The Reincarnation of Virtual

The term *virtual machine* initially described a 1960s operating system concept: a software abstraction with the looks of a computer system's hardware (real machine). Forty years later, the term encompasses a large range of abstractions—for example, Java virtual machines that don't match an existing real machine. Despite the variations, in all definitions the virtual machine is a target for a programmer or compilation system. In other words, software is written to run on the virtual machine.

A CROSS-SECTION VIEW

One way to view the different virtual machine abstractions is as "slices" of the hardware/software stack. A modern computer system is composed of layers, beginning with the hardware and including layers of an operating system and application programs running on top of the operating system (see figure 1). Virtualization software abstracts virtual machines by interposing a layer at various places in the system. Three examples of these *virtualization layers* include hardware-level virtualization, operating system–level virtualization, and high-level language virtual machines.

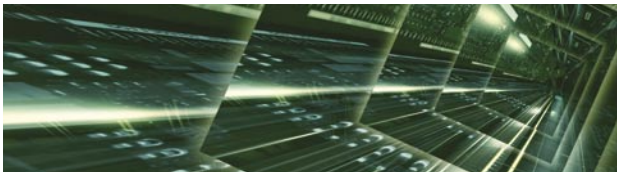
Hardware-level virtualization. Here the virtualization layer sits right on top of the hardware exporting the virtual machine abstraction. Because the virtual machine

Virtualization makes a comeback.

Machines

MENDEL ROSENBLUM,
STANFORD UNIVERSITY AND VMWARE

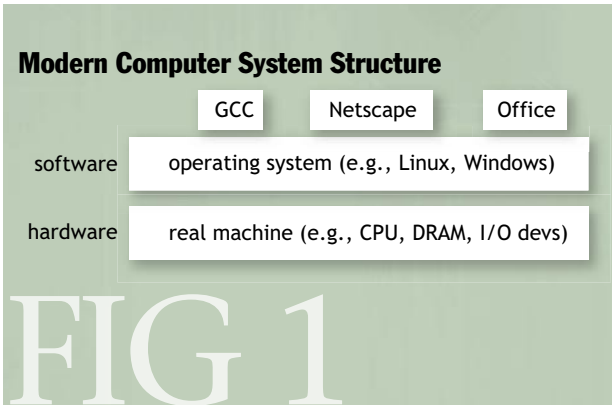
The Reincarnation of Virtual Machines

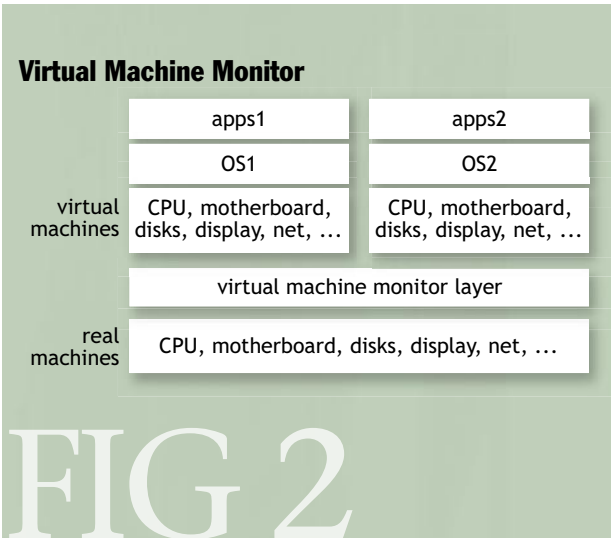


looks like the hardware, all the software written for it will run in the virtual machine. This is actually the original virtual machine definition from the 1960s, including older technology such as VM/370 on IBM mainframes—as well as VMware virtualization technology on x86-based machines, as illustrated in Figure 2. (For more on this, see Bob Supnik's "Simulators: Virtual Machines of the Past {and Future]" on page 52 of this issue.)

Operating system–level virtualization. In this case the virtualization layer sits between the operating system and the application programs that run on the operating system. The virtual machine runs applications, or sets of applications, that are written for the particular operating system being virtualized. FreeBSD Jails are an example of this technology (For more on this, see Poul-Henning Kamp and Robert Watson's "Building Systems to Be Shared" on page 42 of this issue.)

