Nested virtualization for the next-generation cloud
An introduction to nesting with KVM

M. Tim Jones
August 22, 2012

Cloud computing is a reality that has changed both business and development models for online businesses. But current IaaS cloud models place a significant constraint on software developers—namely, dictating the hypervisor they must use. The requirement on virtual machine images removes choice from cloud users and to some is an obstacle for moving to the cloud. But a breakthrough from IBM research will change this model. The technology, called nested virtualization, permits a deeper solution stack where guest virtual machines sit on a guest hypervisor, which in turn runs on the cloud's chosen hypervisor. In this article, we explore the ideas behind nested virtualization and how it will improve the cloud.

Ten years ago, when the topic of the "cloud" was first being introduced, the focus was on simple services within a public infrastructure. But as is typical in technology, these services evolve with their use models. Similarly, the introduction of virtualization on commodity hardware also focused on the simplest usage models, but then evolved as the potential was better understood.

As hardware providers saw the growth of virtualization and the cloud, they also evolved their offerings to more efficiently support the needs. Early x86 processors were not ideal for virtualization, but processor internals have focused on these new usage models and created a more efficient environment for platform virtualization.

Let's begin with a short introduction to cloud architectures and some of the limitations they bring.

Public cloud architectures
Public clouds, or publicly available virtualized infrastructure, are focused on the simple allocation of virtual servers as carved out by a hypervisor for multitenant use. The hypervisor acts as a multiplexer, making a physical platform shareable among multiple users. Multiple offerings are available for hypervisors, from the Kernel Virtual Machine (KVM) to the Xen hypervisor and many others.

Full vs. paravirtualization
Multiple forms of hypervisors exist, from the transparent full virtualization hypervisors to the semitransparent paravirtualizing hypervisors. Although the debate continues as to what is best, it's interesting to note that most hypervisors support paravirtualization through the...
One limitation that exists within virtualized infrastructures is their dependence on a given virtual environment. Amazon Elastic Compute Cloud (Amazon EC2), for example, relies on Xen virtualization. Amazon EC2 expects that any guests that run within its infrastructure will be packaged in a specific way called the Amazon Machine Image (AMI) format. The AMI is the fundamental unit of deployment within Amazon EC2 and can be one of many preconfigured types (based on operating system and application set) or a custom creation with some additional effort.

This virtual machine (VM) format (which consists of metadata and a virtual disk format) can be an obstacle for cloud users. The ability to migrate VMs from private infrastructure to public infrastructure or between public cloud infrastructures is obstructed by this format and dependence on the target’s hypervisor choice.

Therefore, support for nested virtualization creates a new abstraction for cloud users. If clouds support the ability to virtualize a hypervisor on top of another hypervisor, then the VM format becomes irrelevant to the cloud. The only dependence is the format of the guest hypervisor itself. This change evolves first-generation clouds from one-size-fits-all propositions into highly flexible virtualized infrastructures with greater freedom for its users. Figure 1 illustrates the new abstraction in the context of virtual platforms for hypervisors, not just VMs. Note in this figure the nomenclature for the levels of virtualization: L0 represents the bare-metal hypervisor, L1 the guest hypervisors, and L2 the guest VMs.

**Figure 1. Simple illustration of traditional hypervisors vs. nesting hypervisors**

This change creates the ability not simply to package VMs for new infrastructures but to package sets of VMs with their hypervisor, simplifying the ability for users of private cloud infrastructures to migrate functionality (either statically or dynamically) to public cloud infrastructure. This change is shown in Figure 2, with the translation of the private hypervisor into the guest hypervisor in the nested cloud.
Next-generation clouds: Introducing nested virtualization

Nested virtualization is not a new concept but one that has been implemented for some time in the IBM® z/VM® hypervisor. The IBM System z® operating system is itself a hypervisor that virtualizes not just processors and memory but also storage, networking hardware assists, and other resources. The z/VM hypervisor represents the first implementation of practical nested virtualization with hardware assists for performance. Further, the z/VM hypervisor supports any depth of nesting of VMs (with additional overhead, of course). More recently, x86 platforms have been driven toward virtualization assists based on the growing usage models for the technique.

