The Xen-Blanket: Virtualize Once, Run Everywhere

Dan Williams[†‡]
djwill@cs.cornell.edu

Hani Jamjoom[‡]
jamjoom@us.ibm.com

Hakim Weatherspoon[†]
hweather@cs.cornell.edu

[†] Cornell University, Ithaca, NY
[‡] IBM T. J. Watson Research Center, Hawthorne, NY

Abstract

Current Infrastructure as a Service (IaaS) clouds operate in isolation from each other. Slight variations in the virtual machine (VM) abstractions or underlying hypervisor services prevent unified access and control across clouds. While standardization efforts aim to address these issues, they will take years to be agreed upon and adopted, if ever. Instead of standardization, which is by definition provider-centric, we advocate a *user-centric* approach that gives users an unprecedented level of control over the virtualization layer. We introduce the *Xen-Blanket*, a thin, immediately deployable virtualization layer that can homogenize today's diverse cloud infrastructures. We have deployed the Xen-Blanket across Amazon's EC2, an enterprise cloud, and a private setup at Cornell University. We show that a user-centric approach to homogenize clouds can achieve similar performance to a paravirtualized environment while enabling previously impossible tasks like cross-provider live migration. The Xen-Blanket also allows users to exploit resource management opportunities like oversubscription, and ultimately can reduce costs for users.

Categories and Subject Descriptors D.4 [*OPERATING SYSTEMS*]: Organization and Design

General Terms Design, Experimentation, Performance

Keywords Nested Virtualization, Cloud Computing, Xen

1. Introduction

Current Infrastructure as a Service (IaaS) clouds—both public and private—are not interoperable. As a result, cloud users find themselves locked into a single provider, which may or may not suit their needs. Evidenced by the Amazon Elastic Compute Cloud (EC2) downtime in April 2011, outages can affect even large, mature and popular public clouds. A multi-cloud deployment, on the other hand, facilitates fault tolerance strategies that can withstand the failure of an entire provider. Furthermore, multi-cloud deployments can offer new resource management and economic alternatives to would-be cloud users that are already managing their own elaborate private cloud infrastructures. If well-integrated with private clouds, public clouds can provide an outlet for excess load or growth.

Fundamentally, today's clouds lack the homogeneity necessary for cost-effective multi-cloud deployments. That is, a single VM image cannot be deployed—unmodified—on any IaaS cloud. Even worse, there is no consistent set of hypervisor-level services across providers. While some progress towards multi-cloud homogeneity is expected through standardization efforts such as the Open Virtualization Format [11], these *provider-centric* approaches will likely be limited to simple cloud attributes—like image format—and take years to be universally adopted.

We propose a different way of implementing multi-cloud homogeneity. Instead of relying on cloud providers to change their environments, we advocate a *user-centric* view of homogenization, where users are able to run their unmodified VMs on any cloud without any special provider support. As such, users can implement or deploy hypervisor services or management tools on clouds that do not supply them. Towards this goal, we present the Xen-Blanket, a system that transforms existing heterogeneous clouds into a uniform user-centric homogeneous offering. The Xen-Blanket consists of a second-layer hypervisor that runs as a guest inside a VM instance on a variety of public or private clouds, forming a *Blanket layer*. The Blanket layer exposes a homogeneous interface to second-layer guest VMs, called *Blanket guests*, but is completely user-centric and customizable. With the Xen-Blanket, hypervisor-level techniques and management tools, like VM migration, page sharing, and oversubscription, can all be implemented inside the Blanket layer. Meanwhile, the Blanket layer contains *Blanket drivers* that allow it to run on heterogeneous clouds while hiding interface details of the underlying clouds from guests.

Existing nested virtualization techniques (like the Turtles project [2]), focus on an efficient use of hardware virtualization primitives by both layers of virtualization. This re-

EuroSys'12, April 10–13, 2012, Bern, Switzerland.

quires the underlying hypervisor—controlled by the cloud provider—to expose hardware primitives. None of today's clouds currently offer such primitives. In contrast, the Xen-Blanket can be deployed on third-party clouds today, requiring no special support. Thus, the contributions of the Xen-blanket are fundamentally different: the Xen-blanket enables competition and innovation for products that span multiple clouds, whether support is offered from cloud providers or not. Further, the Xen-blanket enables the use of unsupported features such as oversubscription, CPU bursting, VM migration, and many others.

The Xen-Blanket is deployed today on both Xen-based and KVM-based hypervisors, on public and private infrastructures within Amazon EC2, an enterprise cloud, and Cornell University. The Xen-Blanket has successfully homogenized these diverse environments. For example, we have migrated VMs to and from Amazon EC2 with no modifications to the VMs. Furthermore, the user-centric design of the Xen-Blanket affords users the flexibility to oversubscribe resources such as network, memory, and disk. As a direct result, a Xen-Blanket image on EC2 can host 40 CPU-intensive VMs for 47% of the price per hour of 40 small instances with matching performance. Blanket drivers achieve good performance: network drivers can receive packets at line speed on a 1 Gbps link, while disk I/O throughput is within 12% of single level paravirtualized disk performance. Despite overheads of up to 68% for some benchmarks, Web server macrobenchmarks can match the performance of single level virtualization (i.e., both are able to serve an excess of 1000 simultaneous clients) while increasing CPU utilization by only 1%.

In this paper, we make four main contributions:

- We describe how user-centric homogeneity can be achieved at the hypervisor level to enable multi-cloud deployments, even without any provider support.
- We enumerate key extensions to Xen, including a set of *Blanket drivers* and hypervisor optimizations, that transform Xen into an efficient, homogenizing Blanket layer on top of existing clouds, such as Amazon EC2.
- We demonstrate how the Xen-Blanket can provide an opportunity for substantial cost savings by enabling users to oversubscribe their leased resources.
- We discuss our experience using hypervisor-level operations that were previously impossible to implement in public clouds, including live VM migration between an enterprise cloud and Amazon EC2.

The paper is organized as follows. Section 2 further motivates the need for user-centric homogenization to be deployed on today's clouds. Section 3 introduces the concept of a Blanket layer, and describes how the Xen-Blanket provides a user-centric homogenized layer, with the implementation details of the enabling Blanket drivers in Section 4. Some overheads and advantages of the Xen-Blanket are quantified in Section 5, while qualitative practical experience is described in Section 6. Finally, Section 7 identifies future directions, Section 8 surveys related work, and Section 9 concludes.

2. My Cloud, My Way

A cloud user would achieve several benefits from a homogeneous interface to a cloud that encompasses many different cloud providers. If a single VM image can be deployed on every cloud, image management, even upgrading and patching, is simplified. If any service offered by one cloud was available in any other cloud, users would not feel locked-in to a particular vendor. Hypervisor-level resource management techniques and cloud software stacks would emerge that truly span providers, offering users the control to utilize and manage cloud resources to their full potential.

Today's clouds lack homogeneity in three ways. First, VM images—the building blocks of cloud applications—cannot be easily instantiated on different clouds. Second, clouds are becoming diverse in terms of the services they provide to VMs. For example, Amazon EC2 provides tools such as CloudWatch (integrated monitoring), AutoScaling, and Elastic Load Balancing, whereas Rackspace contains support for VM migration to combat server host degradation and CPU bursting to borrow cycles from other instances. Third, a class of resource management opportunities that exist in a private cloud setting—in particular, tools that operate at the hypervisor level—are not consistently available between providers. For example, there is no unified set of tools with which users can specify VM co-location on physical machines [29], page sharing between VMs [16, 23], or resource oversubscription [28].

The desire for a homogeneous interface across cloud providers is *not* a call for standardization. We distinguish between *provider-centric* and *user-centric* homogenization. Standardization is an example of provider-centric homogenization, in which every cloud provider must agree on an image format, services, and management interfaces to expose to users. Standards are emerging; for example, Open Virtualization Format (OVF) [11] describes how to package VM images and `virtio` defines paravirtualized device interfaces. However, until *all* clouds (e.g., Amazon EC2, Rackspace, etc.) adopt these standards, VM configurations will continue to vary depending on the cloud they run on. Even worse, it is unlikely—and infeasible—that the vast array of current and future services available to VMs become standardized across all clouds. Attempts at standardization often lead to a set of functionality that represents the "least common denominator" across all participating providers. Many users will still demand services that are not in the standard set and cloud providers will continue to offer services that differentiate their offering. As a result, standardization, or provider-centric homogenization, is not sufficient.