High-level language virtual machines. In high-level language virtual machines, the virtualization layer sits as an application program on top of an operating system. The layer exports an abstraction of the virtual machine that can run programs written and compiled to the particular abstract machine definition. Any program written in the high-level language and compiled for this virtual machine will run in it. Smalltalk and Java are two examples of this kind of virtual machine (For more on this, see "Interview: James Gosling" on page 24 of this issue.)





ATTRIBUTES OF VIRTUAL MACHINES

Although the chief attractions for running in a virtual machine environment differ among the various types, all share a common set of attributes.

Software compatibility. The virtual machine provides a compatible abstraction so that all software written for it will run on it. For example, a hardware-level virtual machine will run all the software, operating systems, and applications written for the hardware. Similarly, an operating system–level virtual machine will run applications for that particular operating system, and a high-level virtual machine will run programs written in the high-level language.

The virtual machine abstraction frequently can mask differences in the hardware and software layers below the virtual machine. One example is Java's claim that you can "write once, run anywhere."

Isolation. The virtual machine abstraction isolates the software running in the virtual machine from other virtual machines and real machines. This isolation provides that bugs or hackers can be contained within the virtual machine and thus not adversely affect other parts of the system. In addition to data isolation, the virtualization layer can execute performance isolation so that resources consumed by one virtual machine do not necessarily harm the performance of other virtual machines. Traditionally, operating systems are not as fair in performing resource balancing and starvation prevention as virtual

machine environments tend to be.

Encapsulation. The software layer exporting the virtual machine abstraction is an example of what is known as a *level of indirection*. This layer can be used to manipulate and control the execution of the software in the virtual machine. It can also use this indirection to enhance the software or to provide a better execution environment. For example, virtual machines for high-level languages typically support runtime checks that can reduce a class of programming errors. These include type-safe, memory-safe, and garbage-collected memory management. Overall, the layer provides a better execution environment for the programming.

Performance. Adding a layer of software to a system adds overhead, which can adversely affect the performance of the software running in the virtual machine. The benefits of successful virtual machine systems far outweigh any overhead that they introduce.

HARDWARE-LEVEL VMMS: WHY THE COMEBACK?

Although hardware-level virtual machines were popular in both the research and commercial marketplace during the 1960s and 1970s, they virtually disappeared during the 1980s and 1990s. By the time high-level language virtual machines such as Java appeared, few systems were being run, apart from IBM's mainframe and AS/400 business systems.

Hardware-level virtual machines are exported by a thin layer of software called the virtual machine monitor (VMM). The VMM runs directly on the hardware of the real machine, exporting an abstraction of the virtual machine that looks like the hardware. By making the virtual machine match the hardware interface, the virtual machine is capable of running all software written for the hardware.

The chief challenge for the VMM is to pass the real hardware to the virtual machine in a safe, transparent, and efficient way. *Safe* means that regardless of what the software virtual machine does, it should not be able to get out of its isolated environment and affect other virtual machines or the VMM. Since the software running in the virtual machine thinks it's actually running on raw hardware, the VMM is responsible for maintaining this illusion by effectively "lying" to the software about the true state of the hardware.

The trick used during the 1960s was to configure hardware in such a way that the VMM can get control whenever it needs to maintain safe control and to fool the software into thinking that it has the hardware to itself. This hardware support, called *hardware virtualization*, allows the VMM to run virtual machines in an isolated and protected environment. It's also transparent to the software running in the virtual machine, which thinks that it is in exclusive control of the hardware. Much work was done during the 1960s and 1970s to make this mapping simple and fast so that the VMM could function with only small performance and resource overhead.

Since the late 1990s there has been a renewed interest in VMMs, not only in the traditional area of servers, but also as an extension of desktop computing environments. In 1999, for example, VMware introduced the hosted VMM. It was capable of extending a modern operating system to support a virtual machine that acts and runs like the hardware-level VMM of old (see figure 3).

ATTRIBUTES OF VMMS

Primary attributes exported by modern VMMs are software compatibility, isolation capability, encapsulation, and low overhead/high performance.

Software compatibility. By making the virtual machine abstract, the real hardware, all operating systems, and applications developed for the hardware will run in the virtual machine. VMware's products export an x86-based computer capable of running all of Microsoft's operating systems including DOS, Windows 3.1, Windows 95, Windows 98, Windows NT, Windows 2000, Windows ME, and Windows XP—as well as other x86 operating systems, such as Linux and FreeBSD.

