CSE 231—Advanced Operating Systems
"Xen"

Andrew Quinn

Background. Type-1 hypervisors need to support a design like that of Figure 1, in which a hypervisor is deployed above the hardware and supports potentially multiple guests operating systems. Unfortunately, these designs introduce a fundamental dilemma. Operating systems are designed with the assumption that they are the most privileged system software and have full machine capabilities. However, hypervisors inherently limit the privileges of an operating system; resolving this conflict is non-trivial. To make matters worse, the Hypervisors of the late 90s and early 2000s aimed for full virtualization. I.e., these systems exported an virtual machine interface that was identical to the underlying hardware interface. Since hardware at the time had very little support for virtualization, so hypervisors required software solution to support "the illusion of privilege" that operating systems require for correct execution.

Prior systems, namely VMWare's ESX server, dynamically rewrites portions of the guest operating system during execution to insert the required traps for correct virtualization. Dynamic rewriting is an expensive solution that is often used for debugging, intrusion analysis, or software simulations. Performance numbers are not provided, but the Xen paper alludes to the high overhead of ESX.

Xen—Key Idea. The key idea of Xen is to "paravirtualize" operating systems to run above a virtual machine layer. Essentially, paravirtualization means manually porting the operating system to run on a hypervisor (e.g., modifying the way that OSes use privileged instructions) as opposed to the dynamic instrumentation of ESX which automatically ports an OS.

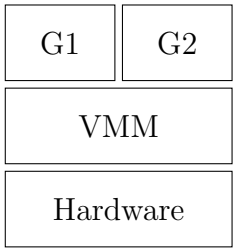

Figure 1: **Type-1 Hypervisor.** A Type-1 hypervisor (VMM) executing two guest operating systems (G1 and G2).

Questions and Comments One key goal of Xen is to leave the ABI exposed by each paravirtualized OS unchanged, so that applications do not need to be modified to run on a virtual machine. However, there are application-level instructions that map poorly to a virtualized world (e.g., `int 0x80/syscall`). Why did Xen look into modifying applications as well to run more efficiently?

Perhaps *the* primary motivation for Xen is the XenoServer project, which a pay-as-you-go cloud-computing interface eerily similar to the offerings of AWS, Google Compute Cloud, and Microsoft Azure. Xen's position is that virtual machines are well suited for enabling diverse customer deployments. In class, we identified the following as properties of virtual machines that enable better deployability:

1. **Scalability.** Virtual machines can be collocated on a large host and thus better take advantage of their resources.

2. **Customizability.** Virtual machines are very customizable for applications (e.g., you can specify a specific library!)

3. **Isolation.** Virtual machines provide isolation across guests.

But, these properties are all goals of the original time-sharing systems! Are virtual machines just re-implementing the same features as an operating systems? The answer is pretty much, yes, virtual machines are realizing the same goals as the original time-sharing operating systems. What's more, they're much higher overhead that an operating system, because virtual machine deployments will duplication of code, data, etc.

And yet, there don't appear to be any cloud computing offerings that allow users to deploy an application with processes as the primary unit

of isolation. Container (e.g., Docker and LXC) are close. They provide operating-system level isolation but allow administrator-like configuration for each user[1]. So, maybe containers are the answer, and virtual machines are dead?

Of course not! Virtual machines are alive and well. Manco et al. provide a compelling reason in their SOSP 2017 paper: safety [1]. The linux application interface is massive, ever growing, and thus it is difficult to ensure that it is safe. In comparison, the machine interfaces provided by virtual machines move much slower and are much easier to secure. Manco and his coauthors argue that the key challenge is performance; containers are smaller and lighter than traditional virtual machines. Their solution? Make virtual machines smaller by turning to UniKernels. Read the paper if you want more details!

References

[1] Filipe Manco, Costin Lupu, Florian Schmidt, Jose Mendes, Simon Kuenzer, Sumit Sati, Kenichi Yasukata, Costin Raiciu, and Felipe Huici. My vm is lighter (and safer) than your container. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, page 218–233, New York, NY, USA, 2017. Association for Computing Machinery.

[1] there's also some resource usage tracking and other features