Cutting the Cost of Hosting Online Services Using Cloud Spot Markets

Xin He, Prashant Shenoy, Ramesh Sitaraman and David Irwin
University of Massachusetts Amherst
{xhe, shenoy, ramesh}@cs.umass.edu, irwin@ecs.umass.edu

ABSTRACT

The use of cloud servers to host modern Internet-based services is becoming increasingly common. Today's cloud platforms offer a choice of server types, including non-revocable on-demand servers and cheaper but revocable spot servers. A service provider requiring servers can bid in the spot market where the price of a spot server changes dynamically according to the current supply and demand for cloud resources. Spot servers are usually cheap, but can be revoked by the cloud provider when the cloud resources are scarce. While it is well-known that spot servers can reduce the cost of performing time-flexible interruption-tolerant tasks, we explore the novel possibility of using spot servers for reducing the cost of hosting an Internet-based service such as an e-commerce site that must *always* be on and the penalty for service unavailability is high.

By using the spot markets, we show that it is feasible to host an always-on Internet-based service at one-third to one-fifth the cost of hosting the same service in the traditional fashion using dedicated non-revocable servers. To achieve these savings, we devise a cloud scheduler that reduces the cost by intelligently bidding for spot servers. Further, the scheduler uses novel VM migration mechanisms to quickly migrate the service between spot servers and on-demand servers to avoid potential service disruptions due to spot server revocations by the cloud provider. Our work provides the first feasibility study of using cloud spot markets to significantly reduce the cost of hosting always-on Internet-based services without sacrificing service availability.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Reliability, availability, and serviceability

Keywords

Cloud computing; spot markets; cost optimization

HPDC'15, June 15–20, 2015, Portland, Oregon, USA.

ACM 978-1-4503-3550-8/15/06 ...$15.00.
http://dx.doi.org/10.1145/2749246.2749275 .

1. INTRODUCTION

Cloud computing has become the paradigm of choice for building low-cost, scalable Internet-based services. Cloud providers such as Amazon AWS, Microsoft Azure [1], and Google Compute Engine [2] operate large, distributed computing infrastructures that provide computing and storage resources that can be leased by service providers. Cloud providers offer a number of benefits to service providers such as a pay-as-you-go model and flexible, on-demand allocation of resources to hosted services. A key business driver for the rapid adoption of cloud computing by service providers is the reduction in infrastructure costs. Unlike the traditional method of buying dedicated infrastructure, which must be provisioned in advance for the peak demand, leasing cloud servers enables the service provider to scale the service as it grows over time and also exploit just-in-time allocation of capacity to handle peak workloads. Consequently, leasing cloud servers is often more economical than building dedicated infrastructure, especially for services with dynamic or growing workloads.

Today's cloud platforms offer a variety of server types to meet the diverse needs of their hosted services. Cloud servers vary in the offered resource configurations, the leasing cost and the service model offered to customers. For instance, *on-demand* servers offer a fixed rental cost and a *non-revocable* model, where the customers pay a fixed cost and can voluntarily relinquish the server when they no longer need it. In contrast, *spot* servers offer a variable rental cost and a *revocable* model, where the customer bids an upper limit on the price they are willing to pay for a server. The cost of these spot servers fluctuates over time and an allocated spot server may be revoked by the cloud provider when its price rises above the bid price the customer is willing to pay for the server. Spot servers allow a cloud provider to offer unused server capacity at a lower price to customers, while allowing the cloud provider to revoke these servers at any time in order to fulfill requests for higher-priced on-demand servers.

Internet-based services that use the cloud vary significantly in their service requirements. At one end of the spectrum lie data-intensive cloud applications that use cloud servers to run large data analytics tasks (e.g., using MapReduce); such "big data" applications often run in batch mode with the results made available within a specified time period. As noted in Amazon's description of their cloud service [3], spot servers are a popular choice for reducing the cost of running "interruption-tolerant" and "time-flexible" tasks, such as data-intensive batch analytics and scientific comput-

ing. Indeed, there has been recent research [6] [19] [23] [13] on using spot markets to provide non-realtime services that can be performed in batch mode at a reduced cost.

At the other end of the spectrum are *always-on* Internet-based services that serve user requests in real-time. Providers of web content such as CNN, video content such as NetFlix, application portals such as Salesforce, e-commerce portals such as Walmart.com, and social networking sites such as Facebook all belong in this category. Traditionally always-on Internet-based services have relied on dedicated deployed servers owned by the service provider or a third-party content delivery network. Recently, in part to reduce costs, there has been a trend for always-on services to use non-revocable on-demand servers from the cloud markets to meet their infrastructure needs. For instance, Netflix uses Amazon's on-demand cloud services to operate their backend origin infrastructure that stores and serves out videos [4].

In this paper, we ask the intriguing question: *can an always-on Internet-based service utilize the cloud spot markets to host their service at a lower cost without sacrificing service availability?* We explore the feasibility of such a proposition and seek to quantify the cost reduction that is possible in comparison with using the more traditional option of non-revocable on-demand servers. Although the use of spot servers can lower the cost of hosting an Internet-based service, our approach raises new challenges since spot servers *can be revoked at any time.* Such a revocation can potentially cause unavailability of the service for which the penalty is high. It is worth noting that a widely-accepted industry requirement for an always-on Internet-based service such as an e-commerce site is to have *at least* four nines (99.99%) of availability. Alternately, the unavailability of the service can be at most one basis point (0.01%), which roughly translates to 4.3 minutes of downtime per month. The need to keep unavailability very low can be understood from the perspective of a large e-tailer, which could lose a significant amount of revenue if their website is down even for a few minutes during a peak hour [14]. To address this key challenge, we design a novel approach that combines intelligent bidding algorithms that reduce the frequency of revocations and combine it with intelligent migration mechanisms that can quickly migrate a service running on a spot server that is being revoked to an on-demand server. We show that as a result the service unavailability can be significantly reduced so as to be within an acceptable range.

Our Contributions: To our knowledge, our work is the first to examine the feasibility of using cloud spot markets to reduce the cost of always-on Internet-based services, while ensuring that service unavailability is acceptably small. Our work provides significant impetus for service providers to build systems that use the spot markets in a novel way to reduce their costs. We make the following specific contributions.

1. While VM migration mechanisms have been studied in other contexts, our work is the first to use clever migration in the cloud context to avoid service unavailability.

2. We propose proactive bidding algorithms that migrate a spot server before it is revoked by the cloud provider, in contrast with reactive algorithms that migrate *after* the spot server is revoked. We show that being proactive reduces both the service cost and unavailability in comparison with being reactive.

3. We compare three different migration mechanisms: memory checkpointing, memory checkpointing with lazy restore and live. We conclude that using checkpointing alone or using it in combination with live migration does not provide sufficiently low service unavailability. However, checkpointing in combination with lazy restore provides a service unavailability that is acceptable, so as to be a viable alternative for always-on Internet-based services. Further, the addition of live migration halves the unavailability even further.

4. We study multiple possible schemes for hosting Internet-based services on the spot market. First, we study hosting an Internet-based service using our cloud scheduler with the option to use only a single spot market in a single geographical region. In this case, the cost achieved by our scheduler is one-third to one-fifth of the baseline cost of hosting the same Internet-based service using only on-demand servers. If our scheduler has the ability to use multiple markets within the same region, the cost decreases further. And, if the scheduler has the ability to use multiple regions, the cost decreases even further. In the latter two cases, our scheduler exploits the lack correlation in the spot prices across different markets and different regions to achieve a lower cost.

