Intel Virtualization Technology

Nicolae-Andrei Vasile

Faculty of Computer Science and Automatic Control

POLITEHNICA University of Bucharest

Bucharest, Romania

***Abstract* – Intel Virtualization Technology provides hardware support for processor virtualization, enabling simplifications of virtual machine monitor software. Resulting VMMs can support a wider range of legacy and future operating systems while maintaining high performance. In this paper, we present the Intel Virtualization Technology.**

I. INTRODUCTION

Virtualizing a computing system's physical resources to achieve improved sharing and utilization has been well established for decades. Full virtualization of all system resources, including processors, memory, and I/O devices, makes it possible to run multiple operating systems on a single physical platform. In a nonvirtualized system, a single OS controls all hardware platform resources. A virtualized system includes a new layer of software, the Virtual Machine Monitor (VMM). The VMM's principal role is to arbitrate accesses to the underlying physical host platform's resources so that multiple operating systems (which are guests of the VMM) can share them. The VMM presents to each guest OS a set of virtual platform interfaces that constitute a Virtual Machine (VM) [1], [2].

Once confined to specialized, proprietary server and mainframe systems, virtualization is now becoming more broadly available and is supported in off-the-shelf systems based on Intel Architecture (IA) hardware. This development is due in part to the steady performance improvements of IA-based systems, which mitigates traditional virtualization performance overheads. Other factors include new creative software approaches that address the difficulties inherent to IA virtualization and the emergence of novel applications for virtualization in both industry and academia [1].

Classic benefits of virtualization include improved utilization, manageability, and reliability of mainframe systems. Several users with differing OS requirements can more easily share a virtualized server, OS upgrades can be staged across VMs to minimize downtime, and failures in guest software can be isolated to the VMs in which they occur.

II. BASIC DEFINITIONS

To accurately characterize the performance of different virtualization technologies we begin with an overview of the major virtualization strategies that are in common use for production computing environments. In general, most virtualization strategies fall into one of four major categories [2]:

Full Virtualization

Also sometimes called hardware emulation. In this case an unmodified operating system is run using a hypervisor to trap and safely translate/execute privileged instructions on-the-fly. Because trapping the privileged instructions can lead to significant performance penalties, novel strategies are used to aggregate multiple instructions and translate them together. Other enhancements, such as binary translation, can further improve performance by reducing the need to translate these instructions in the future.

Paravirtualization

Like full virtualization, paravirtualization also uses a hypervisor, and also uses the term virtual machine to refer to its virtualized operating systems. However, unlike full virtualization, paravirtualization requires changes to the virtualized operating system. This allows the VM to coordinate with the hypervisor, reducing the use of the privileged instructions that are typically responsible for the major performance penalties in full virtualization. The advantage is that paravirtualized virtual machines typically outperform fully virtualized virtual machines. The disadvantage, however, is the need to modify the paravirtualized virtual machine/operating system to be hypervisor-aware. This has implications for operating systems without available source code.

Operating System-level Virtualization

The most intrusive form of virtualization is operating system-level virtualization. Unlike both paravirtualization and full virtualization, operating system-level virtualization does not rely on a hypervisor. Instead, the operating system is modified to securely isolate multiple instances of an operating system within a single host machine. The guest operating system instances are often referred to as virtual private servers (VPS). The advantage to operating system-level virtualization lies mainly in performance. No hypervisor/instruction trapping is necessary. This typically results in system performance of near-native speeds. The primary disadvantage is that all VPS instances share a single kernel. Thus, if the kernel crashes or is compromised, all VPS instances are compromised.

Native Virtualization

Native virtualization leverages hardware support for virtualization within a processor itself to aid in the virtualization effort. It allows multiple unmodified operating systems to run alongside one another, provided that all operating systems are capable of running on the host processor directly. That is, native virtualization does not emulate a processor. This is unlike the full virtualization technique where it is possible to run an operating system on a fictional processor, though typically with poor performance. In x86-64 series processors, both Intel and AMD support virtualization through the Intel-VT and AMD-V virtualization extensions.

