Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Technology 3 (2012) 267 – 272

Procedia Technology

The 2012 Iberoamerican Conference on Electronics Engineering and Computer Science

A summary of virtualization techniques

Fernando Rodríguez-Haro[a,b,*], Felix Freitag[b], Leandro Navarro[b], Efraín Hernández-Sánchez[a], Nicandro Farías-Mendoza[a], Juan Antonio Guerrero-Ibáñez[a], Apolinar González-Potes[a]

[a] *University of Colima, México, Faculty of Engineering*
[b] *Technical University of Catalonia (UPC), Computer Architecture Department, Spain*

Abstract

Nowadays, virtualization is a technology that is applied for sharing the capabilities of physical computers by splitting the resources among OSs. The concept of Virtual Machines (VMs) started back in 1964 with a IBM project called CP/CMS system. Currently, there are several virtualization techniques that can be used for supporting the execution of entire operating systems. We classify the virtualization techniques from the OS view. First, we discuss two techniques that executes modified guest OSs: operating system-level virtualization and para-virtualization. Second, we discuss two techniques that executes unmodified guest OSs: binary translation and hardware assisted. Finally, we present a summary of resource management facilities for capacity planning and consolidation of server applications.

Keywords: virtualization, capacity planning, hypervisor, CPU, tools

1. Introduction

During the last decade, virtualization is a technology that has been widely applied for sharing the capabilities of physical computers by splitting the resources among OSs. The journey started back in 1964 when IBM initiated a project that shaped the meaning of the term *Virtual Machines (VM)*[1]. The initial project was called CP/CMS system and evolved to the Virtual Machine Facility/370. Back in those years, the Control Program (CP) component was an operating system capable to use a computing machine to act like multiples copies of the machine itself. Each copy (VM) was controlled by its own OS (the CMS component).

The IBM project and the efforts from the research community stated the basis of low-level virtualization and by the mid 1970s, Goldberg and Popek published works [2][3] that gave a clear comprehension of the concepts and benefits related to virtualization technologies.

After those years the interest decreased and it was until 1999 when the company VMware Inc presented [4] its product *VMware Virtual Platform* for the x86-32 architecture. Parallel to this commercial product,

[*]Corresponding author: Faculty of Engineering, University of Colima. CP 28860. +52 3143311207
Email addresses: `frodrigu@ac.upc.edu` (Fernando Rodríguez-Haro), `felix@ac.upc.edu` (Felix Freitag), `leandro@ac.upc.edu` (Leandro Navarro)

an unpublished report of 1999 [5] discussed technical implementation issues for the low-level virtualization implementation in the x86-32 architecture.

Xen [6] is a virtual machine monitor (VMM) that allows a physical machine to boot different OSs in the x86-32 architecture. The Xen project is an open source community project which started in the academia at the University of Cambridge Computer Laboratory and currently developed by the Xen community (`http://www.xen.org/community/`).

The x86-32 architecture was not designed to support full virtualization in the sense that Popek and Goldberg described [3], hence the VMM implementations solved this issue by trapping privileged instructions with low-level complex solutions. However, the recent Intel's x86-32 and Itanium architectures support virtualization through extensions called VT-x for x86-32 and VT-i for Itanium [7] and AMD procesor's extensions through its virtualization solution "Pacifica" [8]. Nowadays, most of the virtualization solutions such as Xen and VMware [9] make use of these facilities.

Typically, the OS running in a physical machine is executed in kernel-mode and the user applications in user-mode. Processors have levels of execution so that the OS has access to the full set of instructions (privileged and unprivileged) of the physical processor. Thus, applications running in the unprivileged user-mode have no direct access to the privileged instructions. The x86 architecture has four execution modes (or four rings) where the OS's kernel-mode runs in Ring 0 and the user-mode runs in Ring 3. Depending of the OS, Ring 1 and Ring 2 can be used for additional levels of access.

Currently, there are several virtualization techniques that can be used for supporting the execution of entire operating systems. VMs for execution of guest OSs are also called *System VMs* [10] in contrast of *Process VMs* such as those supported by the Java Virtual Machine. In the following, we summarize the virtualization techniques from the OS view.

2. Execution of modified guest OSs

Operating System-level virtualization.. This technique virtualizes the physical server at the operating system level. Here, the host OS is a modified kernel that allows the execution of multiple isolated Containers -also known as Virtual Private Server (VPS), jail, or virtualized server-. Each Container is an instance that shares the same kernel of the host OS. Some examples that use this technique are Linux-VServer, Solaris Zones, and OpenVZ. This technique shows low overhead and their implementations are widely used. The main drawback is that it does not support multiple Kernels.

