Virtual Machine Monitors: Current Technology and Future Trends

Developed more than 30 years ago to address mainframe computing problems, virtual machine monitors have resurfaced on commodity platforms, offering novel solutions to challenges in security, reliability, and administration.

Mendel Rosenblum
VMware Inc.

Tal Garfinkel
Stanford University

At the end of the 1960s, the virtual machine monitor (VMM) came into being as a software-abstraction layer that partitions a hardware platform into one or more virtual machines.[1] Each of these virtual machines was sufficiently similar to the underlying physical machine to run existing software unmodified.

At the time, general-purpose computing was the domain of large, expensive mainframe hardware, and users found that VMMs provided a compelling way to multiplex such a scarce resource among multiple applications. Thus, for a brief period, this technology flourished both in industry and in academic research.

The 1980s and 1990s, however, brought modern multitasking operating systems and a simultaneous drop in hardware cost, which eroded the value of VMMs. As mainframes gave way to minicomputers and then PCs, VMMs disappeared to the extent that computer architectures no longer provided the necessary hardware to implement them efficiently. By the late 1980s, neither academics nor industry practitioners viewed VMMs as much more than a historical curiosity.

Fast forwarding to 2005, VMMs are again a hot topic in academia and industry: Venture capital firms are competing to fund startup companies touting their virtual-machine-based technologies. Intel, AMD, Sun Microsystems, and IBM are developing virtualization strategies that target markets with revenues in the billions and growing. In research labs and universities, researchers are developing approaches based on virtual machines to solve mobility, security, and manageability problems.

What happened between the VMM's essential retirement and its current resurgence?

In the 1990s, Stanford University researchers began to look at the potential of virtual machines to overcome difficulties that hardware and operating system limitations imposed: This time the problems stemmed from massively parallel processing (MPP) machines that were difficult to program and could not run existing operating systems. With virtual machines, researchers found they could make these unwieldy architectures look sufficiently similar to existing platforms to leverage the current operating systems. From this project came the people and ideas that underpinned VMware Inc. (www.vmware.com), the original supplier of VMMs for commodity computing hardware. The implications of having a VMM for commodity platforms intrigued both researchers and entrepreneurs.

WHY THE REVIVAL?

Ironically, the capabilities of modern operating systems and the drop in hardware cost—the very

0018-9162/05/$20.00 © 2005 IEEE Published by the IEEE Computer Society May 2005

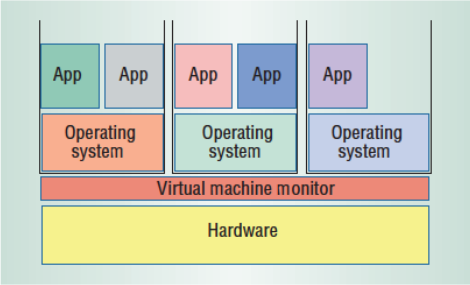

Figure 1. Classic VMM. The VMM is a thin software layer that exports a virtual machine abstraction. The abstraction looks enough like the hardware that any software written for that hardware will run in the virtual machine.

combination that had obviated the use of VMMs during the 1980s—began to cause problems that researchers thought VMMs might solve. Less expensive hardware had led to a proliferation of machines, but these machines were often underused and incurred significant space and management overhead. And the increased functionality that had made operating systems more capable had also made them fragile and vulnerable.

To reduce the effects of system crashes and breakins, system administrators again resorted to a computing model with one application running per machine. This in turn increased hardware requirements, imposing significant cost and management overhead. Moving applications that once ran on many physical machines into virtual machines and consolidating those virtual machines onto just a few physical platforms increased use efficiency and reduced space and management costs. Thus, the VMM's ability to serve as a means of multiplexing hardware—this time in the name of server consolidation and utility computing—again led it to prominence.

Moving forward, a VMM will be less a vehicle for multitasking, as it was originally, and more a solution for security and reliability. In many ways VMMs give operating systems developers another opportunity to develop functionality no longer practical in today's complex and ossified operating systems, where innovation moves at a geologic pace. Functions like migration and security that have proved difficult to achieve in modern operating systems seem much better suited to implementation at the VMM layer. In this context, VMMs provide a backward-capability path for deploying innovative operating system solutions, while providing the ability to safely pull along the existing software base.

