manages the risks associated with using spot servers for users.

SpotCheck provides the illusion of an IaaS platform that offers always-available VMs on demand for a cost near that of spot servers, and supports all types of applications, including interactive ones. SpotCheck's design combines the use of nested VMs with live bounded-time migration and novel server pool management policies to maximize availability, while balancing risk and cost. We implement SpotCheck on Amazon's EC2 and show that it i) provides nested VMs to users that are 99.9989% available, ii) achieves nearly 5× cost savings compared to using equivalent types of on-demand VMs, and iii) eliminates any risk of losing VM state.

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 their 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 limit. Since spot servers incur a risk of unexpected resource loss, they offer weaker availability guarantees than on-demand servers and tend to be cheaper.

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 that enables customers to submit compute tasks. PiCloud charges customers for their compute time, and is able to offer lower prices than on-demand servers by using cheaper spot servers to execute

EuroSys'15, April 21–24, 2015, Bordeaux, France.
Copyright © 2015 ACM 978-1-4503-3238-5/15/04…$15.00.
http://dx.doi.org/10.1145/2741948.2741953

compute tasks. Similarly, Heroku [4] offers a Platform-as-a-Service by repackaging and reselling IaaS resources as containers. As with PiCloud, Heroku constrains the user's programming model—in this case, to containers.

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. Unlike the examples above, SpotCheck does not constrain the programming model by offering 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 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, our algorithms intelligently map customers to multiple pools of spot and on-demand servers of different types, and handle pool dynamics caused by sudden revocations of spot servers or significant price changes.

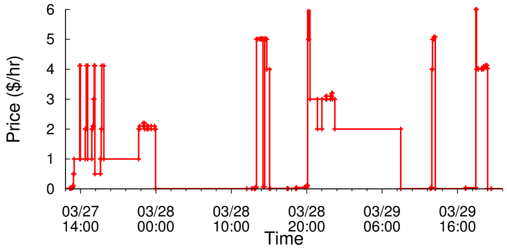


Figure 1. Spot price of the `m1.small` server type in EC2 fluctuates over time and can rise significantly above the on-demand price ($0.06 per hour) during price spikes. Note that the *y*-axis is denominated in dollars and not cents.

Implementation and Evaluation. We implement SpotCheck on Amazon's Elastic Compute Cloud (EC2) and evaluate its migration mechanisms and pool management algorithms. 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 server resources 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 service 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 [26]. 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

[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 [21].

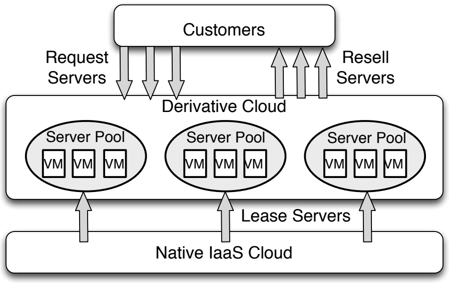


Figure 2. A depiction of a derivative IaaS cloud platform.

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 6 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 customer VMs from one server 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 from the perspective of migrating an individual VM from one native cloud server to another. There are a variety of reasons why such a migration may be necessary or desirable—the native IaaS platform may force a migration by revoking the underlying spot server, or a cheaper spot server may become available, which incentivizes migrating a VM running on a more expensive on-demand server to it. Regardless of the reason, SpotCheck combines several virtualization mechanisms to implement its migration strategies.

3.1 Nested Virtualization

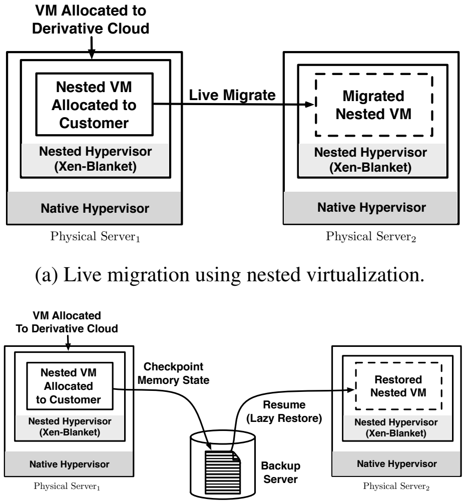
SpotCheck rents VMs from native IaaS platforms that do not expose all of the functionality of the VM hypervisor. For example, EC2 allocates VMs to its customers, but does not expose control over VM placement or support VM migration to different physical servers. To address this limitation, SpotCheck uses *nested virtualization*, where a nested hypervisor runs atop a traditional VM, which itself runs on a conventional hypervisor [11, 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(a). Nested hypervisors provide a uniform and standard platform for repackaging and reselling virtualized server resources. Nested VMs currently provide paravirtualized I/O devices and hide advanced features, such as SR-IOV [7], which may reduce I/O performance. However, as with the native VM platforms, we expect nested VM technology to continue to improve.

