STEVEN HAND, ANDREW WARFIELD,
AND KEIR FRASER

hardware virtualization with Xen

Steven Hand is a Senior Lecturer at the University of Cambridge and a founder of XenSource, the leading open source virtualization company. His interests span the areas of operating systems, networks, and security.

steven.hand@cl.cam.ac.uk

Andrew Warfield completed his Ph.D. at the University of Cambridge in May 2006. He now works as the lead storage architect for XenSource, and is also an Adjunct Professor in the Computer Science Department at the University of British Columbia. Andrew currently lives in Vancouver, Canada.

andrew.warfield@cl.cam.ac.uk

Keir Fraser is an EPSRC academic fellow and lecturer at the University of Cambridge and a founder of XenSource. He completed his Ph.D. in 2004 and now manages the Xen project.

keir.fraser@cl.cam.ac.uk

XEN IS A VIRTUAL MACHINE MONITOR (VMM) that we've been developing at the University of Cambridge for the past several years. As a VMM, Xen allows a single physical computer to be divided up into a number of smaller virtual computers, each running its own operating system and applications. Xen is free, but is also available as part of a number of commercial offerings.

Xen was designed from day one to get every last ounce of performance out of commodity x86 machines. The past year has seen chip vendors such as Intel and AMD launch next-generation processors that provide hardware assistance for virtualization. In this article, we provide a background to Xen, show how these new hardware features can be used to provide high-performance virtualization even for proprietary or legacy operating systems, and look toward the future of hardware virtualization.

Xen: Virtualization for the Masses

System virtualization technology has been around for over four decades. Pioneered by IBM with VM/370, system virtualization allows you to divide a single powerful computer into a number of smaller, less powerful computers called virtual machines. Each virtual machine runs its own operating system and applications and is strongly isolated from other virtual machines. This provides enhanced flexibility, management, and security.

For many years, virtualization was limited to "big iron" machines. However, the increasing power and prevalence of commodity off-the-shelf (COTS) systems have made virtualization an attractive technology for regular x86 boxes. Virtual machine monitors (VMMs) such as Xen and VMware provide system virtualization for COTS systems and are now in use on hundreds of thousands of machines worldwide.

There is, however, a problem: The Intel IA-32 architecture was not designed with virtualization in mind, and so certain instructions which should trap when executed with insufficient privilege simply behave differently, and various privileged states are visible even to user-mode software. This means that traditional system virtualization approaches are insufficient. Instead, new techniques are needed to make x86 VMMs a reality.

THE PROBLEM WITH IA-32

In a classic 1974 paper, Popek and Goldberg describe the basic principles for system virtualization. In particular, they identify three requirements for something to be considered a VMM:

- Equivalence: Software running in a virtual machine should behave exactly as it would on a "real" machine (barring timing effects).
- Performance: The vast majority of machine instructions executed when running within a virtual machine should be executed "natively" on the real hardware, and without intervention from the VMM.
- Resource control: The VMM must be in complete control of the hardware resources.

These requirements typically lead to a "trap and emulate" approach in which the VMM runs hosted operating systems in user mode. Most of the time, the software runs exactly as it would on a real machine, but if the operating system (OS) attempts to perform a privileged operation, a hardware trap will occur. Since the VMM executes in supervisor mode, it can catch this hardware exception, inspect the state of the OS that caused it, and emulate the behavior that would have occurred on real hardware. The VMM can then resume the virtual machine, allowing execution to continue.

This approach will satisfy Popek/Goldberg requirements as long as the processor is guaranteed to trap whenever any privileged operation is attempted in user mode. Unfortunately, the original IA-32 architecture does not guarantee this: Various instructions which should trap don't, and in some cases they simply have different semantics than they would have on a real machine. In addition, certain kinds of privileged machine state (such as page tables and segment descriptor tables) reside in memory and hence are visible to user-mode software.

SOLVING THE PROBLEM

There are two main ways that we can work around these problems with the IA-32 architecture: binary rewriting and paravirtualization. In the former approach, the VMM dynamically scans the memory of the guest OS looking for problematic instructions, and rewrites any it finds with alternative instruction sequences. This approach is costly and fragile, especially as the x86 uses variable-length instructions, but it can be made to work in most cases.