DECOUPLING HARDWARE AND SOFTWARE

As Figure 1 shows, the VMM decouples the software from the hardware by forming a level of indirection between the software running in the virtual machine (layer above the VMM) and the hardware. This level of indirection lets the VMM exert tremendous control over how *guest operating systems* (GuestOSs)—operating systems running inside a virtual machine—use hardware resources.

A VMM provides a *uniform view* of underlying hardware, making machines from different vendors with different I/O subsystems look the same, which means that virtual machines can run on any available computer. Thus, instead of worrying about individual machines with tightly coupled hardware and software dependencies, administrators can view hardware simply as a pool of resources that can run arbitrary services on demand.

Because the VMM also offers complete *encapsulation* of a virtual machine's software state, the VMM layer can map and remap virtual machines to available hardware resources at will and even migrate virtual machines across machines. Load balancing among a collection of machines thus becomes trivial, and there is a robust model for dealing with hardware failures or for scaling systems. When a computer fails and must go offline or when a new machine comes online, the VMM layer can simply remap virtual machines accordingly. Virtual machines are also easy to replicate, which lets administrators bring new services online as needed.

Encapsulation also means that administrators can suspend virtual machines and resume them at arbitrary times or checkpoint them and roll them back to a previous execution state. With this general-purpose undo capability, systems can easily recover from crashes or configuration errors. Encapsulation also supports a very general mobility model, since users can copy a suspended virtual machine over a network or store and transport it on removable media.

The VMM can also provide *total mediation* of all interactions between the virtual machine and underlying hardware, thus allowing strong isolation between virtual machines and supporting the multiplexing of many virtual machines on a single hardware platform. The VMM can then consolidate a collection of virtual machines with low resources onto a single computer, thereby lowering hardware costs and space requirements.

Strong isolation is also valuable for reliability and security. Applications that previously ran together on one machine can now separate into different virtual machines. If one application crashes the operating system because of a bug, the other applications are isolated from this fault and can con-

tinue running undisturbed. Further, if attackers compromise a single application, the attack is contained to just the compromised virtual machine.

Thus, VMMs are a tool for restructuring systems to enhance robustness and security—without imposing the space or management overhead that would be required if applications executed on separate physical machines.

VMM IMPLEMENTATION ISSUES

The VMM must be able to export a hardware interface to the software in a virtual machine that is roughly equivalent to raw hardware and simultaneously maintain control of the machine and retain the ability to interpose on hardware access. Various techniques can help achieve this, each offering different design tradeoffs.

When evaluating these tradeoffs, the central design goals for VMMs are compatibility, performance, and simplicity. Compatibility is clearly important, since the VMM's chief benefit is its ability to run legacy software. The goal of performance, a measure of virtualization overhead, is to run the virtual machine at the same speed as the software would run on the real machine. Simplicity is particularly important because a VMM failure is likely to cause all the virtual machines running on the computer to fail. In particular, providing secure isolation requires that the VMM be free of bugs that attackers could use to subvert the system.

CPU virtualization

A CPU architecture is *virtualizable* if it supports the basic VMM technique of *direct execution*—executing the virtual machine on the real machine, while letting the VMM retain ultimate control of the CPU.

Implementing basic direct execution requires running the virtual machine's privileged (operating-system kernel) and unprivileged code in the CPU's unprivileged mode, while the VMM runs in privileged mode. Thus, when the virtual machine attempts to perform a privileged operation, the CPU traps into the VMM, which emulates the privileged operation on the virtual machine state that the VMM manages.

The VMM handling of an instruction that disables interrupts provides a good example. Letting a guest operating system disable interrupts would not be safe since the VMM could not regain control of the CPU. Instead, the VMM would trap the operation to disable interrupts and then record that interrupts were disabled for that virtual machine. The VMM would then postpone delivering subsequent interrupts to the virtual machine until it reenables interrupts.