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. SpotCheck could also use lighter-weight mechanisms, such as resource containers [9], to isolate partitions of virtualized resources. We chose to use nested VMs in our prototype because the current resource container implementations, e.g., Linux Containers and Docker, do not support the advanced migration features that SpotCheck requires.

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, as depicted in Figure 3(a). Prior work proposes a variety of live VM migration mechanisms and optimizations, such as the pre-copy [16] and post-copy [20] migration variants.

In general, the total latency to live migrate a VM, whether nested or not, is proportional to the size of the VM's memory. Thus, larger VMs with tens of gigabytes of RAM may take several minutes, while smaller VMs with a few giga-



(a) Live migration using nested virtualization.

(b) Memory checkpointing and restoration when using a bounded-time VM migration.

Figure 3. SpotCheck uses live and bounded-time VM migration to migrate nested VMs within an IaaS platform.

bytes of RAM may take tens of seconds. In addition to memory size, the write (or dirtying) rate of memory pages, which depends on application characteristics, also influences live migration latency. As a result, live VM migration is not suitable in all of SpotCheck's migration scenarios. In particular, 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. While the 120 second warning has always been a well-known hidden feature of spot servers, Amazon publicly acknowledged it in January 2015 and now supports official 120 second termination notices for spot servers through its external web services API [10]. Importantly, if the latency to live migrate a VM exceeds the warning period, as it often does with large memory sizes, then the IaaS platform will terminate the spot server and any resident nested VMs before their migrations complete, resulting in the loss of memory state at best and VM failure at worst.

In this scenario, 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 dirtying rate of memory pages. 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(b).

SpotCheck adapts and applies both live [16] and bounded-time VM migration [30, 31] to nested VMs. Depending on the scenario, SpotCheck uses the most appropriate technique for VM migration. 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. SpotCheck then voluntarily relinquishes the native VM as soon as the migration completes. 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. The only exception is for “small" nested VMs that do not use much memory, such that a live migration is able to reliably complete within a spot server's warning period, e.g.,120 seconds for EC2.

Of course, the shorter the warning period, the smaller the nested VM memory size that cannot use a conventional live migration and will require a bounded-time VM migration. SpotCheck may also perform *proactive* migrations from a spot server if it predicts that a revocation is imminent. In this case, the system has less stringent time constraints on the migration latency, since it triggers the migration before the IaaS platform explicitly signals a revocation. Such predictive approaches make it feasible to employ live migration with spot servers and avoid the overhead and complexity of bounded-time VM migration, which requires continually backing up memory state to a remote disk. However, such optimizations incur significant risk of losing VM state unless they are able to predict an imminent revocation with high confidence, e.g., by tracking and predicting a rise in market prices of spot servers that causes revocations.

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 nested VM memory state. As we show in our experiments in Section 6, 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 to customers, SpotCheck employs lazy VM restoration, as proposed in prior work [20, 23], to reduce the downtime to nearly zero. Lazy VM restoration involves reading a small number of initial VM memory pages—the *skeleton*

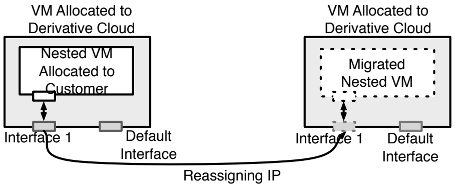

VM Allocated to Derivative Cloud
Nested VM Allocated to Customer
Interface 1
Default Interface
VM Allocated to Derivative Cloud
Migrated Nested VM
Interface 1
Default Interface
Reassigning IP

Figure 4. SpotCheck migrates the network interface of nested VMs from the source to the destination host using VPN functions provided by the underlying IaaS platform.