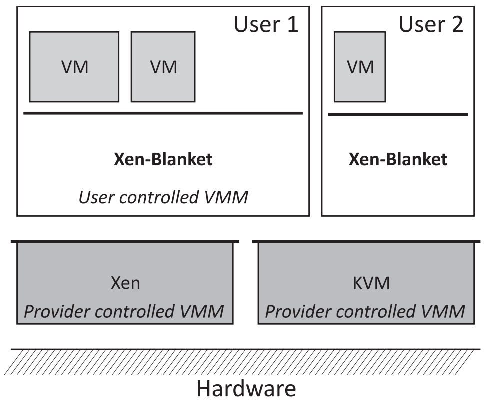

Figure 1. The Xen-Blanket, completely controlled by the user, provides a homogenization layer across heterogeneous cloud providers without requiring any additional support from the providers.

User-centric homogenization, in contrast, enables cloud users to homogenize the cloud and customize it to match their needs. It allows users to select their own VM image format and services, then transform every cloud to support it. The user is not tied into a "least common denominator" of functionality, but quite the opposite: even completely customized services and image formats can be deployed. The user can then develop management tools that work for *their* VMs across *their* (now homogenized) cloud. For example, a user can experiment with new features like Remus [10] across clouds and perhaps achieve high availability even in the case of an entire provider failing.

Finally, any system that implements user-centric homogenization must be immediately and universally deployable. A system that enables user-centric homogenization cannot be dependent on emerging features that are not standard across clouds, such as the low overhead nested virtualization solution proposed in the Turtles Project [2].

3. The Xen-Blanket

The Xen-Blanket leverages nested virtualization to form a *Blanket layer*, or a second layer of virtualization software that provides a user-centric homogeneous cloud interface, as depicted in Figure 1. A Blanket layer embodies three important concepts. First, the *bottom half* of the Blanket layer communicates with a variety of underlying hypervisor interfaces. No modifications are expected or required to the underlying hypervisor. Second, the *top half* of the Blanket layer exposes a single VM interface to Blanket (second-layer) guests such that a single guest image can run on any cloud without modifications. Third, the Blanket layer is completely under the control of the user, so functionality typically implemented by providers in the hypervisor, such as live VM migration, can be implemented in the Blanket layer.

The bottom half of the Xen-Blanket ensures that the Xen-Blanket can run across a number of different clouds without requiring changes to the underlying cloud system or hypervisor. The bottom half is trivial if the following two assumptions hold on all underlying clouds. First, if device I/O is emulated, then the Blanket hypervisor does not need to be aware of the underlying hypervisor's paravirtualized I/O interfaces. Second, if hardware-assisted full virtualization for x86 (called HVM in Xen terminology) is available, then the Blanket hypervisor can run unmodified. However, these assumptions limit the number of clouds that the Blanket layer can cover; for example, we are not aware of any public cloud that satisfies both assumptions.

The Xen-Blanket relaxes the emulated device assumption by interfacing with a variety of underlying cloud paravirtualized device I/O implementations. Paravirtualized device I/O has proved essential for performance and is required by some clouds, such as Amazon EC2. However, there is currently no standard paravirtualized device I/O interface. For example, older Xen-based clouds, including Amazon EC2, require device drivers to communicate with Xen-specific subsystems, such as the XenBus and XenStore, whereas KVM-based systems expect device drivers to interact with the hypervisor through `virtio` interfaces. The Xen-Blanket supports such non-standard interfaces by modifying the bottom half to contain cloud-specific *Blanket drivers*.

On the other hand, the Xen-Blanket does rely on support for hardware-assisted full virtualization for x86 on all clouds. Currently, this assumption somewhat limits deployment opportunities. For example, a large fraction of both Amazon EC2 and Rackspace instances expose paravirtualized, not HVM interfaces, with Amazon EC2 only offering an HVM interface to Linux guests in 4XL-sized cluster instances. EC2 does, however, expose an HVM interface to other sized instances running Windows, which we believe can also be converted to deploy the Xen-Blanket. Further efforts to relax the HVM assumption are discussed as future work in Section 7.

The top half of the Blanket layer exposes a consistent VM interface to (Blanket) guests. Guest VMs therefore do not need any modifications in order to run on a number of different clouds. In order to maximize the number of clouds that the Xen-Blanket can run on, the top half of the Xen-Blanket does not depend on state of the art nested virtualization interfaces (e.g., the Turtles Project [2]). The Xen-Blanket instead relies on other x86 virtualization techniques, such as paravirtualization or binary translation. For our prototype Blanket layer implementation we chose to adopt the popular open-source Xen hypervisor, which uses paravirtualization techniques when virtualization hardware is not available. The Xen-Blanket subsequently inherits the limitations of paravirtualization, most notably the inability to run unmodified

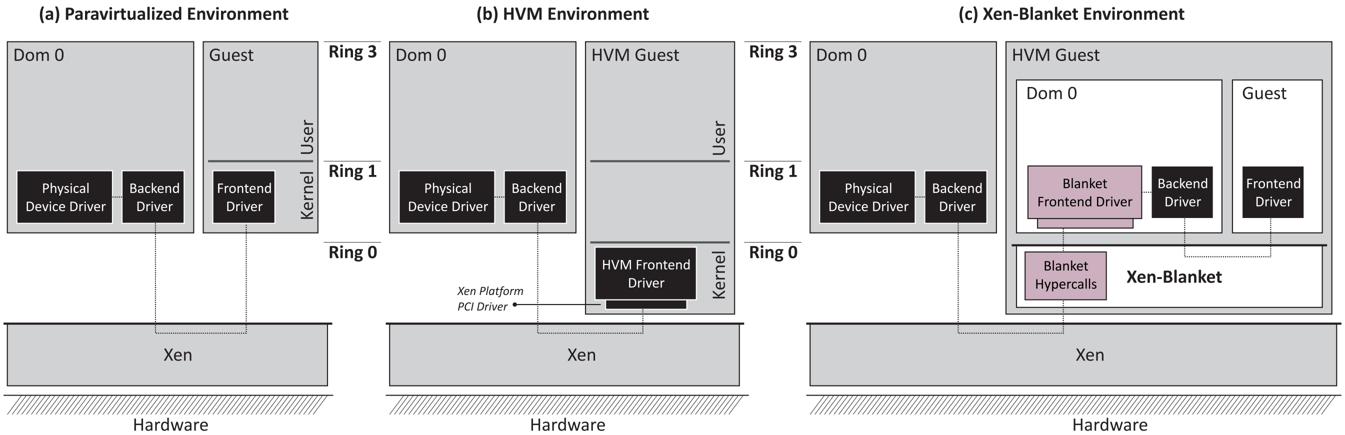

Figure 2. Guests using paravirtualized devices implement a front-end driver that communicates with a back-end driver (a). In HVM environments, a Xen Platform PCI driver is required to set up communication with the back-end (b). The Xen-Blanket modifies the HVM front-end driver to become a *Blanket driver*, which, with support of *Blanket hypercalls*, runs in hardware protection ring 1, instead of ring 0 (c).

operating systems, such as Microsoft Windows.[1] However, this limitation is not fundamental. A Blanket layer can be constructed using binary translation (e.g., a VMWare [21]-Blanket), upon which unmodified operating systems would be able to run. Blanket layers can also be created with other interfaces, such as Denali [25], alternate branches of Xen, or even customized hypervisors developed from scratch.

The Xen-Blanket inherits services that are traditionally located in the hypervisor or privileged management domains and allows the user to run or modify them. For instance, users can issue `xm` commands from the Xen-Blanket. Users can co-locate VMs [29] on a single Xen-Blanket instance, share memory pages between co-located VMs [16, 23], and oversubscribe resources [28]. If Xen-Blanket instances on different clouds can communicate with each other, live VM migration or high availability [10] across clouds become possible.

4. Blanket Drivers

The Xen-Blanket contains Blanket drivers for each of the heterogeneous interfaces exposed by today's clouds. In practice, the drivers that must be implemented are limited to dealing with paravirtualized device interfaces for network and disk I/O. As described in Section 3, Blanket drivers reside in the bottom half of the Xen-Blanket and are treated by the rest of the Xen-Blanket as drivers interacting with physical hardware devices. These "devices" are subsequently exposed to guests through a consistent paravirtualized device interface, regardless of which set of Blanket drivers was instantiated.

This section is organized as follows: we present background on how paravirtualized devices work on existing clouds. Then, we describe the detailed design and implementation of Blanket drivers. Finally, we conclude with a discussion of hypervisor optimizations for the Xen-Blanket and a discussion of the implications of evolving virtualization support in hardware and software.

