CSE 231—Advanced Operating Systems
"Disco"

Andrew Quinn

Background. Virtual machines were introduced in the 60s. In the 70s, Popek and Goldberg introduced a distinction between so called type-1 and type-2 hypervisors [5], which remains widely used today. Type-1 hypervisors, sometimes called 'bare-metal', execute directly on top of the hardware and guests are deployed above (see Figure 1a). Prevalent Type-1 hypervisors include VMWare ESXi, and Xen. In contrast, Type-2 hypervisors execute on top of an existing operating system and are thus essentially equivalent to a process in an OS (see Figure 1b). Example Type-2 hypervisors include VirtualBox, VMWare workstation, and QEMU. Finally, some systems do not neatly fit into either category. For example, KVM [] has both Type-1 and Type-2 characteristics: it is a kernel module that essentially makes Linux a Type-1 hypervisor, but deploys guest OSes on top of Linux. The community continues to rigorously research, Type-1, Type-2, and other uncategorized hypervisors,

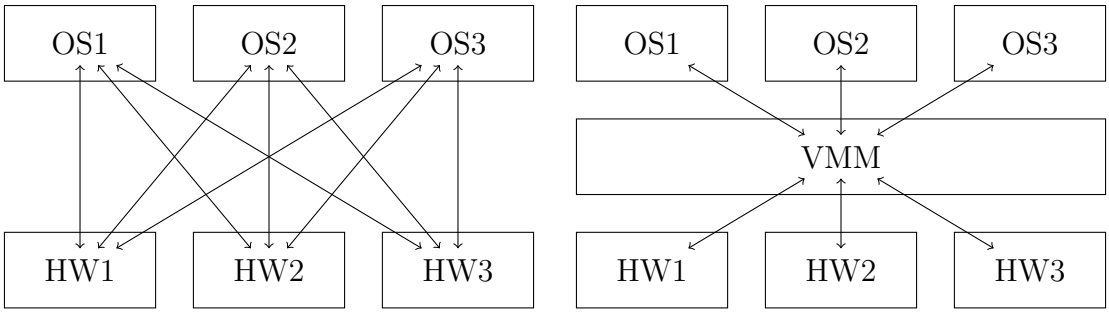
(a) **Type-1 Hypervisor.** A Type-1 hypervisor (VMM) executing two guest operating systems (G1 and G2).

(b) **Type-2 Hypervisor.** A Type-2 hypervisor (VMM) executing alongside a single application (App1) and running a single guest Operating system (G1).

Summary. Disco investigates the use of Virtual Machine Managers (hypervisors) enabling non-uniform memory access (NUMA) multiprocessors. Their high-level vision is that hypervisors present a solution to "computer vendors attempting to provide system software for innovative hardware", with shared-memory NUMA multiprocessors as the particular innovative hardware that they target. The authors propose a number of key solutions to support a fully virtualized platform, in which a guest experiences no difference between running on raw hardware compared to running on the hypervisor. Disco creates virtualization layers for most devices and computer components, including memory, IO devices, and networking

devices. Disco achieves scalability by deploying existing OSes side-by-side and using existing distributed systems protocols for communication (e.g., NFS provides a shared file system across guests). The results are strong, with virtualization coming with relatively meager costs (3%–16% depending upon workload)

Questions/comments. At first glance, the paper's main motivation seems stellar. A layer of indirection that would allow existing systems to seamlessly adopt innovative hardware is clearly a huge win, because it turns an $O(m * n)$ problem into an $O(m + n)$ problem [1](see below).



(a) **No Abstraction.** Each Operating System would need to support each architecture in order adopt innovation.

(b) **VMM Abstraction.** Each OS and architecture only needs to use or support the virtual machine interface.

The problem is that unlocking the performance potential of innovative hardware will almost invariably require operating system and application customization. For example, developers have updated the design of nearly all major operating systems in light of NUMA and multicore systems. Moreover, the magnitude of hardware innovations that can be effectively masked is necessarily minimal. For example, performance dictates that VMs execute directly on the hardware (i.e., there is no emulation layer), but that requires that all hardware innovation be ISA agnostic.

Nevertheless, virtual machines remain an incredibly important primitive and construct in operating systems today. In class, we identified the following reasons why developers and operators love using virtual machines:

1. **Fault Domains.** Virtual machines separate a fault in one operating system from another. This is especially useful when hacking a custom system.

2. **Standardization.** Virtual machines provide standardization in software deployment for both developer and operations teams.

3. **Security.** Virtual Machines, provide separate security domains, at least in principle [].

4. **Migration and Checkpointing.** Virtual Machines are a strong abstraction upon which to implement checkpointing and migration across deployed nodes.

5. **Fault Tolerance.** Virtual Machines are a strong abstraction upon which to build system-level fault tolerance[1].

[1]This terminology is from Sky Computing [6]

6. **Record & Replay** Virtual Machines are a strong abstraction upon which to implement deterministic record and replay, which further enables security forensics [2], debugging [4], intrusion detection [3], etc.

References

[1] T. C. Bressoud and F. B. Schneider. Hypervisor-based fault tolerance. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, SOSP '95, page 1–11, New York, NY, USA, 1995. Association for Computing Machinery.

[2] George W. Dunlap, Samuel T. King, Sukru Cinar, Murtaza A. Basrai, and Peter M. Chen. Revirt: Enabling intrusion analysis through virtual-machine logging and replay. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (Copyright Restrictions Prevent ACM from Being Able to Make the PDFs for This Conference Available for Downloading)*, OSDI '02, page 211–224, USA, 2002. USENIX Association.

[3] Ashlesha Joshi, Samuel T. King, George W. Dunlap, and Peter M. Chen. Detecting past and present intrusions through vulnerability-specific predicates. In *Proceedings of the Twentieth ACM Symposium on Operating Systems Principles*, SOSP '05, page 91–104, New York, NY, USA, 2005. Association for Computing Machinery.

[4] Samuel T. King, George W. Dunlap, and Peter M. Chen. Debugging operating systems with time-traveling virtual machines. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '05, page 1, USA, 2005. USENIX Association.

[5] Gerald J. Popek and Robert P. Goldberg. Formal requirements for virtualizable third generation architectures. *Commun. ACM*, 17(7):412–421, July 1974.

[6] Ion Stoica and Scott Shenker. From cloud computing to sky computing. In *Proceedings of the Workshop on Hot Topics in Operating Systems*, HotOS '21, page 26–32, New York, NY, USA, 2021. Association for Computing Machinery.