Scientific Bulletin of the „Petru Maior" University of Târgu Mureş
Vol. 7 (XXIV) no. 1, 2010, ISSN 1841-9267

VIRTUALIZATION AS THE EVOLUTION OF OPERATING SYSTEMS

Tudor Zaharia
Vrije Universiteit Amsterdam, The Netherlands
tza200@few.vu.nl

ABSTRACT

Although virtualization has been around for more than 50 years, the subject is hotter than ever. Initiated to simulate multiple machines using hardware and software techniques, virtualization today brings server consolidation, security and isolation for multiple operating systems running on the same hardware. This article's main objective is to show that virtual machines are just another step in the evolution of operating systems. I will argue that in fact a virtual machine monitor is a resource manager just like an operating system is. Important points like CPU and memory management are going to be addressed. Although not exhaustive, this paper will discuss the most important attributes of virtual machine monitors as operating systems and my view on how the field should evolve in this new light.

Keywords: virtualization, operating systems, VMM, hypervisor

1. Introduction

The "magic" word "virtualization" is on everybody's lips these days. It seems to provide a solution to many of the problems that computer scientist have had. From server consolidation all the way to running a Real-Time Operating System together with a General Purpose Operating system on a single core chip in a mobile phone [6], virtualization has proven its values time and time again. In High Performance Computing, virtualization improves productivity, performance, reliability, availability and security and decreases software complexity [8]. For embedded systems virtualization means decreased prices for manufacturing, since a decrease in bill of materials is expected [10]. Moreover, live migration [5], [2] improves reliability by enabling an entire running operating system to migrate on another machine with very little downtime.

In datacenters, the Intel 48 core [4] which was demonstrated last year, could finally find its use beyond very intensive computational applications. This is because although novel programming languages such as SAC [3], Occam [11] and others, for one reason or another, did not became mainstream. This means that programmers still use traditional imperative languages extended with Pthreads [9] and MPI [12] which makes parallel programming very difficult and does not take full advantage of the multicore processing power of recent processors. Instead, this power could be leveraged by the virtualization engines.

But virtualization provides a lot of features that operating systems also provide. Above all, a virtual machine monitor (VMM) is a resource manager. It allows multiple operating systems to share the same hardware and this looks very similar to the multi-user, multi-program support offered by operating systems. I will argue that virtualization is an evolution of operating systems and it should be treated like that. This means that some of the functionalities provided by the operating systems should be moved in the VMM and inherently the OSes should be moved higher on the stack (a thing that actually happened already).

Following is an outline of the different types of virtualization. Short details are given about each one.

Virtualization can be achieved in several ways:

1. Emulation/simulation. A software emulator allows computer programs to run on a platform (computer architecture and/or operating system) other than the one for which they were originally written. Multiple such emulators were released under public licenses [16]. The terms emulator and simulator are very close in meaning, and it's beyond this paper's scope to argue about which one is the most appropriate.
2. Partial virtualization simulates multiple instances of much (but not all) of an underlying hardware environment.
3. OS-level virtualization enables multiple isolated and secure virtualized servers to run on a single physical server.
4. Paravirtualization presents a software interface to virtual machines that is similar but not identical to that of the underlying

hardware. This means that the "guest" operating system needs to be modified in order to run in the paravirtualized hardware (to support the para-API). The term "paravirtualization" was first used in association with the Denali virtual machine [19].

5. Full virtualization provides a complete simulation of the underlying hardware. The result is a system in which all software capable of execution on the raw hardware can be run in the virtual machine.

Increased productivity, performance, reliability and all of the words are used alongside virtualization. But they used to be the qualities of operating systems as well. So what did exactly change why is a VMM different from an Operating System?

In this paper I will argue that virtual machines are just the evolution of the operating systems. For this I will start by looking at the definition of an operating system. [15] defines the operating systems regarding two individual aspects:

- an extension of a machine's functionality
- a resource administrator

These are also the main characteristics of a virtual machine. In fact, virtualization is just the latest development in the operating system's continuous adaptation to different forms of multitenancy. This started as the need to provide multiuser support and protected memory. I will argue that, against all recent trends which favor full virtualization, a new paradigm in operating systems is needed that will change the face of the operating systems. This is because virtualization should take over some functionality provided by the operating systems since it is already providing them.

This evolution does not exclude the recent advances in virtualization support provided by the CPUs, but the evolution should be made by rethinking each component's functionalities and together provide improved performance.

2. The virtual machine monitor is a resource manager

The extension of a machine's functionalities is made by the operating system through its process, memory and I/O management. A virtual machine manager does the same things, but instead of process management, it is doing operating system management. A virtual machine manager multiplexes resources in two modes: in time and in space, exactly as an operating system does [15].

This section will describe in detail each one and how some of the functionality actually moved from the operating system side to the virtual machine monitor. I will argue that the implementation of these functionalities should be rethought in order to provide better usage of resources. I will also try to give some answers on why these changes were not made yet (and might as well never be done).