4.1 Background

To understand Blanket drivers, we first give some background as to how paravirtualized device drivers work in Xen-based systems.[2] First, we describe device drivers in a fully paravirtualized Xen, depicted in Figure 2(a). The Xen-Blanket uses paravirtualization techniques in the Blanket hypervisor to provide guests with a homogeneous interface to devices. Then, we describe paravirtualized device drivers for hardware assisted Xen (depicted in Figure 2(b)), an underlying hypervisor upon which the Xen-Blanket successfully runs.

Xen does not contain any physical device drivers itself; instead, it relies on device drivers in the operating system of a privileged guest VM, called Domain 0, to communicate with the physical devices. The operating system in Domain 0 multiplexes devices, and offers a paravirtualized device interface to guest VMs. The paravirtualized device interface follows a *split driver* architecture, where the guest runs a *front-end* driver that is paired with a *back-end* driver in Domain 0. Communication between the front-end and back-end driver is accomplished through shared memory ring buffers and an event mechanism provided by Xen. Both the guest and Domain 0 communicate with Xen to set up these communication channels.

In hardware assisted Xen, or HVM Xen, paravirtualized device drivers are called PV-on-HVM drivers. Unlike paravirtualized Xen, guests on HVM Xen can run unmodified,

[1] Despite the limitations of paravirtualization and the increasingly superior performance of hardware assisted virtualization, paravirtualization remains popular. Many cloud providers, including Amazon EC2 and Rackspace, continue to offer paravirtualized Linux instances.

[2] A discussion of the paravirtualized drivers on KVM, which are similar, is postponed to the end of Section 4.2.

so by default, communication channels with Xen are not initialized. HVM Xen exposes a *Xen platform PCI device*, which acts as a familiar environment wherein shared memory pages are used to communicate with Xen and an IRQ line is used to deliver events from Xen. So, in addition to a front-end driver for each type of device (e.g. network, disk), an HVM Xen guest also contains a Xen platform PCI device driver. The front-end drivers and the Xen platform PCI driver are the only Xen-aware modules in the HVM guest.

4.2 Design & Implementation

The Xen-Blanket consists of a paravirtualized Xen inside of either a HVM Xen or KVM guest. We will center the discussion around Blanket drivers for Xen, and discuss the conceptually similar Blanket drivers for KVM at the end of this subsection. Figure 2(c) shows components of Blanket drivers. The Blanket layer contains both a Xen hypervisor as well as a privileged *Blanket Domain 0*. Guest VMs are run on top of the Blanket layer, each containing standard paravirtualized front-end device drivers. The Blanket Domain 0 runs the corresponding standard back-end device drivers. The back-end drivers are multiplexed into the Blanket drivers, which act as set of front-end drivers for the underlying hypervisors.

There are two key implementation issues that prohibit standard PV-on-HVM front-end drivers from acting as Blanket drivers.[3] First, the Xen hypercalls required to bootstrap a PV-on-HVM PCI platform device cannot be performed from the Blanket Domain 0 hosting the Blanket drivers because the Blanket Domain 0 does not run with the expected privilege level of an HVM guest OS. Second, the notion of a physical address in the Blanket Domain 0 is not the same as the notion of a physical address in a native HVM guest OS.

Performing Hypercalls

Typically, the Xen hypervisor proper runs in hardware protection ring 0, while Domain 0 and other paravirtualized guests run their OS in ring 1 with user spaces in ring 3. HVM guests, on the other hand, are designed to run unmodified, and can use non-root mode from the hardware virtualization extensions to run the guest OS in ring 0 and user space in ring 3. In the Xen-Blanket, in non-root mode, the Blanket Xen hypervisor proper runs in ring 0, while the Blanket Domain 0 runs in ring 1, and user space runs in ring 3 (Figure 2(c)).

In normal PV-on-HVM drivers, hypercalls, in particular `vmcall` instructions, are issued from the OS in ring 0. In the Xen-Blanket, however, Blanket drivers run in the OS of the Blanket Domain 0 in ring 1. The `vmcall` instruction must be issued from ring 0. We overcome this by augmenting the second-layer Xen to contain *Blanket hypercalls* that issue their own hypercalls to the underlying Xen on behalf of the Blanket Domain 0.

[3] Our implementation also required renaming of some global variables and functions to avoid namespace collisions with the second-layer Xen when trying to communicate with the bottom-layer Xen.

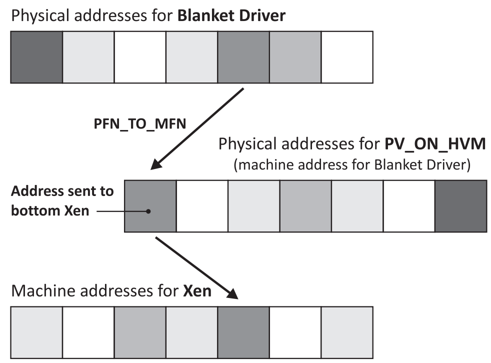

Figure 3. The PV-on-HVM drivers can send physical addresses to the underlying Xen, whereas the Blanket drivers must first convert physical addresses to machine addresses.

Physical Address Translation

Guest OSs running on top of paravirtualized Xen, including Domain 0, have a notion of physical frame numbers (PFNs). The PFNs may or may not match the actual physical frame numbers of the machine, called machine frame numbers (MFNs). The relationship between these addresses is shown in Figure 3. However, the guest can access the mapping between PFNs and MFNs, in case it is necessary to use a real MFN, for example, to utilize DMA from a device. HVM guests are not aware of PFNs vs. MFNs. Instead, they only use physical frame numbers and any translation necessary is done by the underlying hypervisor.

For this reason, PV-on-HVM device drivers pass physical addresses to the underlying hypervisor to share memory pages with the back-end drivers. In the Xen-Blanket, however, the MFN from the Blanket Domain 0's perspective, and thus the Blanket drivers', matches the PFN that the underlying hypervisor expects. Therefore, Blanket drivers must perform a PFN-to-MFN translation before passing any addresses to the underlying hypervisor, either through hypercalls or PCI operations.

Blanket Drivers for KVM

The implementation of Blanket drivers for KVM is very similar. Paravirtualized device drivers in KVM use the `virtio` framework, in which a PCI device is exposed to guests, similar to the Xen platform PCI device. Unlike the Xen platform PCI device, all communication with the underlying KVM hypervisor can be accomplished as if communicating with a physical PCI device. In particular, no direct hypercalls are necessary, simplifying the implementation of Blanket drivers. The only modifications required to run `virtio` drivers in the Xen-Blanket are the addition of PFN-to-MFN translations.

4.3 Hypervisor Optimizations

The Xen-Blanket runs in non-root mode in an HVM guest container. As virtualization support improves, the performance of software running in non-root mode becomes close to running on bare metal. For example, whereas page table manipulations would cause a `vmexit`, or trap, on early versions of Intel VT-x processors, a hardware feature called extended page tables (EPT) has largely eliminated such traps. However, some operations continue to generate traps, so designing the Blanket layer to avoid such operations can often provide a performance advantage.

For example, instead of flushing kernel pages from the TLB on every context switch, the x86 contains a bit in the `cr4` control register called the "Page Global Enable" (PGE). Page Global Enable allows certain pages to be mapped as "global" so that they do not get flushed automatically. Xen enables then disables the PGE bit in order to flush the global TLB entries before doing a domain switch between guests. Unfortunately, these `cr4` operations each cause `vmexits` to happen, generating high overhead for running Xen in an HVM guest. By not using PGE and instead flushing all pages from the TLB on a context switch, `vmexits` are avoided, because of the EPT processor feature in non-root mode.

4.4 Implications of Future Hardware and Software

As discussed above the virtualization features of the hardware, such as EPT, can have a profound effect on the performance of the Xen-Blanket. Further improvements to the HVM container, such as the interrupt path, may eventually replace hypervisor optimizations and workarounds or enable even better performing Blanket layers.

Other hardware considerations include thinking about non-root mode as a place for virtualization. For example, features that aided virtualization before hardware extensions became prevalent, such as memory segmentation, should not die out. Memory segmentation is a feature in 32 bit x86 processors that paravirtualized Xen leverages to protect Xen, the guest OS, and the guest user space in the same address space to minimize context switches during system calls. The 64 bit x86_64 architecture has dropped support for segmentation except when running in 32 bit compatibility mode. Without segmentation, two address spaces are needed to protect the three contexts from each other, and two context switches are required on each system call, resulting in performance loss.