Roadmap: The remainder of this paper is organized as follows. Section 2 presents background on cloud platforms and markets. Section 3 presents the design of our cloud scheduler, and Section 4 presents our experimental results. We present related work in Sec 7 and conclude in Sec 8.

2. BACKGROUND

In this section, we present background on cloud platforms and cloud markets and then describe the research problem addressed in this paper.

2.1 Cloud Platforms and Markets

Our work targets infrastructure clouds that lease server resources to service providers. An infrastructure cloud is a virtualized data center where the cloud provider allocates virtual machines (also referred to as virtual servers) to customers using the underlying physical servers. An infrastructure cloud typically supports different types of virtual servers that vary in their hardware configurations—for instance, Amazon's EC2 cloud supports over a dozen different virtual server configurations that differ in the amount of CPU, memory, disk and network allocations. The cost of a cloud server depends on the chosen configurations and is billed based on time of usage (e.g., hourly).

A cloud service provider can request any server type in one of two modes: on-demand and spot. On-demand cloud servers incur a fixed cost and are non-revocable. For instance, the fixed hourly price of on-demand server varies from 6 cents per hour for the small configuration to as much as $2.19 per hour for the double-extra large configuration. Importantly, once allocated, on-demand servers are non-revocable and the service provider is guaranteed availability to a server until it is no longer needed and voluntarily terminated. Since cloud platforms are provisioned with sufficient capacity to handle peak seasonal demands, they often have many unallocated and unutilized servers, which results in lost revenues due to lack of usage. Cloud providers such as Amazon have begun to offer this unused server capacity at significantly lower prices in the form of spot servers. Spot markets were first introduced by Amazon's EC2 cloud in

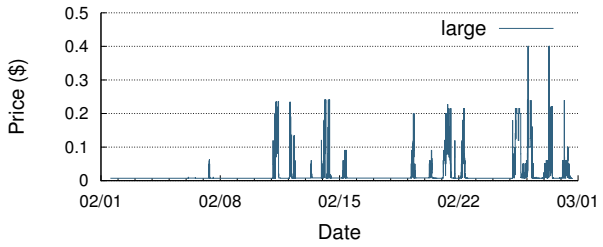
2009. Unlike on-demand servers, a spot server incurs a variable price and is revocable. A cloud-based service provider may request a spot server of any configuration by specifying the maximum hourly price they are willing to pay for such a server (also known as the bid price). Since the price of spot servers varies continuously, the request is granted only if the current price is below the customer's bid price. Furthermore, if the spot price rises above the bid price at any point in the future, the server is revoked. As shown in Figure 1, the price of a spot server fluctuates over time based on supply-demand considerations. Prices are low when there is plenty of unused capacity in relation to demand and the price rises when there is more demand for spot servers or increased demand for on-demand servers, both of which causes the customers with the low bid prices to lose these allocated servers (which are then re-allocated to higher paying on-demand customers).

Researchers have studied the dynamics of spot markets. Each server configuration has its own spot market with fluctuating prices. The different spot markets exhibit different types of dynamics and the price can also spike up during periods of extreme scarcity. As shown in Figure 1, the price of a large server can be as low as few cents per hour for long periods and can spike to as much as $3/hr during high-demand periods. Other than the variable price and revocable nature, spot servers are identical to on-demand servers in all other respects such as their resource configurations. They are also billed on an hourly basis, based on the spot price (*not the bid price*) at the beginning of each hour. Partial hours are not billed if a spot server is revoked before the end of an hourly billing period. Researchers have also observed that upon being revoked, a spot server is given upto a 2 minute grace period to save all unsaved memory state to disk and execute a graceful shutdown (failing which it is forcibly terminated) [12]—while this was an "undocumented" feature of spot servers, Amazon has recently made this grace period official policy by providing an explicit two minute warning prior to revoking a spot server.

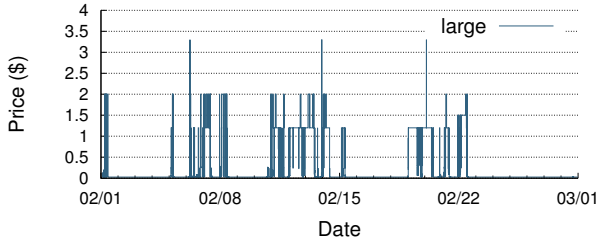
2.2 Problem Statement

Conventional wisdom has held that always-on services should be hosted using either dedicated hardware or non-revocable on-demand servers and that spot servers may not be suitable for this purpose due to potential service interruptions caused by server revocations. In contrast, batch jobs such as MapReduce-style data analytics tasks that have highly elastic deadlines can exploit spot servers to lower their costs while potentially increasing completion time; such tasks can employ checkpointing methods to periodically save their state to disk and resume from the most recent checkpoint if the computation was interrupted by the revocation of spot servers. Thus, as noted by Amazon, spot servers were designed for performing time-flexible and interruption-tolerant tasks.

In this paper, we study the feasibility and benefits of using spot servers for running always-on Internet services, which are *neither* time-flexible nor interruption-tolerant. We study how a service provider can exploit recent advances in OS and virtualization techniques such as nested virtualization and fast migration of virtual machine state to quickly move a service from spot servers to on-demand servers upon revocation and back to spot servers when they are available again. We seek to design clever bidding algorithms that exploit the low costs of spot servers and yet proactively migrate



(a) Varying spot price of a small server



(b) Varying spot price of a large server

Figure 1: Spot prices over a month long period in Amazon's US East-1 region. The prices across markets even within the same region are not strongly correlated, a fact we use in our multi-market bidding algorithms.

the service to on-demand instances when faced with risk of revocation. We also seek to quantify the service unavailability due to downtimes when the cloud platform revokes spot servers. Our overall objectives are to quantify the cost savings and service unavailability and determine whether combining clever bidding and migration technique enable spot servers to be used in a novel fashion for always-on services.

3. A CLOUD SCHEDULER FOR ALWAYS-ON SERVICES

We design a cloud scheduler that procures servers in the cloud markets to host an always-on Internet-based service while minimizing both the cost and the unavailability of the service. A naive approach for using spot servers to host an always-on service is depicted in Figure 3. In this case, the service runs on a spot server for a period of time and the spot server is then revoked by the cloud provider, resulting in the service to be unavailable. Upon revocation, the cloud scheduler immediately requests an on-demand server to replace the revoked spot server. When the on-demand server is (eventually) allocated by the cloud provider, the Internet service is restarted on the new server. This naive baseline approach has two limitations: (i) any memory state of the spot server is lost upon revocation, and (ii) the service is unavailable from the time of revocation to the instant where the service is restarted on a new on-demand server. Note that even in this naive approach, the *disk state of the service is preserved*, since we assume that networked storage volumes are used by the service, so all data on the storage volume is preserved when the server is revoked and the volume can simply be re-attached to the new on-demand server (such networked storage volumes are referred to as EBS volumes in Amazon's EC2 cloud).