2.1 Operating systems management

A virtual machine manager deals with multiple operating system instances, which can vary a lot with regards to the functionalities that they provide and/or the way they are implemented. But so do processes that run inside an operating system. To run in an operating system, processes make system calls which form the API of an operating system. To run in a (virtual) machine, operating systems make calls that form the API of that (virtual) machine. The VMM has just implemented another level of abstraction that allows multiple operating systems to share the same resources (i.e. to run at the same time on the same machine).

Considering this, operating systems lost the access to the privileged instruction in favor of the VMM that now runs in ring0 as it is called by VMware or dom0 in XEN.

If virtual machines calls could someday be standardized, every operating system developer could provide its version for that "architecture". One could argue that this calls are already standardized in a way (by the x86 architecture), and operating systems should use that. But this comes against the trend that has moved the operating system higher on the stack and its place was taken by the virtual machine manager.

In fact, even without the standardized interface, developers should port their operating systems to these diverse virtual machines architectures (just as they did by porting them to various CPU architectures). It would just seem fair since the application developers were constrained to port their application to particular operating systems (as POSIX standard was not adopted by all the operating system developers and therefore application developers had to port their code to each OS).

Of course, since the world of computer science depends on the business models as well, it is very improbable that we will see redesigned OSes so that they give up functionality in favor of VMMs.

I believe this happens because the major players in operating systems do not want a standardized API for virtualization nor they want to provide less functionality. Microsoft still has a very big share of the market [7] and it also entered the virtualization field. It is expected that most of their operating systems users would prefer to acquire their virtualization products.

On the other hand, VMware, the biggest virtualization company wants to impose its own API.

I believe these are the main reasons that hold back the rethinking the operating systems in general. Naturally, another reason is backward compatibility, for which full virtualization is the preferred choice.

2.2 CPU access management

Operating systems have a scheduling algorithm that decides which process/thread runs next. Similarly, a VMM should decide which operating system runs next.

Nowadays the most common thing to have is a hybrid, where a host operating system runs a type 2 (hosted) hypervisor and the guest operating systems are installed on top of this hypervisor. New solutions that make a VMM behave more like an OS are already available. WMware's variant is called ESX server. These solutions provide significantly higher performance [17].

An important penalty hit in virtualization comes from multiple switches between the virtual machine monitor and operating system. This is because some calls are only allowed in ring 0, the most privileged of all. This is similar to operating systems, where some instructions are only allowed to be made from a certain context. If the operating systems would be built keeping in mind that they are running over virtualized hardware, a lot of calls that trap to the VMM could be avoided.

2.3 Memory management

Virtual memory was introduced as an automatic alternative to the overlays used by a programmer to run software that would not fit in the machine's main memory [14]. A Memory Management Unit translates physical addresses into machine addresses. In a virtualized system, memory is virtualized by the hypervisor, which is another level introduced. It is called Shadow Page Tables (SPTs). These pages provide a level of indirection between virtual and machine addresses.

Intel and AMD both developed hardware support for these SPTs. Intel has called it's system "Extended page tables" and AMD uses the name "Rapid Virtualization Indexing". Another level of indirection only increases overhead, especially since this level could be entirely moved from the operating system to the VMM. Since the hypervisor is the one that runs in privileged mode, it should be the one that deals completely with the hardware resources. This means that operating systems should be, again, designed with this in mind. As it is right now, hacks like "ballooning" [17] are used by the VMM in order to reclaim memory. These again, introduce more overheard.

In my opinion SPTs are (another) step made in the wrong direction. The reasons are again business based. The OS developers do not want to modify their products while CPU manufacturers are always eager to support new features requested by the software.

2.4 I/O management

It is somewhat ironic that device drivers live on the application facing side (in the guest OS) which makes the hypervisor look like a microkernel. This if we would not consider that it runs its own drivers.

Currently there are three techniques used for I/O management, although only the first two are equivalent:

- User space device emulation.
- Hypervisor-based device emulation.
- Device passthrough.

In the first case, the device emulation is implemented in user space. QEMU [1] provides device emulation and it is used by a large number of independent hypervisors like KVM [13] and VirtualBox [18]. Device emulation is totally independent from the hypervisor, which makes this solution more secure than hypervisor-based device emulation.

The second solution, although faster, it burdens the hypervisor with this functionality.

Device passthrough can be used when only one virtual machine needs access to a particular device. In this case sharing becomes more efficient, as the virtualization engine provides isolation of devices to a given guest operating system so that the device can be used only by the designated guest. Improved performance and isolation of devices that cannot be shared are the main benefits of this approach.

Intel and AMD both provide support for device passthrough in their recent CPU architecture developments. "Virtualization Technology for Directed I/O" (VT-d) from Intel and "I/O Memory Management Unit" (IOMMU) from AMD provide the means to map PCI physical addresses to guest virtual machines addresses. This mapping ensures that the access is exclusively granted to that particular virtual machine which can use it as if it was a non-virtualized system.

I think this solution could be adapted so that it is used by the memory manager of a hypervisor. I can imagine a system with multiple hosts where each operating system has "passthrough" access to a part of the memory. In this case paging should (and probably will) be kept also on the OS level. An analogous example is a system with multiple (physical) USB ports.

