The Virtualization Reality

A number of important challenges are associated with the deployment and configuration of contemporary computing infrastructure. Given the variety of operating systems and their many versions—including the often-specific configurations required to accommodate the wide range of popular applications—it has become quite a conundrum to establish and manage such systems.

Significantly motivated by these challenges, but also owing to several other important opportunities it offers, virtualization has recently become a principal focus for computer systems software. It enables a single computer to host multiple different operating system stacks, and it decreases server count and reduces overall system complexity. EMC's VMware is the most visible and early entrant in this space, but more recently XenSource, Parallels, and Microsoft have introduced virtualization solutions. Many of the major systems vendors, such as IBM, Sun, and Microsoft, have efforts under way to exploit virtualization. Virtualization appears to be far more than just another ephemeral marketplace trend. It is poised to deliver profound changes to the way that both enterprises and consumers use computer systems.

What problems does virtualization address, and moreover, what will you need to know and/or do differently to take advantage of the innovations that it delivers? In this article we provide an overview of system virtualization, taking a closer look at the Xen hypervisor and its paravirtualization architecture. We then review several challenges in deploying and exploiting computer systems and software applications, and we look at IT infrastructure management today and show how virtualization can help address some of the challenges.

A POCKET HISTORY OF VIRTUALIZATION

All modern computers are sufficiently powerful to use virtualization to present the illusion of many smaller VMs (virtual machines), each running a separate operating system instance. An operating system virtualization environment provides each virtualized operating system (or *guest*) the illusion that it has exclusive access to the underlying hardware platform on which it runs. Of course, the virtual machine itself can offer the guest a different view of the hardware from what is really available, including CPU, memory, I/O, and restricted views of devices.

Virtualization has a long history, starting in the mainframe environment and arising from the need to provide isolation between users. The basic trend started with time-sharing systems (enabling multiple users to share a single expensive computer system), aided by innovations in operating system design to support the idea of processes that belong to a single user. The addition of user and supervisor modes on most commercially relevant

SIMON CROSBY, XENSOURCE and DAVID BROWN, SUN MICROSYSTEMS


Are
hypervisors
the new
foundation
for
system
software?

The Virtualization Reality

processors meant that the operating system code could be protected from user programs, using a set of so-called "privileged" instructions reserved for the operating system software running in supervisor mode. Memory protection and, ultimately, virtual memory were invented so that separate address spaces could be assigned to different processes to share the system's physical memory and ensure that its use by different applications was mutually segregated.

These initial enhancements could all be accommodated within the operating system, until the day arrived when different users, or different applications on the same physical machine, wanted to run *different* operating systems. This requirement could be satisfied only by supporting multiple VMs, each capable of running its own operating system. The virtualization era (marked by IBM's release of VM for the System/360 in 1972) had dawned.

VIRTUALIZATION BASICS

Operating system virtualization is achieved by inserting a layer of system software—often called the *hypervisor* or *VMM (virtual machine monitor)*—between the guest operating system and the underlying hardware. This layer is responsible for allowing multiple operating system images (and all their running applications) to share the resources of a single hardware server. Each operating system believes that it has the resources of the entire machine under its control, but beneath its feet the virtualization layer, or hypervisor, transparently ensures that resources are properly and securely partitioned between different operating system images and their applications. The hypervisor manages all hardware structures, such as the MMU (memory management unit), I/O devices, and DMA (direct memory access) controllers, and presents a virtualized abstraction of those resources to each guest operating system.

EMULATED VIRTUALIZATION

The most direct method of achieving virtualization is to provide a complete emulation of the underlying hardware platform's architecture in software, particularly involving the processor's instruction set architecture. For the x86 processor, the privileged instructions—used exclusively by the operating system (for interrupt handling, reading and writing to devices, and virtual memory)—form the dominant class of instructions requiring emulation. By definition, a user program cannot execute these instructions. One technique to force emulation of these instructions is to execute all of the code within a virtual machine, including the operating system being virtualized, as user code. The resident VMM then handles the exception produced by the attempt to execute a privileged instruction and performs the desired action on behalf of the operating system.