Consequently, the key to providing virtualizable architecture is to provide trap semantics that let a VMM safely, transparently, and directly use the CPU to execute the virtual machine. With these semantics, the VMM can use direct execution to create the illusion of a normal physical machine for the software running inside the virtual machine.

The central design goals for VMMs are compatibility, performance, and simplicity.

Challenges. Unfortunately, most modern CPU architectures were not designed to be virtualizable, including the popular x86 architecture. For example, x86 operating systems use the x86 `POPF` instruction (pop CPU flags from stack) to set and clear the interrupt-disable flag. When it runs in unprivileged mode, `POPF` does not trap. Instead, it simply ignores the changes to the interrupt flag, so direct execution techniques will not work for privileged-mode code that uses this instruction.

Another challenge of the x86 architecture is that unprivileged instructions let the CPU access privileged state. Software running in the virtual machine can read the code segment register to determine the processor's current privilege level. A virtualizable processor would trap this instruction, and the VMM could then patch what the software running in the virtual machine sees to reflect the virtual machine's privilege level. The x86, however, doesn't trap the instruction, so with direct execution, the software would see the wrong privilege level in the code segment register.

Techniques. Several techniques address how to implement VMMs on CPUs that can't be virtualized, the most prevalent being paravirtualization[2] and direct execution combined with fast binary translation. With paravirtualization, the VMM builder defines the virtual machine interface by replacing nonvirtualizable portions of the original instruction set with easily virtualized and more efficient equivalents. Although operating systems must be ported to run in a virtual machine, most normal applications run unmodified.

Disco,[3] a VMM for the nonvirtualizable MIPS architecture, used paravirtualization. Disco designers changed the MIPS interrupt flag to be simply a special memory location in the virtual machine rather than a privileged register in the processor. They replaced the MIPS equivalent of the x86 `POPF` instruction and the read access to the code segment register with accesses to this special memory location. This replacement also eliminated virtualization overhead such as traps on privileged instructions, which resulted in increased performance.

Building a VMM that offers full compatibility and high performance is a significant engineering challenge.

The designers then modified a version of the Irix operating system to take advantage of this paravirtualized version of the MIPS architecture.

The biggest drawback to paravirtualization is incompatibility. Any operating system run in a paravirtualized VMM must be ported to that architecture. Operating system vendors must cooperate, legacy operating systems cannot run, and existing machines cannot easily migrate into virtual machines. With years of excellent backward-compatible x86 hardware, huge amounts of legacy software are still in use, which means that giving up backward compatibility is not trivial.

In spite of these drawbacks, academic research projects have favored paravirtualization because building a VMM that offers full compatibility and high performance is a significant engineering challenge.

To provide fast, compatible virtualization of the x86 architecture, VMware developed a new virtualization technique that combines traditional direct execution with fast, on-the-fly binary translation. In most modern operating systems, the processor modes that run normal application programs are virtualizable and hence can run using direct execution. A binary translator can run privileged modes that are nonvirtualizable, patching the nonvirtualizable x86 instructions. The result is a high-performance virtual machine that matches the hardware and thus maintains total software compatibility.

Others have developed binary translators[4] that translate code between CPUs with different instruction sets. VMware's binary translation is much simpler because the source and target instruction sets are nearly identical. The VMM's basic technique is to run privileged mode code (kernel code) under control of the binary translator. The translator translates the privileged code into a similar block, replacing the problematic instructions, which lets the translated block run directly on the CPU. The binary translation system caches the translated block in a *trace cache* so that translation does not occur on subsequent executions.

The translated code looks much like the results from the paravirtualized approach: Normal instructions execute unchanged, while the translator replaces instructions that need special treatment, like `POPF` and reads from the code segment registers with an instruction sequence similar to what a paravirtualized virtual machine would need to run. There is one important difference, however: Rather than applying the changes to the source code of the operating system or applications, the binary translator applies the changes when the code first executes.

While binary translation does incur some overhead, it is negligible on most workloads. The translator runs only a fraction of the code, and execution speeds are nearly indistinguishable from direct execution once the trace cache has warmed up.