The latter approach, used by Xen, modifies the operating system source code to make it aware that it is running on top of a VMM. The resulting enlightened operating system can run extremely efficiently in a virtual machine environment: Typically an overhead of just 1% is observed. It can also work in cooperation with the VMM to provide advanced features such as CPU, memory, and device hotplug, or even live migration, seamlessly relocating a running virtual machine from one physical node to another.

At the time of writing, there are enlightened versions of modern Linux, BSD, and Solaris operating systems that run efficiently on Xen. Furthermore, Microsoft has announced that it is working on an enlightened version of its forthcoming "Longhorn" operating system.

Nonetheless, there is a large existing base of legacy operating systems that cannot use the paravirtualized technique. To support these, we need to look to the processor vendors and their recently introduced hardware support for virtualization.

Hardware Virtualization

To work around the problems with the original IA-32 architecture, both major processor vendors have recently introduced hardware extensions. Intel's technology is called VT-x, or VT for short, and ships in most recent processors including the Xeon 51xx series, the Xeon 71xx series, and the Core Duo and Core 2 Duo processors. The equivalent AMD technology, called AMD-V, ships in recent (stepping F2) Opteron and AMD64 processors. (Although there are some important differences between VT-x and AMD-V, this article will avoid them in the interests of simplicity. More technical details on both technologies are available from the references given at the end of the article.)

These hardware virtualization (HV) technologies both operate by making the processor aware of multiple virtual machine contexts (VMCs). A VMC is analogous to a process control block (PCB) in an operating system: a copy of the state required to resume or schedule that virtual machine. The VMC holds a strict superset of the contents of a PCB, however; for example, in addition to the values of general purpose and floating-point registers and flags, the VMC will contain the values of the processor control registers (such as cr0, cr4, and cr8). The VMC will also include the values of certain model-specific registers (MSRs) such as CSTAR and EFER, as well as an expanded version of each segment selector.

Hence with hardware virtualization technology, the VMM acts somewhat like a traditional operating system, but scheduling virtual machines instead of processes. The HV extensions include instructions to launch and/or resume a given VMC, which causes the hardware to load the relevant processor state and continue execution. The new execution environment includes its own privilege levels ("rings" in IA-32 terminology), so the operating system kernel can operate in what it believes is supervisor mode and runs its own applications in what it believes is user mode. The new execution environment can also operate in a completely independent processor mode; for example, the VMM can run in 64-bit mode, one VM in 32-bit paged mode, and another VM in 16-bit real mode.

The act of launching and/or resuming a VMC is sometimes called entering a virtual machine and, as previously mentioned, can be seen as analogous to scheduling a process in an operating system. However, things are different when we consider the opposite case: exiting a virtual machine. Whereas in an operating system a process will usually only be descheduled as a result of an interrupt or system call, a VMM wishes to intercept execution in a much wider range of situations. Examples include instructions that manipulate processor interrupt state, interactions with the TLB, instructions that access or update control registers or MSRs, and attempts to put the processor into a halt state.

To allow maximum flexibility, hardware virtualization allows the VMM to select precisely which events it wants to intercept. The selected events will cause a vmexit, effectively a trap from the running virtual machine into the VMM. Since the set of allowable events includes all privileged x86 instructions, this allows implementation of the classic trap-and-emulate scheme, and hence it enables efficient virtualization of nonparavirtualized operating systems.

XEN: HARDWARE VIRTUAL MACHINES

Xen uses the hardware virtualization technologies described above to enable support for legacy or proprietary operating systems. It uses the trap-and-emulate approach to deal with privileged instructions, which enables efficient virtualization without the overhead or fragility of binary rewriting.

However, existing hardware virtualization support only provides part of the solution required to enable the execution of hardware virtual machines. In particular we can consider a modern COTS system as comprising three main components:

- The processor
- The memory subsystem
- The I/O subsystem

Current VT-x and AMD-V technologies help with processor virtualization, but they do not deal with memory or I/O. Software support within Xen is required to complete the picture.