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. A background process also runs in parallel and proactively reads memory pages into RAM to reduce the frequency of page faults. 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 [16]. 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 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.

While many IaaS platforms still treat IP address creation and assignment as privileged operations, a few platforms, including EC2, have introduced virtual private networking (VPN) functions to provide users control over their own private IP address space. EC2 supports VPNs through its Virtual Private Cloud (VPC) feature, which enables users to directly assign IP addresses to their VMs. SpotCheck creates a VPC and places all of its spot and on-demand servers into it. As a result, SpotCheck is able to create a private IP address for each nested VM. Upon migration, SpotCheck uses available VPC functions to deallocate the IP address associated with a nested VM on its source server, and reassign it to a new (unused) network interface on the destination server, as depicted in Figure 4. This ensures the IP address of nested VMs remains unchanged after migration. SpotCheck currently allocates a subnet within a shared data plane, defined by the VPC, to each customer. By default, SpotCheck assigns one public IP address per customer, attached to a designated "head" nested VM, to provide access to the public Internet from within the private VPC subnet.

3.5 Putting it all together

SpotCheck combines nested virtualization, virtual private networks, VM migration, and lazy VM restoration to implement its migration strategies, as summarized below. Upon initial allocation, SpotCheck assigns a backup server to each nested VM on a spot server, which stores its memory state, unless the nested VM's memory size is small enough to ensure a live migration completes within the warning period. SpotCheck might also not assign a backup server if it decides to migrate nested VMs proactively in advance of a revocation. Nested VMs hosted on on-demand servers do not require a backup server, since they are always capable of a live migration. If the underlying IaaS platform revokes a spot server, SpotCheck must migrate each resident 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, discussed in Section 4. 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 5. If SpotCheck employs bounded-time VM migration, it uses lazy VM restoration to minimize the migration downtime.

4. Server Pool Management

SpotCheck rents VM servers from native IaaS platforms under different service contracts that specify different levels of price and availability, and then repackages and resells their resources to its customers. The ability to rent and manage servers of different types, and intelligently multiplex their resources across multiple customers is central to the design of any derivative cloud, including SpotCheck. Note that, similar to traditional virtualization, nested virtualization enables multiple nested VMs to run on a host VM, such that the

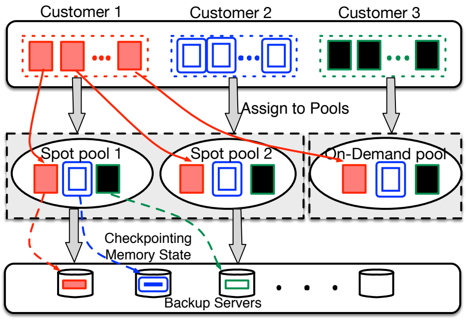


Figure 5. SpotCheck's architecture using multiple pools.

nested hypervisor in the host VM isolates the nested VMs and prevents cross-VM attacks. As with a native IaaS platform, SpotCheck controls the nested hypervisor and has full access to the memory state of each of its customer's nested VMs. In this section, we describe the techniques SpotCheck uses to manage resources from multiple pools of servers.

4.1 SpotCheck Architecture

At an architectural level, SpotCheck 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, comprising spot and on-demand servers of the same type, respectively. 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. 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. Given the architecture above, we next describe the techniques and algorithms SpotCheck employs to manage server pools and handle pool dynamics.

4.2 Mapping Customers to Pools and Pools to Backups

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. As a result, the price of a spot server that is two or four times the size of a requested nested VM may be less (or more) than two to four times the price of a smaller spot server of the requested type.

By default, SpotCheck uses 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. Thus, if a customer requests a medium server, and the price of a large server, which can be sliced into two medium-sized nested VMs, is less than twice the spot price of a medium server, then SpotCheck allocates a large spot server from the native IaaS platform. SpotCheck then allocates one nested VM to satisfy the current customer request, and reserves the additional slot in order to rapidly allocate a medium VM to fulfill a subsequent customer request. In contrast, if market conditions are such that medium spot servers are the cheapest option, SpotCheck directly allocates them to satisfy customer requests for 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 satisfy a customer request. The more volatile the prices of a particular spot server type, the greater the chance of a price spike, and the higher the frequency of revocations. 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.