On the software side, support for nested virtualization of unmodified guests [2] may begin to be adopted by cloud providers. While this development could eventually lead to fully virtualized Blankets such as a KVM-Blanket, relying on providers to deploy such a system is provider-centric: every cloud must incorporate such technology before a KVM-Blanket becomes feasible across many clouds. It may be possible, however, for exposed hardware virtualization extensions to be leveraged as performance accelerators for a system like the Xen-Blanket.

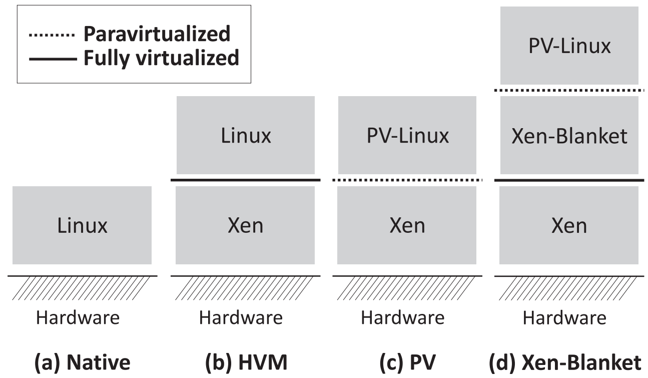

Figure 4. We run benchmarks on four different system configurations in order to examine the overhead caused by the Xen-Blanket. *Native* represents an unmodified CentOS 5.4 Linux. *HVM* represents a standard single-layer Xen-based virtualization solutions using full, hardware-assisted virtualization. *PV* represents a standard single-layer Xen-based virtualization solutions using paravirtualization. *Xen-Blanket* consists of a paravirtualized setup inside of our Xen-Blanket HVM guest.

5. Evaluation

We have built Blanket drivers and deployed the Xen-Blanket on two underlying hypervisors, across three resource providers. In this section, we first examine the overhead incurred by the Xen-Blanket. Then, we describe how increased flexibility resulting from a user-centric homogenization layer can result in significant cost savings—47% of the cost per hour—on today's clouds, despite overheads.

5.1 Overhead

Intuitively, we expect some amount of degraded performance from the Xen-Blanket due to the overheads of running a second-layer of virtualization. We compare four different scenarios, denoted by *Native*, *HVM*, *PV*, and *Xen-Blanket* (Figure 4). The Native setup ran an unmodified CentOS 5.4 Linux. The next two are standard single-layer Xen-based virtualization solutions using full, hardware-assisted virtualization (HVM, for short) or paravirtualization (PV, for short), respectively. The fourth setup (Xen-Blanket) consists of a paravirtualized setup inside an HVM guest.[4] All experiments in this subsection were performed on a pair of machines connected by a 1 Gbps network, each with two six-core 2.93 GHz Intel Xeon X5670 processors,[5] 24 GB of memory, and four 1 TB disks. Importantly, the virtualization capabilities of the Xeon X5670 include extended page table support (EPT), enabling a guest OS to modify page tables without generating `vmexit` traps. With the latest hardware virtualization support, HVM is expected to outperform

[4] We have also run experiments on KVM with comparable results, but focus on a single underlying hypervisor for a consistent evaluation.

[5] Hyperthreading causes the OS to perceive 24 processors on the system.

	Native	HVM	PV	Xen-Blanket
Processes (μs)				
null call	0.19	0.21	0.36	0.36
null I/O	0.23	0.26	0.41	0.41
stat	0.85	1.01	1.19	1.18
open/close	1.33	1.43	1.84	1.86
slct TCP	2.43	2.79	2.80	2.86
sig inst	0.25	0.39	0.54	0.53
sig hndl	0.90	0.79	0.94	0.94
fork proc	67	86	220	258
exec proc	217	260	517	633
sh proc	831	1046	1507	1749
Context Switching (μs)				
2p/0K	0.40	0.55	2.85	3.07
2p/16K	0.44	0.57	3.03	3.46
2p/64K	0.45	0.66	3.18	3.46
8p/16K	0.74	0.85	3.60	4.00
8p/64K	1.37	1.18	4.14	4.53
16p/16K	1.05	1.10	3.80	4.14
16p/64K	1.40	1.22	4.08	4.47
File & Virtual Memory (μs)				
0K file create	4.61	4.56	4.99	4.97
0K file delete	3.03	3.18	3.19	3.14
10K file create	14.4	18.1	19.9	28.8
10K file delete	6.17	6.02	6.01	6.08
mmap latency	425.0	820.0	1692.0	1729.0
prot fault	0.30	0.28	0.38	0.40
page fault	0.56	0.99	2.00	2.10

Table 1. The Xen-Blanket achieves performance within 3% of PV for simple `lmbench` operations, but incurs overhead up to 30% for file creation microbenchmarks.

PV, because of reduced hypervisor involvement. Therefore, since the Xen-Blanket setup contains a PV setup, PV can be roughly viewed as a best case for the Xen-Blanket.

System Microbenchmarks

To examine the performance of individual operations, such as null system calls, we ran `lmbench` in all setups. In order to distinguish the second-layer virtualization overhead from CPU contention, we ensure that one CPU is dedicated to the guest running the benchmark. To clarify, one VCPU backed by one physical CPU is exposed to the guest during single-layer virtualization experiments, whereas the Xen-Blanket system receives two VCPUs backed by two physical CPUs: one is reserved for the second-layer Domain 0 (see Figure 2(c)), and the other one for the second-layer guest.

Table 1 shows the results from running `lmbench` in each of the setups. For simple operations like a null syscall, the performance of the Xen-Blanket is within 3% of PV, but even PV is slower than native or HVM. This is because a syscall in any paravirtualized system first switches into (the top-most) Xen before being bounced into the guest OS. We stress that, for these operations, nesting Xen does not introduce additional overhead over standard paravirtualization. All context switch benchmarks are within 12.5% of PV,

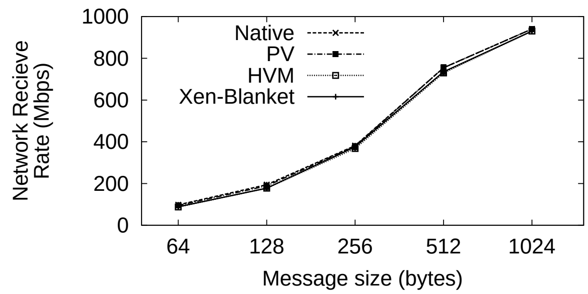

Figure 5. Network I/O performance on the Xen-Blanket is comparable to a single layer of virtualization.

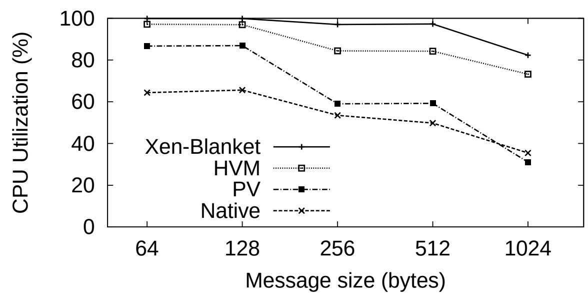

Figure 6. CPU utilization while receiving network I/O on the Xen-Blanket is within 15% of a single layer of virtualization.

with most around 8% of PV. Eliminating `vmexits` caused by the second-layer Xen is essential to achieve good performance. For example, if the second-layer Xen uses the `cr4` register on every context switch, overheads increase to 70%. Worse, on processors without EPT, which issue `vmexits` much more often, we measured overheads of up to 20×.

Blanket Drivers

Device I/O is often a performance bottleneck even for single-layer virtualized systems. Paravirtualization is essential for performance, even in fully-virtualized environments. To examine the network and disk performance of the Xen-Blanket, we assign each of the configurations one VCPU (we disable all CPUs except for one in the native case). Figure 5 and Figure 6 show the UDP receive throughput and the corresponding CPU utilization[6] under various packet sizes. We use `netperf` for the throughput measurement and `xentop` in the underlying Domain 0 to measure the CPU utilization of the guest (or Xen-Blanket and guest). The CPU utilization of the native configuration is determined using `top`. Despite the two layers of paravirtualized device interfaces, guests running on the Xen-Blanket can still match the network throughput of all other configurations for all packet sizes, and receive network traffic at full capacity over a 1 Gbps

[6] Errorbars are omitted for clarity: all CPU utilization measurements were within 1.7% of the mean.