While some CPUs were carefully architected with operating system virtualization in mind (the IBM 360 is one such example), many contemporary commodity processor architectures evolved from earlier designs, which did not anticipate virtualization. Providing full virtualization of a processor in such cases is a challenging problem, often resulting in so-called "virtualization holes." Virtualization of the x86 processor is no exception. For example, certain instructions execute in both user mode and supervisor mode but produce different results, depending on the execution mode. A common approach to overcome these problems is to scan the operating system code and modify the offending instruction sequences, either to produce the intended behavior or to force a trap into the VMM. Unfortunately, this patching and trapping approach can cause significant performance penalties.

PARAVIRTUALIZATION

An alternative way of achieving virtualization is to present a VM abstraction that is similar but not identical to the underlying hardware. This approach has been called *paravirtualization*.

In lieu of a direct software emulation of the underlying hardware architecture, the concept of paravirtualization is that a guest operating system and an underlying hypervisor collaborate closely to achieve optimal performance. Many guest operating system instances (of different configurations and types) may run atop the one hypervisor on a given hardware platform. This offers improved performance, although it does require modifications to the guest operating system. It is important to note, however, that it does not require any change to the ABI (application binary interface) offered by the guest system; hence, no modifications are required to the guest operating system's *applications*.

In many ways this method is similar to the operating

system virtualization approach of VM for the IBM 360 and 370 mainframes.[1,2] Under pure virtualization, you can run an unmodified operating-system binary *and* unmodified application binaries, but the resource consumption management and performance isolation is problematic—one guest operating system and/or its apps could consume all physical memory and/or cause thrashing, for example. The paravirtualization approach requires some work to port each guest operating system, but rigorous allocation of hardware resources can then be done by the hypervisor, ensuring proper performance isolation and guarantees.

The use of *paravirtualization* and the complementary innovation of processor architecture extensions to support it (particularly those recently introduced in both the Intel and AMD processors, which eliminate the need to "trap and emulate") now permit high-performance virtualization of the x86 architecture.

PARAVIRTUALIZATION AND THE XEN HYPERVISOR

An example of paravirtualization as applied on the x86 architecture is the Xen hypervisor (figure 1). Xen was initially developed by Ian Pratt and a team at the University of Cambridge in 2001-02, and has subsequently evolved into an open source project with broad involvement.

Any hypervisor (whether it implements full hardware emulation or paravirtualization) must provide virtualization for the following system facilities:

- CPUs (including multiple cores per device)
- Memory system (memory management and physical memory)
- I/O devices
- Asynchronous events, such as interrupts

Let's now briefly examine Xen's approach to each of these facilities. (For further detail, we recommend the excellent introduction to and comprehensive treatment of Xen's design and principles presented in Pratt et al.'s paper.[3])

CPU AND MEMORY VIRTUALIZATION

In Xen's paravirtualization, virtualization of CPU and memory and low-level hardware interrupts are provided by a low-level efficient hypervisor layer that is implemented in about 50,000 lines of code. When the operating system updates hardware data structures, such as the page table, or initiates a DMA operation, it collaborates with the hypervisor by making calls into an API that is offered by the hypervisor.

This, in turn, allows the hypervisor to keep track of all changes made by the operating system and to optimally decide how to manage the state of hardware data structures on context switches. The hypervisor is mapped into the address space of each guest operating system, meaning that there is no context-switch overhead between the operating system and the hypervisor on a hypercall.

Finally, by cooperatively working with the guest operating systems, the hypervisor gains insight into the intentions of the operating system and can make it aware that it has been virtualized. This can be a great advantage to the guest operating system—for example, the hypervisor can tell the guest that real time has passed between its last run and its present run, permitting it to make smarter rescheduling decisions to respond appropriately to a rapidly changing environment.

Xen makes a guest operating system (running on top of the VMM) virtualization-aware and presents it with a slightly modified x86 architecture, provided through the so-called hypercall API. This removes any difficult and costly-to-emulate privileged instructions and provides equivalent, although not identical, functionality with explicit calls into the hypervisor. The operating system must be modified to deal with this change, but in a well-structured operating system, these changes are limited to its architecture-dependent modules, most typically a fairly small subset of the complete operating system implementation. Most importantly, the bulk of the operating system and the entirety of application programs remain unmodified.