Regardless of the actual policy to determine what type of native server to use to satisfy a customer request, SpotCheck spreads the nested VMs belonging to each of its customers across multiple different server pools. A revocation event due to a price spike for a particular type of spot server can cause many 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, distributing a customer's nested VMs across multiple pools decreases risk by reducing the probability of a revocation storm. Revocation storms degrade nested VM performance and increase downtime by overloading backup servers, which must simultaneously broker the migration of every revoked nested VM. This policy is akin to managing a financial portfolio by distributing assets across uncorrelated, independent asset classes to reduce risk. 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)

Finally, 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. 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. SpotCheck employs a simple round-robin policy to map nested VMs within each pool across the set of backup servers. Once every 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. Of course, a backup server is not necessary for running stateless services on nested VMs, 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 the applications that run on it. This does mean that SpotCheck may incur slightly higher costs than necessary for stateless services, since these servers can use spot servers directly without incurring extra costs for a backup server or requiring any application modifications.

4.3 Handling Pool Dynamics

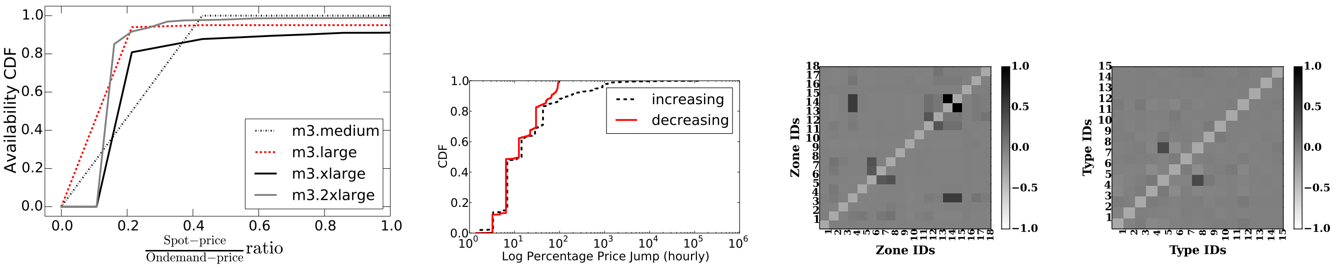
After the initial mapping of a nested VM onto a server in a pool, SpotCheck will likely migrate it to servers in other pools over the course of its lifetime due to pool dynamics. There are two types of pool dynamics caused by changing spot prices that SpotCheck must handle. The first is *revocation dynamics*, which cause the sudden revocation of one or more spot servers within a pool due to prices rising above the bid price. The second is *allocation dynamics*, which dictates when to transition a nested VM back from an on-demand to a spot server when a price spike abates and the spot price again drops below the on-demand price. Note that, in EC2, spot prices often rise substantially above the on-demand price during a price spike, as depicted in Figure 1.

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 [12, 21, 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 bid the equivalent on-demand price for a spot server or bid k times the on-demand price. With the first policy, SpotCheck retains spot servers in a pool as long as those servers' spot price remains below the equivalent on-demand price of the servers. If the spot price rises above the on-demand price, the IaaS platform revokes the spot servers in the pool, which forces SpotCheck to migrate them to on-demand servers. Of course, this revocation only occurs if the equivalent on-demand servers are now cheaper than the spot servers, so migrating to on-demand servers at these times is the cheapest, most cost-effective strategy.

The second policy bids a price that is k times the on-demand price, where $k > 1$. In general, the higher the bid price, the lower the probability of an IaaS platform revoking the spot servers in a pool. Bidding a high price that exceeds the on-demand price lowers a pool's revocation frequency at the expense of a higher cost. This policy also makes proactive migrations more feasible, since SpotCheck can periodically monitor prices and proactively trigger live migrations to on-demand servers whenever prices rise above the on-demand price, but are still lower than the bid price. Thus, SpotCheck currently only uses proactive migrations in conjunction with this second policy.

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.

In either case, in SpotCheck's current implementation, all servers within a spot pool have the *same* bid price. As



(a) CDF of prices

(b) Price changes are large

(c) Correlations of prices between zones.

(d) Correlations between instance types.