Binary translation is also a way to optimize direct execution. For example, privileged code that frequently traps can incur significant additional overhead when using direct execution since each trap transfers control from the virtual machine to the monitor and back. Binary translation can eliminate many of these traps, which results in a lower overall virtualization overhead. This is particularly true on CPUs with deep instruction pipelines, such as the modern x86 CPUs, where traps incur high overhead.

Future support. In the near term, both Intel with its Vanderpool technology and AMD with its Pacifica technology have announced hardware support for x86 CPU VMMs. Rather than making existing execution modes virtualizable, both the Intel and AMD technologies add a new execution mode to the processor that lets a VMM safely and transparently use direct execution for running virtual machines. To improve performance, the mode attempts to reduce both the traps needed to implement virtual machines and the time it takes to perform the traps.

When these technologies become available, direct-execution-only VMMs could be possible on x86 processors, at least for operating system environments that do not use these new execution modes.

If this hardware support works as well as the IBM mainframe virtualization support of the early days, it should be possible to decrease performance overhead even more, as well as simplifying the implementation of virtualization techniques.

Lessons from the past indicate that adequate hardware support can decrease overhead, even without paravirtualization, to the point that the value of having a fully compatible virtual machine abstraction overrides any performance benefits from breaking compatibility.

Memory virtualization

The traditional implementation technique for virtualizing memory is to have the VMM maintain a shadow of the virtual machine's memory-management data structure. This data structure, the *shadow page table*, lets the VMM precisely control which pages of the machine's memory are available to a virtual machine.

When the operating system running in a virtual machine establishes a mapping in its page table, the VMM detects the changes and establishes a mapping in the corresponding shadow page table entry that points to the actual page location in the hardware memory. When the virtual machine is executing, the hardware uses the shadow page table for memory translation so that the VMM can always control what memory each virtual machine is using.

Like a traditional operating system's virtual memory subsystems, the VMM can page the virtual machine to a disk so that the memory allocated to virtual machines can exceed the hardware's physical memory size. Because this effectively lets the VMM overcommit the machine memory, the virtual machine workload requires less hardware. The VMM can dynamically control how much memory each virtual machine gets according to what it needs.

Challenges. The VMM's virtual memory subsystem constantly controls how much memory goes to a virtual machine, and it must periodically reclaim some of that memory by paging a portion of the virtual machine out to disk. The operating system running in the virtual machine (the GuestOS), however, is likely to have much better information than a VMM's virtual memory system about which pages are good candidates for paging out. For example, a GuestOS might note that the process that created a page has exited, which means nothing will access the page again. The VMM operating at the hardware level does not see this and might wastefully page out that page.

To address this problem, VMware's ESX Server[5] adopted a paravirtualization-like approach, in which a *balloon process* running inside the GuestOS can communicate with the VMM. When the VMM wants to take memory away from a virtual machine, it asks the balloon process to allocate more memory, essentially "inflating" the process. The GuestOS then uses its superior knowledge about page replacement to select the pages to give to the balloon process, which the process then passes to the VMM for reallocation. The increased memory pressure caused by inflating the balloon process causes the GuestOS to intelligently page memory to the virtual disk.

A second challenge for memory virtualization is the size of modern operating systems and applications. Running multiple virtual machines can waste considerable memory by storing redundant copies of code and data that are identical across virtual machines.

To address this challenge, VMware designers developed content-based page sharing for their server products. In this scheme, the VMM tracks the contents of physical pages, noting if they are identical. If so, the VMM modifies the virtual machine's shadow page tables to point to only a single copy. The VMM can then deallocate the redundant copy, thereby freeing the memory for other uses.

As with a normal copy-on-write page-sharing scheme, the VMM gives each virtual machine its own copy of the page if the contents later diverge. To give an idea of potential savings, an x86 computer might have 30 virtual machines running Microsoft Windows 2000 but only one copy of the Windows kernel in the computer's memory—a significant reduction in physical memory use.

Resource management holds great promise as an area for future research.

Future support. Operating systems make frequent changes to their page tables, so keeping shadow copies up to date in software can incur undesirable overhead. Hardware-managed shadow page tables have long been present in mainframe virtualization architectures and would prove a fruitful direction for accelerating x86 CPU virtualization.

