JON CROWCROFT, KEIR FRASER, STEVEN HAND, IAN PRATT, AND ANDREW WARFIELD

# the inevitability of Xen

Jon Crowcroft is the Marconi Professor of Networked Systems in the Computer Laboratory of the University of Cambridge. Prior to that he was professor of networked systems at UCL in the Computer Science Department.

- *jon.crowcroft@cl.cam.ac.uk*

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

- *keir.fraser@cl.cam.ac.uk*

Steven Hand is a lecturer at the University of Cambridge. His interests span the areas of operating systems, networks, and security.

- *steven.hand@cl.cam.ac.uk*

Ian Pratt is a senior lecturer at the University of Cambridge. Irreverently dubbed the "XenMaster" by his students, he has been the lead researcher on the Xen project since its inception.

- *ian.pratt@cl.cam.ac.uk*

Andrew Warfield is a Ph.D. student at the University of Cambridge, working on the Xen project. He hopes to finish his degree later this year.

- *andrew.warfield@cl.cam.ac.uk*

**XEN IS A VIRTUAL MACHINE MONITOR** (VMM) that we have developed at the University of Cambridge over the past five years. As a VMM, Xen allows a single physical computer to be partitioned into a set of isolated virtual computers, each running its own operating system and applications. Xen has received a fair bit of attention recently, and we have even spun out a company to support the commercial use of the software.

This article isn't just about our VMM, though. Xen is the core part of a much larger vision for public computing that has been behind a lot of our research in the 21st century. In this article we articulate this vision, the motivation behind Xen, and cover the details of the current VMM, the context in which it was conceived, and the future uses that we anticipate.

## Xen: Master and Servant

Xen is the crucial component of the Xenoserver world of public computing. The Internet provides connectivity between all the networks in the world, and the Web provides glue between all of the information resources connected to these networks. In both of those contexts, there is mutual benefit to participants: Sometimes referred to as Metcalfe's Law, the value of N nodes joining in a network is $N^2$, which usually offsets the risks, added security costs, denial of service, and so forth, associated with being connected to a global and largely unregulated network.

There are already a great number of resources attached to these networks, and an interest in constructing services (online games, file sharing, Internet telephony, and even the search for extra-terrestrial life) built by combining distributed resources. However, if we want equipment owners to share their computational resources, the benefit is often less obvious and the risks are certainly greater—it's a big bad world out there, full of worms, viruses, and ill-willed teenagers. To offset the risks, we must provide one key feature on each host: isolation. If a user is to offer CPU resources, for instance, she must be assured there is no negative impact on her normal applications. In SETI@Home this is relatively easy—the service is included as a screen-saver; the managing organization is generally believed to be capable, trustworthy, and benign; and there is only one application. For a general service that permits arbitrary applications to run, this is a more difficult problem.

This is exactly the problem that the Xenoservers project is attempting to solve. Using the strong isolation provided by a virtual machine monitor as a base, we intend to build a service platform that allows computers, from common desktops to high-end servers in commercial hosting facilities, to safely host arbitrary applications managed completely by an external party.

The key word in that last sentence is "arbitrary." Users have applications that run on all types of hardware, are written in all types of languages, and use many different libraries and operating systems. For a service platform to be truly generic for public computing services, it must be agnostic on each and every one of these factors.

The idea of a global service platform is not new. The rich history of such efforts, both academic and commercial, is littered with solutions that address only a subset of the problems of isolation and generality.

### NEW PROGRAMMING LANGUAGE APPROACHES

Users are told that if they simply change programming language, then they can run their programs on any machine that has a VM for the byte code for that programming language, be it Java, C#, or some other flavor of the decade. This is fine, provided they rewrite all their applications, someone has ported the JVM to the OS, and the OS actually supports isolation. For small, new Web-service type applications, this may be the approach of choice, but there are many large software systems out there that cannot reasonably be expected to be rewritten. Such difficulties in rewriting applications and retraining developers to a new language, and the benefits of software diversity, all point to the limited applicability of this technique.

### NEW OPERATING SYSTEM APPROACHES

Users have been told that if they simply change which OS they use, then they will find their applications running in an environment that will be ported to all known hardware platforms some day; of course, they will have to change all the applications to call a new API to make known the resource needs so that the OS isolation mechanisms know what to do. History has shown that while isolation has worked quite well in some of the newer OSes, the time taken for them to reach the market is longer by far than the time for system performance to outstrip the multiple demands a user has.

### NEW HARDWARE APPROACHES

Users have been offered the possibility that if they simply change all their hardware to use a new processor such as the Transmeta Crusoe, then it can emulate all known processors (up to a couple of years ago) and may one day have the resources to provide isolation between the virtual CPUs it implements. This last approach is really very attractive, but unless one has the resources of a major semi-conductor outfit, and a decade to wait, the functionality in the VLIW CPU microcode falls short of the generality needed.