Of all the virtualization software forms described here, the hardware-level virtualization runs the most software. To maintain compatibility, it needs only to match the hardware interface, which tends to evolve much slower than the software interfaces, such as those between the application and the operating system.

Isolation capability. VMMs are able to leverage the

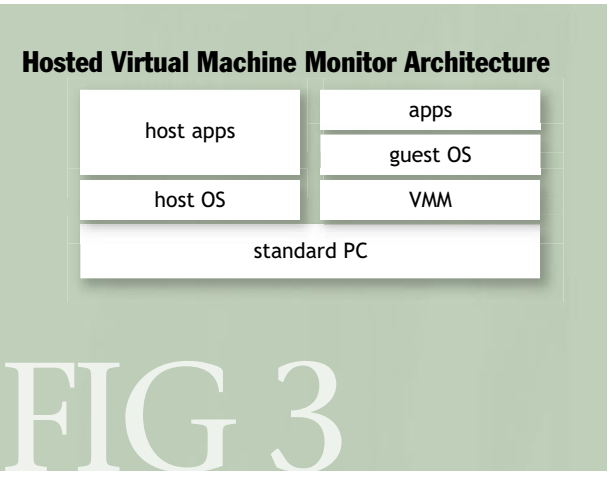
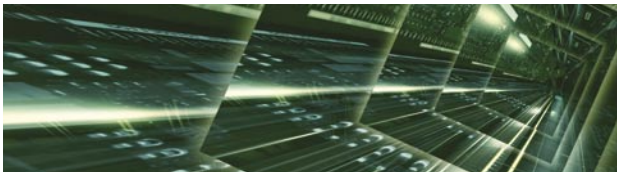


FIG 3

The Reincarnation of Virtual Machines



hardware production mechanism of the machine to isolate virtual machines from one another. Traditional VMMs use the MMU (memory management unit) of the CPU, as well as other production mechanisms, to control the access of software running in a virtual machine. This approach leads to a relatively small amount of software responsible for the isolation of virtual machines.

The small size of the isolation code helps provide a very high level of assurance that true isolation is achieved. Code that runs in a virtual machine cannot access other virtual machines or the monitor. The VMM is able to control which resources are accessible to each virtual machine. This isolation can handle both accidents and malicious attacks. Regardless of what the software in the virtual machine does, it cannot break the protection and illusions set up by the VMM.

Compared with the size and complexity of a modern operating system, which can consist of millions of lines of code, the small size of the VMM further increases the confidence of achieving true isolation. For example, the code responsible for maintaining isolation and running in the most privileged mode of the CPU in a modern operating system is hundreds of thousands of lines. The isolation code of a traditional VMM that has good virtualization support from the hardware can be measured in the tens of thousands of lines.

Traditionally, the isolation provided by the VMM has been compared with the isolation provided by having separate physical machines. This is much higher than that experienced with modern operating systems.

Encapsulation. Hardware-level virtual machines encapsulate all software that runs on the hardware, thus giving the VMM the unique ability to manage the hardware resources, as well as manipulate and control the entire software stack. The monitor allows the software running in the virtual machine to be effectively decoupled from the hardware. This decoupling provides for many of the unique features of hardware-level virtual machines.

Because all of the virtual machine software is encapsulated, the monitor can transparently manage the software and hardware in the virtual machine. The monitor can use this capability to run multiple virtual machines simultaneously on the same physical machines—or migrate running virtual machines between different hardware platforms. The monitor effectively controls all of the hardware. It also maps it to whatever virtual machines need the resource.

The virtualization layer can also smooth out minor differences between hardware platforms to allow the same virtual machine to run on them. The VMM provides a conversion layer than can map the virtual devices of a virtual machine onto different physical devices.

Finally, the encapsulation of all the software in the virtual machine allows the monitor to manage the entire software stack. The virtual machine abstraction can be used to provision software on a machine, as well as manage and load-balance it. The monitor can save or checkpoint the execution state of a virtual machine and restore it some other time. This capability allows it to effectively undo the execution of a virtual machine.

Low overhead/high performance. The techniques used by VMMs to safely map the virtual machine directly onto the hardware of the real machine result in performance close to that of the real machine. The job of the VMM is to set up the hardware so that the virtual machine's virtual hardware is mapped directly onto the real hardware resources. Because the two interface definitions (virtual and real) match, an executing virtual machine can run at full speed on the real hardware. For example, the virtual CPU is emulated by the VMM by simply scheduling the virtual machine to run a real CPU. Similarly, virtual I/O devices such as disks are emulated using real disks.