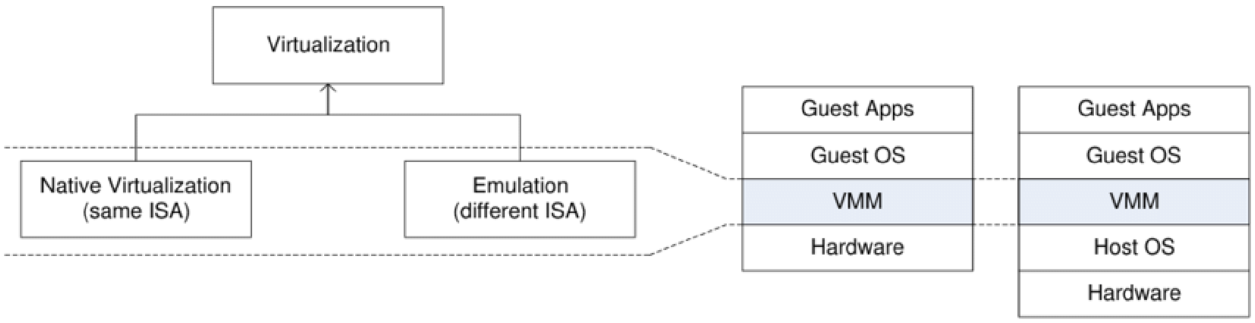

Figure 1. Virtualization

Figure 1 portraits the native virtualization, emulation and the virtual machine monitor, also known as hypervisor, as an abstraction layer.

III. INTEL VIRTUALIZATION TECHNOLOGY

Intel virtualization technology is a native virtualization technology. It is the processor's hardware ability to divide and isolate its computing capacity for multiple host virtual machines and their operating systems. Two architectures are provided by Intel and they are discussed in this chapter [1].

VT-x architecture overview

VT-x augments IA-32 with two new forms of CPU operation: VMX root operation and VMX non-root operation. A VMM runs in VMX root operation and its guests in VMX non-root operation. Intel microprocessors provide protection based on the concept of a 2-bit privilege level, using 0 for most-privileged software and 3 for the least privileged. Both forms of operation support all four privilege levels, allowing a guest OS to run at its intended privilege level and providing a VMM with the flexibility to use multiple privilege levels. VMX root operation is similar to IA-32 without VT-x. Software running in VMX non-root operation is deprivileged in certain ways, regardless of privilege level.

VT-x defines two new transitions: a transition from VMX root operation to VMX non-root operation, from VMM to guest, called a VM entry, and a transition from VMX non-root operation to VMX root operation, from guest to VMM, called a VM exit.

The virtual-machine control structure (VMCS) is a new data structure that manages VM entries and VM exits and processor behavior in VMX non-root operations. The VMCS is logically divided into sections, two of which are the guest-state area and the host-state area. These areas contain fields corresponding to different components of processor state. VM entries load processor state from the guest-state area. VM exits save processor state to the guest-state area and then load processor state from the host-state area.

Processor behavior changes substantially in VMX non-root operation. Most importantly, many instructions and events cause VM exits. Some instructions cannot be executed in VMX non-root operation because they cause VM exits unconditionally; these include CPUID, MOV from CR3, RDMSR, and WRMSR. Other instructions, interrupts, and exceptions can be configured to cause VM exits conditionally, using VM-execution control fields in the VMCS.

VM-execution control fields

The VM-execution control fields allow a VMM the flexibility to specify the instructions and events that cause VM exits. There are separate controls for each of the following instructions: HLT, INVLPG, MOV CR8, MOV DR, MWAIT, RDPMC, and RDTSC. These controls support a variety of virtualization strategies. Additional controls allow selective protection of CR0, CR3, and CR4 [1].

VT-x includes two controls that support interrupt virtualization. When the external interrupt exiting control is set, all external interrupts cause VM exits; in addition, the guest cannot mask interrupts. When the interrupt-window exiting control is set, a VM exit occurs whenever guest software indicates that it is ready to receive interrupts. To support VMM flexibility, VT-x includes bitmaps that allow a VMM selectivity regarding some causes of VM exits. One of these is the exception bitmap, which contains 32 entries for the IA-32 exceptions. It allows a VMM to specify which exceptions should cause VM exits and which should not. Another bitmap allows per-port control of I/O instructions.

VMCS details

The guest-state area contains the state of the virtual CPU associated with the VMCS. It includes fields corresponding to the IA-32 registers that manage processor operation, such as the segment registers, CR3, and IDTR. In addition, the guest-state area includes fields corresponding to certain components of non-register processor state, for example, the descriptor caches for the segment registers. Inclusion of these components allows the VMM to record their values when a VM is not running and to restore them when the VM is restarted.

A VMM references the VMCS with a physical, not linear address. This eliminates the need to locate the VMCS in the guest's linear-address space, which can be different from the VMM's linear address space.

