An Overview of Xen Virtualization

Xen virtualization technology—available for the Linux® kernel—is designed to consolidate multiple operating systems to run on a single server, normalize hardware accessed by the operating systems, isolate misbehaving applications, and migrate running OS instances from one physical server to another. This article provides an overview of Xen 3.0 architecture and how it is expected to utilize Intel® Virtualization Technology to enhance manageability and optimize resource utilization in Linux environments.

BY TIM ABELS, PUNEET DHAWAN, AND BALASUBRAMANIAN CHANDRASEKARAN

Related Categories:

Dell PowerEdge servers

Linux

Virtualization

Visit www.dell.com/powersolutions for the complete category index.

Recent advances in virtualization technologies—enabling data centers to consolidate servers, normalize hardware resources, and isolate applications on the same physical server—are driving rapid adoption of server virtualization in Linux environments. This article provides an overview of Xen 3.0 architecture, which is near-native-speed virtualization software for Intel x86 architectures.

Virtualization software abstracts the underlying hardware by creating an interface to *virtual machines* (VMs), which represent virtualized resources such as CPUs, physical memory, network connections, and block devices. Software stacks including the OS and applications are executed on top of the VMs. Several VMs can run simultaneously on a single physical server. Multiplexing of physical resources between the VMs is enforced by a VM monitor, which is also designed to provide the required translations of operations from the VMs to the physical hardware.

Full virtualization versus para-virtualization

There are several ways to implement virtualization. Two leading approaches are full virtualization and para-virtualization. *Full virtualization* is designed to provide total abstraction of the underlying physical system and creates a complete virtual system in which the guest operating systems can execute. No modification is required in the guest OS or application; the guest OS or application is not aware of the virtualized environment so they have the capability to execute on the VM just as they would on a physical system. This approach can be advantageous because it enables complete decoupling of the software from the hardware. As a result, full virtualization can streamline the migration of applications and workloads between different physical systems. Full virtualization also helps provide complete isolation of different applications, which helps make this approach highly secure. Microsoft® Virtual

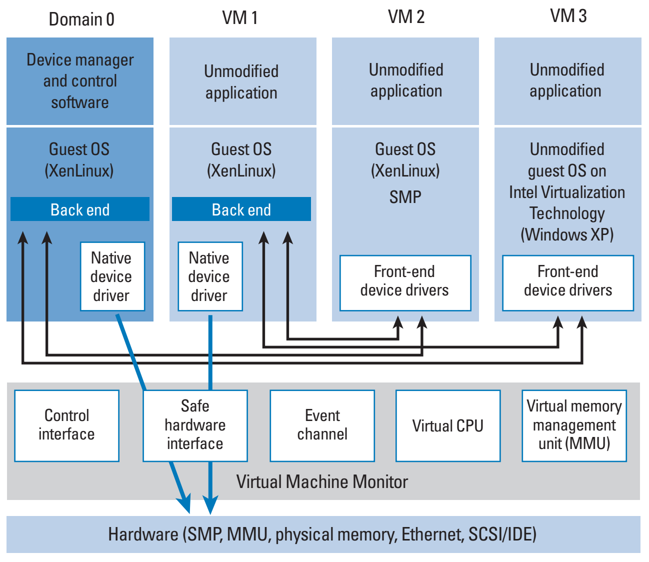

Figure 1. Xen 3.0 architecture: Hosting four VMs

Server and VMware® ESX Server™ software are examples of full virtualization.

However, full virtualization may incur a performance penalty. The VM monitor must provide the VM with an image of an entire system, including virtual BIOS, virtual memory space, and virtual devices. The VM monitor also must create and maintain data structures for the virtual components, such as a shadow memory page table. These data structures must be updated for every corresponding access by the VMs.

In contrast, *para-virtualization* presents each VM with an abstraction of the hardware that is similar but not identical to the underlying physical hardware. Para-virtualization techniques require modifications to the guest operating systems that are running on the VMs. As a result, the guest operating systems are aware that they are executing on a VM—allowing for near-native performance. Para-virtualization methods are still being developed and thus have limitations, including several insecurities such as the guest OS cache data, unauthenticated connections, and so forth.

Xen 3.0 architecture

Xen is an open source virtualization software based on para-virtualization technology. This section provides an overview of the Xen 3.0 architecture.[1]

Figure 1 shows the architecture of Xen 3.0 hosting four VMs (Domain 0, VM 1, VM 2, and VM 3). This architecture includes the Xen Virtual Machine Monitor (VMM), which abstracts the underlying physical hardware and provides hardware access for the different virtual machines. Figure 1 shows the special role of the VM called Domain 0. Only Domain 0 can access the control interface of the VMM, through which other VMs can be created, destroyed, and managed. Management and control software runs in Domain 0. Administrators can create virtual machines with special privileges—such as VM 1—that can directly access the hardware through secure interfaces provided by Xen. Administrators can create other virtual machines that can access the physical resources provided by Domain 0's control and management interface in Xen.

In this example, the guest operating systems in VM 1 and in VM 2 are modified to run above Xen and also have Xen-aware drivers to enable high performance. Near-native performance can be achieved through this approach. Unmodified guest operating systems are also supported, as discussed in the "Xen and Intel Virtualization Technology" section in this article. In addition, the developers of Xen 3.0 plan to include support for virtual machines with symmetric multiprocessing (SMP) capabilities, 64-bit guest operating systems, Accelerated Graphics Port (AGP), and Advanced Configuration and Power Interface (ACPI).