Para-virtualization.. This technique adds a special set of instructions (named Hypercalls) that replaces instructions of the real machine's instruction set architecture. The advantage is that it has low virtualization overhead. Examples of Para-virtualization solutons are Denali [11], Xen [6], and Hyper-V [12]. In the x86 architecture, the VMM (or Hypervisor) runs just above the physical hardware (Ring 0) so that guest OSs run in higher levels. Even though this technique supports multiple Kernels, the main drawback is that to take full advantage of this technique the kernel of the guest OSs needs to be modified in order to make use of the Hypercalls.

3. Execution of unmodified guest OSs

Binary translation.. This technique is used for the emulation of a processor architecture over another processor architecture. Thus, it allows executing unmodified guest OSs by emulating one instruction set by another through translation of code. An example of multiplatform emulation is QEMU [13] which is a processor emulator developed by Fabrice Bellard. Currently, QEMU supports full CPU emulation of x86, x86-64, ARM, SPARC, PowerPC, MIPS, and m68k (Coldfire) though there are more architectures that are in current development [14]. That is, OSs originally developed for these processors can be fully executed by QEMU. Nowadays, QEMU has full host CPU support of x86, x86-64, and PowerPC and there are versions for Linux, Windows, Mac OS X, and OpenSolaris. The advantage of full processor emulation is the multiplatform portability of application code and OSs, the disadvantage is the overhead caused by the emulation of the complete instruction set.

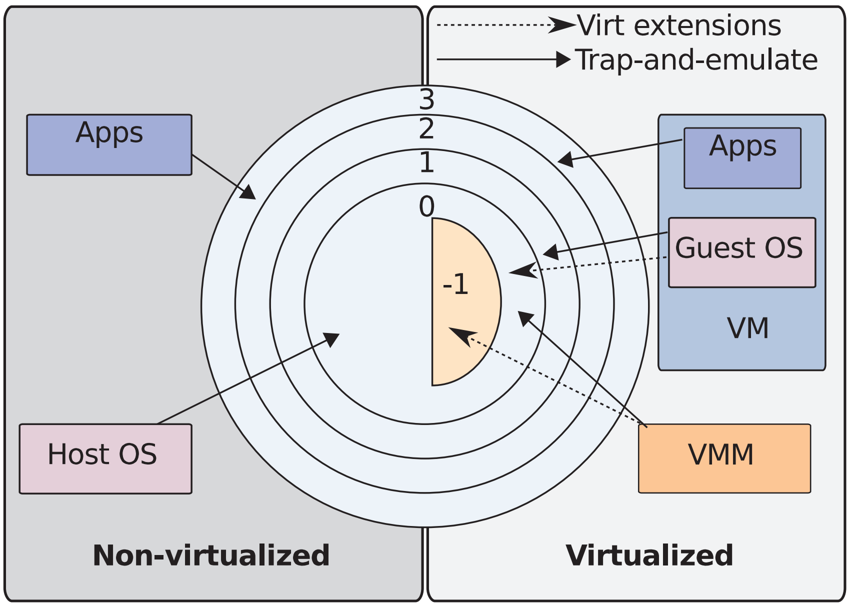

Fig. 1. CPU protection ring levels.

Binary translation (BT) is also used for virtualization. By using BT in conjunction with direct execution the purpose of virtualization of having copies (VMs) of the physical machine is achieved. Unlike emulation, BT is used to translate or emulate a small set of the processor instructions. That is, the code that needs privileged execution (Ring 0) such as kernel-mode and real mode. The rest of the instructions are executed directly by the host CPU. Examples that use similar techniques are VMware virtualization solutions [15] and QEMU Accelerator (KQEMU) [16]. This approach outperforms full emulation and shows low overhead. On the other hand, unlike emulation the support of unmodified guest OSs is limited to those that run in the host CPU.

Hardware assisted. Hardware assisted virtualization is not a new concept. It was first implemented when the Virtual Machine Facility/370 became available in 1972 for the IBM System/370 [1]. Recently, processor vendors such as AMD and Intel introduced virtualization extensions to their line of products in 2006. Hardware assisted virtualization implements a Ring with a higher privileged mode in the processor architecture, see Figure 1. The CPU extensions for virtualization support allows executing unmodified guest OSs in Ring 0 (non-root mode) and the VMM or Hypervisor in Ring -1 (root mode). Hardware assisted virtualization enhances CPUs to support virtualization without the need of binary translation or Para-virtualization. The advantage is that this technique reduces the overhead caused by the trap-and-emulate model, instead of doing it in software here it is done in hardware. Examples of virtualization that uses hardware assisted are Kernel-based Virtual Machine (KVM), VirtualBox, Xen, Hyper-V, and VMware products.