Resource management holds great promise as an area for future research. Much work remains in investigating ways for VMMs and guest operating systems to make cooperative resource management decisions. In addition, research must look at resource management at the entire data center level, and we expect significant strides will be made in this area in the coming decade.

I/O virtualization

Thirty years ago, the I/O subsystems of IBM mainframes used a channel-based architecture, in which access to the I/O devices was through communication with a separate channel processor. By using a channel processor, the VMM could safely export I/O device access directly to the virtual machine. The result was a very low virtualization overhead for I/O. Rather than communicating with the device using traps into the VMM, the software in the virtual machine could directly read and write the device. This approach worked well for the I/O devices of that time, such as text terminals, disks, card readers, and card punches.

Challenges. Current computing environments, with their richer and more diverse collection of I/O devices, make virtualizing I/O much more difficult. The x86-based computing environments support a huge collection of I/O devices from different vendors with different programming interfaces. Consequently, the job of writing a VMM layer that talks to these various devices becomes a huge effort. In

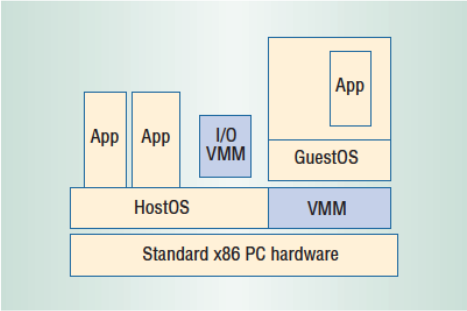

Figure 2. VMware's hosted architecture. Rather than running as a layer below all other software, the hosted architecture shares the hardware with an existing operating system (HostOS).

addition, some devices such as a modern PC's graphics subsystem or a modern server's network interface have extremely high performance requirements. This makes low-overhead virtualization an even more critical prerequisite for widespread acceptance.

Exporting a standard device interface means that the virtualization layer must be able to communicate with the computer's I/O devices. To provide this capability, VMware Workstation, a product targeting desktop computers, developed the hosted architecture[6] shown in Figure 2. In this architecture, the virtualization layer uses the device drivers of a host operating system (HostOS) such as Windows or Linux to access devices. Because most I/O devices have drivers for these operating systems, the virtualization layer can support any I/O device.

When the GuestOS gives the command to read or write blocks from the virtual disk, the virtual layer translates the command into a system call that reads or writes a file in the HostOS's file system. Similarly, the I/O VMM renders the virtual machine's virtual display card in a window on the HostOS, which lets the HostOS control, drive, and manage the virtual machine's I/O display devices regardless of what devices the GuestOS thinks are present.

The hosted architecture has three important advantages. First, the VMM is simple to install because users can install it like an application on the HostOS rather than on the raw hardware, as with traditional VMMs. Second, the hosted architecture fully accommodates the rich diversity of I/O devices in the x86 PC marketplace. Third, the VMM can use the scheduling, resource management, and other services the HostOS environment offers.

The disadvantages of the hosted architecture became material when VMware started to develop products for the x86 server marketplace. The hosted architecture greatly increases the performance overhead for I/O device virtualization. Each I/O request must transfer control to the HostOS environment and then transition through the HostOS's software layers to talk to the I/O devices. For server environments with high-performance network and disk subsystems, the resulting overhead was unacceptably high.

Another problem is that modern operating systems such as Windows and Linux do not have the resource-management support to provide performance isolation and service guarantees to the virtual machines—a feature that many server environments require.

ESX Server[5] adopts a more traditional VMM approach, running directly on the hardware without a host operating system. In addition to sophisticated scheduling and resource management, ESX Server has a highly optimized I/O subsystem for network and storage devices.

The ESX Server kernel can use device drivers from the Linux kernel to talk directly to the device, resulting in significantly lower virtualization overhead for I/O devices. VMware could use this approach because relatively few network and storage I/O devices have passed certification to run in major x86 vendor server machines. Limiting support to these I/O devices makes directly managing the I/O devices feasible for servers.