Using hardware virtualization results in VMM overhead measured in a small percentage. Software running in a virtual machine spends most of its execution time directly using the hardware resources; the performance is the same as the underlying hardware. The VMM overhead occurs when the VMM needs to get control to maintain safe isolation, or maintain the transparent illusion. These events occur relatively infrequently in most workloads.

Hardware virtualization was common during the 1960s and 1970s, but by the 1990s had disappeared from most popular hardware platforms. It appears to be making a comeback, however.

VMM TECHNOLOGY

Today, VMMs are seeing a resurgence in use in more traditional server environments, as well as on desktop

machines. Some of the reasons for this are application compatibility, program testing and development, accelerated application deployment, data isolation, and logical partitioning.

Application compatibility. When upgrading an operating system or migrating to a different operating system, it's not uncommon to find that some applications don't work on the new operating system. Maintaining 100 percent backward compatibility is extremely difficult when modifying complex software systems such as modern operating systems. This becomes an impossible task when the modifications include fixing bugs and closing security holes that the application has come to rely on.

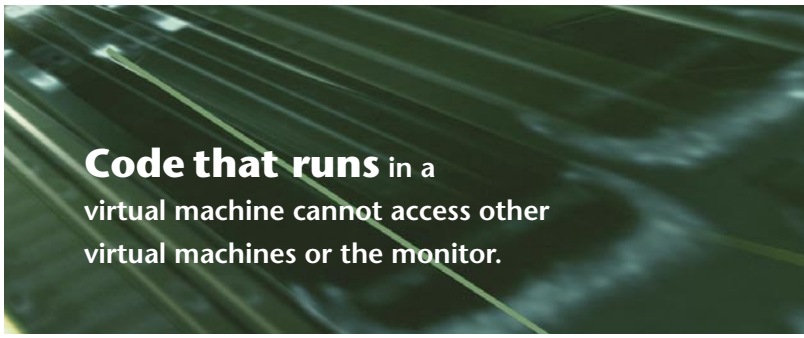
By using a hosted VMM such as the VMware's Workstation, it's possible to run both the new and old operating systems simultaneously on the same personal computer. Application programs that will not run on the new operating system can be run in the old environment, which is contained in a virtual machine.

Program testing and development. The ability to handle multiple, different, complete operating system environments is extremely useful in the development and testing of software. A software engineer can keep a library of virtual machines of different software configurations to develop or test against. Similarly, a system administrator can keep an older version, the current version, and maybe newer versions of software on hand to aid in deployment and backward-compatibility phases.

Accelerated application deployment. Preconfigured virtual machines that contain a suite of application programs that are configured and ready to run can be used to accelerate the deployment of applications. The virtual machine can be run without the time-consuming tasks of installing and configuring the suite of other dependent software applications on existing machines. This deployment mechanism can be particularly useful for demonstrating complex software environments—for example, big application suites or client-server applications that would otherwise require multiple machines. Consider a modern Web server that may need a full SQL database, middleware applications, PHP servers, Perl modules, and much more, all installed and running before the new application of a discussion board can be installed.

Data isolation. The strong isolation properties of hardware-level virtual machines can be used to segregate data within one system. The NetTop system from the NSA (National Security Agency) uses the isolation property to allow the same workstation to access both classified networks and the public Internet. Two virtual machines are used—one connected only to the classified network, and the other connected to the public network. The isolation property assures that a malicious attacker coming from the public network cannot access classified data.

Logical partitioning. Hardware-level virtualization can be used to partition a physical box to support multiple virtual machines with a technique called *logical partitioning*. By using another technique called *server consolidation*, you can consolidate multiple servers into a smaller number of boxes and then manage them from the central console. Not only is less hardware needed to support the same number of servers, the management console can use the control features of the layer to observe, stop, reboot, load-balance, and otherwise manage the server with greater ease than if they were running on physical boxes. The smaller number of physical machines also reduces the power, cooling, floor space, etc. of the servers—which can result in reduced costs. Finally, the isolation property of virtual machines means that a failure or compromise in one server will not affect the rest of the servers on the machine. For example, a failure of a print server will not take down the Exchange server. Of course, this logical partitioning assumes that the separate application servers tend to be underutilized and idle most of the time.