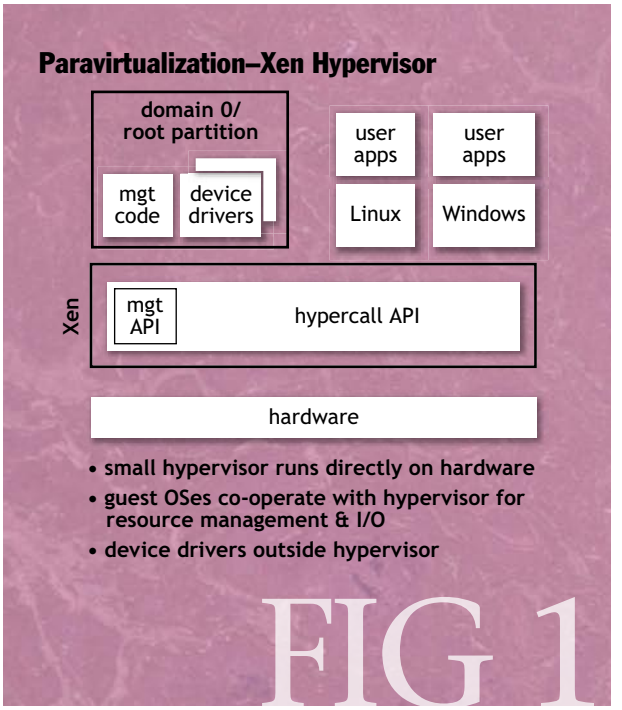

FIG 1

The Virtualization Reality

For Linux, the Xen hypercall API takes the form of a jump table populated at kernel load time. When the kernel is running in a native implementation (i.e., *not* atop a paravirtualizing hypervisor), the jump table is populated with default native operations; when the kernel is running on Xen, the jump table is populated with the Xen hypercalls. This enables the same kernel to run in both native and virtualized forms, with the performance benefits of paravirtualization but without the need to recertify applications against the kernel.

Isolation between virtual machines (hence, the respective guest operating systems running within each) is a particularly important property that Xen provides. The physical resources of the hardware platform (such as CPU, memory, etc.) are rigidly divided between VMs to ensure that they each receive a guaranteed portion of the platform's overall capacity for processing, memory, I/O, and so on. Moreover, as each guest is running on its own set of virtual hardware, applications in separate operating systems are protected from one another to almost the same degree that they would be were they installed on separate physical hosts. This property is particularly appealing in light of the inability of current operating systems to provide protection against spyware, worms, and viruses. In a system such as Xen, nontrusted applications considered to pose such risks (perhaps such as Web browsers) may be seconded to their own virtual machines and thus completely separated from both the underlying system software and other more trusted applications.

I/O VIRTUALIZATION

I/O virtualization in a paravirtualizing VMM such as Xen is achieved via a single set of drivers. The Xen hypervisor exposes a set of clean and simple device abstractions, and a set of drivers for all hardware on the physical platform is implemented in a special domain (VM) outside the core hypervisor. These drivers are offered via the hypervisor's abstracted I/O interface for use within other VMs, and thus are used by all guest operating systems.[4]

In each Xen guest operating system, simple paravirtualizing device drivers replace hardware-specific drivers for the physical platform. Paravirtualizing drivers are independent of all physical hardware but represent each type of device (e.g., block I/O, Ethernet). These drivers enable high-performance, virtualization-safe I/O to be accomplished by transferring control of the I/O to the hypervisor, with no additional complexity in the guest operating system. It is important to note that the drivers in the Xen architecture run outside the base hypervisor, at a lower level of protection than the core of the hypervisor itself. The hypervisor is thus protected from bugs and crashes in device drivers (they cannot crash the Xen VMM) and can use any device drivers available on the market. Also, the virtualized operating system image is much more portable across hardware, since the low levels of the driver and hardware management are modules that run under control of the hypervisor.