Figure 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).

a result, 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. An alternative approach is to request spot servers of a different, larger type where prices are stable, and then migrate to new spot servers. 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. For example, empirical studies have shown that it takes up to 90 seconds to start up a new on-demand server in EC2 [27], while the warning period for a spot server is two minutes, which leaves only 30 seconds to migrate the spot server's 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 is able to maintain a pool of hot spares to immediately receive nested VMs from revoked spot servers without waiting for a new server to come online. Hot spares increase SpotCheck's overhead cost, while reducing the risk of downtime. 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. 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 running on them. 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. Of course, 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. While native IaaS platforms attempt to provision resources to stay ahead of the demand curve, they occasionally run out of on-demand servers if the demand for them exceeds their supply.

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 [17], 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 in this paper is 100% available.

4.4 Cost and Availability Analysis

SpotCheck's goal is to provide resources, in the form of nested VMs, with high availability that resembles that of on-demand servers, but at low prices that resembles those of spot servers. In this section, we analyze the costs incurred by SpotCheck's server pool management and migration strategies, and their resulting availability.

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 $E(c)$ of an individual nested VM as $E(c) = (1-p)E(c_{spot}(t)) + p \cdot c_{od}$, where p denotes the probability of a revocation when it resides on a spot server, $c_{spot}(t)$ denotes the variable price of the spot server, 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., $p = P(c_{spot}(t) > bid)$, which is given by the cumulative distribution shown in Figure 6(a) that we derive empirically for different spot pools.

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.

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. Thus, there is no incentive for EC2 to hinder (or prevent) SpotCheck by reducing (or eliminating) the warning notification for spot servers.

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 such 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.

5. 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 to its customers for managing virtualized cloud servers, although the servers are provisioned in the form of nested VMs.

SpotCheck Controller. The controller, which we implement in `python`, is SpotCheck's main component, and interfaces between customers and the underlying native IaaS platform. The controller is centralized, 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. While we do not expect the controller's performance to be a bottleneck, if it is, replicating it by partitioning customers across multiple independent controllers is straightforward. In addition, we do not include controller costs in our estimates, since we expect them to be negligible, as they are amortized across all the VMs of all the customers.

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 and manage its server pools. 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 load of 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). SpotCheck assumes that the customer-provided disk image used to boot the nested VM resides on a network-attached disk volume in EBS. Due to the use of Xen as the nested hypervisor, the image must support Xen's paravirtualization extensions.

Since network-attached storage is the primary storage medium in many IaaS platforms, including EC2, our current prototype requires the VM to use one (or more) network-attached EBS volumes to store the root disk and any persistent state, and does not support backing up local storage to a remote disk. However, since the speed of the local disk and a backup server's disk are similar in magnitude, EC2's warning period permits asynchronous mirroring of local disk state to the backup server, e.g., using DRBD [19], without significant performance degradation. Our experiments primarily focus on memory-intensive workloads, since using a backup server to store the live in-memory state of multiple nested VMs imposes a significantly larger cost and performance overhead than maintaining disk backups.

To implement SpotCheck, we modified XenBlanket to support bounded-time VM migration in addition to live migration. For the former, we adapt a version of the bounded-time VM migration technique 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. In Yank [30], the VM pauses execution and incurs downtime when transferring the stale state after receiving a warning. To reduce this downtime, our implementation increases the checkpointing frequency after receiving a warning, which reduces the amount of dirty pages the nested VM must transfer. By gradually increasing the checkpointing frequency, we reduce downtime at the cost of slightly degrading VM performance during the warning period.

SpotCheck configures nested VMs mapped to a spot server pool to use bounded-time VM migration, while it configures those mapped to an on-demand pool to 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. As a result, we optimize each backup server's file system and kernel memory management options for write-heavy traffic. Specifically, we use the ext4 filesystem, and avoid costly metadata updates by using the write-back journalling mode and the `noatime` option. This is safe, since the backup server stores a small number of large files, representing the memory state of each nested VM it backs up, with no read/write concurrency, i.e., the files storing VM memory state are either being written or read but not both. To maximize the use of the page cache and absorb write storms, we set a high `dirty_ratio` and `dirty_background_ratio`, which retains file data in the page cache for a long period, allowing the I/O scheduler to increase batching of write requests.