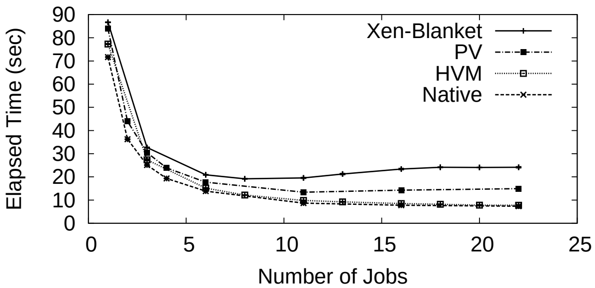

Figure 7. The Xen-Blanket can incur up to 68% overhead over PV when completing a `kernbench` benchmark.

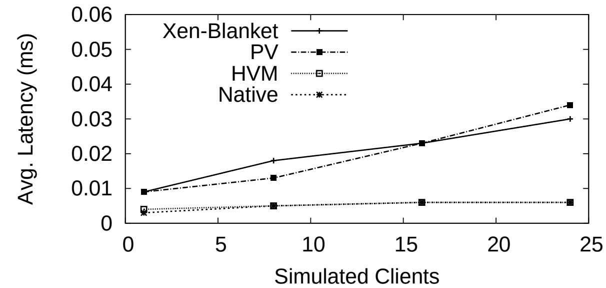

Figure 9. The average latency for ReadX operations during the `dbench` benchmark for Xen-Blanket remains comparable to PV.

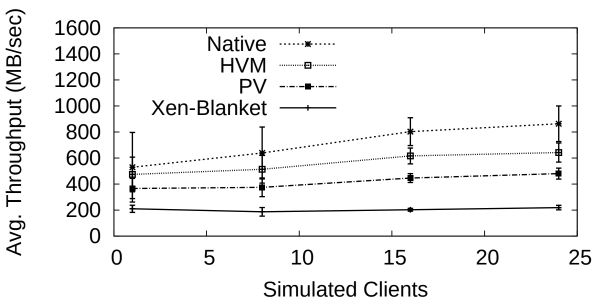

Figure 8. The Xen-Blanket can incur up to 55% overhead over PV when performing the `dbench` filesystem benchmark.

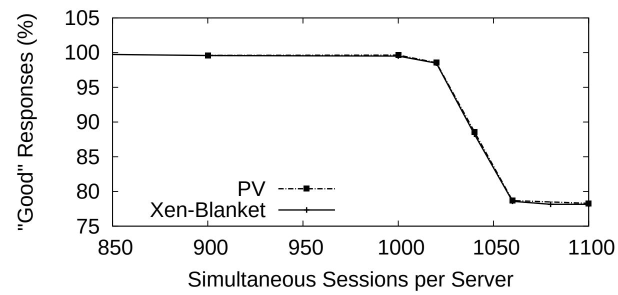

Figure 10. The Xen-Blanket performs just as well as PV for the SPECweb2009 macrobenchmark.

link. The Xen-Blanket does incur more CPU overhead because of the extra copy of packets in the Blanket layer. We also ran `dd` to get a throughput measure of disk I/O. System caches at all layers were flushed before reading 2GB of data from the root filesystem. Native achieved read throughput of 124.6 MB/s, HVM achieved 86.3 MB/s, PV achieved 76.6 MB/s, and the Xen-Blanket incurred an extra overhead of 12% over PV, with disk read throughput of 67.6 MB/s.

Macrobenchmarks

Macrobenchmarks are useful for demonstrating the overhead of the system under more realistic workloads. For these experiments, we dedicate 2 CPUs and 8 GB of memory to the lowest layer Domain 0. The remaining 16 GB of memory and 22 CPUs are allocated to single layer guests. In the case of the Xen-Blanket, we allocate 14 GB of memory and 20 CPUs to the Blanket guest, dedicating the remainder to the Blanket Domain 0. Unlike the microbenchmarks, resource contention does contribute to the performance measured in these experiments.

`kernbench`[7] is a CPU throughput benchmark that consists of compiling the Linux kernel using a configurable number of concurrent jobs. Figure 7 shows the elapsed time for the kernel compile. With a single job, the Xen-Blanket stays within 5% of PV, however, performance falls to about 68% worse than PV for high concurrency. The performance loss here can be attributed to a high number of `vmexits` due to APIC (Advanced Programmable Interrupt Controller) operations to send inter-processor-interrupts (IPIs) between VCPUs. Despite this overhead, the flexibility of the Xen-Blanket enables reductions in cost, as described in Section 5.2.

`dbench`[8] is a filesystem benchmark that generates load on a filesystem based on the standard `NetBench` benchmark. Figure 8 show the average throughput during load imposed by various numbers of simulated clients. Figure 9 shows the average latency for ReadX operations, where ReadX is the most common operation during the benchmark. PV and the Xen-Blanket both experience significantly higher latency than HVM. The advantage of HVM can be attributed to the advantages of hardware memory management because of extended page tables (EPT). The Xen-Blanket incurs up to 55% overhead over PV in terms of throughput, but the latency is comparable.

Finally, we ran the banking workload of SPECweb2009 for a web server macrobenchmark. For each experiment, a client workload generator VM running on another machine connected by a 1 Gbps link drives load for a server that runs PHP scripts. As SPECweb2009 is a Web server benchmark, the back-end database is simulated. A valid SPECweb2009

[7] version 0.50

[8] version 4.0

Type	CPU (ECUs)	Memory (GB)	Disk (GB)	Price ($/hr)
Small	1	1.7	160	0.085
Cluster 4XL	33.5	23	1690	1.60
Factor	33.5×	13.5×	10×	18.8×

Table 2. The resources on Amazon EC2 instance types do not scale up uniformly with price. The user-centric design of Xen-Blanket allows users to exploit this fact.

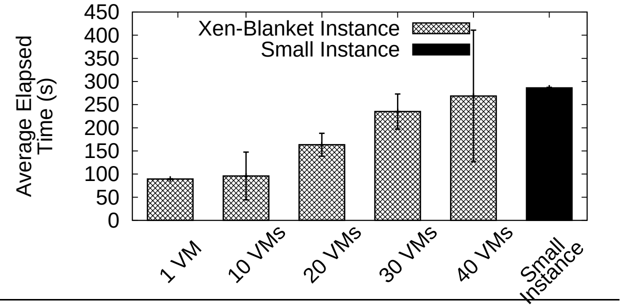

Figure 11. The Xen-Blanket gives the flexibility to oversubscribe such that each of 40 VMs on a single 4XL instance can simultaneously complete compilation tasks in the same amount of time as a small instance.

run involves 95% of the page requests to compete under a "good" time threshold (2s) and 99% of the requests to be under a "tolerable" time threshold (4s). Figure 10 shows the number of "good" transactions for various numbers of simultaneous sessions. VMs running in both PV and Xen-Blanket scenarios can support an identical number of simultaneous sessions.[9] This is because the benchmark is I/O bound, and the Blanket drivers ensure efficient I/O for the Xen-Blanket. The SPECweb2009 instance running in the Xen-Blanket does utilize more CPU to achieve the same throughput, however: average CPU utilization rises from 4.3% to 5.1% under 1000 simultaneous client sessions.

5.2 User-defined Oversubscription

Even though running VMs in the Xen-Blanket does incur overhead, its user-centric design gives a cloud user the flexibility to utilize cloud resources substantially more efficiently than possible on today's clouds. Efficient utilization of cloud resources translates directly into monetary savings. In this subsection, we evaluate oversubscription on the Xen-Blanket instantiated within Amazon EC2 and find CPU-intensive VMs can be deployed for 47% of the cost of small instances.

Table 2 shows the pricing per hour on Amazon EC2 to rent a small instance or a quadruple extra large cluster compute instance (cluster 4XL). Importantly, while the cluster 4XL instance is almost a factor of 19 times more expensive than a small instance, some resources are greater than 19 times more abundant (e.g. 33.5 times more for CPU) while other resources are less than 19 times more abundant (e.g 10 times more for disk). This suggests that if a cloud user has a number CPU intensive VMs normally serviced as small instances, it may be more cost efficient to rent a cluster 4XL instance and oversubscribe the memory and disk. This is not an option provided by Amazon; however, the Xen-Blanket is user-centric and therefore gives the user the necessary control to implement such a configuration. A number of would-be small instances can be run on the Xen-Blanket within a cluster 4XL instance, using oversubscription to reduce the price per VM.

