가상화의 발전에 대한 서베이

박주영*, 신동주*, 김종국*
*고려대학교 전기전자공학부
nehalem@korea.ac.kr, alansynn@korea.ac.kr, jongkook@korea.ac.kr

A Survey on the Advancement of Virtualization Technology

JooYoung Park*, DoangJoo Synn*, JongKook Kim*
*Dept. of Electric and Electronics Engineering, Korea University

Abstract

This paper is a survey on the advancement of virtualization technology. Virtualization of resources was an inevitable path in modern computer systems. This abstraction of hardware allowed the decoupling of the operating system that manages the hardware and applications' requirements by adding a layer between them. It also led to the application-centric view of computing and light virtual machines, where each represents a computer networking device. As virtualization technology ripens, the performance of virtual machines can only improve. This paper will be introducing how virtualization technology has evolved from Xen to LightVM and Firecracker.

1. Introduction

In the early stages of kernels, monolithic kernels were outperforming μ-kernel, a cornerstone for virtualization. Monolithic kernel architecture had the entire OS run in kernel space, while μ-kernel architecture minimized the kernel and implemented server applications outside the kernel. Efforts to share resources across applications lead to the advancement of Exokernel, which is crucial for virtualization. Exokernel binds multiple library OS and hardware resources directly and securely. Xen, the x86 virtual machine monitor, was then presented to run multiple OSes on a single machine. Xen implemented hypervisor, which distributes hardware resources across multiple virtual machines, and it runs on host hardware directly. KVM, Kernel-based Virtual Machine which switches Linux system to hypervisor, and QEMU, an open-source emulator, was combined to propose a new type of hypervisor, which runs on a host OS as a software layer or application. Container OS could not run multiple OSes, but it could move containers from one kernel version to another while running with live migration. Each virtual machine was much lighter than other implementations. Borg and Kubernetes successfully offered a new method of viewing clusters as application-oriented by using containers as a unit of management, not machine-oriented. ClickOS was developed to meet network function virtualization needs, using modified Xen as hypervisor and Click as a programming abstraction. To minimize the tradeoff between performance and security made by container and virtual machines, LightVM and Firecracker were introduced. LightVM is based on Xen but heavily modified to reduce the overhead caused by Xen. Firecracker is based on KVM and QEMU, replacing QEMU with a modified version of Google's Chrome OS Virtual Machine Monitor.

2. μ-kernel is not a delusion

A kernel, in traditional perspective, is a part of OS that is mandatory and common to all other software. μ-kernel's basic idea is to minimize the kernel and to implement server applications outside the kernel if possible. This approach presented some software technological advantages, which are: (1) makes kernel interface and structure modular, (2) servers can use features provided by μ-kernel, which makes server programs run just like user programs, (3) system becomes more flexible and customizable, as different API and strategies can coexist in the system.

Although there were many implementation attempts, they lacked efficiency, which led to bad performance, and restricted flexibility. It was widely believed that inefficiency of μ-kernel is inherent. However, Jochen Liedtke [1] discovered that these inefficiencies were delivered due to inadequate implementation. He argued that μ-kernels were built on a thin hardware-dependent layer, which made kernel machine-independent but created performance overheads. Also, a lack of appropriate abstractions cost μ-kernels flexibility and performance. In the modified implementation, a lightweight kernel allows appropriate abstraction and isolation methods and moves server applications to user space to gain flexibility. This suggested approach provided the implementation of arbitrary operating systems and exploitation of varieties of hardware.

3. How to manage machine resources

In traditional operating systems, fixed interface and operating system abstractions lead to applications' flexibility, performance, and functionality limitations. They create virtual machines in which applications execute. In Exokernel [2], they implement OS abstractions as library at the application-level. They export hardware resources to application with two techniques. One is secure binding, which binds application to machine resources, and another is abort protocol, which allows Exokernel to break secure binding.

Exokernel comes with an operating system called Library OS, which works above Exokernel, implementing high-level abstraction. Library OS is tightly coupled to applications because it can be modified by users, easy to be specialized, and as Exokernel does not trust Library OS, Library OS can trust applications, not hurting application performance. The overall system structure is illustrated in Figure 1. Each Library OS implements its own system objects and policies.