During revocations, the backup server prepares for nested VM restoration by loading images into memory using `fadvise`, setting the `WILL_NEED` flag, and using the appropriate `RANDOM` or `SEQUENTIAL` access flags, depending on whether SpotCheck is lazily restoring the VM or not. 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. To allow the hypervisor to trap accesses to missing memory pages during execution, our implementation of lazy restoration enables shadow paging during the restore process. As a result, 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 reads all other unrestored pages without waiting for them to be paged in by the executing VM.

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`. We conducted these experiments when there was no explicit documentation of a revocation warning for spot servers on EC2. Our measurements found that, at that the time, EC2 provided 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 replaced the default shutdown script with our own script, which EC2 would invoke upon revocation to notify SpotCheck of the two minute warning. However, as we mention previously, as of January 2015 [10], 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 in the next section.

	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

Table 1. Latency for various SpotCheck operations on EC2 for the `m3.medium` server type based on 20 separate measurements executed over a one week period.

6. 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. As mentioned previously, SpotCheck uses Virtual Private Clouds (VPCs) in EC2 to create and assign IP addresses to nested VMs. We run all the microbenchmark experiments in a single EC2 availability zone, while our simulations include cross-availability zone experiments within a single region. Since XenBlanket is only compatible with servers that have HVM capabilities, SpotCheck is only capable of using HVM-enabled EC2 servers. Thus, for our experiments, 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. The `m3.medium` is the smallest HVM-enabled server. 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.

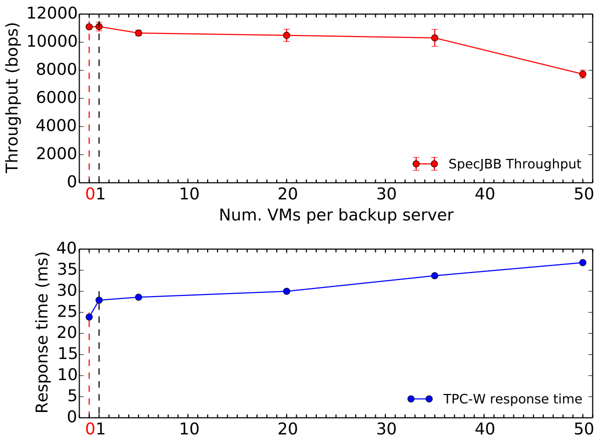


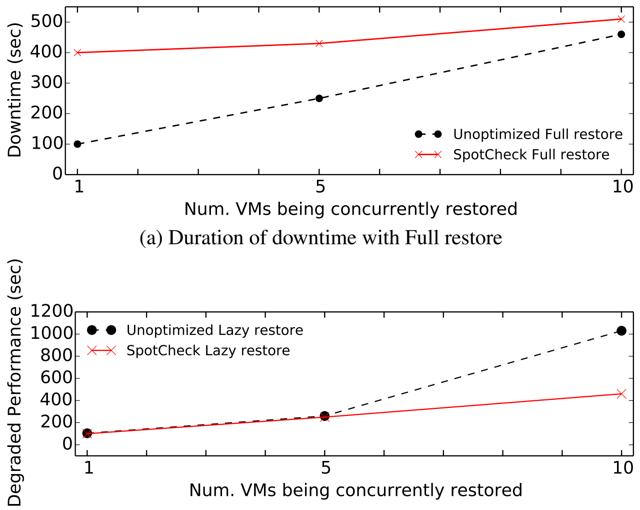
Figure 7. Effect on performance as the number of nested VMs backing up to a single backup server increases.

6.1 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 (see Section 4.4). 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 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



(a) Duration of downtime with Full restore

(b) Duration of degraded performance with Lazy restore

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

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 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. We evaluate SpotCheck's bidding and pool assignment policies below.

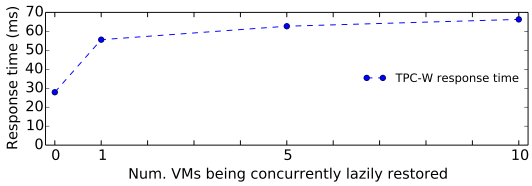


Figure 9. Effect of lazy restoration on VM performance.

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.

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

Finally, 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.*

6.2 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

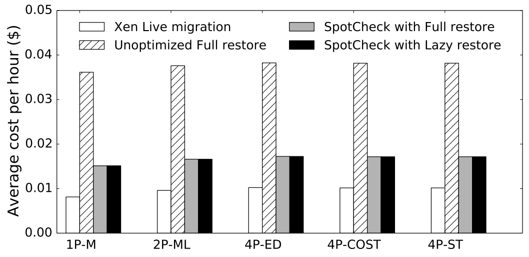


Figure 10. Average cost per VM under various policies.