To illustrate this point, we ran a CPU-intensive macrobenchmark, `kernbench`, simultaneously in a various numbers of VMs running inside a single cluster 4XL instance with the Xen-Blanket. We also ran the benchmark inside a small EC2 instance for a comparison point. The benchmark was run without concurrency in all instances for consistency, because a small instance on Amazon only has one VCPU. Figure 11 shows the elapsed time to run the benchmark in each of these scenarios. Each number of VMs on the Xen-Blanket corresponds to a different monetary cost. For example, to run a single VM, the cost is $1.60 per hour. 10 VMs reduce the cost per VM to $0.16 per hour, 20 VMs to $0.08 per VM per hour, 30 VMs to $0.06 per VM per hour, and 40 VMs to $0.04 per VM per hour. Running a single VM, the benchmark completes in 89 seconds on the Xen-Blanket, compared to 286 seconds for a small instance. This is expected, because the cluster 4XL instance is significantly more powerful than a small instance. Furthermore, the average benchmark completion time for even 40 VMs remains 33 seconds faster than for a small instance. Since a small instance costs $.085 per VM per hour, this translates to 47% of the price per VM per hour. It should be noted, however, that the variance of the benchmark performance significantly increases for large numbers of VMs on the same instance.

In some sense, the cost benefit of running CPU intensive instances inside the Xen-Blanket instead of inside small instances simply exploits an artifact of Amazon's pricing scheme. However, other benefits from oversubscription are possible, especially when considering VMs that have uncorrelated variation in their resource demands. Every time one VM experiences a burst of resource usage, others are likely quiescent. If VMs are not co-located, each instance must operate with some resources reserved for bursts. If VMs are co-located, on the other hand, a relatively small amount of resources can be shared to be used for bursting behavior, resulting in less wasted resources.

Co-location of VMs also affect the performance of enterprise applications, made up of a number of VMs that may heavily communicate with one another [20]. To demonstrate the difference that VM placement can make to network performance, we ran the `netperf` TCP benchmark between

[9] PV and Xen-Blanket run the same VM and thus the same configuration of this complex benchmark. We omit a comparison with native and HVM to avoid presenting misleading results due to slight configuration variation.

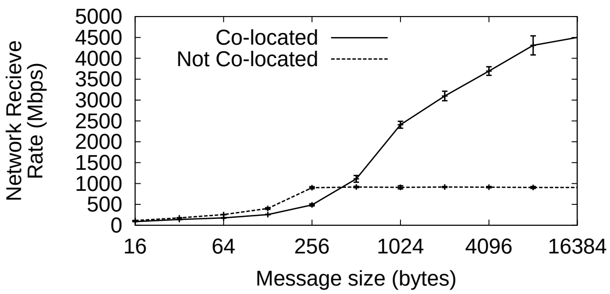

Figure 12. Co-location of VMs to improve network bandwidth is another simple optimization made possible by the user-centric approach of the Xen-Blanket.

two VMs. In the first setup, the VMs were placed on two different physical servers on the same rack, connected by a 1 Gbps link. In the second, the VMs were co-located on the same physical server. Figure 12 shows the network throughput. The co-located servers are not limited by the network hardware connecting the physical machines. By enabling co-location, the Xen-Blanket can increase inter-VM throughput by a factor of 4.5. This dramatic result is without any modification to the VMs. The user-centric design of the Xen-Blanket enables other optimization opportunities, including CPU bursting, page sharing and resource oversubscription, that can offset the inherent overhead of the approach.

6. Experience with Multi-Cloud Migration

The Xen-Blanket homogenizes and simplifies the process of migrating a VM between two clouds managed by two different providers. While it is possible to migrate VMs between multiple clouds today, the process is cloud-specific and fundamentally limited. For example, it is currently impossible to live migrate [8, 18] a VM between cloud providers. We give a qualitative comparison to illustrate the difficulty faced in migrating a Xen VM from our private Xen environment to Amazon EC2 with and without the Xen-Blanket. We also show how one can reintroduce live migration across multi-clouds using the Xen-Blanket. In our experiment, we use a VM housing a typical legacy LAMP-based[10] application that contained non-trivial customizations and approximately 20 GB of user data.

6.1 Non-Live Multi-Cloud Migration

Figure 13 summarizes the four steps involved in a migration: *modifying* the VM's disk image to be compatible with EC2, *bundling* or compressing the image to be sent to EC2, *uploading* the bundled image, and *launching* the image at the new location. In both scenarios, bundling, uploading and launching took one person about 3 hrs. However, the modify step caused the scenario without the Xen-Blanket to be much more time consuming: 24 hrs additional work as compared to no additional work with the Xen-Blanket.

[10] Linux, Apache, MySQL, and PHP

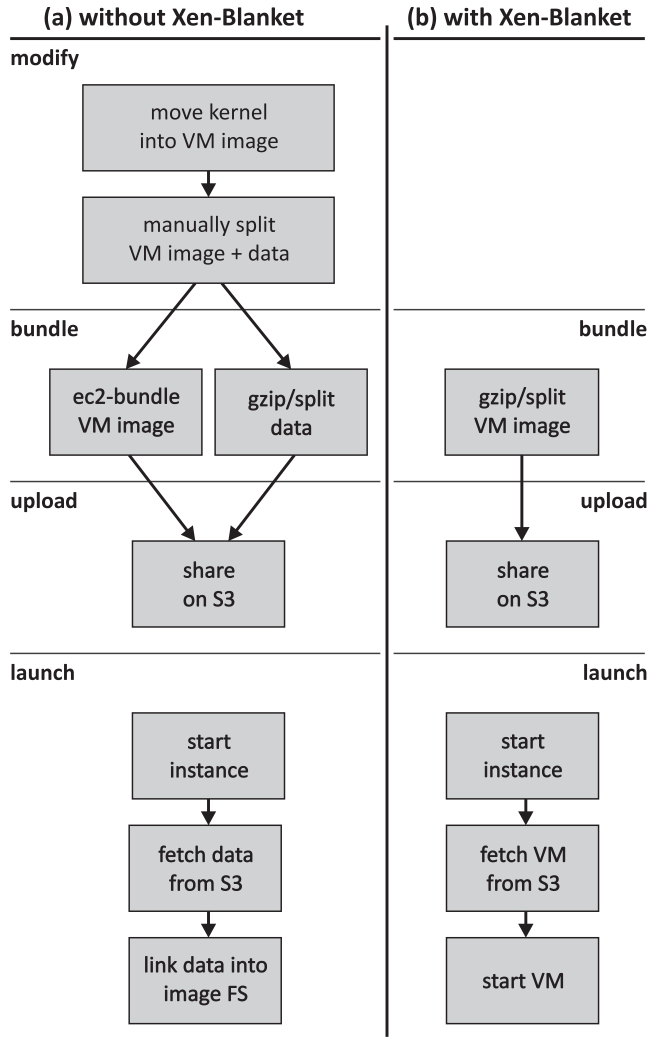

Figure 13. Comparison between the steps it takes to migrate (offline) an image into Amazon's EC2 with and without the Xen-Blanket

Migrating a VM image from our private setup to Amazon EC2 is relatively straightforward given the Xen-Blanket. No image modifications are required, so the process begins with bundling, or preparing the image to upload for use in EC2. The image was compressed with `gzip`, split into 5 GB chunks for Amazon's Simple Storage Service (S3), and uploaded. Then, we started an EC2 instance running the Xen-Blanket, retrieved the disk image from S3, concatenated the pieces of the file, and unzipped the image. The VM itself was created using standard Xen tools, such as `xm create`.

Without the Xen-Blanket, there currently exists a EC2-specific process to create an Amazon Machine Image (AMI) from an existing Xen disk image, roughly matching the bundle, upload, and launch steps. Before that, two modifications were required to our VM image. First, we had to modify the image to contain the kernel because no compatible kernel was offered by EC2.[11] This task was complicated by the fact that our private Xen setup did not have the correct tools to boot the kernel within the image. Second, we had to shrink our 40 GB disk image to fit within the 10 GB image limit on EC2. This involved manually examining the disk image in order to locate, copy, and remove a large portion of the application data, then resizing the VM's filesystem and image. After the modifications were complete, we used an AMI tool called `ec2-bundle-image` to split the VM image into pieces and then compressed, split and uploaded the relocated data to S3. We then started an EC2 instance with our new AMI, configured it to mount a disk, and reintegrated the user data from S3 into the filesystem.