In full-virtualization (emulation) implementations, the platform's physical hardware devices are emulated, and the unmodified binary for each guest operating system is run, including the native drivers it contains. In those circumstances it is difficult to restrict the respective operating system's use of the platform's physical hardware, and one virtual machine's runtime behaviors can significantly impact the performance of the others. Since all physical access to hardware is managed centrally in Xen's approach to I/O virtualization, resource access by each guest can be marshaled. This provides the consequential benefit of performance isolation for each of the guest operating systems.

Those who have experience with microkernels will likely find this approach to I/O virtualization familiar. One significant difference between Xen and historical work on microkernels, however, is that Xen has relaxed the constraint of achieving a complete and architecturally pure emulation of the x86 processor's I/O architecture. Xen uses a generalized, shared-memory, ring-based I/O communication primitive that is able to achieve very high throughputs by batching requests. This I/O abstraction has served well in ports to other processor architectures, including the IA-64 and PowerPC. It also affords an innovative means to add features into the I/O path, by plumbing in additional modules between the guest virtual device and the real device driver. One example in the network stack is the support of full OSI layer 2 switching, packet filtering, and even intrusion detection.

HARDWARE SUPPORT FOR VIRTUALIZATION

Recent innovations in hardware, particularly in CPU, MMU, and memory components (notably the hardware

virtualization support presently available in the Intel VT-x and AMD-V architectures, offered in both client and server platforms), provide some direct platform-level architectural support for operating system virtualization. This has enabled near bare-metal performance for virtualized guest operating systems.

Xen provides a common HVM (hardware virtual machine) abstraction to hide the minor differences between the Intel and AMD technologies and their implementations. HVM offers two key features: First, for unmodified guest operating systems, it avoids the need to trap and emulate privileged instructions in the operating system, by enabling guests to run at their native privilege levels, while providing a hardware vector (called a VM EXIT) into the hypervisor whenever the guest executes a privileged instruction that would unsafely modify the machine state. The hypervisor begins execution with the full state of the guest available to it and can rapidly decide how best to deal with the reason for the VM EXIT. Today's hardware takes about 1,000 clock cycles to save the state of the currently executing guest and to transition into the hypervisor, which offers good, though not outstanding, performance.

A second feature of the HVM implementations is that they offer guest operating systems running with a paravirtualizing hypervisor (in particular, their device drivers) new instructions that call directly into the hypervisor. These can be used to ensure that guest I/O takes the fastest path into the hypervisor. Paravirtualizing device drivers, inserted into each guest operating system, can then achieve optimal I/O performance, even though neither Intel's nor AMD's virtualization extension for the x86 (Intel VT and AMD-V, respectively) offers particular performance benefits to I/O virtualization.

VIRTUALIZATION AS A SOLUTION

A number of chronic challenges are associated with deployment and management of computer systems and their applications, especially in the modern context of larger-scale, commercial, and/or enterprise use. Virtualization provides an abstraction from the physical hardware, which breaks the constraint that only a single instance of an operating system may run on a single hardware platform. Because it encapsulates the operating environment, virtualization is a surprisingly powerful abstraction.

SERVER VIRTUALIZATION

The past decade has witnessed a revolutionary reduction in hardware costs, as well as a significant increase in both capacity and performance of many of the basic hardware platform constituents (processors, storage, and memory). Ironically, in spite of the corresponding widespread adoption of these now relatively inexpensive, x86-based servers, most enterprises have seen their IT costs and complexity escalate rapidly.

While the steady march of Moore's law has markedly decreased hardware's cost of acquisition, the associated proliferation of this inexpensive computing has led to tremendous increases in complexity—with the costs of server configuration, management, power, and maintenance dwarfing the basic cost of the hardware. Each server in the data center costs an enterprise on average $10,000 per year to run when all of its costs—provisioning, maintenance, administration, power, real estate, hardware, and software—are considered. In addition, the artifacts of current operating-system and system-software architecture result in most servers today running at under 10 percent utilization.

