Virtualization Basics: Understanding Techniques and Fundamentals

Hyungro Lee
School of Informatics and Computing, Indiana University
815 E 10th St.
Bloomington, IN 47408
lee212@indiana.edu

ABSTRACT
Virtualization is a fundamental part of cloud computing, especially in delivering Infrastructure as a Service (IaaS). Exploring different techniques and architectures of the virtualization helps us understand the basic knowledge of virtualization and the server consolidation in the cloud with x86 architecture. This paper describes virtualization technologies, architectures and optimizations regarding the sharing CPU, memory and I/O devices on x86 virtual machine monitor.

Categories and Subject Descriptors
C.0 [**General**]: Hardware/software interface; C.4 [**Performance of systems**]: Performance attributes; D.4.7 [**Operating Systems**]: Organization and design

General Terms
Performance, Design

Keywords
Virtualization, Virtual Machine Monitor, x86, Cloud Computing

1. INTRODUCTION
Back in 1974, Popek and Goldberg virtualization requirements were introduced in the article "Formal Requirements for Virtualizable Third Generation Architectures" [7] and it still provides guidelines for virtualizing hardware resources and terms. Virtualization is now the foundation of cloud computing [1] to provide highly scalable and virtualized resources. Numerous systems and projects in industry, academia and communities have adopted a virtualization technology with cloud computing services to reduce under-utilized hardware resources and achieve efficient use of systems. The FutureGrid project funded a US National Science Foundation (NSF), a Nebula platform launched by NASA, or Kasumigaseki Cloud supported by Japanese government is a good example of growing involvement in virtualization technologies with the cloud. Early technologies and developments in the virtualization have been accomplished by some companies such as IBM from 1967 and VMware from 1998. In open source communities, Xen, KVM, Linux-vServer, LXC and others have supported virtualization in different platforms with different approaches. In this paper, x86 architecture virtualization will be discussed with these historical changes.

In cloud computing, Infrastructure-as-a-Service (IaaS) provides on-demand virtual machine instances with virtualization technologies. IaaS has been broadly used to provide required compute resources in shared resource environments. Amazon Web Services (AWS), Google Compute Engine, Microsoft Windows Azure, and HP Cloud offer commercial cloud services. OpenStack, Eucalyptus, SaltStack, Nimbus, and many others provide private open source cloud platforms with community support in development. Since the virtualization is sharing resources, many concerns have been raised regarding security, isolation and performance compared to native systems.

2. VIRTUALIZATION
Virtualization typically refers to the creation of virtual machine that can virtualize all of the hardware resources, including processors, memory, storage, and network connectivity [7]. With the virtualization, physical hardware resources can be shared by one or more virtual machines. According to the requirements from Popek and Goldberg, there are three aspects to satisfy the virtualization. First, the virtualization should provide an equivalent environment to run a program compared to a native system. If the program shows a different behavior under the virtualization, it may not be eligible as a virtualized environment. The virtualization also needs to provide a secured control of virtualized resources. Having a full control of resources is important to protect data and resources on each virtual environment from any threats or performance interference in sharing physical resources. Virtualization often expects performance degradation due to the additional tasks for virtualization, but good performance should be achieved with a software or hardware support in handling privileged instructions. With these requirements, efficient virtualization is guaranteed. In the following section, different types of hypervisors are explained with the implementation level of virtualization. Virtualized resource is also presented in cpu, memory and I/O transactions.