In a virtual data center framework, CPU, memory, and I/O components need to be virtualized. Xen 3.0 is designed to enable para-virtualization of all three hardware components.

Para-virtualization techniques require modifications to the guest operating systems that are running on the VMs. As a result, the guest operating systems are aware that they are executing on a VM—allowing for near-native performance.

CPU operations. The Intel x86 architecture provides four levels of privilege modes. These modes, or *rings,* are numbered 0 to 3, with 0 being the most privileged. In a non-virtualized system, the OS executes at ring 0 and the applications at ring 3. Rings 1 and 2 are typically not used. In Xen para-virtualization, the VMM executes at ring 0, the guest OS at ring 1, and the applications at ring 3. This approach helps to ensure that the VMM possesses the highest privilege, while the guest OS executes in a higher privileged mode than the applications and is isolated from the applications. Privileged instructions issued by the guest OS are verified and executed by the VMM.

Memory operations. In a non-virtualized environment, the OS expects contiguous memory. Guest operating systems in Xen para-virtualization are modified to access memory in a non-contiguous

[1] Note that Xen 3.0 is still under development by a large community of open source developers, and the actual features and architectural details may vary from what is discussed in this article. For more information, visit wiki.xensource.com.

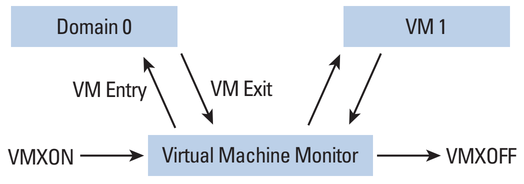

Figure 2. VMM support in Intel Virtualization Technology

manner. Guest operating systems are responsible for allocating and managing page tables. However, direct writes are intercepted and validated by the Xen VMM.

I/O operations. In a fully virtualized environment, hardware devices are emulated. Xen para-virtualization exposes a set of clean and simple device abstractions. For example, I/O data to and from guest operating systems is transferred using shared-memory ring architecture (memory is shared between Domain 0 and the guest domain) through which incoming and outgoing messages are sent.

Modifying the guest OS is not feasible for non–open source platforms such as Microsoft Windows® 2000 or Windows Server™ 2003 operating systems. As a result, such operating systems are not supported in a para-virtualization environment. The following section explains how Xen works with Intel Virtualization Technology to support unmodified operating systems.

Xen and Intel Virtualization Technology

Intel Virtualization Technology (IVT), which is expected to be available in future Intel processors, is designed to support virtualization. IVT-enabled processors will have additional instructions that can be used by the VMM to create and support VMs. In terms of ring architecture, IVT places the VMM one level below ring 0, allowing the VMs to execute in ring 0.

Figure 2 depicts a simple processor state of IVT. The system starts the VMM by executing the VMXON instruction. The VMM can then schedule any VM by executing the VM Entry instruction. The processor state of the VMM is automatically stored and the saved processor state of the VM is loaded. This level of hardware support enables faster context switching, which can enhance overall system performance. Under certain conditions, the VM may relinquish control to the VMM. Such conditions are called VM Exits and can be caused by several factors, including external interrupts, nonmaskable interrupts, page faults, and other high-privileged instructions executed by the VM.

Processor-level support for virtualization allows Xen 3.0 to support an OS whose source code cannot be modified. In such cases, the Xen implementation would have the capability to enable full virtualization with support from IVT. That is, Xen is designed to provide a fully abstracted VM—including virtual BIOS and virtual devices—to allow full support for unmodified guest operating systems.

Xen 3.0 management tools

Two important management tools in Xen 3.0 are xend and xm. The Xen daemon, xend, is a Python program that forms a central point of control for starting and managing VMs. Major functions include invoking setup of virtual networking for VMs, providing a console server, and maintaining the Xen events log.

The Xen command-line interface, xm, provides functionality to create, destroy, save, restore, shut down, migrate domains, and so forth. It also enables administrators to configure the CPU scheduler, list active domains, adjust the memory footprint using ballooning, and call an xend HTTP application programming interface (API) directly.

Xen looking forward

Xen enables administrators to implement two types of virtualization environments: para-virtualized VMs, which can enhance performance but require guest OS modifications, and fully virtualized VMs, which are highly portable and do not require guest OS modifications. Developers are planning to include Xen in the Linux 2.6.12 kernel so that organizations can take advantage of this powerful virtualization tool in their Linux environments.

Tim Abels is a senior software architect currently developing scalable enterprise computing systems at Dell. Tim has an M.S. in Computer Science from Purdue University.

Puneet Dhawan is a systems engineer on the Scalable Enterprise Team at Dell. Puneet has a bachelor's degree in Electrical Engineering from Punjab Engineering College (PEC) in Chandigarh, India, and a master's degree in Computer Engineering from Texas A&M University.

Balasubramanian Chandrasekaran is a systems engineer in the Scalable Enterprise Computing Lab at Dell. His research interests include virtualization of data centers, high-speed interconnects, and high-performance computing. Balasubramanian has an M.S. in Computer Science from The Ohio State University.

FOR MORE INFORMATION

Xen virtual machine monitor:
xen.sf.net

XenSource:
www.xensource.com