Several opportunities arise directly from the rapid performance and capacity increase seen in the contemporary commodity hardware platforms. Last decade's trend in commercial IT infrastructure was an expanding hardware universe: achieving performance and capacity by "horizontal" scaling of the hardware. Given the dramatic performance available on a single commodity box today, we may now be witnessing a contraction of this universe—still a horizontal trend, but in reverse. Whereas it may have required many servers to support enterprise-wide or even department-wide computing just five years ago, virtualization allows many large application loads to be placed on one hardware platform, or a smaller number of platforms. This can cut both per-server capital cost and the overall lifetime operational costs significantly.

The 10-percent utilization statistic reveals that server consolidation can achieve a tenfold savings in infrastructure cost, not simply through reduced CPU count but more importantly through its consequent reductions in switching, communication, and storage infrastructure, and power and management costs. Since virtualization allows multiple operating system images (and the applications associated with each that constitute software services) to share a single hardware server, it is a basic enabler for server consolidation.

The virtual I/O abstraction is another important component of server virtualization. In the past, when multiple servers and/or multiple hardware interfaces per server were used to support scalability, physical hardware devices could be individually allotted to guarantee a certain performance, specific security properties, and/or other configuration aspects to individual operating-sys-

The Virtualization Reality

tem and application loads. Nowadays, a single device may have significantly higher performance (e.g., the transition from Fast Ethernet to inexpensive Gigabit or even 10-Gigabit network interface cards), and just one or a much smaller number of physical devices will likely be present on a single server or server configuration.

In such configurations, where individual physical hardware devices are shared by multiple hosted VMs on a single server, ensuring that there is proper isolation between their respective demands upon the shared hardware is critical. Strict allocation of the shared CPU, memory, and I/O resources, as well as the assurance of the security of both the platform and the guests, are key requirements that fall on the hypervisor.

Beyond its immediate application for server consolidation, server virtualization offers many further benefits that derive from the separation of virtual machines (an operating system and its applications) from physical hardware. These benefits (several of which have yet to be exploited fully in application) include dynamic provisioning, high availability, fault tolerance, and a "utility computing" paradigm in which compute resources are dynamically assigned to virtualized application workloads.

VIRTUAL APPLIANCES

Once an operating system and its applications have been encapsulated into a virtual machine, the VM can be run on any computer with a hypervisor. The ability to encapsulate all states, including application and operating-system configuration, into a single, portable, instantly runnable package provides great flexibility. For a start, the application can be provisioned and the VM saved in a "suspended" state, which makes it instantly runnable without further configuration. The image of one or more applications that have been properly configured in a VM and are ready to run can be saved, and this may then be used as a highly portable distribution format for a software service.

The administrative tasks of installing and configuring an operating system and the necessary applications prior to instantiating and launching a software service on a platform are no longer needed. The preconfigured and saved VM image is simply loaded and launched. VMware led the industry with its appliance concept, which aims to use packaged VMs as a new software distribution technique. VMware offers more than 200 prepackaged appliances from its Web site.

Within the enterprise, the packaged VM offers additional benefits: Software delivered by an engineering group can be packaged with the operating system it requires and can be staged for testing and production as a VM. Easily and instantly provisioned onto testing equipment, the application and the operating system against which it is certified can be quickly tested in a cost-efficient environment before being made available as a packaged VM, ready for deployment into production.

A key problem in the data center is the ability to get new applications quickly into production. New applications typically take 60 to 90 days to qualify. To make it from testing into the data center, IT staff must acquire a new machine, provision it with an operating system, install the application, configure and test the setup for the service in question, and only then, once satisfied, rack the resulting server in the data center.

This packaging approach provides an avenue to a solution. Once new software has been packaged as an appliance, it can be deployed and run instantly on any existing server in the data center that has sufficient capacity to run it. Any final testing or qualification can still be done before the service is made available for production use if required, but the lead times to acquire, install, and/or customize new hardware at its point of use are removed.

LIVE RELOCATION

Virtual appliances accelerate software provisioning and portability. Live relocation—the ability to move a running VM dynamically from one server to another, without stopping it—offers another benefit: When coupled with load-balancing and server resource optimization software, this provides a powerful tool for enabling a "utility computing" paradigm. When a VM is short of resources, it can be relocated dynamically to another machine with more resources. When capacities are stretched, additional copies of an existing VM can be cloned rapidly and deployed to other available hardware resources to increase overall service capacity. Instantaneous load considerations are a notorious challenge in the IT administrative world. Grid engines, as applied on distributed virtualized

servers, where spare resources are held in reserve, can be used to spawn many instances of a given application dynamically to meet increased load or demand.