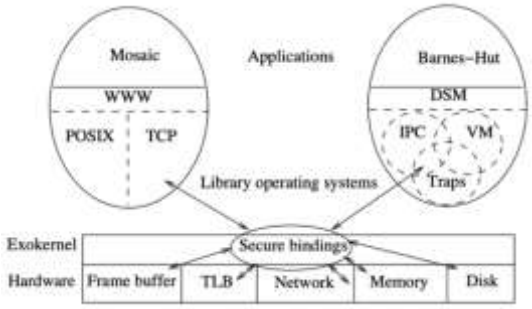

Figure 1: Exokernel-based system Overview [2]

4. How can a system hold multiple OSes?

Modern computers use virtual machines, which run on separate operating system instance. There are several challenges to execute multiple OS concurrently:(1) virtual machines must be isolated from one another, (2) wide range of OSes should be supported, (3) minimize performance overhead due to virtualization.

Xen [3], an x86 virtual machine monitor, was presented as a solution. Xen is placed between hardware and virtual machine (Figure 2), multiplexing hardware at the granularity of an entire operating system and providing performance isolation among virtual machines. Xen also implements paravirtualization for several reasons. First is to meet the requirement of running 100 virtual machines, and second is due to architectural reasons. x86 architecture is not fully virtualizable in the first place, as they did not take virtualization into account when they were designed. However, paravirtualization requires modifying guest OS, but Linux and XP only had to modify 1.36%, 0.04% of their codebase [3].

Xen design principles were; (1) Support for running unmodified application binary, (2) supporting full OS, (3) obtaining high performance and resource isolation, and abstracting resource completely which lowers correctness and performance (e.g., hardware page walker needs full availability of resources). By designing Xen to separate policy from mechanism wherever possible, Xen's overhead became negligible close to the baseline Linux system. Hypervisor was designed to provide only basic controls, not involved in or even aware of the actual event.

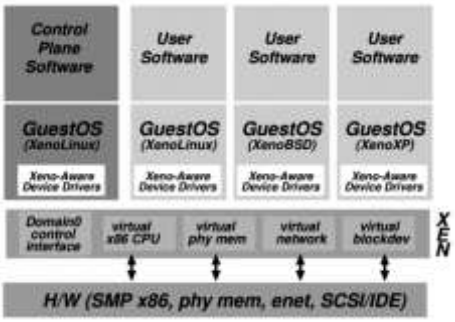

Figure 2: The structure of a machine running the Xen hypervisor, running multiple different guest OS [3]

5. QEMU and KVM, a way to new type of hypervisor

QEMU [4] is a machine emulator, which lets the user run target OS on top of another OS in a virtual machine. QEMU consists of many emulators and a dynamic translator. Dynamic translator performs runtime conversion of target CPU instructions into the host instruction set. This begins by splitting target CPU instructions into micro-operations. And then while they are translated into host functions, they bundle translated functions into Translated Blocks (TBs). TB contains codes between codes that modify CPU state and cached for later use. And they are chained to accelerate common cases.

KVM [5] makes Linux system a hypervisor. KVM adopted new execution mode, "guest mode", which has all regular privilege levels, except some execution gets trapped. Hardware state switch is implemented for speed, and MMU is virtualized, but not in way of classic shadow page table sync manner. KVM write protects guest OS's pages, and when a write to a guest page is trapped, KVM emulates precise effect on both guest and shadow page table. KVM also supports live migration, transporting a virtual machine from one host to another without interrupting guest execution for more than a few tens of milliseconds.

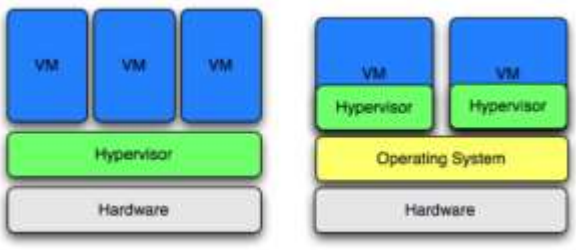

Figure 3: Type 1 (left) and Type 2 (right) Hypervisor [6]

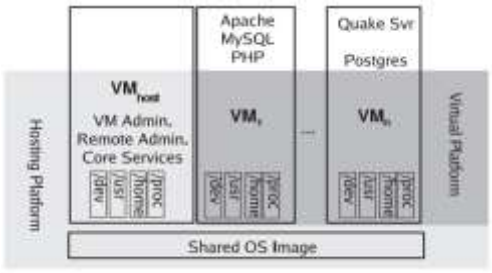

Figure 4: Container Overview [7]

QEMU and KVM combined, KVM enables virtualization capabilities provided by hardware, and QEMU emulates entire memory and I/O. They work together to simulate virtual machine hardware. As they got together, they form a type-2 hypervisor (see figure 3), which is much simple to manage, and easy to be used for testing. The QEMU+KVM approach is a different structure compared to Xen, which is the type-1 hypervisor. Type-1 hypervisors have to provide some OS functionality, but type-2 hypervisors can depend on the host operating system.