2.1 Hypervisor

To understand virtualization, hypervisor should be addressed first. Hypervisor enables communication between hardware and a virtual machine so that the virtualization accomplishes with this abstraction layer (hypervisor). Hypervisor is originally called virtual machine monitor (VMM) from [7]. These two terms (Hypervisor and VMM) are typically treated as synonyms, but according to the distinction from Agesen et al [1], a virtual machine monitor (VMM) is a software that manages CPU, memory, I/O data transfer, interrupt, and the instruction set on a given virtualized environment. A hypervisor may refer to an operating system (OS) with the VMM. There is a slight distinction between hypervisor and VMM but in this paper, we consider these terms to have identical meanings to represent a software for virtual machine. Typically, a hypervisor can be divided into Type 1 and Type 2 hypervisor based on the different level of implementation. Type 1 is sitting on hardware and the communication between hardware and virtual machine is direct. The host operating system is not required in Type 1 hypervisor since it runs directly on a physical machine. Due to this reason, it is sometimes called a 'bare metal hypervisor'. VMware vSphere/ESXi, Microsoft Windows Server 2012 Hyper-V, Citrix XenServer, Red Hat Enterprise Virtualization (RHEV) and open-source Kernel-based Virtual Machine (KVM) are identified in this category. Type 2 hypervisor is on the operating system to manage virtual machine easily with the support of hardware configuration from operating system. The extra layer between hardware and virtual machine in the type 2 hypervisor causes inefficiency compared to the type 1 hypervisor. VirtualBox and VMware Workstation are in this category. The terms of Host or Guest machine (or domain) are used in the hypervisor to describe different roles. Host machine (domain) contains a hypervisor to manage virtual machines, and Guest machine (domain) means each virtual machine sitting on a hosted machine in a secure and isolated environment with its own logical domain. With these separated roles, the hypervisor is able to offer resource boundaries to multiple virtual machines on the same physical machine. In other words, the hypervisor is a software layer that creates a virtual environment with virtualized CPU, memory and I/O (storage and network) devices by abstracting away the underlying physical hardware. Virtual machine (VM) typically refers to an encapsulated entity including the operating system and the applications running in it as well.

2.2 x86 privilege levels

x86 architecture has certain restrictions related to resource access such as kernel execution for machine instructions, memory and I/O functions. x86 operating systems can perform these tasks by running in a most privileged level, although user applications do not have access to the level. These different levels of access to hardware resources provide certain protection from software failure. These privilege levels are also called protection rings since it protects access to hardware resources with four different layers of privileges. The privilege levels in x86 architecture with two-level modes i.e. ring 0 and ring 3 can protect memory, I/O ports and certain machine instructions against accidental damage of the system by user programs [9]. Ring zero is for kernel execution and device drivers which is the most privileged and Ring three is for user mode applications in the least privileged layer. Any interrupt from ring 0 cannot transfer control to ring 1, 2 or 3. With these protection rings, memory access or I/O data transfer is only doable via the kernel execution. For example, User application can transfer control to the kernel by making a system call when it opens a file or allocates memory.

Hypervisor has to manage these privilege levels to offer virtualized resources. Virtualization offers different techniques for handling the privilege levels, and the resource sharing can be achieved with a software, a hardware support, or both. Binary translation, shadow page tables, and I/O emulation are used as a software-assisted virtualization. Binary translation runs VMM on ring 0 mode to have the most privileged level but guest operating system runs on Ring 1 to trap OS calls in VMM. This was introduced by VMware to offer better performance in virtualization. Shadow page tables are used for mapping guest physical memory to the actual machine memory. The guest OS is not allowed to access to the hardware page tables so that the hypervisor keeps the mappings between the guest memory and the host physical memory. Using Shadow page tables consumes system memory but it accelerates the mappings with a one-step lookup with translation lookaside buffer (TLB) hardware. Emulated device is used to deliver requests from guest OS to a real hardware across different platforms even if the device is not supported. Hardware-assisted virtualization uses a higher privilege level than ring 0 to run VMM at the level, i.e. ring -1. With hardware-assisted virtualization the x86 operating system has direct access to resources without binary translation or emulation.

On the x86 architecture, there are several techniques based on different approaches for handling privilege rings and emulating devices. Full virtualization, paravirtualization, hardware-assisted virtualization and operating system-level virtualization are introduced.