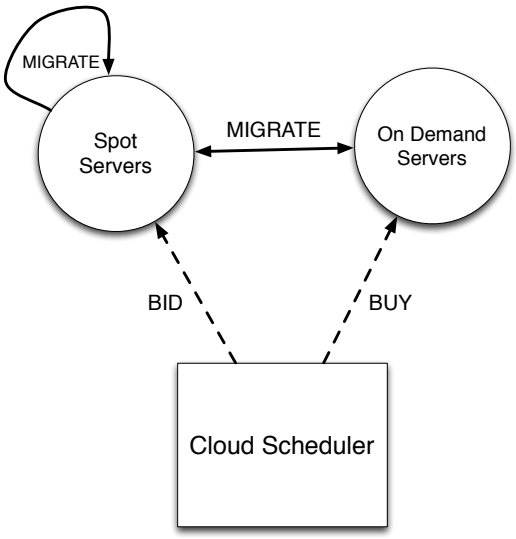


Figure 2: Interactions between the cloud scheduler and the cloud markets.

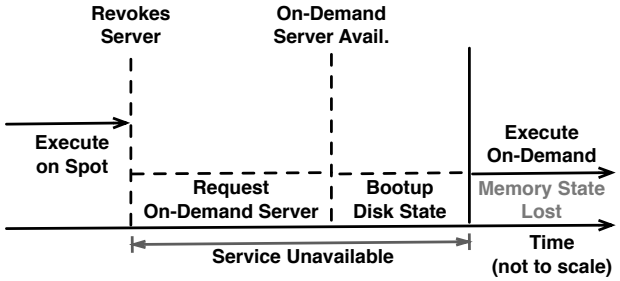


Figure 3: A naive approach to migrating from spot to on-demand server that involves substantial service unavailability and loss of memory state.

Our cloud scheduler uses a combination of intelligent bidding strategies and OS and virtualization-based techniques to address the two drawbacks of the naive approach. Our scheduler seeks to (i) eliminate any loss of memory state by migrating any such state to the new server, and (ii) reduce service unavailability or eliminate it completely in some scenarios. Figure 2 provides an overview of the server transitions implemented by the cloud scheduler. We next describe the two key components of the scheduler.

3.1 Bidding Algorithms

The cloud scheduler's bidding algorithm seeks to achieve two goals: (i) determine what prices to bid when acquiring spot servers so as to reduce the frequency of revocations and achieve cost savings over solely using on-demand servers, and (ii) determine when to transition from spot servers to on-demand servers and vice versa. As noted earlier, when requesting a spot server, the cloud platform requires a maximum price p_b to be specified by the service provider. Since the cost of a spot server fluctuates over time based on supply and demand considerations, this maximum price p_b, also known as the bid price, is the upper limit that the service provider is willing to pay for the spot server. Hence, when the instantaneous spot price $p_{sp}(t)$ rises above the bid price p_b, the spot server is revoked by the cloud provider.

The bidding algorithm must intelligently choose the bid price p_b to achieve its goals. In general, a higher bid price reduces the chances that the spot price will rise above the bid and reduces the chances (and frequency) of server revocation. However, there is a risk that the spot price could increase but still stay below in the bid price, resulting in more cost and lower savings when compared to a pure on-demand model. In contrast, a lower bid price increases the chances of a revocation but can also lower costs.

The cloud scheduler implements two variants of the bidding algorithm. Note that whenever the spot price rises above the price of an on-demand servers, the cost savings vanish and it is more cost-effective to transition to an on-demand server and pay the fixed on-demand price over paying a even higher spot price. In the *reactive* version, the bid price is set to the price of an on-demand server i.e., $p_b = p_{on}$, where p_{on} denotes the cost of an on-demand server. Hence, setting $p_b = p_{on}$ ensures that the cloud platform will revoke the spot server whenever the spot price increases above the on demand price–forcing a migration (transition) to an on-demand server.

An alternative approach, which we refer to the *proactive* version, the bid price is set to a value that is higher than the on-demand price: $p_b = k \cdot p_{on}, \quad k > 1$. In this case, the bidding algorithm continuously tracks the fluctuating spot price $p_{sp}(t)$ and whenever the spot price rises above the on-demand price, the algorithm *voluntarily and proactively* transitions to an on-demand server to pay the fixed on-demand price over paying the higher spot price. Since the migration to an on-demand server is voluntary, the cloud scheduler has more time and flexibility to make the transition, which in turn allows *service unavailability to be virtually eliminated*. Note that in the reactive approach, the transition must be made within a limited time duration before the server is revoked, while in the proactive case, the cloud scheduler can wait until the migration has completed before relinquishing the spot server. In the extreme case of the proactive version, the bidding algorithm can bid the highest bid that is allowed by the cloud platform (e.g., a large multiple k of the on-demand price) which gives the greatest flexibility[1] Regardless of the actual bid, a large sharp spike of the spot price above the bid price will cause the spot server to be revoked by the cloud platform before the proactive algorithm can begin (or finish) its voluntary migration.

After transitioning to an on-demand server, the bidding algorithm continues to monitor the spot price $p_{sp}(t)$ and can again request a spot server when $p_{sp}(t)$ falls below the on-demand price p_{on} and initiate a reverse migration from an on-demand to the spot server; such migrations are also voluntary and can take as long as needed to migrate the service.

Thus, both the reactive and proactive version of the bidding algorithms involve the following steps:

1. *Forced Migration.* If the $p_{sp}(t) > p_b$ and the algorithm holds a spot server, then the spot instance is terminated by the cloud provider. The algorithm is forced to migrate the spot server to an on-demand server.

[1]Note that cloud providers do not allow an infinite bid price. The largest bid price currently allowed by Amazon is four times the on-demand price which we use in our proactive algorithm.

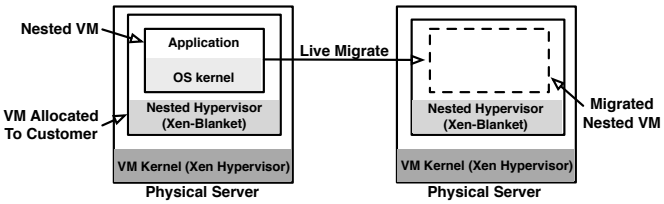


Figure 4: Nested virtualization and live migration of a nested virtual machine.

2. *Planned Migration.* If $p_b \geq p_{sp}(t) \geq p_{on}$ near the end of a billing period (i.e., billing hour) and the algorithm holds a spot server, it reduces cost by voluntarily migrating to an on-demand server.

3. *Reverse Migration.* If $p_{on} > p_{sp}(t)$ near the end of a billing hour and the algorithm currently holds an on-demand server, it reduces cost by re-procuring and migrating back to a spot server.

Note that planned migrations are more desirable than forced migrations, since there is more time to migrate the service in the former, resulting in less disruption to the service. Whereas with a forced migration there is only a short time window before the spot server is terminated.

3.2 OS Mechanisms

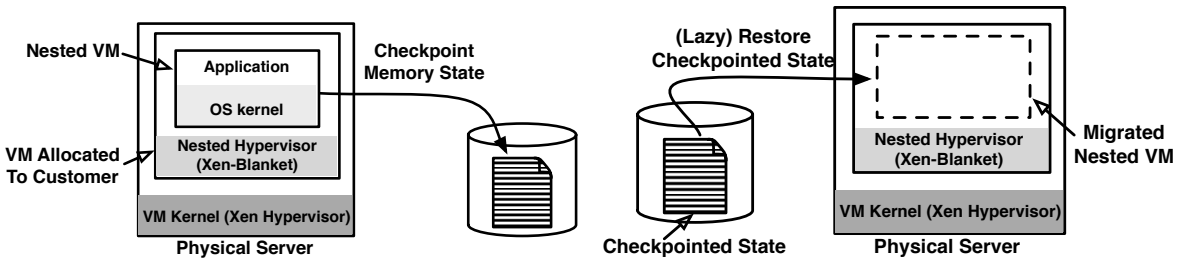
The cloud scheduler uses four well-known OS-level mechanisms to implement migrations from spot servers to on-demand servers and vice versa. While these OS-level mechanisms were proposed elsewhere, they have not been used in cloud platforms previously, nor has this novel combination been studied previously in the cloud context.