It should be noted that subsequent launches of the migrated VMs do not require all of the steps outlined above. However, if a modified or updated version of the VM is released, the entire process must be redone. Even worse, we expect migrating to other clouds to be similarly arduous and provider-specific, if possible at all. In contrast, using the Xen-Blanket, the migration process will always be the same, and can be reduced to a simple remote copy operation.

6.2 Live Multi-Cloud Migration

Live migration typically relies on memory tracing: a hypervisor-level technique. Such techniques are not available across clouds.[12] The Xen-Blanket enables immediate implementation of live migration across cloud providers. We have experimented with live migration between an enterprise cloud and Amazon EC2. We note that since live migration between two clouds is not currently possible without provider support or the Xen-Blanket, we do not have a comparison point to present.

Beyond the ability to implement hypervisor-level features like memory tracing, there are two key challenges to implement live multi-cloud migration on the Xen-Blanket. First, Xen-Blanket instances in different clouds are in different IP subnets, causing communication issues before and after Blanket guest migrations. Second, Xen-Blanket instances in different clouds do not share network attached storage, which is often assumed for live VM migration.

To address the networking issues, each Xen-Blanket instance runs a virtual switch in Domain 0 to which the virtual network interfaces belonging to Blanket guest VMs are attached. A layer-2 tunnel connects the virtual switches across the Internet. The result is that VMs on either of the two Xen-Blanket instances appear to be sharing a private LAN. A few basic network services are useful to introduce onto the virtual network. A gateway server VM can be run with two virtual network interfaces: one attached to the virtual switch and the virtual network; the other attached to the externally visible interface of the Xen-Blanket instance. The gateway server VM, shown in Figure 14, runs `dnsmasq` as a lightweight DHCP and DNS server.

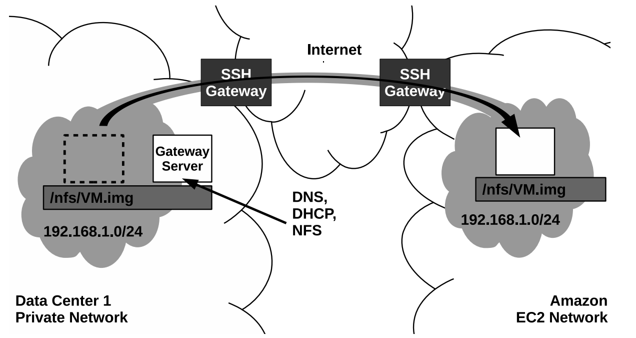

Figure 14. Xen-Blanket instances are connected with a layer-2 tunnel, while a gateway server VM provides DNS, DHCP and NFS to the virtual network, eliminating the communication and storage barriers to multi-cloud live migration.

Once VMs on the Xen-Blanket can communicate, the storage issues can be addressed with a network file system, such as NFS. NFS is useful for live VM migration because it avoids the need to transfer the entire disk image of the VM at once during migration. In our setup, the gateway server VM also runs an NFS server. The NFS server exports files onto the virtual network and is mounted by the Domain 0 of each Xen-Blanket instance. Both Xen-Blanket instances mount the NFS share at the same location. Therefore, during VM migration, the VM root filesystem image can always be located at the same filesystem location, regardless of the physical machine.

With Xen-Blanket VMs able to communicate, maintain their network addresses, and access storage within either cloud, live VM migration proceeds by following the typical procedure in the Blanket hypervisor. However, while we have successfully live-migrated a VM from an enterprise cloud to Amazon EC2 and back, this is simply a proof-of-concept. It is clearly inefficient to rely on a NFS disk image potentially residing on another cloud instead of a local disk. Moreover, the layer-2 tunnel only connects two machines. More sophisticated wide-area live migration techniques exist [5], that can, as future work, be implemented and evaluated on the Xen-Blanket.

7. Future Work

Two limitations of the current Xen-Blanket are the inability to support unmodified guest OSs (such as Microsoft Windows) and the reliance on fully virtualized (HVM) containers. To address the first limitation, as discussed earlier, unmodified guests can be supported with binary translation, for example a VMWare-Blanket. The performance implications of such a system is a subject of future research. The second

[11] Until recently, Amazon EC2 only allowed a limited selection of a few standard kernels and initial ramdisks for use outside the image. Luckily, in July 2010, Amazon EC2 began to support kernels stored within the image.

[12] While some providers expose an interface for users to use live migration within their own cloud, as with other provider-centric approaches, standardization may take years.

limitation can be addressed by allowing a version of the Xen-Blanket to run on a paravirtualized interface, while continuing to export a homogeneous interface to guests. Paravirtualizing the Blanket layer is technically feasible, but may encounter performance issues in the memory subsystem where hardware features such as Extended Page Tables (EPT) cannot be used and is another subject of future research.

More broadly, a user-centric, homogeneous Blanket layer enables future projects to examine features such as cross-provider live migration (see Section 6.2), high availability [10], and security [12] on one or more existing clouds. It also offers researchers an environment within which novel systems and hypervisor level experimentation can be performed. We also plan to research issues in running entire cloud stacks, such as Eucalyptus [14] or OpenStack [1], in nested environments and across multiple clouds.

8. Related Work

There are several techniques that exist today to deploy applications on multiple clouds, but none afford the user the flexibility or level of control of the Xen-Blanket. Conversely, there are also a number of existing systems that offer a user similar levels of control as the Xen-Blanket. However, none of these systems are able to be deployed on today's public clouds.

Nested virtualization is leveraged by the Xen-Blanket in order to allow a user to implement its own version of homogeneity, including hypervisor-level services. Graf and Roedel [15] and the Turtles Project [2] are pioneers of enabling nested virtualization with one or more levels of full virtualization, on AMD and Intel hardware, respectively. Berghmans [3] describes the performance of several nested virtualization environments. CloudVisor [30] explores nested virtualization in a cloud context, but for security, where the provider controls both layers. The Xen-Blanket sacrifices full nested virtualization for immediate deployment on existing clouds.

8.1 Multi-Cloud Deployments

Using tools from Rightscale [9], a user can create ServerTemplates, which can be deployed on a variety of clouds and utilize unique features of clouds without sacrificing portability. However, users are unable to homogenize the underlying clouds, particularly hypervisor-level services.

Middleware, such as IBM's Altocumulus [17] system homogenizes both IaaS clouds like Amazon EC2 and Platform as a Service (PaaS) clouds like Google App Engine into a PaaS abstraction across multiple clouds. However, without control at the IaaS (hypervisor) level, the amount of customization possible by the cloud user is fundamentally limited.

fos [24], deployed on EC2 today and potentially deployable across a wide variety of heterogeneous clouds, exposes a single system image instead of a VM interface. However, users must learn to program their applications for fos; the familiar VM interface and legacy applications contained within must be abandoned.

Eucalyptus [14] and AppScale [7] are open-source cloud computing systems that can enable private infrastructures to share an API with Amazon EC2 and Google App Engine respectively. However, the user cannot implement their own multi-cloud hypervisor-level feature. OpenStack [1] is another open-source implementation of an IaaS cloud, with the same limitation.

The RESERVOIR project [19] is a multi-cloud agenda in which two or more independent cloud providers create a *federated cloud*. A provider-centric approach is assumed; standardization is necessary before federation can extend beyond the testbed. With the Xen-Blanket, such an agenda could be applied across today's public clouds.

8.2 User-Centric Design and Control

OpenCirrus [6] is an initiative that aims to enable cloud targeted system level research—deploying a user-centric cloud—by allowing access to bare hardware, as in Emulab [26], in a number of dedicated data centers. However, OpenCirrus is not aimed at applying this ability to existing cloud infrastructures.

Cloud operating systems such as VMWare's vSphere [22] allow the administrator of a private cloud to utilize a pool of physical resources, while providing features like automatic resource allocation, automated failover, or zero-downtime maintenance. These features, which are examples of hypervisor-level services, cannot easily be integrated with current public cloud offerings.

Finally, the Xen-Blanket is an instantiation of our extensible cloud, or xCloud, proposal [27], which is influenced by work on extensible operating systems. For example, SPIN [4] allows extensions to be downloaded into the kernel safely using language features, while Exokernels [13] advocate hardware to be exposed to a library OS controlled by the user. However, these extensibility strategies are provider-centric, and unlikely to be incorporated in today's clouds. The combination of deployment focus and user-level control sets the Xen-Blanket apart from existing work.

9. Conclusion

Current IaaS clouds lack the homogeneity required for users to easily deploy services across multiple providers. We have advocated that instead of standardization, or provider-centric homogenization, cloud users must have the ability to homogenize the cloud themselves. We presented the Xen-Blanket, a system that enables user-centric homogenization of existing cloud infrastructures.

