A Comparison of Software and Hardware Techniques for x86 Virtualization

Keith Adams, Ole Ageson
kma@vmware.com, ageson@vmware.com

10 July 2007

Author:
Michael Wallner
mwallner@cosy.sbg.ac.at
Department of Computer Sciences
University of Salzburg

Abstract

This article is a survey of the paper *A Comparison of Software and Hardware Techniques for x86 Virtualization* (1), which was presented at ASPLOS 2006. Virtualization is a technique for hiding the physical characteristics of computing resources from the way in which other systems, applications, or end users interact with those resources.

1 Introduction

Currently we experience hype on virtualization in commerce, which also results in new research on this topic. There are two main trends in the development of virtual machine systems:

- *Paravirtualization*, where the guest operating systems are modified so that they can be run concurrently with each other, and
- *Full System Virtualization*, where the hardware architecture is completely replicated virtually. Research on full system virtualization follows some new techniques on software and hardware side. The two leading processor manufactures – namely Intel and AMD – have imbedded explicit support for virtualization into their current product designs.

The current rise of virtualization is driven by a variety of different motivations:

- *Application and OS debugging* without the requirement of having multiple machines for each platform. It enables investigation on a crashed or compromised system running in a virtual machine and bidirectional debugging of the interaction between applications and the operating system.
- *Distributed load balancing*, especially interesting for data centers and hosting provider
- *Performance and data isolation* among the virtual machines
- *Reducing hardware costs* for data centers and software developers
- *Reliability and robustness* through live migration from one to another machine upon hardware failures
- *Sandboxing of untrusted applications* to provide security and reliability.
- *Simplify application deployment* by bundling a whole environment into a package to avoid complications with dependencies and versioning.

2 History

2.1 Classical virtualization

Popek and Goldberg defined a standard, how to consider a software system to be a virtual machine monitor. It must meet the following three criteria:

- *Equivalent execution*. Programs running in a virtual environment run identically to running natively, barring differences in resource availability and timing.
- *Performance*. A major subset of instructions must be executed directly on the CPU.
- *Safety*. A virtual machine monitor must completely control the system resources.

An early technique for virtualization was *trap and emulate*. While this approach was successful at offering an equivalent execution environment, its performance was brutally lacking since each virtual instruction could require dozens of native instructions to emulate.

2.2 First Approaches

The pioneer on virtualization was IBM with his VM/370 system which provides each client (user) a full virtualized replication of the underlying IBM System/370 hardware (2). The system has a specialized architecture that aids virtualization by using a time-sharing approach. This hardware extension brings a boost in performance against the pure simulation of the instruction for each virtual machine. Another benefit is that these instructions safely run in hardware without to interference other virtual machines. This technique has influenced the current approaches to efficient virtualization.

3 Full System Virtualization

A fully virtualized system is a replication of the system's hardware so that an operating system and its applications may run on the virtual hardware exactly as if they would have been executed on the original hardware. This new execution environment must meet the expectations of the guest operating system that are executed in it.

3.1 Virtualization of x86

Virtualization on the x86 architecture requires unnecessary complexity as a result of its native deficit to support virtual machines, since it was never intended to be virtualized. (3)

To support the virtualization through the hardware, some architectural challenges have to be addressed:

- *Address space compression*. The virtual machine monitor must protect the portions of address spaces that are used from the guest. Otherwise, the guest can discover that it is running in a virtual machine or compromise the virtual machine's isolation.
- *Interrupt virtualization*. The virtual machine monitor wants to maintain control of system and therefore masks and unmasks external interrupts. Some guest OSes frequently do these themselves, which results in poor performance if a switch to the virtual machine monitor is required for each masking instruction.
- *Non-privileged sensitive instructions*. The x86 ISA supports sensitive instructions that are not privileged and therefore do not trap to the virtual machine monitor for correct handling.
- *Ring aliasing*. The current privilege level of a guest OS is exposed, contrary to the guest's belief that it is running in ring 0.
- *Ring compression*. To provide isolation among virtual machines, the virtual machine monitor runs in the highest privileged ring and the virtual machines run in a lower privileged ring. This results in ring compression, when running a guest OS in ring 3, unprotected from the user application.
- *Silent privilege failures*. Some privileged accesses, rather than trapping to the virtual machine monitor, fail silently without faulting.

Here are the main procedures that overcome this weakness:

- *Non-sensitive and non-privileged instructions may run directly on the processor*. Instructions that are known to be safe can be directly executed on the processor without intervention.
- *Sensitive and privileged instruction trap*. As the virtual machine is run in user mode, it will be interrupted by the CPU if it executes a privileged instruction. The virtual machine monitor traps this instruction and performs the necessary steps to emulate the instruction for the virtual machine.
- *Sensitive and non-privileged instructions are detected*. The x86 ISA has some problem instructions, which are extremely sensitive on running in a virtual machine, but they cannot be trapped.

Much effort has been put into the virtualization of the IA-32 as it is the most dominate computer architecture nowadays. While the IA-64 (Itanium) architecture is not widely popular yet, some groups are beginning to consider the IA-64 platform's ability to host virtual machines. IA-64 provides one important feature which is not available on IA-32: ring compression.

3.1.1 Intel VT-x

Intel calls its virtualization extension *VT-x* (4) which was primarily codenamed Vanderpool. The technique introduces new modes of CPU operation:

- *VMX root operation*, which is the host mode and intended for virtual machine monitors.
- *VMX non-root operation*, which is essentially a guest mode targeting virtual machines.