Yet another performance optimization in VMware's products is the ability to export special highly optimized virtual I/O devices that don't correspond to any existing I/O devices. Like the paravirtualization approach for CPUs, this use of paravirtualization requires that GuestOS environments use a special device driver to access the I/O devices. The result is a more virtualization-friendly I/O device interface with lower overhead for communicating the I/O commands from the GuestOS and thus higher performance.

Future support. Like CPU trends, industry trends in I/O subsystems point toward hardware support for high-performance I/O device virtualization. Discrete I/O devices, such as the standard x86 PC keyboard controller and IDE disk controllers that date back to the original IBM PC, are giving way to channel-like I/O devices, such as USB and SCSI. Like the IBM mainframe I/O channels, these I/O interfaces greatly ease implementation complexity and reduce virtualization overhead.

With adequate hardware support, safely passing these channel I/O devices directly to the software in the virtual machine should be possible, effectively eliminating all I/O virtualization overhead. For this to work, I/O devices will need to know about virtual machines and be able to support multiple virtual interfaces so that the VMM can safely map the

interface into the virtual machine. In this way, the virtual machine's device drivers will be able to communicate directly with the I/O device without the overhead of trapping into the VMM.

I/O devices that perform direct memory access will require address remapping. The remapping ensures that the memory addresses that the device driver running in the virtual machine specifies will get mapped to the locations in the computer's memory that the shadow page tables specify. For the isolation property to hold, the device should be able to access only memory belonging to the virtual machine regardless of how the driver in the virtual machine programs the device.

In a system with multiple virtual machines using the same I/O device, the VMM will need an efficient mechanism for routing device completion interrupts to the correct virtual machine. Finally, virtualizable I/O devices will need to interface to the VMM to maintain isolation between hardware and software and ensure that the VMM can continue to migrate and take a checkpoint of the virtual machines. I/O devices that provide this kind of support could minimize virtualization overhead, allowing the use of virtual machines for even the most I/O-intensive workloads. Besides performance, a significant benefit is the improved security and reliability gained from removing complex device driver code from the VMM.

WHAT'S AHEAD?

An examination of current products and recent research provides some interesting insights into the future of VMMs and the demands they will place on virtualization technology.

Server side

In the data center, administrators will be able to quickly provision, monitor, and manage thousands of virtual machines running on hundreds of physical boxes—all from a single console. Rather than configuring individual computers, system administrators will create new servers by instantiating a new virtual machine from an existing template and mapping these virtual machines onto physical resources according to specific administration policies. Rather than thinking of any computer as providing a particular fixed service, administrators will view computers simply as part of a pool of generic hardware resources. An example of this technology is VMware's Virtual Center.

This mapping of a virtual machine to hardware resources will be highly dynamic. Hot migration capabilities, such as those in VMware's VMotion technology, will let virtual machines move rapidly between physical machines according to the data center's needs. The VMM can handle traditional hardware-management problems, such as hardware failure, simply by placing the virtual machines running on the failed computer onto other correctly functioning hardware. The ability to move running virtual machines also eases some hardware challenges, such as scheduling preventive maintenance, dealing with equipment lease ends, and deploying hardware upgrades. Administrators can use hot migration to perform these tasks without service interruptions.

Virtual machines provide a powerful unifying paradigm for restructuring desktop management.

Today, manual migration is the norm, but the future should see a virtual machine infrastructure that automatically performs load balancing, detects impending hardware failures and migrates virtual machines accordingly, and creates and destroys virtual machines according to demand for particular services.

Beyond the machine room

As the pervasive use of virtual machines moves from the server room to the desktop, their effects on computing will become even more profound. Virtual machines provide a powerful unifying paradigm for restructuring desktop management.[7] The provisioning benefits that VMMs bring to the machine room apply equally to the desktop and help solve the management challenges that large collections of desktop and laptop machines impose.

Solving problems in the VMM layer benefits all software running in the virtual machine, regardless of the software's age (legacy or latest release) or its vendor. This operating system independence also reduces the need to buy and maintain redundant infrastructure. Instead of *n* versions of help desk or backup software, for example, only one version—the one that operates at the VMM level—would require support.