CLIENT SECURITY AND MOBILITY

On the client, virtualization offers various opportunities for enhanced security, manageability, greater worker mobility, and increased robustness of client devices. Virtualization of clients is also made possible through the hosting of multiple client operating system instances on a modern server-class system. Offering each client environment as a virtualized system instance located on a server in the data center provides the user with a modern-day equivalent of the thin client. Mobility of users is a direct result of their ability to access their virtualized workload remotely from any client endpoint. Sun's Sun Ray system is an example of one such implementation.

Increased security of data, applications and their context of use, and reduced overall cost of administration for client systems are important aspects of this technology. Enhanced reliability and security can be achieved, for example, by embedding function-specific, hidden VMs on a user's PC, where the VM has been designed to monitor traffic, implement "embedded IT" policies, or the like. The packaging of applications and operating-system images into portable appliances also provides a powerful metaphor for portability of application state: Simply copying a suspended VM to a memory stick allows the user to carry running applications to any virtualization-ready device. VMware's free Player application is a thin, client-side virtualization "player" that has the ability to execute a packaged VM. Examples include prepackaged secure Web browsers that can be discarded after per-session use (to obtain greater security) and secured, user-specific or enterprise-specific applications.

CONCLUSION

The use of virtualization portends many further opportunities for security and manageability on the client. The examples presented here only begin to illustrate the ways in which virtualization can be applied. Virtualization represents a basic change in the architecture of both systems software and the data center. It offers some important opportunities for cost savings and efficiency in computing infrastructure, and for centralized administration and management of that infrastructure for both servers and clients. We expect it to change the development, testing, and delivery of software fundamentally, with some immediate application in the commercial and enterprise context. Q

ACKNOWLEDGMENTS

We are particularly indebted to the team at the University of Cambridge, including Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Derek McAuley, Rolf Neugebauer, Ian Pratt, Andrew Warfield, and Matt Williamson, who have developed and evolved the Xen system. This article reports on their work.

REFERENCES

1. Gum, P. H. 1983. System/370 extended architecture: Facilities for virtual machines. *IBM Journal of Research and Development* 27(6): 530-544.
2. Seawright, L., MacKinnon, R. 1979. VM/370—a study of multiplicity and usefulness. *IBM Systems Journal* 18(1): 4-17.
3. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A. 2003. Xen and the art of virtualization. In *Proceedings of the 19th ACM SOSP* (October): 164-177.
4. Fraser, K., Hand, S., Neugebauer, R., Pratt, I., Warfield, A., Williamson, M. 2004. Safe hardware access with the Xen virtual machine monitor. Cambridge, UK: University of Cambridge Computer Laboratory; www.cl.cam.ac.uk/research/srg/netos/papers/2004-oasis-ngio.pdf.

LOVE IT, HATE IT? LET US KNOW

feedback@acmqueue.com or www.acmqueue.com/forums

SIMON CROSBY is CTO of XenSource where he is responsible for XenEnterprise R&D, technology leadership, and product management, and maintaining a close affiliation with the Xen project run by Ian Pratt, the founder of XenSource. Crosby was a principal engineer at Intel where he led research in distributed autonomic computing and platform security and trust. Before Intel, Simon founded CPlane Inc., a network optimization software vendor. He was a tenured faculty member at the University of Cambridge, where he led research on network performance and control, and multimedia operating systems.

DAVID BROWN is a member of the Solaris Engineering group at Sun Microsystems. He led the Solaris ABI compatibility program and more recently has worked on several projects to support Sun's AMD x64- and Intel-based platforms. Earlier he was a founder of Silicon Graphics and the Workstation Systems Engineering group at Digital Equipment Corporation. He introduced and described the unified memory architecture approach for high-performance graphics hardware in his Ph.D. dissertation at the University of Cambridge.