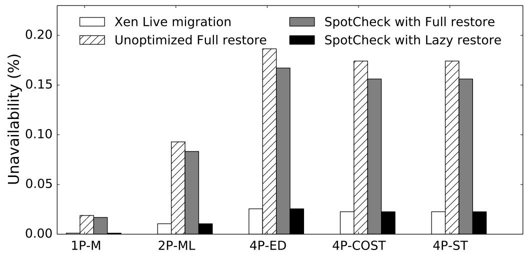


Figure 11. Unavailability for live migration and SpotCheck (with and without optimizations and lazy restore).

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 either their weighted historical (rather than current) 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×.

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 11 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 demonstrates 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` 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 12 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 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

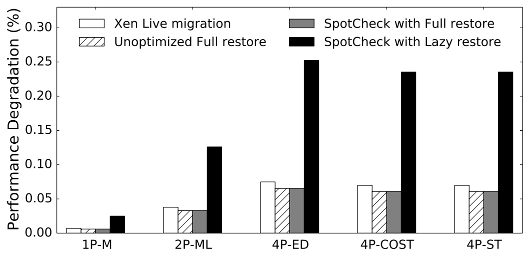


Figure 12. Performance degradation during migration.

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 enables SpotCheck to lower 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.*

Policy Comparison. Our results demonstrate that each of SpotCheck's policies provide similar cost savings (Figure 10) and availability (Figure 11). Performance degradation is lowest for single-pool policy (1P-M), but negligible even for the worst-performing policy (4P-ED as shown in Figure 12), while the four-pool policies drastically reduce the risk of mass migration events (from Table 3).

7. Related Work

Designing Derivative Clouds. Prior work on inter-clouds [13] and superclouds [24, 36] propose managing resources across multiple IaaS platforms by using nested virtualization [11, 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 [29], 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 research in market-based resource allocation [14], which envisions systems with a fluid mapping of software to hardware that enable computation and data to flow wherever prices are lowest.

	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

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

Spot Market Bidding Policies. Prior work on optimizing bidding policies for EC2 spot instances are either based on analyses of spot price history [12, 21, 37] or include varying assumptions about application workload, e.g., job lengths, deadlines [28, 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 in SpotCheck.

Virtualization Mechanisms. Prior work handles the sudden revocation of spot servers either by checkpointing application state at coarse intervals [22, 34, 38] or eliminating the use of local storage [15, 25]. In some cases, application modifications are necessary to eliminate the use of local storage for storing intermediate state, e.g., MapReduce [15, 25]. SpotCheck adapts a recently proposed bounded-time VM migration mechanism [30, 31],which is based on Remus [18] 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 [20] and SnowFlock [23].

8. Conclusion

SpotCheck is a derivative IaaS cloud that offers low-cost, high-availability servers using cheap but volatile servers from a native IaaS platforms. To do this, SpotCheck must simultaneously ensure high availability, reduce the risk of mass server revocations, maintain high performance for applications, and keep its costs down. We design SpotCheck to balance these competing goals. By combining recently proposed virtualization techniques, 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 nearly 5× less than the equivalent on-demand servers

Acknowledgements. We thank all the reviewers and our shepherd Derek Murray for their insightful comments, which improved the quality of this paper. 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] Single Root I/O Virtualization. https://www.pcisig.com/specifications/iov/single_root/, May 1st 2014.

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

[9] G. Banga, P. Druschel, and J. Mogul. Resource Containers: A New Facility for Resource Management in Server Systems. In *OSDI*, February 1999.

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

[11] 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.

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

[13] 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.

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

[15] 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.

[16] 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*, May 2005.

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

[18] 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.

[19] DRBD. DRBD: Software Development for High Availability Clusters. http://www.drbd.org/, September 2012.

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

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

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

[23] 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*, April 2009.

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

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

[26] Z. Liu, A. Wierman, Y. Chen, B. Razon, and N. Chen. Data Center Demand Response: Avoiding the Coincident Peak via Workload Shifting and Local Generation. 70(10), 2013.

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

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

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

[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*, July 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.