These are individually isolated to given domains, so each virtual machine has exclusive access to certain ports.

Further evolution of the virtualization could bring virtualization-aware devices that would eliminate I/O virtualization overhead by employing the adequate hardware support. The devices should export multiple interfaces that can be mapped to virtual interfaces inside the virtual machines. Communication between the operating system and the device would be made directly, without trapping into the VMM. This is like DMA in a way, but not from the device to the memory, but from the operating system to the device.

3. Conclusions

Virtualization provides a lot of benefits to its user. Most significant, it brings server consolidation. Since the development of live VM migration, automatic load balancing is trivial and a robust model for dealing with hardware failures is real.

The benefits are visible also in mobile phones, where virtualization can decrease the costs of devices. Virtualization is a solution to provide true isolation among operating systems.

But there's more to virtualization. In this paper I have shown how virtualization is in fact a step in the operating systems' evolution. The main argument was that a virtual machine monitor is first of all a resource manager that multiplexes access to resources to multiple operating systems. I have argued that since the operating systems have moved higher on the stack, they should be designed accordingly. The main points that I discussed covered operating system, CPU, memory and I/O management made by the VMM.

But the trend is actually the opposite. The same functionality continues to be provided at the operating system's level and at the virtual machine monitor's one. Advances in CPU designs are made to move some of the burden into the hardware instead of rethinking the whole hardware and software stack. This might be because Operating Systems failed to provide the real process isolation, and the problem was solved through virtualization. Moving functionalities from the operating system to the VMM could open up old wounds and, moreover, turn operating systems into "bags of drivers" as Marc Anddressen once said.

References:

[1] Fabrice Bellard. *Qemu, a fast and portable dynamic translator*. In ATEC '05: Proceedings of the annual conference on USENIX Annual Technical Conference, pages 41–41, Berkeley, CA, USA, 2005. USENIX Association.

[2] Brendan Cully, Geoffrey Lefebvre, Dutch Meyer, Mike Feeley, Norm Hutchinson, and Andrew Warfield. *Remus: high availability via asynchronous virtual machine replication*. In NSDI'08: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation, pages 161–174, Berkeley, CA, USA, 2008. USENIX Association.

[3] Clemens Grelck and Sven-Bodo Scholz. Sac: *a functional array language for efficient multi-threaded execution*. Int. J. Parallel Program, 34(4):383–427, 2006.

[4] intel.com. *Futuristic intel chip could reshape how computers are built, consumers interact with their pcs and personal devices*: http://www.intel.com/pressroom/archive/releases/20091202comp sm.htm.

[5] Christopher Clark Keir, Christopher Clark, Keir Fraser, Steven H, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. *Live migration of virtual machines.* In In Proceedings of the 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI, pages 273–286, 2005.

[6] linuxfordevices.com. *Singlecore linux phone hits the market.* http://www.linuxfordevices.com/c/a/News/SinglecoreLinuxphonehi tsthemarket/.

[7] marketshare.hitslink.com. *Os market share*. http://marketshare.hitslink.com/os-market-share.aspx?qprid=9.

[8] Mark F. Mergen, Volkmar Uhlig, Orran Krieger, and Jimi Xenidis. *Virtualization for high-performance computing*. SIGOPS Oper. Syst. Rev., 40(2):8–11, 2006.

[9] Institute of Electrical and Inc. Information Technology Electronic Engineers. *Portable Operating Systems Interface (POSIX) Part: System Application Program Interface (API) Amendment 2: Threads Extension [C Language]*. IEEE, IEEE, 1995.

[10] ok labs.com. *Ok labs enables world's first virtualized smartphone, with mobile virtualization solution : Open kernel labs.* http://www.ok-labs.com/releases/release/ ok-labs-enables-worlds-first-virtualized-smartphone-with-mobile-virtualizat.

[11] A. W. Roscoe and C. A. R. Hoare. *The laws of occam programming*. Theor. Comput. Sci., 60(2):177–229, 1988.

[12] Marc Snir and Steve Otto. *MPI-The Complete Reference: The MPI Core.*
MIT Press, Cambridge, MA, USA, 1998.

[13] Sun Microsystems Inc. *The k virtual machine (kvm).* White paper, 1999.

[14] Andrew S. Tanenbaum. *Structured Computer Organization (5th Edition).* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2005.

[15] Andrew S. Tanenbaum. *Modern Operating Systems.* Prentice Hall Press, Upper Saddle River, NJ, USA, 2007.

[16] Jeffrey van der Hoeven, Bram Lohman, and Remco Verdegem. *Emulation for digital preservation in practice: The results.* International Journal of Digital Curation, 2(2), 2007.

[17] Carl A. Waldspurger. *Memory resource management in vmware esx server.*

[18] Jon Watson. *Virtualbox: bits and bytes masquerading as machines.* Linux J., 2008(166):1, 2008.

[19] Andrew Whitaker, Marianne Shaw, and Steven D. Gribble. *Denali: Lightweight virtual machines for distributed and networked applications*. In Proceedings of the USENIX Annual Technical Conference, 2002.