3. CPU VIRTUALIZATION

Virtualization in x86 architecture needs to manage virtual machines (VMs) by the additional layer (hypervisor) between the vms and physical hardware. Virtual machines have in-direct access to the cpu, memory, and other hardware resources through the hypervisor so the privileged instructions in the guest operating system can be executed with or without translating and trapping. Full virtualization, para-virtualization and hardware-assisted virtualization are one of the techniques handling these privileged instructions in hypervisor. Some other techniques are also introduced in this section.

3.1 Full virtualization

Full virtualization provides virtualization without modifying guest operating system. In x86 architecture, dealing with privileges instructions is a key factor for virtualizing hardware. VMware offers binary translation of operating system requests so that virtualizing privileged instructions can be completed without supports from either hardware or operating system. There are Microsoft Virtual Server and VMware ESXi using this technique.

3.2 Para-virtualization

Xen group initially developed paravirt-ops (later called by paravirtualization) to support high performance and powerful resource isolation with slight modifications to the guest operating system. Xen noticed that full virtualization supported guest domains without a change in the operating system, but there were negative influences on performance due to the use of shadow page tables. [2] Paravirtualization (PV) requires the modified OS kernel with system calls to deal with privileged instructions. Xen registers guest OS page tables directly with the MMU with a read-only access to avoid the overhead and complexity regarding the updating shadow page tables in full virtualization. With the interface between a virtual machine and a hypervisor, paravirtualization achieves high performance without the assist from hardware extensions on x86. VMware introduced Virtual Machine Interface (VMI) which is a communication protocol between a hypervisor and a virtual machine. Xen used separated device drivers and paravirt operations extensions (PVOPS) in the Linux kernel to avoid emulating system's devices such as network cards or disk drives. Paravirtualization, in cooperation with Kernel modifications on the guest OS, provides improved performance for CPU, memory and I/O virtualization compared to full virtualization, but this requires special kernel support. Paravirtualization supports unmodified application binary interface (ABI) so that user applications do not require any changes. Paravirtualization is also called operating system assisted virtualization because of the awareness of a hypervisor on guest OS. Xen, UML and VMware support paravirtualization.

3.3 Hardware Assisted virtualization

To improve performance of virtualization, Intel and AMD provides virtualization extensions to x86 processors. Intel Virtualization Technology (VT) and AMD Virtualization (AMD-v) are accelerations for privileged instructions including memory management unit (mmu), directed I/O devices (iommu). With these hardware-assisted virtualization technology, modified guest OS is unnecessary to enable virtualization because VMM manages privilege instruction at a root mode which is a ring -1 without affecting the guest OS. Using Second Level Address Translation (SLAT), nested paging in Intel EPT (Extended Page Table) or AMD RVI (Rapid Virtualization Indexing), memory management has been enhanced and the overhead of translating guest physical addresses to real physical addresses has been reduced. Early CPUs for x86 do not have virtualization extensions which are not included in hardware assisted virtualization.

3.4 Operating System-level virtualization

(Shared Kernel Virtualization) Operating system provides isolated partitions to run virtual machines in the same kernel. With a chroot operation, which is a transition of a root directory for a certain process with an isolation to outside directories, OS-level virtualization enables isolation between multiple virtual machines on a shared OS. Overhead is very limited in this model due to the benefits of running under operating systems with a shared kernel. Emulating devices or communicating with VMM is not necessary. The guest os and the host os should have the same OS or kernel. Running Windows on Linux host is incompatible. There are Solaris Containers, BSD Jails, LXC, Linux vServer, and Docker.

3.5 Guest operating system

The virtualization application also engages in a process known as binary rewriting which involves scanning the instruction stream of the executing guest system and replacing any privileged instructions with safe emulations. There are VirtualBox and VMware Server. Clearly, the multiple layers of abstraction between the guest operating systems and the underlying host hardware are not conducive to high levels of virtual machine performance. This technique does, however, have the advantage that no changes are necessary to either host or guest operating systems and no special CPU hardware virtualization support is required.

