Virtualization

Fred Douglis
EMC Backup Recovery Systems

Orran Krieger
Boston University

One of the most famous adages in computer science is that "any problem in computer science can be solved by another level of indirection." (This phrase, commonly attributed to Roger Needham, was in fact attributed by Needham to David Wheeler. Thus, it proves itself.) Increasingly, that level of indirection takes the form of virtualization, in which a resource's consumers are provided with a virtual rather than physical version of that resource. This layer of indirection has helped address myriad problems, including efficiency, security, high availability, elasticity, fault containment, mobility, and scalability.

Virtualization has been a part of the computing landscape for nearly half a century. In the 1960s and 1970s, IBM developed the Control Program/Cambridge Monitor System (CP/CMS) which led into VM/370. These systems let each user run what appeared to be an isolated system, but all within one timeshared computing environment. Language-level virtualization was introduced around the 1980s to support application-level portability and isolation (Smalltalk-80, developed by Xerox PARC, is probably the best example of this). The Java Virtual Machine, introduced by Sun in the 1990s, was in a unique position: coming at the start of the World Wide Web, it offered developers an opportunity to add executable content to the Web in a portable and secure manner. Although language-level virtualization had a huge impact, a significant gap exists between a VM for a programming language runtime environment and one for an entire operating system. But once the challenges of virtualizing modern computer systems were addressed, the march toward widespread adoption of machine-level virtualization was rapid and inexorable. The renaissance for VMs can most likely be attributed to Disco,[1] a Stanford research project that ultimately led to VMware.

Although VMs are the most obvious example of virtualization, others include desktop sharing (Virtual Network Computing), virtual networks, virtual storage, and many more. All these have an enormous impact on Internet computing. Two personal anecdotes are illustrative of how virtualization can dramatically simplify problems.

Mobility

One of us (Fred Douglis) once worked extensively on the problem of process mobility (known as *process migration*). Migrating workloads from one physical

Published by the IEEE Computer Society
1089-7801/13/$31.00 © 2013 IEEE

machine to another is important for balancing load, improving performance (for instance, by colocating frequently communicating processes), dealing with failures, and so on. The work was successful in specialized systems such as V,[2] Accent,[3] Sprite,[4] Mach,[5] and Mosix;[6] however, migration was extremely fragile and hard to maintain as a system evolved. In 2000, Douglis coauthored a survey article on process migration[7] with a section discussing why it had failed to achieve commercial acceptance. While briefly acknowledging VMs' potential to facilitate process migration (as mentioned in the Disco work), the article didn't realize that virtualization would be the key to making process migration commercially practical.

In 2001, Peter Chen and Brian Noble published a position paper, "When Virtual Is Better Than Real,"[8] arguing for the benefits of virtual environment migration; in 2002, Stanford researchers published "Optimizing the Migration of Virtual Computers"[9] and ushered in the new era of virtual migration. It turns out that by encapsulating the state of the applications you want to migrate as an autonomous VM, you can solve many of the odd interprocess dependencies that otherwise make process migration so difficult and fragile.

Migration has now become ubiquitous and a critical enabling technology, and is being employed in production in ways the original research didn't envision. It has allowed the industry to increase the use of data center servers by exploiting workload variability to host more workloads on fewer machines. It allows better failure models, for instance, by offloading ongoing processes prior to upgrades or other downtime. It's critical to green computing, letting hosts power off when demand is low. It's even being used today across data centers and — experimentally, as described in one article in this issue — between clouds. Without virtualization, the promise of workload migration would have gone unrealized.

Support for Games

Virtualization is normally thought to provide benefits to enable individual VMs to achieve larger goals, but at a cost in performance. However, one of us (Orran Krieger) first experienced VMs in a very different context. When Sony and IBM starting working together on the Cell/Playstation 3, they faced an enormous technical and cultural problem on how to develop software for the platform. The fundamental problem was that the console game model, in which the game gets total control of the hardware, was incompatible with the platform's expected usage.