The Xen-Blanket leverages a second-layer Xen hypervisor—completely controlled by the user—that utilizes a set of provider-specific Blanket drivers to execute on top of existing clouds without requiring any modifications to the provider. Blanket drivers have been developed for both Xen and KVM based systems, and achieve high performance: network and disk throughput remain within 12% of paravirtualized drivers in a single-level paravirtualized guest. The Xen-Blanket is currently running on Amazon EC2, an enterprise cloud, and private servers at Cornell University. We have migrated VM images between the three different sites with no modifications to the images and performed live migration to and from Amazon EC2. We have exploited the user-centric nature of the Xen-Blanket to oversubscribe resources and save money on EC2, achieving a cost of 47% of the price per hour of small instances for 40 CPU-intensive VMs, despite the inherent overheads of nested virtualization.

We have only scratched the surface in terms of the applications and functionality made possible by user-centric homogenization, and the Xen-Blanket in particular. The Xen-Blanket project website is located at `http://xcloud.cs.cornell.edu/`, and the code for the Xen-Blanket is publicly available at `http://code.google.com/p/xen-blanket/`. We hope other projects adopt the Xen-Blanket and look forward to expanding the Xen-Blanket to cover even more underlying cloud providers.

Acknowledgments

This work was partially funded and supported by an IBM Faculty Award received by Hakim Weatherspoon, DARPA, NSF TRUST and NSF FIA. Also, this work was performed while Dan Williams was an intern at the IBM T. J. Watson Research Center in Hawthorne, NY. We would like to thank our shepherd, Andreas Haeberlen, and the anonymous reviewers for their comments.

References

[1] OpenStack. `http://www.openstack.org/`, Oct. 2010.

[2] M. Ben-Yehuda, M. D. Day, Z. Dubitzky, M. Factor, N. Har'El, A. Gordon, A. Liguori, O. Wasserman, and B.-A. Yassour. The turtles project: Design and implementation of nested virtualization. In *Proc. of USENIX OSDI*, Vancouver, BC, Canada, Oct. 2010.

[3] O. Berghmans. Nesting virtual machines in virtualization test frameworks. *Masters thesis, University of Antwerp*, May 2010.

[4] B. N. Bershad, S. Savage, P. Pardyak, E. G. Sirer, M. E. Fiuczynski, D. Becker, C. Chambers, and S. Eggers. Extensibility, safety and performance in the SPIN operating system. In *Proc. of ACM SOSP*, Copper Mountain, CO, Dec. 1995.

[5] R. Bradford, E. Kotsovinos, A. Feldmann, and H. Schiöberg. Live wide-area migration of virtual machines including local persistent state. In *Proc. of ACM VEE*, San Diego, CA, June 2007.

[6] R. Campbell, I. Gupta, M. Heath, S. Y. Ko, M. Kozuch, M. Kunze, T. Kwan, K. Lai, H. Y. Lee, M. Lyons, D. Milojicic, D. O'Hallaron, and Y. C. Soh. Open cirrus^{TM} cloud computing testbed: federated data centers for open source systems and services research. In *Proc. of USENIX HotCloud*, San Diego, CA, June 2009.

[7] N. Chohan, C. Bunch, S. Pang, C. Krintz, N. Mostafa, S. Soman, and R. Wolski. Appscale: Scalable and open appengine application development and deployment. In *Proc. of ICST CLOUDCOMP*, Munich, Germany, Oct. 2009.

[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 *Proc. of USENIX NSDI*, Boston, MA, May 2005.

[9] T. Clark. Rightscale. http://www.rightscale.com, 2010.

[10] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield. Remus: high availability via asynchronous virtual machine replication. In *Proc. of USENIX NSDI*, San Francisco, CA, Apr. 2008.

[11] Distributed Management Task Force, Inc. (DMTF). Open virtualization format white paper version 1.00. `http://http://www.dmtf.org/sites/default/files/standards/documents/DSP2017_1.0.0.pdf`, Feb. 2009.

[12] G. W. Dunlap, S. T. King, S. Cinar, M. A. Basrai, and P. M. Chen. ReVirt: Enabling intrusion analysis through virtual-machine logging and replay. In *Proc. of USENIX OSDI*, Boston, MA, Dec. 2002.

[13] D. R. Engler, M. F. Kaashoek, and J. W. O'Toole. Exokernel: An operating system architecture for application-level resource management. In *Proc. of ACM SOSP*, Copper Mountain, CO, Dec. 1995.

[14] Eucalyptus Systems, Inc. Eucalyptus open-source cloud computing infrastructure - an overview. `http://www.eucalyptus.com/pdf/whitepapers/Eucalyptus_Overview.pdf`, Aug. 2009.

[15] A. Graf and J. Roedel. Nesting the virtualized world. In *Linux Plumbers Conference*, Portland, OR, Sept. 2009.

[16] D. Gupta, S. Lee, M. Vrable, S. Savage, A. C. Snoeren, G. Varghese, G. M. Voelker, and A. Vahdat. Difference engine: Harnessing memory redundancy in virtual machines. In *Proc. of USENIX OSDI*, San Diego, CA, Dec. 2008.

[17] E. M. Maximilien, A. Ranabahu, R. Engehausen, and L. C. Anderson. IBM altocumulus: a cross-cloud middleware and platform. In *Proc. of ACM OOPSLA Conf.*, Orlando, FL, Oct. 2009.

[18] M. Nelson, B.-H. Lim, and G. Hutchins. Fast transparent migration for virtual machines. In *Proc. of USENIX Annual Technical Conf.*, Anaheim, CA, Apr. 2005.

[19] B. Rochwerger, D. Breitgand, A. Epstein, D. Hadas, I. Loy, K. Nagin, J. Tordsson, C. Ragusa, M. Villari, S. Clayman, E. Levy, A. Maraschini, P. Massonet, H. Muñoz, and G. Tofetti. Reservoir - when one cloud is not enough. *IEEE Computer*, 44(3):44–51, 2011.

[20] V. Shrivastava, P. Zerfos, K. won Lee, H. Jamjoom, Y.-H. Liu, and S. Banerjee. Application-aware virtual machine migration in data centers. In *Proc. of IEEE INFOCOM Mini-conference*, Shanghai, China, Apr. 2011.

[21] J. Sugerman, G. Venkitachalam, and B.-H. Lim. Virtualizing I/O devices on VMware workstation's hosted virtual machine monitor. In *Proc. of USENIX Annual Technical Conf.*, Boston, MA, June 2001.

[22] VMware. "VMware vsphere, the first cloud operating system, provides an evolutionary, non-disruptive path to cloud computing". `http://www.vmware.com/files/pdf/cloud/VMW_09Q2_WP_Cloud_OS_P8_R1.pdf`, 2009.

[23] C. A. Waldspurger. Memory resource management in VMware ESX server. In *Proc. of USENIX OSDI*, Boston, MA, Dec. 2002.

[24] D. Wentzlaff, C. Gruenwald, III, N. Beckmann, K. Modzelewski, A. Belay, L. Youseff, J. Miller, and A. Agarwal. An operating system for multicore and clouds: mechanisms and implementation. In *Proc. of ACM SoCC*, Indianapolis, IN, June 2010.

[25] A. Whitaker, M. Shaw, and S. D. Gribble. Scale and performance in the Denali isolation kernel. In *Proc. of USENIX OSDI*, Boston, MA, Dec. 2002.

[26] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *Proc. of USENIX OSDI*, Boston, MA, Dec. 2002.

[27] D. Williams, E. Elnikety, M. Eldehiry, H. Jamjoom, H. Huang, and H. Weatherspoon. Unshackle the cloud! In *Proc. of USENIX HotCloud*, Portland, OR, June 2011.

[28] D. Williams, H. Jamjoom, Y.-H. Liu, and H. Weatherspoon. Overdriver: Handling memory overload in an oversubscribed cloud. In *Proc. of ACM VEE*, Newport Beach, CA, Mar. 2011.

[29] T. Wood, G. Tarasuk-Levin, P. Shenoy, P. Desnoyers, E. Cecchet, and M. D. Corner. Memory buddies: Exploiting page sharing for smart colocation in virtualized data centers. In *Proc. of ACM VEE*, Washington, DC, Mar. 2009.

[30] F. Zhang, J. Chen, H. Chen, and B. Zang. CloudVisor: Retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization. In *Proc. of ACM SOSP*, Cascais, Portugal, Oct. 2011.