The first hypervisor for commodity hardware to implement nested virtualization was the KVM. This addition to KVM was performed under IBM's Turtles project and permitted multiple unmodified hypervisors to run on top of KVM (itself a hypervisor as a tweak of the Linux® kernel). The Turtles project was motivated in part by a desire to use commodity hardware in a way that IBM pioneered for IBM System p® and System z operating systems. In this model, the server runs an embedded hypervisor and allows the user to run the hypervisor of his or her choice on top of it. The approach has gained interest from the virtualization community, as the capabilities (modifications to KVM) are now part of the mainline Linux kernel.

Architecture for nested virtualization

Nested virtualization introduces some unique problems not seen before. Let's explore some of these issues and how they've been addressed within KVM.

Processor virtualization

A disadvantage to current virtualization support in processor architectures is the focus on dual-level virtualization (VMs stacked on a single hypervisor). Turtles stretches this support through the simple process of multiplexing. Recall from Figure 1 that three levels exist (L0 as the host hypervisor, L1 as the guest hypervisor, and L2 as the guest VM). With today's processors, L0 and L1 are efficiently handled, but efficiency is lost at L2. Rather than maintaining this strict stacking, Turtles multiplexes entities at L1 and in essence allows the host hypervisor to multiplex the guest hypervisor and guest VMs at L1. Therefore, rather than virtualizing the virtualization instructions, the hardware assists available in the processor are used efficiently to support the three layers (see Figure 3).
But exploiting the virtualization assets of the processor was not the only obstacle. Let’s explore some of the other issues and their solutions within KVM.

Nested virtualization introduces some interesting problems in this space. Note that traditional virtualization partly addresses the instruction set, directly executing certain instructions on the processor and emulating others through traps. In nested virtualization, another level is introduced at which certain instructions continue to execute directly on hardware and others are trapped but managed in one layer or another (with the overhead of transitioning between the layers).

This setup has exposed strengths and weaknesses in the processor implementations of virtualization, as the Turtles project found. One such area was management of VM control structures (VMCSs). In Intel’s implementation, reading and writing these structures involves privileged instructions that require multiple exits and entries across the layers of the nested stack. These transitions introduce overhead, which is expressed as loss of performance. AMD’s implementation manages VMCS through regular memory reads and writes, which means that when a guest hypervisor (L1) modifies a guest VM’s VMCS (L2), the host hypervisor (L0) is not required to intervene.

Without processor support for nesting, the Turtles approach to multiplexing also minimizes transitions between layers. Transitions in virtualization occur through special instructions to enter or exit VMs (VMexit and VMentry) and are expensive. Certain exits require that the L1 hypervisor be involved, but other conditions (such as external interrupts) are handled solely by L0. Minimizing some of the transitions from L2 to L0 to L1 results in improved performance.

**MMU and memory virtualization**

Prior to page table assists in modern processors, hypervisors emulated the behavior of the memory management unit (MMU). Guest VMs created guest page tables to support their translation of guest virtual addresses into guest physical addresses. The hypervisor maintained shadow page tables to translate guest physical addresses into host physical addresses. All of this required trapping changes to the page tables so that the hypervisor could manage the physical tables in the CPU.

Intel and AMD solved this issue through the addition of two-dimensional page tables called extended page tables (EPTs) by Intel and nested page tables (NPTs) by AMD. These assists allow the secondary page tables to translate guest physical addresses to host physical addresses (while the traditional page tables continue to support guest virtual-to-guest physical translation).
The Turtles project introduced three models to deal with nesting. The first and least efficient is the use of shadow page tables on top of shadow page tables. This option is used only when hardware assists are not available where both the guest and host hypervisor maintain the shadow tables. The second method uses shadow tables over the two-dimensional page tables, which are managed by L0. Although more efficient, page faults in the guest VM result in multiple L1 exits and overhead. The final method virtualizes the two-dimensional page tables for the L1 hypervisor. By emulating the secondary page tables in L1 (where L0 uses the physical EPT/NPT), there are fewer L1 exits and less overhead. This innovation from the Turtles project was called multidimensional paging.

**I/O device virtualization**

Virtualizing I/O devices can be one of the most costly aspects of virtualization. Emulation (as provided by QEMU) is the costliest, where approaches like paravirtualization (making the guest aware and coordinating I/O with the hypervisor) can improve overall performance. The most efficient scheme uses hardware assists such as the AMD I/O MMU (IOMMU) to provide transparent translation of guest physical addresses to host physical addresses (for operations such as direct memory access [DMA]).