In previous generation consoles, games took total control of the machine with no intervening operating system. This is important for two fundamental reasons. First, top game programmers were used to having absolute control of the machine to achieve the best possible performance. Second, without an underlying operating system, a game that worked could be guaranteed to work through a console's entire lifetime. In contrast, in the PC world, an OS upgrade would frequently break or require upgrades to user applications.

The game community felt that giving games full control over the console was a critical requirement needed on all future machines. On the other hand, the Playstation 3 was expected to have persistent storage and network connectivity; it's unacceptable that a single, poorly designed game could corrupt a system. Many felt that an operating system isolated from the game was necessary for the platform.

In the end, the surprising solution for a 2006 consumer product was to adopt the 1960s mainframe technology of virtualization. The developers created a hypervisor for the platform that let games be deployed directly to virtualized hardware with no underlying operating system. Meanwhile, the network and persistent storage were protected from a buggy application by an operating system running on top of the same hypervisor.

The Playstation 3 is one of the most successful game consoles on the market. Virtualization preserves all the best properties of previous generation consoles while still allowing for secure access to storage and the network.

In This Issue

These two examples illustrate how virtualization's level of indirection makes previously intractable problems solvable. In the first, virtualization helped a previously fragile technology, workload migration, become commercially viable. In the second, virtualization enabled game deployment in a way that maximized performance and isolated games from platform upgrades.

In this special issue, we consider three other uses of virtualization; specifically, we include articles on language-level virtualization exploited to automatically parallelize applications, network virtualization used to enable tenant-specific network customization, and a virtualization layer employed above existing clouds to enable a multicloud grid.

The primary motivation for language-level virtualization is to let programs be portable across platforms. However, just as with other forms of virtualization, once a well-defined interface is enforced, the developer of the virtualized service can innovate below that interface. Developers of virtualized services have provided security services, load balancing, and scheduling guarantees, among other examples.

One of the greatest challenges facing software today is how to exploit the parallelism of modern processors. Although such processors have an ever-increasing number of cores, most programmers have trouble writing code that can exploit these machines. In our first article, "Using Speculation to Enhance JavaScript Performance in Web Applications," Jan Kasper Martinsen, Håkan Grahn, and Anders Isberg use language-level virtualization to automatically exploit multiple cores for JavaScript applications. Even though JavaScript is a sequential programming language, a great deal of potential parallelism exists in many Web applications. The authors' approach generates new threads using method-level speculation; if the speculation is wrong, the thread's effect is automatically undone, resulting in the same functionality as the sequential program. This approach improved performance for all the Web applications studied and, in some cases, achieved dramatic speedups.

Although virtualization is often associated with processors, it's exploited at all layers of the computing stack. In the networking space, for example, virtual LANs (VLANs) have been used for decades to provide different isolated virtual networks on a single physical network. In the past few years, software-defined networking (SDN)[10] has introduced a major new use for virtualization. Just as with machine virtualization, SDN separates the control of virtual networks from the services performed within them. This enables various new services to be embedded directly into the network and opens up innovation at multiple levels.

Our second article, "Scalable Network Virtualization in Software-Defined Networks," by Dmitry Drutskoy, Eric Keller, and Jennifer Rexford, describes the FlowN architecture. FlowN gives each tenant the illusion of its own virtual network, including virtual network elements (switches/routers/servers), ports, links, and its own OpenFlow[11] network controller. For efficiency, the authors use a "container-based" approach to controller virtualization in which multiple distinct controller applications can share the same controller. The separation of control into software lets FlowN adopt a traditional relational database for storing its metadata: the graph relationships of the virtual networks and their mappings onto physical network links/switches. This approach both greatly simplifies implementation and lets the authors adopt existing relational database techniques to achieve good scalability.