We assume that migrations from spot to on-demand servers and back is implemented at the virtual machine level. Virtual machine migration is transparent to the OS and the applications and does not require *any* modifications to either, allowing the technique to apply to all applications (here, cloud services) and operating systems unmodified. Our cloud scheduler employs three variants of virtual machine migration, as described below, to achieve different goals.

Nested virtualization. All common cloud platforms are virtualized and allocate virtual servers in the form of virtual machines. As a result, migration of virtual machines (VMs) is feasible in cloud platforms. Unfortunately, however, today's cloud platforms do not expose migration capabilities of virtual machines to customers and retain this control for themselves. Since the ability to migrate virtual machines from spot to on-demand servers and back is central to our approach, the cloud scheduler uses a mechanism called *nested virtualization* to achieve this goal in today's cloud platforms. Nested virtualization involves running a virtual machine *inside* another virtual machine and the application runs inside the nested virtual machine (see Figure 4). The advantage of nested virtualization is that it allows complete control of the nested virtual machine to the user without requiring any privileged access to the native virtual machine. Since cloud platforms allow a customer to run any OS kernel inside their virtual servers, a customer can easily run a nested virtual machine kernel, instead of a regular OS kernel, and run the second, nested VM inside the virtual server. In such a scenario, we only need to migrate the nested virtual machine from one virtual server to another (e.g., spot to on-demand) without migrating the outside virtual machines. Nested virtualization was proposed in [20] and has been implemented in Xen, a widely used open-source virtualization platform, in the form of Xen-Blanket, which is compatible with Amazon's cloud servers that also use Xen. Experiments reported in [20] show only a modest overhead due to the second nested virtualization layer.

Live migration. Live virtual machine migration is a technique where an entire virtual machine is migrated from one physical server to another while the OS and resident applications continue to execute without requiring any downtime. Live VM migration techniques were proposed over a decade ago and are now supported by most common commercial and open-source virtual machine products (e.g., VMWare, Xen) [7]. Live migration is implemented by interactively copying the memory pages of the virtual machine from the source server to a destination server while the OS and applications continue to run. Since the VM is running during this migration process, memory pages will continue to be modified. Hence, live migration operates in rounds, where each round involves sending memory pages modified since the previous round. After several round of incremental transfers, the difference between the source and destination servers shrinks, and the virtual machine is momentarily paused to send the final set of changed memory pages. The VM at the destination is resumed and the source VM is terminated. This allows the application and its network connections to smoothly transition to the new server and no network reconfiguration is needed (the IP address remains unmodified when transferring the VM within a LAN). We note that VM migration techniques typically only transfer memory state of the virtual machine and do not transfer disk state since disk state is assumed to be stored on a network disk that can simply be re-attached to the destination server. By using nested virtualization, our cloud scheduler can live migrate nested virtual machines as shown in Figure 4. Further, the cloud scheduler uses techniques such as virtual private cloud [3] that allow customer control over the assignment of IP addresses to one's virtual machines to ensure that the address assigned to the nested VM on a spot server can be transparently reassigned to an on-demand server upon migration and vice versa.

Bounded memory checkpointing. Live migration is an attractive and straightforward method for transparently migrating a virtual machine from one server to another. The main limitation of the approach, however, is the latency involved in memory copying may be large, especially for larger server configurations that have substantial amount of memory (e.g., tens of GB of RAM). While these longer latencies can be easily accommodated during planned or reverse migrations initiated by the bidding algorithm, where there is flexibility in determining how much in advance to start the migration process, they may not be feasible for forced migrations. When a cloud platform revokes a spot server, there is a limited window of time to execute a graceful shutdown and this period may not suffice to live migrate a nested VM with large amounts of memory. Consequently a different approach is needed to quickly save memory state during forced migrations.



(a) Checkpoint (b) Restore

Figure 5: VM Memory checkpointing and restoration.

Memory checkpointing of a virtual machine in the form of *suspend-resume* involves writing out the entire memory contents of a VM to disk prior to suspending the virtual machine, and then resuming it at a later time by loading the checkpointed memory state (see Figure 5). Such suspend-resume support is already built into all virtual machine products and is an alternate approach to live migration for capturing memory state and resuming the VM on a different machine. However, writing of the memory contents of a virtual machine to a network disk can also involve a substantial latency for servers with substantial amounts of RAM. Fortunately, we need not wait to initiate memory checkpointing until a revocation is in progress and can instead run memory checkpointing periodically in the background on a continuous basis. In this case, memory contents are asynchronously written to disk in the background periodically and upon a renovation, only the incremental modified memory state since the most recent checkpoint needs to be written out, making the capturing of memory state very quick (and well suited to the limited time window available during a forced migration). Our cloud scheduler uses a recently proposed checkpointing technique called Yank [18] that provides an *upper bound* on the time needed to complement a checkpoint – given a bound τ, it dynamically adjusts the periodicity of the background checkpointing process to ensure that the incremental state does not exceed a threshold and can always be written out within the bound of τ seconds. By setting the bound to a small time window allowed by the cloud platform during a revocation, our cloud scheduler can ensure that all of the memory contents are safely captured to disk and the nested VM can be resumed on a different cloud server; we assume that network disks are used for the purpose of capturing memory state so that the disk is still available after a spot server has terminated.

Lazy VM restore. While bounded memory checkpointing allows suspension of the VM's memory state to complete within a small, bounded time period, the resume part of the suspend-resume process can still involve a latency of tens of seconds—since it requires reading the saved memory state from disk into RAM prior to the resumption of the nested VM. For larger cloud server configurations, this may involve reading tens of gigabytes of RAM state from disk. Hence, we employ an OS mechanism called lazy restore that substantially speeds up the resumption of a virtual machine from its saved memory state. Lazy restore [10, 24, 11] involves reading in only a small subset of the memory pages and resuming execution. The remaining memory state is read concurrently in the background as the VM executes. In the event the executing VM accesses a memory page that has not yet been read from disk, the corresponding memory page is fetched on-demand from disk (akin to how a page fault is handled in traditional operating systems). Lazy restore only requires a small fraction of the memory state to be read from disk before execution can be resumed, allowing for fast resumes and very small downtimes. Of course a downside is that the VM execution may be slower for a period of time due to the page faults that are seen while the remaining memory state is being loaded from disk in the background.

This novel combination of the four OS mechanisms makes it feasible to implement forced, planned and reverse migrations of our bidding algorithm in today's cloud platforms.