For the interaction between the host and the guest mode Intel introduced the in-memory data structure *Virtual Machine Control Block* (VMCB) which contains both the guest and the host state. The introduced instruction `vmrun` loads the guest state from the VMCB and switches from host into guest mode. The guest execution proceeds until some pre-defined conditions expressed in the VMCB are met.

Adams and Agesen demonstrate in their paper that software techniques for virtualization outperform a hardware-based virtual machine monitor. The hardware virtual machine monitor performs better in some of the experiments, but overall the software virtual machine monitor provides a better high-performance virtualization solution. Reasons for these results are:

- *Maturity*. Hardware assisted virtualization on the x86 architecture is still an up-and-coming technology while software techniques have been around long.
- *Page faults*. Maintaining the integrity of the shadow page tables can be expensive and produce many virtual machine exits.
- *Statelessness*. virtual machine monitor must rebuild the source for a virtual machine exit from information in the VMCB.

3.1.2 AMD-V

AMD's extension got the name *AMD-V* (5) which was codenamed Pacifica. Its functionality is quite similar to Intel's VT-x but provides an incompatible instruction set architecture. In contrast to Intel's solution it provides an I/O Memory Mapping Unit (IOMMU) which extends the microcontroller with an additional memory mapping unit and provides access protection on direct memory access.

3.2 Full System Virtualization Drawbacks

Full system virtualization has the benefit of running operating system and applications unmodified. This has one big drawback, that special tricks must be used to virtualize the underlying hardware for each virtual machine since virtualization was no design goal of the original x86 architecture.

Efficient virtual memory management is also extremely difficult since the virtual machine monitor must catch all memory access and translate the address space of the virtual machine into the real memory of the system.

4 Paravirtualization

Full system virtualization is extremely complex. *Paravirtualization* overcomes the problem by modifying the operating system in a way that protected tasks are performed on the virtual machine monitor instead of the CPU. Paravirtualization gives some virtual machine information to its guest operating systems to enable them to make more informed decisions on things like page replacement. Paravirtualization significantly simplifies the process of virtualization by eliminating special hardware features and instructions that are difficult to virtualize.

4.1 Denali

The term paravirtualization was introduced with the Denali virtual machine monitor developed at the University of Washington (6). It provides fast containers for running virtual machines within. The Denali VMM does not supply an exact replication of the underlying hardware and therefore requires the client operating system to be modified. For that reason Denali can outperform full system virtualization. The most important techniques that are used in Denali are listed below:

- *Generic I/O devices*. Denali provides a small set of generic devices instead of a specialized access to devices on the physical system
- *Idle loops*. When the virtual machine calls the idle instruction a context switch to the virtual machine monitor occurs so that it can schedule other virtual machines
- *Interrupt queuing*. Denali queues interrupts so that the interrupt is dispatched the next time the virtual machine, where the interrupt is addressed to, is run.
- *Interrupt semantics*. As a side effect of interrupt queuing, Denali must alter the semantic of interrupts to mean "something happened recently" instead of "something just happened".
- *No Bios*. Denali provides system information in read-only register.
- *No virtual memory*. Each virtual machine has its own address space.

4.2 Xen

The (leading) paravirtualization system *Xen* is an high performance virtual machine monitor which is developed by the University of Cambridge (7). It provides an high performance virtual machine monitor with performance isolation. Guest operating systems that are intended to run on Xen must be ported to have everything behave as expected. These modifications target only the operating system and do not require changes in the application binary interface (ABI). Xen presents a virtual machine abstraction that is similar but not identical to the underlying hardware system.

Xen offers the following capabilities:

- Generic I/O devices,
- Lower privileged levels,
- No hardware interrupts,
- Partial access to hardware page tables,
- System calls registered with processors,
- Trap handlers registered with virtual machine monitor

Starting with Xen version 3.0 an abstraction layer is introduced, which enables to run unmodified operating systems within Xen virtual machines on machines with a hardware virtualization extension like Intel's VT-x (8). This allows proprietary operating systems where the kernel cannot be ported to be virtualized.

5 Conclusion

Virtualization will be a hot topic over the next years and will lead into more research topics as the market will require fast and reliable solutions. The current hardware extensions to the x86 architecture do not provide the expected performance gains through some design faults but it is an import step into the goal of achieving true native execution speed in a virtual machine on the x86 architecture. It relies to the engineers form the chip manufactures to bring up new solutions and techniques in their up-coming products and to the VMM software developer to efficiently use them.

It will be interesting to see if hardware virtualization techniques can outperform software-only virtualization techniques, or if the optimal solution might be a hybrid approach.

References

1. *A Comparison of Software and Hardware Techniques for x86 Virtualization.* **Adams, Keith and Agesen, Ole.** ASPLOS 2006.

2. **Creasy, Robert J.** The origin of the vm/370 time-sharing. *IBM Journal of Research and Development.* 1981, Vol. 25, 5.

3. **Fisher-Ogden, John.** *Hardware Support for Efficient Virtualization.* San Diego : s.n.

4. **Neiger, Gil, et al.** Intel Virtualization Technology: Hardware support for efficient processor virtualization. *Intel Technology.* 2006, Vol. 10, 3.

5. **AMD.** *Amd64 virtualization codenamed "pacifica" technology: Secure virtual machine architecture reference.* 2005.

6. **Whitaker, Andrew, Cox, Richard S. and Gribble, Steven D.** Denali: Lightweight Virtual Machines for Distributed and Networked Applications. *Univ. of Washington Technical Report.* 2002, Vols. 02-02-01.

7. *Xen and the Art of Virtualization.* **Paul, Barham T., et al.** SOSP'03.

8. **Dong, Yaozu, et al.** Extending Xen with Intel® Virtualization Technology. *Intel® Virtualization Technology.* 2006, Vol. 10, 3.