### VMMS

Users have been offered virtual machines at the level of the CPU. Virtual machine monitors have been around for decades, offering multiplexing of the processor and other system resources among multiple copies of the same OS, and between different concurrent operating systems on the same host. Most VMs, however, only provide the functional isolation necessary to multiplex resources safely; they do not typically consider the performance isolation required to manage access to CPU and devices.

Xen is a VMM that offers paravirtualization: The operating system must be modified slightly to run on top of Xen, which does not present an exact replica of the underlying hardware as so-called "pure virtualization" packages (such as VMware) do. By changing the OS-to-hardware interface, Xen is able to make considerable performance improvements, accounting for the fact that the x86 architecture was not built with virtualization in mind. Xen currently allows Linux, FreeBSD, and NetBSD OS instances to share a common physical host in isolation from one another. We'll look at this in more detail in the next section.

Meanwhile, if we really want owners to share their computational resources, we need to offer more than isolation. We need to provide incentives. To this end, the Xenoserver system was conceived.

The Xenoserver model consists of a number of control-plane components that together provide resource trading, resource registration and discovery, deployment of guest OS and applications, OS migration, and virtualization-supporting storage. These components rely on the local mechanisms on each node running Xen. Mediated through local policies, they allow the owner of resources to manage what is visible and usable in public and what is isolated and private.

## Xen: The Master Platform

Xen is a VMM that paravirtualizes the x86 architecture. Figure 1 shows the structure of a machine run-

ning Xen, hosting a number of different guest operating systems, including *Domain0* running control software in a XenoLinux environment.

As a hardware architecture to virtualize, the x86 is probably best described as uncooperative. Virtualizing the platform efficiently has presented interesting technical challenges with almost every aspect of the hardware: Instruction execution, memory management, and device access have all required careful consideration and design to virtualize effectively—detailed war stories are available in our research papers. The end result of Xen, though, is a system that provides efficient virtualization using slightly modified OSes. Xen currently supports Linux, NetBSD, FreeBSD, and Plan9. The application binary interface ABI remains unchanged, and so applications may be run unmodified. In fact, many of the leading Linux vendors are including Xen in their distributions.

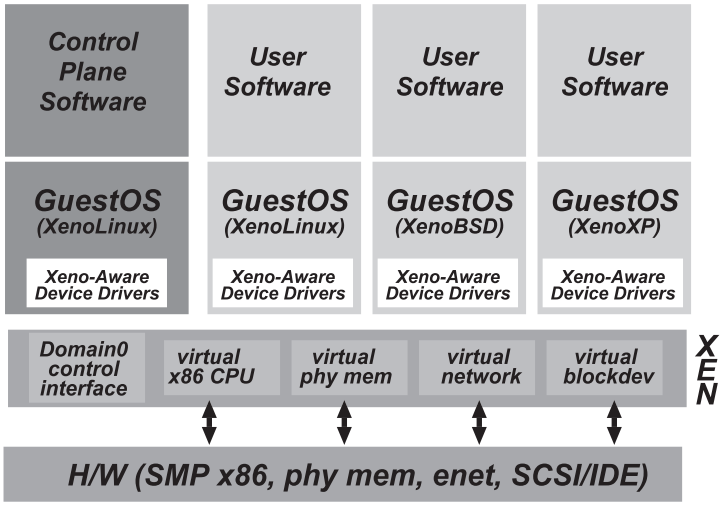


FIGURE 1. THE XEN HYPERVISOR

Other virtualization projects such as VMware and Denali make different cuts in the software stack. VMware chooses to avoid the requirement of rebuilding the OS by presenting an exact virtualization of the underlying hardware. The benefit of this technique is the ability to support unmodified, closed-source OSes such as Windows. The cost, as mentioned above, is the inability to make many performance-enhancing improvements at the virtualization layer. Denali's approach is in the opposite direction: the ABI is not maintained, and so applications must also be recompiled to run on the virtual architecture.

As mentioned above, the key property that Xen provides to guest OSes is isolation. Xen rigidly divides CPU resources between VMs to ensure that they each receive an allotted amount of processing time. Moreover, as each guest is running on its own set of virtual hardware, applications in separate OSes are protected from one another to almost the same degree that they would be were they installed on separate physical hosts. This property has attracted considerable attention in light of the inability of current OSes to protect applications against spyware, worms, and viruses: Untrusted applications such as Web browsers may be seconded to their own virtual machines and completely separated from other, more trusted applications.

This strong isolation has also proved very useful in solving two major problems with device drivers: driver availability and reliability. Xen is capable of allowing individual virtual machines to have direct access to specific pieces of hardware. We have taken the approach of using a single virtual machine to run the physical driver for a device (such as a disk or network interface) and then exporting a virtualized version of the device to all of the other guest OSes that are running on the host. This approach means that a device need only be supported on a single platform (Linux, for instance) and may be available to all the OSes that Xen runs. Each guest implements an idealized disk and network device, which are capable of connecting to the hardware-specific driver in an isolated device domain. This approach also has the benefit of making drivers, a major source of bugs in operating systems, more reliable. By running a driver in its own VM, driver crashes are limited to the driver itself—other applications may continue to run. Device domains can even be rebooted to recover failed drivers, resulting in downtimes on the order of hundreds of milliseconds in cases where the entire machine would previously have crashed.