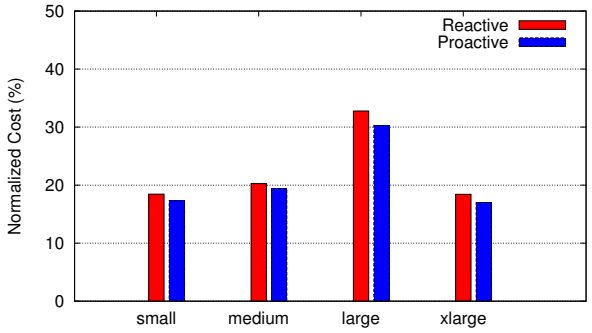
4. EVALUATION OF THE CLOUD SCHEDULER

We use empirical micro-benchmark measurements on Amazon's EC2 cloud as well as simulations seeded by Amazon's spot price traces to drive our evaluation. We evaluate the bidding algorithms and migration mechanisms employed by our cloud scheduler in three different scenarios. The simplest is the *single-region single-market* scenario where the cloud scheduler procures servers of a single size from a single spot market at a single geographical region, migrating to on-demand servers of the same size when necessary. More complex is the *single-region multi-market* scenario where the cloud scheduler has the option to buy servers of different sizes from different spot markets, though all of the servers are hosted at a single region. The most complex situation is the *multi-region multi-market* scenario where the cloud scheduler can procure servers of different sizes from different markets across any of the regions offered by the cloud provider. Intuitively, the cost reduction attainable should increase with each scenario since the cloud scheduler has more options for lowering the cost. However, the migration becomes more complex—a multi-market strategy involves packing multiple nested VMs onto a larger spot or on-demand server, while multi-region involves migration across regions that could be more complex and expensive.[2]

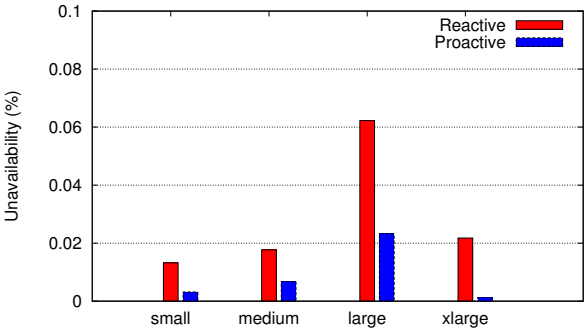
4.1 Microbenchmarks

We ran the XenBlanket nested hypervisor on Amazon's cloud servers and conduct a series of micro benchmark measurements to capture the overheads of various migration mechanisms; these measured values are then used to param-

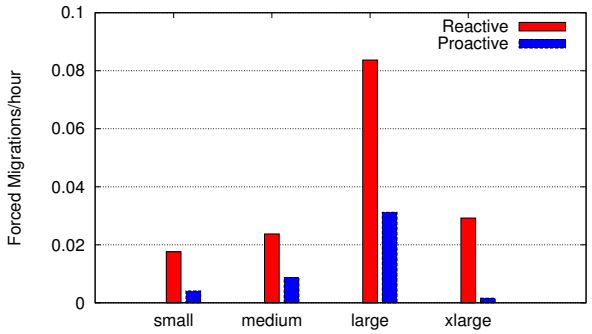
[2]WAN VM migration across regions involves additional network reconfigurations [21] that also add to the overheads.



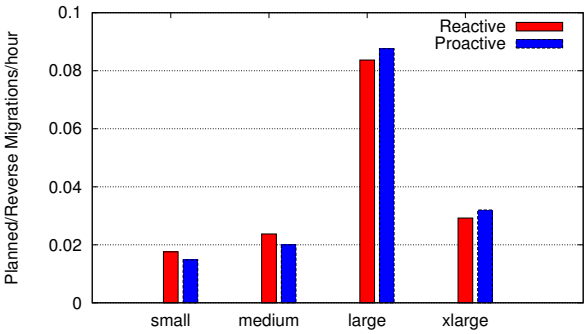
(a) Both proactive and reactive provide significantly smaller cost than the baseline.



(b) Proactive mitigates service unavailability better than reactive.



(c) Proactive has a smaller number of forced migrations per server per hour.



(d) Proactive and reactive have similar number of planned/reverse migrations per server per hour.

Figure 6: A comparison of proactive versus reactive bidding algorithms.

Instance type	US east (s)	US west (s)	EU west (s)
On-demand	94.85	93.63	98.08
Spot	281.47	219.77	233.37

Table 1: Average Start-up Time of On-demand and Spot instances

	Live migrate (s)	Memory checkpointing (s)	Disk copy (s)
Inside US East	58.5	28.9	–
Inside US West	57.1	28.8	–
Inside EU West	58.2	28.05	–
US East to US West	73.7	–	122.4
US East to EU West	74.6	–	140.5
US West to EU West	140.2	–	171.6

Table 2: Overhead of migration mechanisms.

eterize subsequent simulation experiments. We first measure the latency to allocate an on-demand and spot server of different sizes in different regions. Table 1 shows the mean measured values across multiple runs and shows that typical allocation times are around 1.5 minutes for an on demand server and between 3.5 to 4.5 minutes for spot servers. Next, we measure the latency to live migrate a nested VM with 2GB of RAM within and across regions. Table 2 shows that live migration latency is around 1 minute for intra-data center migration and vary from 73 to 140 seconds for cross region migrations. While LAN migration can use networked storage and do not require disk state transfers, cross-region WAN migration do and the table shows that cross-datacenter copying of disk state take between 2 to 3 minutes per GB of disk state. We also benchmark the latency of memory checkpointing, which involve writing memory pages sequentially to a network attached disk and observe a latency of 28s per GB of memory state (VM restoration latencies which read this data back from disk are similar). In contrast, we assume a lazy restoration latency of 20s, which is independent of memory size, based on measurements reported in [10].

In our microbenchmarks, we conducted multiple runs to obtain estimated startup times and migration times. In addition to these measured parameters, we also gathered published spot price history for Amazon's spot servers. In our simulations, we sampled the empirically observed distributions and used a different sample for each simulation run. We report results for small, medium, large and xlarge spot servers at four Amazon regions: US East 1A, US East 1B, US West 1A and Europe West 1A.

4.2 Proactive versus Reactive Bidding

We start with the simplest scenario where our bidding algorithm described in Section 3.1 uses a single market (either small, medium, or large) in a single region (us-east). The bidding algorithm alternately uses servers procured in the chosen spot market in the us-east region or an on-demand server obtainable at the same region. We study the two variants of the bidding algorithm described earlier, proactive and reactive, both using the bounded checkpointing

with lazy restore for migration.[3] To estimate the cost savings from using the spot market, we use the cost of using only on-demand servers to host the service as the baseline. As shown in Figure 6(a), both proactive and reactive approaches show a significant reduction in cost achieving 17% to 33% of the baseline cost of not using the spot market at all. However, proactive does achieve a slightly smaller cost than reactive in all three markets. More importantly, the proactive algorithm achieves significantly less service unavailability than the reactive algorithm in *all* markets (cf. Figure 6(b)). Specifically, the unavailability of the proactive algorithm is smaller by a factor that ranges from 2.5 to 18 when compared to the reactive algorithm. The reason is that the proactive algorithm significantly reduces the number of forced migrations in comparison with the reactive algorithm as shown in Figure 6(c). Specifically, the proactive algorithm migrates its servers from the spot market to the on-demand market before it is forced to do so, giving it more time to perform the migration, in turn reducing the possibility of the service being unavailable during the migration process. Figure 6(d) shows that proactive and reactive algorithms have similar number of planned/reverse migrations.

The results for other regions are also similar to what we presented above for us-east. Thus, we conclude that it is better to be proactive rather be reactive, both from the perspectives of cost and unavailability. Henceforth, we will use the proactive bidding algorithm and its variants in all our subsequent evaluations.