Virtual machines could also significantly change how users think about computers. If ordinary users can easily create, copy, and share virtual machines, the use models could be vastly different from those in computing environments with hardware availability constraints. Software developers, for example, can use products like VMware Workstation to easily set up a network of machines for testing, or they can keep their own set of test machines for every target platform.

The increased mobility of virtual machines will also significantly change machine use. Projects such as The Collective[7] and Internet Suspend/Resume[8]

VMMs offer the potential to restructure existing software systems to provide greater security.

demonstrate the feasibility of migrating a user's entire computing environment over the local and wide area. The availability of large-capacity, inexpensive removable media in the form of USB hard drives might mean that users can bring their computing environments with them wherever they go.

The increasingly dynamic character of virtual machine-based environments will also require more dynamic network topologies. Virtual switches, virtual firewalls, and overlay networks will be an integral part of a future in which the logical computing environment is decoupled from the physical location.

Security improvements

VMMs offer the potential to restructure existing software systems to provide greater security, while also facilitating new approaches to building secure systems. Current operating systems provide poor isolation, leaving host-based security mechanisms subject to attack. Moving these capabilities outside a virtual machine—so that they run alongside an operating system but are isolated from it—offers the same functionality but with much stronger resistance to attack. Two research examples of such systems are Livewire,[9] a system that uses a VMM for advanced intrusion detection on the software in the virtual machines, and ReVirt,[10] which uses the VMM layer to analyze the damage hackers might have caused during the break-in. These systems not only gain greater attack resistance from operating outside the virtual machine, but also benefit from the ability to interpose and monitor the system inside the virtual machine at a hardware level.

Placing security outside a virtual machine provides an attractive way to quarantine the network—limiting a virtual machine's access to a network to ensure that it is neither malicious nor vulnerable to attack. By controlling network access at the virtual machine layer and inspecting virtual machines before permitting (or limiting) access, virtual machines become a powerful tool for limiting the spread of malicious code in networks.

Virtual machines are also particularly well suited as a building block for constructing high-assurance systems. The US National Security Administration's NetTop architecture, for example, uses VMware's VMM to isolate multiple environments, each of which has access to separate networks with varying security classifications. Applications like this illustrate the need to continue researching and developing support for building ever smaller VMMs with increasingly higher assurance.

VMMs are particularly interesting in that they support the ability to run multiple software stacks with different security levels. Because they can specify the software stack from the hardware up, virtual machines provide maximum flexibility in trading off performance, backward compatibility, and assurance. Further, specifying an application's complete software stack simplifies reasoning about its security. In contrast, it is almost impossible to reason about the security of a single application in today's operating systems because processes are poorly isolated from one another. Thus, an application's security depends on the security of every other application on the machine.

These capabilities make VMMs particularly well suited for building trusted computing, as the Terra system[11] demonstrates. In Terra, the VMM can authenticate software running inside a virtual machine to remote parties, in a process called *attestation*.

Suppose, for example, that a user's desktop machine is running multiple virtual machines simultaneously. The user might have a relatively low-security Windows virtual machine for Web browsing, a higher-security virtual machine with a hardened Linux virtual machine for day-to-day work, and a still higher-security virtual machine comprising a special-purpose high-security operating system and a dedicated mail client for sensitive internal mail.

A remote server could require attestation from each virtual machine to confirm its contents; for example, the company file server might allow only the hardened Linux virtual machine to interact with it, while the secure-mail virtual machine might be able to connect only to a dedicated mail server. In both scenarios the servers are also likely to be running in virtual machines, permitting mutual authentication to take place.

Finally, the flexible resource management that VMMs provide can make systems more resistant to attack. The ability to rapidly replicate virtual machines and dynamically adapt to large workloads can provide a powerful tool for dealing with the scaling demands that flash crowds and distributed denial-of-service attacks can impose.

Software distribution

For the software industry, the ubiquitous deployment of VMMs has significant implications. The VMM layer provides exciting possibilities for software companies to distribute entire virtual machines containing complex software environments. Oracle, for example, has distributed more than 10,000 fully functional copies of its latest database environment

in virtual machines. Rather than having to install the entire complex environment to test the software, users simply boot the virtual machine.