VIRTUALIZING MEMORY

Most operating systems expect a contiguous range of physical memory (RAM) starting from address 0x0. When running on top of a VMM, however, many operating systems are run concurrently, and will be allocated varying amounts of physical memory from the overall pool. One job for the VMM then is to translate between physical addresses as seen by an individual virtual machine ("guest physical addresses" or just "physical addresses" for short) and the actual physical addresses as seen by the real hardware ("machine addresses").

There are two interesting cases to consider depending on which mode the virtual machine is executing in: (1) real mode or protected mode or (2) paged mode. In the former case, addresses generated by the virtual machine are physical addresses (albeit modified by segment translation); in the latter case the virtual machine generates virtual addresses, which it expects to be translated via the processor's paging mechanism. Xen handles both of these cases by the same means: shadow page tables.

The basic idea is simple: The guest creates and manages its own page tables, which translate from virtual to guest physical addresses. When the guest wishes to use an address space for the first time, it will update its cr3 register to point to the root page table. Using hardware virtualization, this causes a vmexit, which allows Xen to create a shadow copy of the root page table. Unlike the guest version, the version used by Xen translates from virtual addresses directly to machine addresses, and so it can be used by the "real" (hardware) MMU.

For space efficiency, it is not necessary to make shadow copies of every part of the current page table; instead, copies can be made on demand as the operating system (or its hosted processes) access various parts of the virtual address space. In addition, Xen must be able to track any updates made by the guest to its page tables and reflect the appropriate changes in the shadow copies. For these reasons, Xen ensures that guest page table pages are always mapped read-only. As a consequence, any page table modification attempted by the guest will result in a fault into the VMM. Xen can then intercept the access and maintain coherence between guest and shadow page tables.

The current implementation (in Xen 3.0.3) has been designed for high performance and includes a number of optimizations above and beyond the

scheme just described. It also incorporates support for other modes of operation, which may be used for the live migration of virtual machines. Interested readers can learn more from the references given at the end of the article.

VIRTUALIZING I/O

The final part of the picture entails dealing with the I/O subsystem. This includes simple platform devices (such as timers and interrupt controllers), disk drives, video cards, USB controllers, and network interface cards.

COTS systems expect to access such devices either via I/O instructions (direct or memory mapped) or via memory-mapped PCI bus addresses. As with page table updates, Xen intercepts any such accesses and emulates the behavior of device hardware. For platform devices, this is relatively straightforward since they perform no actual I/O per se. Other devices are more complex and may require the ability to send packets on a real network interface card or read data from a real storage device.

Xen supports these I/O devices by instantiating a device model process for each virtual machine. This emulates the behavior of the rest of the platform hardware, which can be configured to include the desired number and type of network interface cards, IDE controllers, graphics cards, and USB controllers. These virtual devices handle any accesses made by device drivers running in the virtual machine, mirroring the state transitions that would be made by an equivalent piece of hardware. They also interact with a—potentially virtualized—instance of that hardware: For example, a disk device can be represented as a sparse file.

Providing an emulated platform allows operating systems to run without requiring that they are at all aware of virtualization. However, emulation can be rather slow, particularly for devices such as network interface cards, which can require many (emulated) bus cycles to, for example, transmit a packet.

Hence Xen also provides the ability to load new, virtualization-aware device drivers into the operating system after it has been installed. These paravirtualized drivers understand the underlying VMM, and hence they can more directly interact with the virtual hardware. In the case of networking, this can increase performance by an order of magnitude.

Next Steps in Hardware Virtualization

We've seen how Xen uses existing hardware virtualization of the processor to efficiently and robustly run unmodified operating systems, augmenting this with software support for memory virtualization (shadow page tables) and I/O virtualization (device model).

Looking ahead, we envision a number of further hardware enhancements: hardware support for memory virtualization, platform virtualization, and device virtualization.

NESTED/EXTENDED PAGE TABLES

Even though shadow page tables can be implemented efficiently, they still require a number of transitions between the VMM and the guest. These transitions can cost hundreds or even thousands of cycles, and so it is desirable to keep their number to an absolute minimum. To this end, both Intel and AMD have recently announced hardware support for virtualizing

the MMU. Intel's scheme is called extended page tables (EPT); AMD's is called nested page tables (NPT).

Both operate by adding an extra level of translation; in essence, a new page table (called the EPT or NPT, respectively) is introduced to translate between (guest) physical and machine addresses. There is one of these per virtual machine, since all address spaces within that virtual machine share the same physical to machine mapping. In addition, the hardware is now explicitly aware of the guest page tables.