4.3 Evaluating the Migration Mechanisms

We next evaluate the efficacy of four different combinations of migration mechanisms for the proactive bidding algorithm: memory checkpointing (with standard restore), memory checkpointing with lazy restore, live migration with checkpointing and live migration with checkpointing and lazy restore. The service unavailability of each combination is shown in Figure 7 for small servers in the US East 1a region; we report results for normal case as well as a pessimistic case where all migration mechanisms exhibit worst case behavior. Pure checkpointing alone has the worst unavailability of 0.018% due to the long latency needed to read the save memory state from disk prior to resuming the virtual machine. The unavailability improves significantly to 0.004% when lazy restore is used to speed up the resumption of a checkpointed VM. Similarly live migration with checkpointing has higher unavailability of 0.0095% since any forced migrations employ checkpointing with its longer downtimes. The final combination of using live migration when possible, and checkpointing with lazy restore for any forced migration has the smallest unavailability of 0.002% (roughly factor of two better than checkpointing with lazy restoration alone). According to [8] and [15], in the worst case, the downtime during migration of a 4GB virtual machine can be 10s, and migration of a 2GB VM causes down time of as much as 4s. The worst case of memory restore is copying the whole memory to the new VM while restoring. In our measurement, the time to copy a 2GB disk file which is less than 120s inside a region. The pessimistic scenarios, which assume pessimistic values of a 10s outage for live migration, and 120s latency for lazy restoration, see uniformly higher unavailability for all mechanisms, with the best unavailability of 0.017% for live migration with checkpointing and lazy restore. Thus we conclude that pure checkpointing is not desirable due to its higher unavailability, when used alone or in combination with live migration. However, when used with lazy restoration, the technique provides unavailability values that make it feasible for always-on services, with live migration further halving the unavailability of the service.

[3]Results for planned live migrations are similar and omitted here.

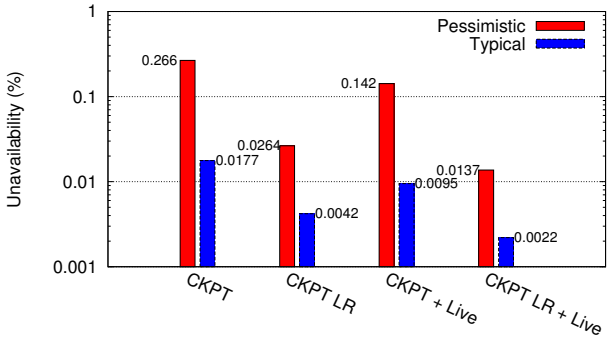


Figure 7: Comparison of different migration mechanisms using proactive bidding. (Unavailability percent is plotted in log-scale)

4.4 Multi-Market Bidding Strategies

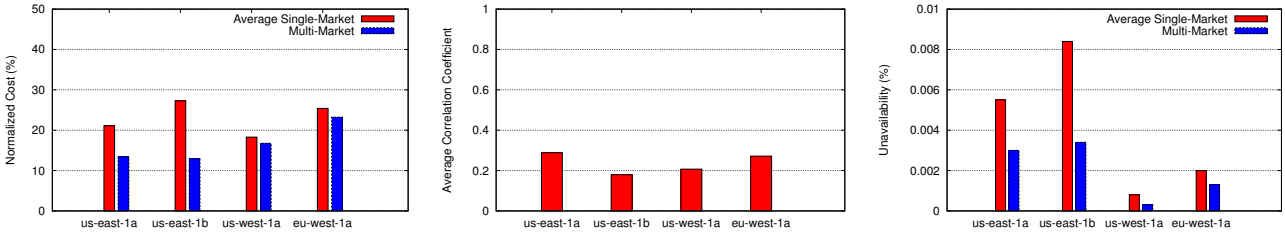
We study the benefit of bidding in multiple spot markets in comparison with bidding in a single spot market within a given region. The intuitive reason why multiple markets can decrease the cost is that when one spot market has a price rise the other markets in the same region *may* not experience a similar rise. So, our cloud scheduler can move its servers from the pricier spot market to one of the cheaper ones.

We modified our proactive bidding algorithm of Section 3.1 to use multiple markets within the same region as follows. In the planned migration step, we look to see if there is *any* spot market in the same region that has a cheaper price than the on-demand price. If so, the algorithm bids in the cheapest available spot market in that region and migrates the spot server to that market. If not, the algorithm migrates the spot server to the on-demand server as it is currently cheaper than any of the spot servers. The forced and reverse migration steps work the same as before.

We evaluated our multi-market bidding algorithm in all regions and show the results in Figure 8. As shown in Figure 8(a), a multi-market scheme was able to reduce the cost by 8% for us-west-1a to 52% for us-east-1b in comparison with the average cost of the single-market schemes in those regions. The reason for the reduction is that price correlation between the different markets is low as shown in Figure 8(b), i.e., when the price spikes up in one of the spot markets, another of the spot markets in the same region may not have an equivalent increase. Our multi-market bidding algorithm exploits the lack of correlation to move servers from the costlier to the cheaper spot market.

4.5 Multi-Region Bidding Strategies

We study multi-region bidding algorithms that can move servers between spot markets both within a given region



(a) Bidding in multiple markets decreases the cost in comparison with single-market schemes.

(b) The price correlation between the different spot markets within a region is low.

(c) Bidding in multiple markets decreases unavailability in comparison to single-market schemes.

Figure 8: The benefits of bidding in multiple markets within the same region.

as well as *across* different regions. Our multi-region algorithm is identical to the multi-market algorithm described in Section 4.4 except that the algorithm looks for the cheapest market both within and across regions for migration. We evaluate our multi-region bidding algorithm on pairs of regions and we show the results in Figure 9. To normalize the cost achieved by our multi-region algorithm, we use the lowest on-demand cost available in the two allowable regions as the baseline. As we show in Figure 9(a), our multi-region strategy achieves 12-17% of the baseline cost, resulting in a significant cost reduction in comparison to the baseline of not using the spot markets at all. Further, our multi-region algorithm results in a normalized cost that is 5-28% smaller than the average cost achieved by the single-region bidding algorithm operating in each of the two regions. The reason for the additional cost savings is that the prices across two regions have a low correlation as shown in Figure 9(b). Therefore, when the spot price increases in one region, our multi-region algorithm is able find cheaper prices in the other region.

However, service unavailability can actually *increase* in some cases with multi-region bidding as can be seen in Figure 9(c). The reason is that regions such as us-east-1a and us-east-1b that tend to have cheaper prices, also have greater variability in those prices (cf. Figure 10). Whereas the eu-west region tends to be more expensive but the prices are more stable. Since our multi-region bidding algorithm migrates its servers to spot markets primarily based on a lower price, it can sometimes migrate to lower cost regions (such as us-east) with more volatile prices. Markets with larger price volatility can cause more migrations as the prices fluctuate making these markets more expensive at times than the other markets. The increased migration causes more unavailability. Bidding algorithms that also consider price stability instead of greedily opting for the cheapest price is a topic for future research.

5. COST AND AVAILABILITY ANALYSIS

In the previous section, we show that by using nested VMs and migrating between on-demand instances and spot instances, we can achieve a significant reduction in cost over using on-demand instances alone. In this section, we go a step further to show the advantage of our method over current spot market.

Figure 11 compares our proactive method to using spot instances alone. We find that although using spot instances reduce cost in some markets, its availability is quite bad.