6. Container-based OS

The most significant difference between the hypervisor and container-based OS(COS)[7] is that hypervisors support multiple kernels on one system, but COS does not. Furthermore, COS goes one step further in migrating hypervisors because they support live-update, which means migration from one kernel version to another.

Figure 4 is an architecture of COS. COS shares a host kernel, and each container has its own file system, including chroot barrier. Thus, namespace is isolated, and resource is also isolated per container. VServer filesystem unifies files common to more than one VM, and the only drawback is virtual machine dying and destroying files.

7. From Borg to Kubernetes

Borg [8] is a cluster manager that runs thousands of jobs across thousands of clusters, each with thousands of machines. Borg hides the detail of resource management and handles failure automatically; therefore, it operates with very high reliability and availability.

Users submit workloads to Borg in jobs, which consists of one or more tasks, and each job runs in one Borg cell, which consists of thousands of machines. Each task is mapped to a set of Linux processes running in a container on a machine. Ineach cell (see figure 5), there are redundant, five copies of BorgMaster, multi processed schedulers, and a Borglet, local Borg agents that reside in each machine. BorgMaster saves states of all machines in cell and all changes are snapshotted. Scheduler does two jobs; first is feasibility checking, which is finding machines that task could run, and second is scoring, picking one of feasible machines. Borglet sends BorgMaster its health and state every few seconds.

While building and running Borg, there were some benefits we could get. As container abstracted away many details of machines and OSes, clusters became application-oriented, not machine-oriented, which improved application development and introspection. Application developers, infrastructure teams were both relieved from compatibility issues, and telemetries collected in application units.

Kubernetes [9] is an open-source, improved software-engineered version of Borg, easy to deploy and efficiently manage complex distribution systems. Kubernetes' goal is to give programmers productivity, and ease manual and automated system management.

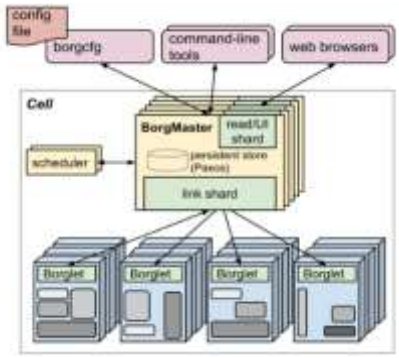

Figure 5: Borg Overview [8]

8. ClickOS and Network Function Virtualization

There are many middleboxes deployed in access and enterprise networks which have many downsides: expensive, unable to add new features without replacing them, cannot scale up or down quickly, and even takes a significant investment to develop new device. Due to these drawbacks, demands toward network function virtualization, NFV were widely accepted. Requirements were: (1) flexibility to run different types of software middleboxes, (2) isolation to support multi-tenant, (3) high throughput and low delay as middleboxes are mostly deployed at operator environment, (4) scalability to scale up and down quickly following traffic difference.

ClickOS[10] adopted Xen as a hypervisor, Click modular router software [11] as a programming abstraction, and a tailor-made guest OS to run Click. There were number of overhauls done to Xen network interface (see Figure 6), like changing Open vSwitch to ClickOS switch for exposing per port ring packet buffers, removing netback from pipe, only keeping control plane, and modifying netfront driver to map directly to exposed memory space, and netback driver to proxy event channel notifications netfront driver.

ClickOS is proof that software solution alone is enough to speed up NFV processing significantly. Small, quick-to-boot virtual machines made it possible to offer personalized processing to a large number of users with comparatively little hardware.