4. Hypervisors

A Hypervisor or Virtual Machine Monitor (VMM) is a “software layer that virtualizes all of the resources of a physical machine, thereby defining and supporting the execution of multiple virtual machines (VMs)” [11]. The properties *efficiency, resource control, and equivalence* of a VMM were defined in 1974 [3]. There are two types of VMMs type I and II [17], see Figure 2. A Hypervisors type I runs directly above the hardware in the Ring of highest privilege and controls all VMs. These hypervisors are also known as native and they support *Classic system VMs*. Some examples of Hypervisors Type I are VMware ESX, Hyper-V, and Xen [18]. Hypervisors type II, also known as hosted, run within an OS along with the Ring of

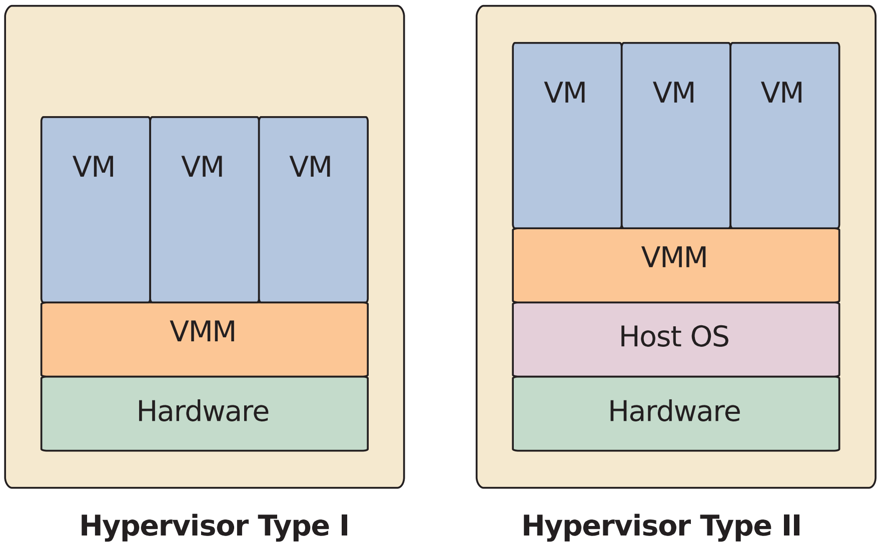

Fig. 2. System VMs.

the Host OS and support *Hosted VMs*[10]. Some examples of Hypervisors Type II are VMware Server and VirtualBox.

Hypervisors type I have resource management components such as memory, CPU, and I/O. For instance, there is a VM scheduler which is used for CPU resource management. On the contrary, Hypervisors type II rely on the OS process scheduler, i.e. each running VM is another OS process.

5. Capacity planning and consolidation

It is clear that the decision of using one virtualization or another one is up to the infrastructure needs vs the facilities offered by each solution: portability, migration, snapshots, administration, etc.

Xen and VMware ESX provides similar CPU resource management facilities. Xen's Credit scheduler and VMware ESX scheduler [19] have the following features: SMP support, both uses *shares* to define the amount of resource that each VM can consume (proportional share), processor affinity support, and maxCPU limit. VMware ESX, additionally supports resource pools and minCPU limit. Regarding to dynamic memory management, it is currently supported with memory balloning [20, 21] by Xen, VMware ESX, and KVM.

With Xen or VMware ESX solutions we can consolidate, in a physical node, the workloads of different applications and use the specific-purpose VM scheduler to handle the CPU resource management. We can apply the infrastructure policies (static capacity planning) to define the amount of each resource assigned to the VM (e.g. CPU share or memory) during the VM creation. This proportion of the assigned resources will remain static unless we modify the assigned share through the administrative privileged console. However, any change to achieve a certain performance level needs knowledge of all running VMs in advance: the number of VMs, their specific application's needs and the type of workloads. Finally, the gathered information can be used to apply dynamic capacity planning in order to recalculate the new proportion of each resource needed by each VM. In a dynamic environment this is a problem for administrators.

The features offered by virtualization platforms like Xen and VMWare have allowed the usage of clusters of VM-based resource providers. However, each application has different behavior (burst, batch, IO intensive, CPU intensive, network intensive). Moreover, each application requires different hardware (amount of memory, CPU, bandwidth) and software (operating system, deployment packages, network configuration). These dynamic features in a cluster of VM-based resource providers make it challenging to assign properly according to certain optimization criteria the physical resources to the VMs (i.e. the administrators face that the task of manual management becomes unfeasible).

Even though administrators have virtualization management tools to set the proportion of physical resources, it is certainly true that it is necessary to have a certain expertise and read the suggested guidelines [22] to properly set the low-level parameters that match the needed resources of each VM (memory, CPU, disk, etc).