4. MEMORY VIRTUALIZATION

x86 CPUs include a Translation Lookaside Buffer (TLB) and Memory Management Unit (MMU) to deal with translation of virtual memory addresses to physical memory addresses with page tables. TLB is a cache of recently used mappings from the page tables, and MMU is a hardware unit that contains page table entries (PTEs) to map virtual address to physical address. In virtualized environments, guest OSs do not have access to the native direct access to physical machine memory, and the hypervisor offers another mapping between physical location of machine memory and physical location of guest memory. This shadow page tables in the hypervisor create overhead in memory virtualization because the shadow page tables need to be synchronized with the guest page tables when the guest OS updates. This overhead of two level memory mapping can be reduced with the hardware assist, for example Intel's Extended Page Tables (EPT), or AMD's RVI (Rapid Virtualization Indexing). EPT or RVI provides nested page tables which contain mapping between virtual memory of guest OS and physical memory of machine in the TLB with a special tag for VM.

5. NETWORK VIRTUALIZATION

Network virtualization provides logical devices in software on top of a physical network. The virtual devices support I/O sharing across multiple VMs with its serialized access path. As a part of I/O virtualization, networking in virtualized systems requires isolation between virtual machines and physical machines. Unlike a virtualizing storage device which is block based, a virtualizing network is packet based which expects intermixed packet streams. Virtual Machine Monitor (VMM) offers Network Interface Card (NIC) with a emulation or a para-virtualization to deliver logical network devices. This additional step compared to a native single o/s over a single physical machine generates performance degradation and increases network access latency.

In virtualization technologies such as Xen and VMware, I/O instructions are virtualized and managed by VMM. Software-based techniques like para-virtualized NIC (address translation) or emulated disk (instruction translation) enable data protection integrity to the VMS sharing I/O devices but performance degradation and bandwidth loss are expected. [8] Massimo Cafaro performed the experiment of the effect of the device virtualization architecture with httperf and netperf. 70% of drop on the throughput is identified in para-virtualized systems, and 50% of available bandwidth loss is identified in the consolidated VMs. [3]

5.1 Network Virtualization in Xen

In Xen implementation, Driver Domain handles I/O data transfer between a virtual machine (VM) and a physical device. It requires multiplexing and demultiplexing packets between multiple VMs and multiple physical devices. Due to this additional layer, there is a significant overhead on network communication in the network virtualization architecture. Menon et al [6] identifies the performance degradation, and it shows about 65% of receive overhead in Xen VM (guest domain) and 81% of transmit overhead compared to the native devices.

5.2 Xen Architecture for Networking

Figure 1 describes the paravirtualized network interfaces in Xen. Each virtual machine has a virtual interface provided by Xen drivers instead of having a physical network interface in a native operating system. With the virtual communication channel between the frontend device and the backend device, the traffic from guest domain to physical network interfaces occurs through bridging, routing or Network Address Translation (NAT). The unprivileged driver domain has security and reliability advantages due to its separation.

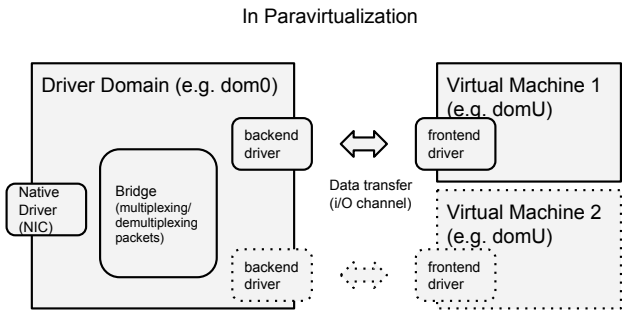

Figure 1: Xen Networking with Paravirtualized Network Devices

5.3 Performance Overhead in Xen

Menon et al [5] provides performance benchmark by measuring the transmit (outbound) bandwidth and the receive (inbound) bandwidth. Table 1 indicates a significant overhead for processing packets in Xen's virtualized network interfaces. To transfer packets from the physical interface to the virtual interfaces in a guest domain, 70% of the execution time is required in the driver domain and the Xen hypervisor. For the outbound packets, 60% of the network processing time is wasted. These additional time are mainly caused by multiplexing/demultiplexing overheads and bridging network packets in the driver domain. High memory miss rates are also reported such as TLB miss or L2 cache misses.