Consequently, on a TLB miss, the hardware can walk the guest page tables directly, using the additional EPT or NPT to translate the physical addresses contained within page table entries. On completion of the walk, the TLB is updated with the resulting virtual to machine mapping and execution continues.

Note that even though this extra level of indirection does involve more lookups, it does not require any vmexits, hence improving overall efficiency. The hardware can also cache intermediate translation results to further improve performance.

VIRTUALIZING THE PLATFORM

Help is also coming for platform virtualization. First in line are extensions from AMD and Intel that allow enhanced protection from DMA-capable devices.

In today's COTS systems, devices are not subject to any translation or protection checks when they access memory. This means that a malicious or buggy device driver can program a device to read or write any piece of memory in the system, bypassing the VMM and any installed security policy.

Currently shipping AMD-V chips include support for device exclusion vectors (DEVs), which addresses this risk. A DEV is a bitmap with 1 bit for every 4K page of physical (host) memory. Any attempted memory access by a device first causes a lookup (based on the device and bus ids) to a protection domain; this is then used to select an appropriate DEV, and the target address is checked against the appropriate bit. If the bit is set, the access is disallowed. This can be used by a VMM to protect itself and any other key data (such as security policies) from rogue DMA accesses.

Similarly, Intel has announced VT-d, a forthcoming technology aimed at providing enhanced support for platform virtualization. VT-d is also a northbridge-based approach that interposes on device accesses, and it also maps devices to protection domains. However, VT-d takes a more generalized IOMMU approach: In particular, device-issued DMA addresses are no longer "physical" addresses but are instead translated through a hardware table. This allows protection as well as arbitrary remapping of the bus address space. VT-d support is expected to ship in 2007.

VIRTUALIZING DEVICES

Finally, there is work on making I/O devices themselves virtualization-aware, to allow direct yet safe sharing between multiple virtual machines. This is particularly of interest for high-throughput, low-latency devices such as gigabit network interface cards and next-generation graphics cards.

Some of this work involves proposed extensions to PCIe being developed by the PCI-SIG. These extensions include address translation and the

introduction of virtual functions within PCI devices. There is also ongoing development of "smart" I/O devices that provide translation, protection, and multiplexing between multiple clients. Early results indicate that bare-metal performance can be maintained without sacrificing safety.

Conclusion

As virtualization continues to grow as an important technique for managing modern systems, the software and hardware used to provide it are maturing at a dramatic rate: The original version of Xen stemmed from a research project at the University of Cambridge and allowed a specific handful of modified operating systems to be efficiently virtualized on uncooperative x86 hardware. Nearly four years later, Xen is a mature and robust VMM supporting paravirtualized OSes that are increasingly maintained by the OS developers themselves; Xen has further been incorporated as a core feature in the major Linux distributions, being directly included with their release kernels.

Chip makers have also embraced virtualization and have released hardware features to assist VMMs. Xen now includes support for both Intel's VT and AMD's V processor extensions, allowing unmodified legacy OSes to be efficiently and safely virtualized. As a result, Xen can now host nonparavirtualized OSes, such as Microsoft Windows, on modern hardware. Hardware will continue to evolve in support of virtualization in the immediate future, providing more direct support for both memory and I/O devices. We look forward to incorporating these features into Xen as they become available, as they promise to provide even greater performance and stability for the virtualization of COTS systems.

REFERENCES

The following resources are useful for finding out more about hardware virtualization and Xen:

"Intel Virtualization Technology," *Intel Technology Journal*: http://www.intel.com/technology/itj/2006/v10i3/index.htm.

AMD64 Architecture Programmers Manual, Volume 2: System Programming: http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/24593.pdf.

Xen downloads: http://xensource.com/download.

Symposium on Operating System Principles (SOSP) 2003 paper on Xen: http://www.cl.cam.ac.uk/netos/papers/2003-xensosp.pdf.

Shadow2 presentation at Fall 2006 Xen Summit: http://www.xensource.com/files/summit_3/XenSummit_Shadow2.pdf.

Xen Source Code Repository: http://xenbits.xensource.com.