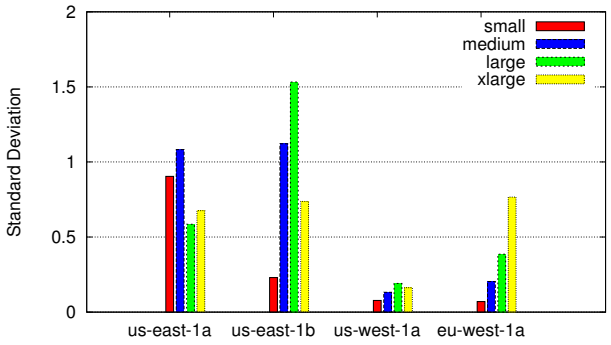


Figure 10: The prices of us-east are more variable than us-west or eu-west.

	Cost	Availability
Only On-demand	High	High
Only Spot	Low	Low
Using migration mechanisms	Low	High

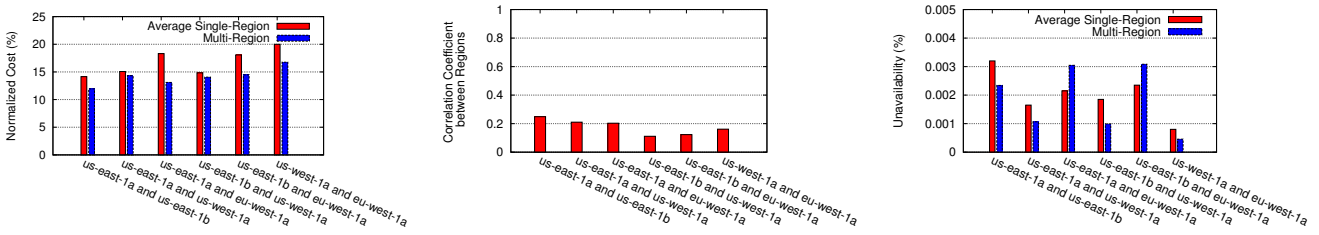
Table 3: By using a combination of on-demand and spot servers, we can achieve low cost and high availability for online services

In small, medium and large markets, unavailability is over 1% which is not acceptable for always-on internet services. Further, since the price may be over bid limit for a long period, services can be unavailable for hours or even days. Hence, using spot instances alone are not a good choice for hosting always-on internet services, as conventional wisdom has held.

As table 3 shows, our method combines the advantage of on-demand and spot market and provides a solution with low cost and high availability to host always-on internet services in current cloud platforms.

6. IMPACT OF SYSTEM PERFORMANCE ON COST

Although nested VMs on spot instances provide good savings, nested virtualization can also impose system performance overheads. In this section, we quantify these system performance overheads and study their impact on the eventual cost savings.

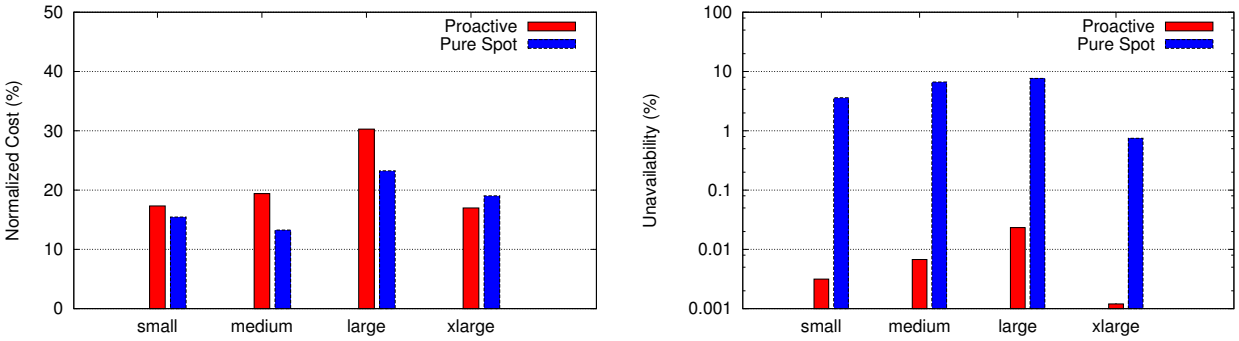


(a) Bidding in multiple regions decreases the cost in comparison with single-region schemes.

(b) The price correlation across regions is low, enabling the multi-region algorithm to avoid price hikes by switching to a cheaper alternate region.

(c) Multi-region unavailability can increase in cases when the lower-priced markets such as us-east happen to also be less stable.

Figure 9: A comparison of multi-region versus single-region bidding algorithms. Both algorithms bid in multiple markets within their allowable regions.



(a) Using pure spot instances can slightly reduce the total cost

(b) Using pure spot instances largely increases unavailability

Figure 11: A comparison of proactive method versus using only spot servers.

	Amazon VM (Mbps)	Nested VM (Mbps)
Network TX	304	304
Network RX	316	314
Disk Read	304.6	297.6
Disk Write	280.4	274.2

Table 4: Network and Disk I/O performance of nested VMs is comparable to Amazon's native VMs

6.1 Disk and Network I/O Overheads

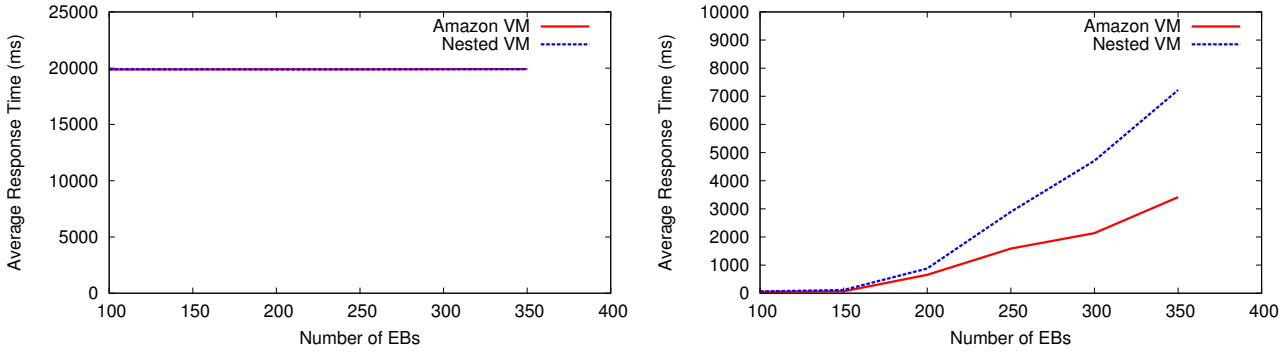
Since we use a second hypervisor to host our nested VMs, our system will incur performance overheads. We compare the system performance of Amazon VMs and nested VMs (using the xen-blanket nested hypervisor). In our experiments, we use Amazon EC2 m3.medium VMs which has 1 virtual CPU and 3.75 GB memory, using HVM virtualization instances and Elastic Block Store (EBS). When creating a nested VM, we only distribute 3 GB memory to it because dom0 needs some memory to hold its service. Network address translation (NAT) is used to provide transparent network access to the nested VMs.

We first measure the network I/O and disk I/O overhead. We use iperf to get a measurement of network throughput. From Table 4 we can see that both the transmitting rate and receiving rate of nested VM matches the throughput of Amazon VM. Then we ran *dd* to measure disk I/O. System caches at all layers were flushed before reading and writing 2GB of data from the root file system. Table 4 shows that disk I/O performance is only degraded by 2%. These results show that disk and network I/O performance of nested virtual machine instances is close to Amazon's native VMs.