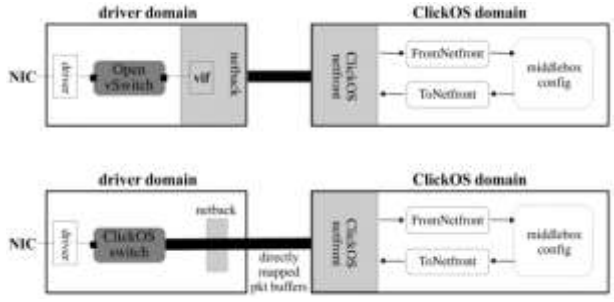

Figure 6: Standard Xen network I/O pipe (top), and optimized ClickOS one (bottom) [11]

9. Middle ground between VM and container

In a traditional approach, there is a choice between containers and virtualization. Containers tend to provide light overhead lack security due to kernel syscall API, and virtual machines provide sound isolation and security features, but with heavyweight. Some projects were promoted to get free of these tradeoffs. There were some approaches to hypervisor-based virtualization, one was type-1 hypervisor, which was implemented by LightVM [12] by slimming down Xen, and another is type-2 hypervisor, implemented by Firecracker [13] by using KVM and minimalized virtual machine monitor.

LightVM did a significant overhaul on Xen architecture to avoid major overhead parts, XenStore interaction and device creation. First, LightVM does not use the XenStore for VM creation or boot because the hypervisor already keeps the most necessary information about a VM. Secondly, LightVM splits VM creation functionality to prepare and execute phase, reducing the amount of work done on VM creation. With these implementations, LightVM achieved comparable boot time to fork/exec implementation in Linux, and almost constant creation and boot times regardless of the number of running VMs.

Firecracker, on the other hand, did large-scale modification to KVM/QEMU architecture by replacing QEMU with a modified version of Google's Chrome OS Virtual Machine Monitor which device drivers are removed. Firecracker provides limited number of emulated devices since it does not require most of them, and block devices for storage instead of filesystem passthrough due to security problem. Firecracker used KVM to support modern Linux hosts and guests. Reason for this is that Firecracker had to rely on component built into Linux due to implementation cost and operation knowledge from the previous infrastructure. Firecracker achieved primary goals to successfully replace its predecessor, which used Linux containers to isolate functions and virtualization to isolate between customer account, leading to a tradeoff between security, compatibility, and efficiency.

10. Conclusion

This paper presented a survey on the advancement of virtualization. Virtualization was first developed to share resources across multiple applications. The advancement of virtualization allowed the shift from machine-oriented view to an application-oriented approach for computing services. Future research should focus on both abstraction and flexibility, which hides out as much machine-centric view from programmers but allows direct access to hardware if needed.

Reference

[1] Liedtke, Jochen. "On micro-kernel construction." *ACM SIGOPS Operating Systems Review* 29.5 (1995): 237-250.

[2] Engler, Dawson R., M. Frans Kaashoek, and James O'Toole Jr. "Exokernel: An operating system architecture for application-level resource management." ACM SIGOPS Operating Systems Review 29.5 (1995): 251-266

[3] Barham, Paul, et al. "Xen and the art of virtualization." *ACM SIGOPS operating systems review* 37.5 (2003): 164-177.

[4] Bellard, Fabrice. "QEMU, a fast and portable dynamic translator." *USENIX annual technical conference, FREENIX Track*. Vol. 41. 2005.

[5] Kivity, Avi, et al. "kvm: the Linux virtual machine monitor." Proceedings of the Linux symposium. Vol. 1. No. 8. 2007.

[6] Fenn, Michael, et al. "An evaluation of KVM for use in cloud computing." *Proc. 2nd International Conference on the Virtual Computing Initiative, RTP, NC, USA*. 2008.

[7] Soltesz, Stephen, et al. "Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors." Proceedings of the 2Nd ACM SIGOPS/EuroSys european conference on computer systems 2007. 2007.

[8] Verma, Abhishek, et al. "Large-scale cluster management at Google with Borg." Proceedings of the Tenth European Conference on Computer Systems. 2015.

[9] Burns, Brendan, et al. "Borg, Omega, and Kubernetes: Lessons learned from three container-management systems over a decade." Queue 14.1 (2016): 70-93.

[10] Martins, Joao, et al. "ClickOS and the art of network function virtualization." 11th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 14). 2014.

[11] Kohler, Eddie, et al. "The Click modular router." ACM Transactions on Computer Systems (TOCS) 18.3 (2000): 263-297.

[12] Manco, Filipe, et al. "My VM is Lighter (and Safer) than your Container." *Proceedings of the 26th Symposium on Operating Systems Principles*. 2017.

[13] Agache, Alexandru, et al. "Firecracker: Lightweight virtualization for serverless applications." *17th {usenix} symposium on networked systems design and implementation ({nsdi} 20)*. 2020.