The Turtles project improved performance by giving the L2 guest direct access to the physical devices available to L0. The L0 host hypervisor emulates an IOMMU for the L1 guest hypervisor. This approach minimizes guest exits, resulting in reduced overhead and improved performance.

**Nested performance**

Nesting within KVM has found negligible overhead depending on use model. Workloads that drive VM exits (such as external interrupt processing) tend to be the worst offenders, but the optimizations within KVM result in 6% to 14% overhead. This overhead is certainly reasonable given the new capabilities that nested virtualization provides. Advancements in processor architectures will likely improve on this further.

**Where can you find nested virtualization?**

Today, a number of hypervisors support nested virtualization, though not as efficiently as they could. The Linux KVM supports nesting on recent virtualization-enabled processors. The Xen hypervisor has also been modified to support nested virtualization, so the open source community has quickly moved to adopt this capability and its potential usage models.

From a production standpoint, it's safe to say that this capability is in the early stages of development. In addition, scaling virtualization with nesting implies heavier loading on the physical host and therefore should use servers with more capable processors.

Note also that it's possible to perform nested virtualization in other contexts. In a recent OpenStack article, nesting was demonstrated using VirtualBox as the host hypervisor and QEMU (providing emulation) as the guest. Although not the most efficient configuration, the article demonstrates the basic capability on more modest hardware. See Related topics for more details.
Other applications

A hypervisor as a standard portion in firmware on a desktop or server may be commonplace in the future. This usage model implies that the embedded hypervisor can support an operating system (as a VM) or another hypervisor of the user's choice.

The user of a hypervisor in this fashion also supports new security models (as the hypervisor exists underneath the user's and the hacker's code). This concept was originally used for nefarious purposes. The "Blue Pill" rootkit was an exploit from Joanna Rutkowska that inserted a thin hypervisor underneath a running instance of an operating system. Rutkowska also developed a technique called Red Pill that could be used to detect when a "Blue Pill" was inserted below a running operating system.

Going further

The Turtles project proved that nested virtualization of hypervisors was not only possible but efficient under many conditions, using the KVM hypervisor as a test bed. Work has continued with KVM, and it is now a model for the implementation of nesting within a hypervisor, supporting the execution of multiple guest hypervisors simultaneously. As processor architectures catch up with these new requirements, nested virtualization could be a common usage model in the future, not only in enterprise servers in next-generation cloud offerings but on commodity servers and desktops.
Related topics

• The Turtles Project: Design and Implementation of Nested Virtualization is a useful read on how IBM Research Haifa and the IBM Linux Technology Center modified and optimized the KVM to support nested virtualization. Also interesting is the OSDI Presentation on Turtles.
• The paper Architecture of Virtual Machines by R. P. Goldberg was one of the earliest definitions of recursive VMs. This transcript is quite old (from 1973) but well worth a read.
• Blue Pill and Red Pill were introduced by Joanna Rutkowska in 2006. Blue Pill was a malware approach to inserting a thin hypervisor underneath a running operating system as a rootkit. The source code to Blue Pill has never been publicly released.
• The Linux KVM was the first hypervisor to be mainlined into the Linux kernel. KVM is an efficient, production-quality hypervisor that is widely used in the virtualization community.
• As another demonstration of the flexibility of the Linux kernel, the modifications for KVM translate Linux from a desktop and server kernel into a full-featured hypervisor. Learn more in Anatomy of a Linux hypervisor (M. Tim Jones, developerWorks, May 2009).
• OpenStack is an Infrastructure as a Service cloud offering that takes advantage of nested virtualization. In the article Cloud computing and storage with OpenStack you can see a demonstration of nested virtualization with emulation (running QEMU on VirtualBox).
• In the developerWorks cloud developer resources, discover and share knowledge and experience of application and services developers building their projects for cloud deployment.
• Follow developerWorks on Twitter. You can also follow this author on Twitter at M. Tim Jones.
• See the product images available for IBM SmartCloud Enterprise.
• Evaluate IBM products in the way that suits you best: Download a product trial, try a product online, or use a product in a cloud environment.