One of the most significant transformations that virtualization has led to is infrastructure-as-a-service (IaaS) cloud computing. Users can access virtually unlimited computing power whenever they need it, paying for only what they use. Providers have the benefit of economies of scale and massively automated infrastructure. The economics are so compelling that many believe that all computing will eventually move to the cloud, much like electric power's transition in the 1800s from individual electricity generators to electric utilities to today's electric grid. Virtualization is critical to the cloud. It lets producers efficiently support many tenants while strongly isolating them from each other. Consumers are isolated from the specifics of providers' physical capacity, allowing, for example, VMs to move between different computers and even clouds.

In "Plug into the Supercloud," Dan Williams, Hani Jamjoom, and Hakim Weatherspoon focus on using virtualization in cloud computing and transitioning to a grid of clouds similar to the electrical grid. Unfortunately, today's clouds are in many ways incompatible. Rather than relying on standards, this article proposes recursively applying virtualization to solve the problems inherent to incompatible clouds. The authors deploy their own hypervisor running within VMs on each cloud, and then deploy (paravirtualized) operating systems to these VMs. They demonstrate the ability to move VMs between clouds and implement oversubscription on top of today's clouds.

These three articles demonstrate very different approaches to virtualization, from programming environments to networks to service providers, each solving very different problems. Looking forward, we expect virtualization to have an ever-increasing impact on computing, and we'll undoubtedly revisit the topic many times in the coming years.

References

1. E. Bugnion et al., "Disco: Running Commodity Operating Systems on Scalable Multiprocessors," *ACM Trans. Computing Systems*, vol. 15, no. 4, 1997, pp. 412–447.
2. M.M. Theimer, K.A. Lantz, and D.R. Cheriton, "Preemptable Remote Execution Facilities for the V-System," *Proc. 10th ACM Symp. Operating Systems Principles* (SOSP 85), ACM, 1985, pp. 2–12.
3. E. Zayas, "Attacking the Process Migration Bottleneck," *SIGOPS Operating Systems Rev.*, vol. 21, no. 5, 1987, pp. 13–24.
4. F. Douglis and J. Ousterhout, "Transparent Process Migration: Design Alternatives and the Sprite Implementation," *Software Practice and Experience*, vol. 21, no. 8, 1991, pp. 757–785.
5. D.S. Milojičić et al., "Task Migration on the Top of the Mach Microkernel," *Proc. Usenix Mach III Symp.*, Usenix Assoc., 1993, pp. 273–290.
6. A. Barak, S. Guday, and R.G. Wheeler, *The MOSIX Distributed Operating System: Load Balancing for UNIX*, Springer, 1993.
7. D.S. Milojičić et al., "Process Migration," *ACM Computing Surveys*, vol. 32, no. 3, 2000, pp. 241–299.
8. P.M. Chen and B.D. Noble, "When Virtual Is Better than Real," *Proc. 8th Workshop Hot Topics in Operating Systems* (HOTOS 01), IEEE CS, 2001, p. 133.
9. C.P. Sapuntzakis et al., "Optimizing the Migration of Virtual Computers," *Proc. 5th Symp. Operating Systems Design and Implementation* (OSDI 02), ACM, 2002, pp. 377–390.
10. M. Casado et al., "Ethane: Taking Control of the Enterprise," *Proc. ACM SIGCOMM*, ACM, 2007, pp. 1–12.
11. N. McKeown et al., "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Computer Communication Rev.*, vol. 38, no. 2, 2008, pp. 69–74.

Fred Douglis is a consultant software engineer at EMC Backup Recovery Systems. His research interests are in storage, distributed systems, and Internet tools and performance. Douglis has a PhD in computer science from the University of California, Berkeley. He's a former editor in chief of *IEEE Internet Computing* and is a senior member of IEEE and a member of ACM and Usenix. Contact him at f.douglis@computer.org.

Orran Krieger is a research professor and the director of the Center for Cloud Innovation at Boston University. His research interests include operating systems, parallel architecture and software, and cloud computing. Krieger has a PhD in electrical and computer engineering from the University of Toronto. He's a member of IEEE, ACM, and Usenix. Contact him at okrieg@cs.bu.edu.