6. Conclusions

A virtualization solution can have different CPU management facilities, most of them rely on schedulers that support weighted shares and maxCPU limit. Some of them support minCPU limit and additional facilities. Therefore, capacity planning for each VM in large-scale scenarios is a challenge and requires additional guidelines[23] and expertise to properly set the configuration parameters and amounts which better match each workload. At the end, a VMM manages the physical hardware and it does not need to know which workload each VM is running in. Therefore, upper management layers can leverage the VMM facilities in order to implement application-aware resource managers. We are working towards a management layer for cloud computing providers. The challenge is to abstract virtualization facilities in order to support high-level application requirements at each of the physical machines.

7. Acknowledgements

This work is supported in part by the Ministry of Education of Mexico, the program PROMEP of the Public Education Secretary under Contract PROMEP/103.5/10/7336, the University of Colima, and the Computer Architecture Department of the Technical University of Catalonia.

References

[1] R. J. Creasy, The origin of the vm/370 time-sharing system, IBM Journal of Research and Development 25 (5) (1981) 483–490.

[2] R. P. Goldberg, Architecture of virtual machines, in: Proceedings of the workshop on virtual computer systems, ACM Press, New York, NY, USA, 1973, pp. 74–112. doi:http://doi.acm.org/10.1145/800122.803950.

[3] G. J. Popek, R. P. Goldberg, Formal requirements for virtualizable third generation architectures, Commun. ACM 17 (7) (1974) 412–421. doi:http://doi.acm.org/10.1145/361011.361073.

[4] VMWare Inc, Introducing vmware virtual platform, technical white paper (February 1999).

[5] K. P. Lawton, Running multiple operating systems concurrently on an ia32 pc using virtualization techniques, unpublished (November 1999).
URL `http://www.ece.cmu.edu/~ece845/sp05/docs/plex86.txt`

[6] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield, Xen and the art of virtualization, in: SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles, ACM Press, New York, NY, USA, 2003, pp. 164–177. doi:http://doi.acm.org/10.1145/945445.945462.

[7] R. Uhlig, G. Neiger, D. Rodgers, A. L. Santoni, F. C. Martins, A. V. Anderson, S. M. Bennett, A. Kagi, F. H. Leung, L. Smith, Intel virtualization technology, Computer 38 (5) (2005) 48–56. doi:http://doi.ieeecomputersociety.org/10.1109/MC.2005.163.

[8] Advanced Micro Devices, AMD64 Virtualization Codenamed "Pacifica" Technology. Secure Virtual Machine Architecture Reference Manual (May 2005).

[9] VMWare, `http://www.vmware.com` (2006).

[10] J. E. Smith, R. Nair, The architecture of virtual machines, Computer 38 (5) (2005) 32–38. doi:http://doi.ieeecomputersociety.org/10.1109/MC.2005.173.

[11] A. Whitaker, M. Shaw, S. D. Gribble, Denali: Lightweight virtual machines for distributed and networked applications, Tech. rep. (Feb. 08 2002).

[12] M. MSDN, Hyper-V Architecture, `http://msdn.microsoft.com/en-us/library/cc768520.aspx` (2008).

[13] F. Bellard, QEMU, a fast and portable dynamic translator, in: USENIX Annual Technical Conference, FREENIX Track, USENIX, 2005, pp. 41–46.
URL `http://www.usenix.org/events/usenix05/tech/freenix/bellard.html`

[14] F. Bellard, QMEU: Status of supported CPUs and guest OSs, `http://bellard.org/qemu/status.html/` (2009).

[15] VMWare, Understanding Full Virtualization, Paravirtualization, and Hardware Assist. Whitepaper, `http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf` (2007).

[16] F. Bellard, QEMU Accelerator (KQEMU), http://bellard.org/qemu/kqemu-tech.html (2009).
[17] IBM, IBM systems virtualization. version 2 release 1, http://publib.boulder.ibm.com/infocenter/eserver/v1r2/topic/eicay/eicay.pdf (2005).
[18] Xen, www.cl.cam.ac.uk/research/srg/netos/xen (2006).
[19] VMWare, ESX Server Performance and Resource Management for CPU-Intensive Workloads. Whitepaper, http://www.vmware.com/pdf/ESX2_CPU_Performance.pdf (2006).
[20] C. A. Waldspurger, Memory resource management in vmware esx server, SIGOPS Oper. Syst. Rev. 36 (SI) (2002) 181–194. doi:http://doi.acm.org/10.1145/844128.844146.
[21] D. Magenheimer, Memory overcommit... without the commitment. Extended abstract for Xen summit, http://oss.oracle.com/projects/tmem/dist/documentation/papers/overcommit.pdf (2008).
[22] VMWare, Performance Tuning Best Practices for ESX Server 3. Guide., http://www.vmware.com/pdf/vi_performance_tuning.pdf (2007).
[23] VMWare, ESX Performance Tips and Tricks. Whitepaper. Latest Revision: Feb 7, 2005, http://www.vmware.com/pdf/esx_performance_tips_tricks.pdf (2005).