Table 1: Network performance under Xen (Menon et al [5])

Configuration	Inbound*	Outbound*
Native Linux	2508	3760
Driver Domain	1738 (30%)	3760 (0%)
Guest Domain(VM)	820 (67%)	750 (80%)

* Overheads shown in parentheses

5.4 Improvements

Menon et al [5] suggests three optimizations towards the Xen network virtualization architecture based on the problems identified earlier. The optimizations achieved 300% of improvements on the transmit throughput and 35% of improvements on the receive throughput. The techniques used in the optimizations include TCP Segmentation Offload (TSO) and data copy (packet transfer) instead of the address remap operation per packet and a new memory allocator in a guest domain, which tries to group together all memory pages into contiguous memory so that superpages can be enabled. Concurrent direct network access (CDNA) proposed direct connection between a guest domain and its own network interface by CDNA NIC. With the multiplexing network traffic on the NIC, the software multiplexing overheads in the driver domain disappears. [10] Efficient and scalable paravirtual I/O system (ELVIS) identifies two issues. First, context switch costs are high between the hypervisor and the guest for exits and entries in interrupt handling. The lack of scalability in the multi-core systems is the other issue. Exit-Less Interrupts (ELI) technique is introduced [4].

6. CONCLUSIONS

Over the past few decades, virtualization on x86 architecture has been grown with software and hardware support to ensure server consolidation, resource management and provisioning over the multiple operating systems. CPU manufacturers introduced hardware virtualization support such as Intel Virtualization Technology (VT-x) and AMD Virtualization (AMD-v) with new instruction sets so that the virtual machine execution has been improved with software support. In addition, many optimizations and techniques such as Binary Translation, Segmentation, and Shadow Page Tables have been added to enhance the performance of the virtualization. We expect this paper to provide some lessons to understand historical changes and recent challenges of the virtualization.

7. REFERENCES

[1] O. Agesen, A. Garthwaite, J. Sheldon, and P. Subrahmanyam. The evolution of an x86 virtual machine monitor. *ACM SIGOPS Operating Systems Review*, 44(4):3–18, 2010.

[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. *ACM SIGOPS Operating Systems Review*, 37(5):164–177, 2003.

[3] M. Cafaro and G. Aloisio. *Grids, Clouds, and Virtualization*. Springer, 2011.

[4] A. Gordon, N. Amit, N. Har'El, M. Ben-Yehuda, A. Landau, A. Schuster, and D. Tsafrir. Eli: bare-metal performance for i/o virtualization. *ACM SIGARCH Computer Architecture News*, 40(1):411–422, 2012.

[5] A. Menon, A. L. Cox, and W. Zwaenepoel. Optimizing network virtualization in xen. In *USENIX Annual Technical Conference*, number LABOS-CONF-2006-003, 2006.

[6] A. Menon, J. R. Santos, Y. Turner, G. J. Janakiraman, and W. Zwaenepoel. Diagnosing performance overheads in the xen virtual machine environment. In *Proceedings of the 1st ACM/USENIX*

international conference on Virtual execution environments, pages 13–23. ACM, 2005.

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

[8] S. Rixner. Network virtualization: Breaking the performance barrier. *Queue*, 6(1):37, 2008.

[9] M. D. Schroeder and J. H. Saltzer. A hardware architecture for implementing protection rings. *Commun. ACM*, 15(3):157–170, Mar. 1972.

[10] P. Willmann, J. Shafer, D. Carr, S. Rixner, A. L. Cox, and W. Zwaenepoel. Concurrent direct network access for virtual machine monitors. In *High Performance Computer Architecture, 2007. HPCA 2007. IEEE 13th International Symposium on*, pages 306–317. IEEE, 2007.