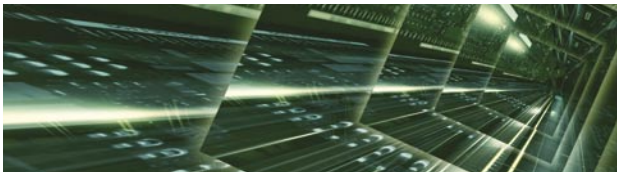


The sophisticated resource management accomplished by the VMM guarantees servers a certain amount of hardware resources. A misbehaving server cannot effect a denial-of-service attack by consuming all resources of the physical machine. In this way, each isolated server is still guaranteed a certain level of performance when combined on the single hardware platform.

LESSONS FROM THE PAST

Hardware-level virtual machine technology has been around for more than 40 years, so it's worthwhile looking at the past to learn some lessons for the future. One of the interesting questions concerns VMMs and operating systems. Both systems "believe" they should control the resources of the computer system. But the experience has

The Reincarnation of Virtual Machines



been that neither VMMs nor modern operating systems alone have been able to do this flawlessly.

For example, IBM's VM/CMS used a VMM running multiple virtual machines, each with a single-user operating system instead of a multi-user operating system such as Unix. When it became clear that a better solution would be a real multi-user operating system, IBM deemphasized VMM-based solutions in favor of MVS, its multi-user operating system. With regard to backward compatibility, modern multi-user operating systems have problems sufficiently serious that VMM technology is being reintroduced. Clearly, we must determine a balance between implementing functionality with a VMM and implementing it within the operating system.

We must also determine which software really controls the hardware resources: the VMM or the operating system. Running both a VMM and an operating system that makes resource decisions inside the VM can result in a suboptimal decision. For example, what should decide when a page of memory is no longer needed and thus can be written (i.e., "paged out") to disk—the VMM or the operating system running in the virtual machine? These resource management decisions become particularly important when running software that's highly sensitive to timing (e.g., realtime control). Solutions to these problems have included letting the software in the virtual machine "know" that it's running in a virtual machine—and communicating with the VMM on resource management decisions.

THE FUTURE OF HARDWARE VIRTUAL MACHINES

Although hardware-level virtualization went from being widely used during the 1970s to near extinction in the 1980s, it has come back in a strong way. The success of VMware's products in the commercial marketplace, together with recent hardware support for virtualization such as Intel's Vanderpool technology and extensions to IBM's Power architecture, indicate that it is a technology just now beginning to be fully realized and that it is here to stay.

Computing trends indicate that the data center of the future will likely include a hardware-level virtualization layer and a control system. Services will run in virtual machines and will be mapped onto available hardware resources. Not only will this greatly ease the management of data centers, it will also ease the handling of new hardware, as well as failed hardware. The failure of a single physical box will reduce the pool of available resources, not the availability of a particular service.

Similarly, virtual machine technology will be used to allow aggressive innovation in the area of system software, providing the ability to maintain backward compatibility. Virtual machines will allow for the support of old applications, as well as the current versions, and will test the deployment of new versions that are all based on the same hardware.

One consequence of Moore's law of semiconductor growth has been the exponential increase in the performance of computing systems. The overhead of a well-tuned hardware virtualization system is extremely small compared with the performance increase. This means that the computing industry can, for only a few percentage points of performance, realize the huge benefits of hardware-level virtualization. Such benefits include the management of both the hardware and the software that runs in virtual machines—currently a large expense in modern computing environments. Q

LOVE IT, HATE IT? LET US KNOW

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

MENDEL ROSENBLUM, associate professor in the computer science department at Stanford University, cofounded VMware in 1998 and serves as its chief scientist. He received a B.A. in mathematics from the University of Virginia (1984) and an M.S. (1989) and Ph.D. (1992) in computer science from the University of California at Berkeley. He was recipient of the 1992 National Science Foundation's National Young Investigator award, the 1994 Alfred P. Sloan Foundation Research Fellowship, and was a cowinner of the 1992 ACM Doctoral Dissertation Award and the 2002 ACM/SIGOPS Mark Weiser Award for creativity and innovation in operating systems research. His research interests include system software, distributed systems, and computer architecture. Rosenblum has published material on disk storage management, computer simulation techniques, scalable operating system structure, virtualization computer security, and mobility.