Although the use of virtual machines as a distribution mechanism is widespread for software demonstration, the model could also work well for production environments, creating a fundamentally different way of distributing software. Administrators using VMware's ACE product can publish virtual machines and control how these virtual machines can be used. The Collective project explored in depth the idea of bundling applications into *virtual appliances*. The idea is to provide file servers, desktop applications, and so on in a form that lets users treat the virtual machines as a standalone application. An appliance maintainer handles issues like patch management, thus relieving normal users of the maintenance burden.

The virtual machine-based distribution model will require software vendors to update their license agreements. Software that is licensed to run on a particular CPU or physical machine will not translate as well into this new environment, relative to licenses based on use or to sitewide licenses. Users and system administrators will tend to favor operating system environments that they can easily and inexpensively distribute in virtual machines, rather than more restrictive and expensive options.

The VMM resurgence seems to be fundamentally altering the way software and hardware designers view, manage, and structure complex software environments. VMMs also provide a backward-capability path for deploying innovative operating system solutions that both meet current needs and safely pull along the existing software base. This capability will be key to meeting future computing challenges.

Companies are increasingly abandoning the strategy of procuring individual machines and tightly bundling complex software environments. VMMs are giving these fragile, difficult-to-manage systems new freedom. In coming years, virtual machines will move beyond their simple provisioning capabilities and beyond the machine room to provide a fundamental building block for mobility, security, and usability on the desktop. Indeed, VMM capabilities should continue to be an important part of the shift in the computing landscape. ■

References

1. R.P. Goldberg, "Survey of Virtual Machine Research," *Computer*, June 1974, pp. 34-45.
2. A. Whitaker, M. Shaw, and S. Gribble, "Scale and Performance in the Denali Isolation Kernel," *ACM SIGOPS Operating Systems Rev.*, vol. 36, no. SI, Winter 2002, pp. 195-209.
3. E. Bugnion et al., "Disco: Running Commodity Operating Systems on Scalable Multiprocessors," *ACM Trans. Computer Systems,* vol. 15, no. 4, 1997, pp. 412-447.
4. R. Sites et al., "Binary Translation," *Comm. ACM*, Feb. 1993, pp. 69-81.
5. C. Waldspurger, "Memory Resource Management in VMware ESX Server," *ACM SIGOPS Operating Systems Rev.*, vol. 36, no. SI, Winter 2002, pp. 181-194.
6. J. Sugerman, G. Venkitachalam, and B. Lim, "Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor," *Proc. Usenix Ann. Technical Conf.*, Usenix, 2002, pp. 1-14.
7. R. Chandra et al., "The Collective: A Cache-Based Systems Management Architecture," *Proc. Symp. Network Systems Design and Implementation*, Usenix, 2005, to appear.
8. M. Kozuch and M. Satyanarayanan, "Internet Suspend/Resume," *Proc. IEEE Workshop Mobile Computing Systems and Applications,* IEEE Press, 2002, pp. 40-46.
9. T. Garfinkel and M. Rosenblum, "A Virtual Machine Introspection-Based Architecture for Intrusion Detection," *Proc. Network and Distributed Systems Security Symp.*, The Internet Society, 2003, pp. 191-206.
10. G. Dunlap et al., "ReVirt: Enabling Intrusion Analysis through Virtual-Machine Logging and Replay," *ACM SIGOPS Operating Systems Rev.,* vol. 36, no. SI, Winter 2002, pp. 211-224.
11. T. Garfinkel et al., "Terra: A Virtual-Machine-Based Platform for Trusted Computing," *Proc. ACM Symp. Operating Systems Principles*, ACM Press, 2003, pp. 192-206.

Mendel Rosenblum *is an associate professor of computer science at Stanford University and a cofounder and chief scientist at VMware Inc. His research interests include system software, distributed systems, computer architecture, and security. Rosenblum received a PhD in computer science from the University of California, Berkeley. Contact him at mendel@cs.stanford.edu.*

Tal Garfinkel *is a PhD candidate in computer science at Stanford University. His research interests include operating systems, distributed systems, computer architecture, and security. He received a BA in computer science from the University of California, Berkeley. Contact him at talg@cs stanford.edu.*