6.2 CPU Overhead Benchmarking

The original Xen-blanket paper [20] provided detailed results on the CPU overheads imposed by the Xen-blanket nested hypervisor. We use TPC-W as an example benchmark application to verify their results in Amazon's EC2 cloud. TPC-W is a web benchmark that emulates an online e-commerce store. We use a Java servlets-based multi-tiered configuration of the TPC-W shopping website. Our experiment injects an "ordering workload" where 50% of the clients only browse the website and the remaining 50% execute order transactions. TPC-W allows us to measure the influence of the extra xen-blanket hypervisor on the response time perceived by the clients of an interactive web application.

We perform the above experiment with two common configurations: 1) browsers fetch images from the server while browsing 2) browsers don't fetch images from the server while browsing. The first configuration emulates a case where the entire website, including the images, is served by our server VMs. The second configuration emulates a case where only the base web page is served by our server VMs and the embedded images are cached and served by a third-party content delivery service. Figure 12 shows



(a) If browsers fetch images while browsing, nested VM's performance is no worse than Amazon VM

(b) If browsers don't fetch images while browsing, nested VM's performance is upto 50% worse than Amazon VM

Figure 12: The overhead of nested VM depends on the type of service it provides

the response time under a varying load imposed by different number of emulated browsers. Figure 12(a) shows the result under the first configuration. We can see that nested VMs can achieve similar performance as Amazon VMs. This is because when browsers get images from the server, the benchmark is I/O bound and xen-blanket can provide efficient I/O. Figure 12(b) gives the result under the second CPU-intensive configuration; in this case, the CPU overhead depends on the load and in the worst case, we see that nested VMs incur up to a 50% overhead over Amazon VMs.

From our system measurements, we observe that disk and network intensive services will see close to native performance and achieve most of the cost savings. For CPU-intensive workloads, the overheads depends on the actual load and can reduce the cost savings (since additional capacity is needed to service a particular load). In the worst case, performance may be halved, yielding actual savings of 12%-34% of the baseline cost. Of course, Xen-Blanket is a research prototype of a nested hypervisor and a commercial ested hypervisor implementation may be able to optimize the performance overhead and yield better savings.

7. RELATED WORK

There has been recent research on cloud spot servers, but much of prior work has focused on interrruption-tolerant batch jobs. The use of spot servers to reduce the cost of data-intensive MapReduce batch jobs has been studied in [12] and [6]. Optimal bidding strategies that minimize completion times of short batch jobs have also been studied in [23], [17], and [19]. Checkpointing techniques for batch jobs running on spot servers were studied in [22]. In contrast, our work focuses on using spot instances of always-on services that interact with users in real-time.

Our work builds on a large body of work in virtualization techniques [5]. Live migration of virtual machines was studied in [7], while checkpointing techniques for virtual machines have been studied in [9, 18]. Nested virtualization in the context of the Xen virtual machine platform was proposed in [20]. Lazy restoration methods have been studied in [24, 10, 11]. SpotCheck [16] is a system that uses nested virtualization and migration mechanisms to manage server pools based on spot and on-demand servers. Our work assumes the presence of such system level mechanisms and examines a range of bidding and migration policies that use these mechanisms in the cloud context.

8. CONCLUSIONS

In this paper we studied the efficacy of using spot servers to lower the cost of hosting always-on Internet services. We proposed a cloud scheduler that combines bidding algorithms and migration techniques to reduce, or nearly eliminate, unavailability by migrating a spot server to an on-demand server when needed. Our results demonstrated the feasibility of using our proactive approach to provide availability levels that are close to levels desirable for always-on services, at nearly one-third to one-fifth of the cost of the traditional approach of using on-demand servers. As part of future work, we plan to design more sophisticated bidding strategies that take spot price stability into account to further reduce server revocation frequency, and hence, service unavailability.

9. ACKNOWLEDGMENTS

We thank all the reviewers for their insightful comments, which improved the quality of this paper. This work is supported in part by NSF grants CNS-1413998, CNS-1422245 and CNS-1229059.

10. REFERENCES

[1] `https://azure.microsoft.com/`.

[2] `https://cloud.google.com/products/compute-engine/`.

[3] `http://aws.amazon.com/`.

[4] `http://aws.amazon.com/solutions/case-studies/netflix/`.

[5] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. *ACM SIGOPS Operating Systems Review*, 37(5):164–177, 2003.

[6] N. Chohan, C. Castillo, M. Spreitzer, M. Steinder, A. Tantawi, and C. Krintz. See spot run: using spot

instances for mapreduce workflows. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, pages 7–7. USENIX Association, 2010.

[7] C. Clark, K. Fraser, S. Hand, J. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfiel. Live migration of virtual machines. In *Proceedings of Usenix NSDI Symp.*, May 2005.

[8] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI'05, pages 273–286, Berkeley, CA, USA, 2005. USENIX Association.

[9] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield. Remus: High availability via asynchronous virtual machine replication. In *Proceedings of the 5th NSDI Symp.*, pages 161–174. San Francisco, 2008.

[10] M. Hines and K. Gopalan. Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning. In *Proceedings of ACM VEE Conference*, March 2009.

[11] A. Lagar-Cavilla et al. Snowflock: rapid virtual machine cloning for cloud computing. In *Proceedings of ACM EuroSys*, pages 1–12, 2009.

[12] H. Liu. Cutting mapreduce cost with spot market. In *3rd USENIX Workshop on Hot Topics in Cloud Computing*, pages 1–5, 2011.

[13] A. Marathe, R. Harris, D. Lowenthal, B. R. de Supinski, B. Rountree, and M. Schulz. Exploiting redundancy for cost-effective, time-constrained execution of hpc applications on amazon ec2. In *Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing*, HPDC '14, pages 279–290, New York, NY, USA, 2014. ACM.

[14] E. Nygren, R. Sitaraman, and J. Sun. The Akamai Network: A platform for high-performance Internet applications. *ACM SIGOPS Operating Systems Review*, 44(3):2–19, 2010.

[15] F. Salfner, P. Troger, and A. Polze. Downtime analysis of virtual machine live migration. In *DEPEND 2011, The Fourth International Conference on Dependability*, pages 100–105, 2011.

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

[17] X. Shi, K. Xu, J. Liu, and Y. Wang. Continuous double auction mechanism and bidding strategies in cloud computing markets. *arXiv preprint arXiv:1307.6066*, 2013.

[18] R. Singh, D. E. Irwin, P. J. Shenoy, and K. K. Ramakrishnan. Yank: Enabling green data centers to pull the plug. In *NSDI*, pages 143–155, 2013.

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

[20] D. Williams, H. Jamjoom, and H. Weatherspoon. The xen-blanket: virtualize once, run everywhere. In *Proceedings of the 7th ACM european conference on Computer Systems*, pages 113–126. ACM, 2012.

[21] T. Wood, K. Ramakrishnan, P. Shenoy, and J. V. der Merwe. Cloudnet: Dynamic pooling of cloud resources by live wan migration of virtual machines. In *Proc. of ACM VEE*, March 2011.

[22] S. Yi, D. Kondo, and A. Andrzejak. Reducing costs of spot instances via checkpointing in the amazon elastic compute cloud. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 236–243. IEEE, 2010.

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

[24] I. Zhang, A. Garthwaite, Y. Baskakov, and K. C. Barr. Fast restore of checkpointed memory using working set estimation. In *ACM SIGPLAN Notices* volume 46, pages 87–98. ACM, 2011.