This approach will no doubt sound familiar to anyone who has worked with micro-kernels in the past; Xen's isolation achieves a similar fragmentation of OS subsystems. One major difference between Xen and historical work on micro-kernels is that we have foregone the architecturally pure fixation on IPC mechanisms in favor of a generalized, shared-memory ring-based communication primitive that is able to achieve very high throughputs by batching requests.

In addition to the benefits of virtualization as a base for service platforms, it is worth noting that virtualization has attracted considerable attention as a development debug and management environment. The pervasive debugger project (PDB) in our lab is building debug support for entire distributed systems. PDB allows both *vertical debugging*, tracing execution through the entire software stack, including OS and application code, and *horizontal debugging*, allowing execution across a complete set of virtual hosts to be examined concurrently. The decoupling of virtual

machines from physical hardware has the additional benefit of allowing the entire state of a system to be saved at arbitrary points in time. This allows debuggers to be built that examine old versions of an executing VM to identify the point at which a bug was introduced, and even to step execution backwards after a crash to quickly establish the root cause.

## Xenoservers: The Service Platform

The larger view of the Xenoservers project is to use Xen-based hosts to manage and deploy distributed applications across large numbers of physical hosts. The two key targets for such deployments are large clusters and the Internet at large. Perhaps surprisingly, these two environments are very similar in that they share the property of desiring an accountable decoupling of application management from the maintenance of physical hardware. Several of the organizations that we have interacted with maintain clusters containing tens of thousands of nodes used by a wide variety of users. The aim of Xenoservers is to provide the necessary higher-level functionality to locate and account for resources and to otherwise facilitate the management of such large distributed environments.

Whether it is a federation of IT data centers within a corporation or a disjoint set of Internet-connected hosts, the integration of a large set of Xen-based hosts into a viable service platform needs to allow diverse sets of hardware facilities (the providers) and application managers (the customers) to work together. Xenoservers take a market-based approach to managing a large distributed system with virtually no central management, and very limited trust between parties.

The remainder of this section briefly discusses the key components of the project.

### RESOURCE REGISTRATION AND DISCOVERY

The first key problem in managing such a potentially fragmented service platform is in keeping track of the resources on offer, and in finding the required resources for a particular application. Xenosearch provides service location, and permits complex queries to find a number of servers meeting some desired constraint, including how far apart they are (e.g., for disaster recovery), as well as the more normal requirement for how near they are to a set of users (e.g., for game serving).

### MIGRATION

While many server applications may be very long-lived, the hardware that they run on will invariably need service from time to time. A major benefit of virtualization is the ability to migrate a running operating system instance from one physical host to another. Migration allows a physical host to be unloaded so that hardware may be serviced, it allows coarse-grained load balancing in a cluster environment, and it allows servers to move closer to the users that they serve. We have demonstrated that migration may be made very fast—experiments migrating a running Quake server have achieved repeatable migration times with outages of less than 100ms.

### VIRTUALIZATION-SUPPORTING STORAGE

Large virtual machine–based systems present many interesting new challenges for the management of storage. Storage must potentially scale to support an order-of-magnitude more hosts from the same number of physical machines. In addition it must provide location-transparent access to allow migration, and in many cases must maintain historical versions of disk images to allow old versions of VMs to be resumed. The Parallax storage system aims to address these problems by unifying storage resources across a set of hosts, and allowing virtual disks to be provided for individual VMs.

## Conclusion

In this article we have attempted to describe our work to date on the Xen virtual machine monitor, and our plans for using Xen as a service platform for large distributed systems in the future. Xen has been publicly available as an open source VMM for over two years, and is now very stable and used in production environments. We enjoy a very active developer community and are always eager to hear about new applications and deployments of Xen in the real world.

### REFERENCES

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

The Xenoservers project page, http://www.cl.cam.ac.uk/xeno/, contains links to publications, by the group at the University of Cambridge, especially: Paul Barham et al., "Xen and the Art of Visualization," *Symposium on Operating System Principles (SOSP) 2003*, http://www.cl.cam.ac.uk/netos/papers/2003-xensosp.pdf.

The team at Clarkson, who patiently reconstructed the results from the SOSP paper and USENIX had the good judgment to publish it: Brian Clark et al., "Xen and the Art of Repeated Research," http://www.clarkson.edu/class/cs644/xen/files/repeatedxen-usenix04.pdf.

Xen on Sourceforge: http://sourceforge.net/projects/xen/.

Xensource, the company: http://xensource.com.