VM entries and exits

VM entries load processor state from the guest-state area of the VMCS. A VMM can optionally configure VM entry to follow this loading by injecting an interrupt or exception. The CPU effects this injection using the guest interrupt descriptor table (IDT), just as if the injected event had occurred immediately after VM entry. This feature removes the need for a VMM to emulate delivery of these events. VM exits save processor state into the guest state area and load processor state from the host-state area. All VM exits use a common entry point to the VMM. To simplify the design of a VMM, every VM exit saves into the VMCS detailed information specifying the reason for the exit; many exits also record an exit qualification, which provides further details. For example, if the MOV CR [1] instruction causes a VM exit, the exit reason would indicate "control-register access"; the exit qualification would indicate (1) the identity of the control register (for example, CR0); (2) whether the MOV was to or from the control register; and (3) which general-purpose register was the source or destination of the instruction. Both VM entries and VM exits load CR3 (the base address of the page-table hierarchy). This implies that the VMM and the guest can run in different linear-address spaces.

VT-i architecture overview

VT-i consists of extensions to the Itanium processor hardware and the processor abstraction layer (PAL) firmware.

Processor status bit PSR.vm

The principal hardware extension is the addition of a new bit (vm) in the processor-status register (PSR). A VMM runs with PSR.vm = 0; it runs its guests with PSR.vm = 1. All four privilege levels can be used regardless of the value of PSR.vm; guest software can run at its intended privilege level, and a VMM has the flexibility to use multiple privilege levels. When PSR.vm = 0, processor operation is similar to operation without VT-i. When PSR.vm = 1, all privileged instructions and some nonprivileged instructions, for example, *thash*, cause a new virtualization fault. PSR.vm is cleared to 0 on all interruptions delivered through the IVT; thus, the VMM or PAL handles all interruptions, even those belonging to guest software. The VMM or PAL can set PSR.vm to 1 by using the *rfi* instruction to return to guest software. VT-i adds a new instruction, *vmsw* (virtual machine switch), which modifies the PSR.vm bit with minimum overhead, reducing the latency of transitions between guest software and a VMM in cooperative virtualization environments. PSR.vm also controls the number of virtual address bits available to software. When a VMM is running, PSR.vm = 0, all implemented virtual address bits are available. When a guest is running, PSR.vm = 1, the uppermost implemented virtual-address bit is not available, and an exception occurs if this bit is used. This reserves some dedicated address space for the VMM that guest software cannot access.

IVT vectors

To facilitate efficient handling of transitions to a VMM, VT-i adds two new vectors to the IVT. The VMM uses the virtualization vector to configure the processor to use two of the processor-banked registers to identify the cause of the virtualization fault and the faulting opcode. With the virtual external interrupt vector, the VMM can use a PAL service to register pending virtual interrupts. If the VMM has registered an interrupt and the guest performs an operation that would unmask it, control is transferred to the virtual external interrupt vector.

PAL firmware layer extensions

VT-i includes additions to the PAL firmware layer that provide a consistent programming interface to a VMM even if the hardware is not implemented identically across processor generations. The PAL extensions include a set of new procedures, the addition of PAL services for high-frequency VMM operations, and a virtual processor descriptor (VPD) table.

VT-i defines PAL procedures for setting up and tearing down a virtual machine environment, initializing and terminating virtual processors, and saving and restoring a virtual processor's state. These procedures follow the same calling convention as other PAL procedures.

VT-i introduces a PAL interface for virtualization called a service. To reduce overhead, PAL services use a new calling convention specifically targeted for VMMs. PAL services provide several functions including synchronizing guest shadow registers and the VPD, saving and restoring a subset of a virtual processor's state, and resuming execution of guest software after a transition to the VMM.

Both the PAL firmware and the VMM can access the VPD, which is located in memory. The VPD contains configuration settings for the virtual processor and the subset of the virtual processor's state that influences its execution characteristics.

IV. CONCLUSIONS

Despite the promise of new and existing virtualization usages, many challenges stand in the way of achieving efficient virtualization of today's IA based systems. Creative software techniques such as binary translation and paravirtualization have addressed some of these problems, but the scope of these challenges has meant that these solutions are either highly complex, and potentially less robust, or are incomplete in their ability to run unmodified legacy operating systems.

VT-x and VT-i are the first components of Intel Virtualization Technology, a series of processor and chipset innovations soon to become available in IA-based client and server platforms. VT-x and VT-i offer solutions to the problems inherent in IA-32 and Itanium processor virtualization, enabling simpler, more robust, and more secure VMM software to support a wider range of legacy and future operating systems while maintaining high levels of performance.

V. REFERENCES

[1] J. P. Walters, V. Chaudhary, M. Cha, S. Guercio and S. Gallo, "A Comparison of Virtualization Technologies for HPC," in *22nd International Conference on Advanced Information Networking and Applications*, Gino-wan, Japan, 2008.

[2] R. Uhlig, G. Neiger, D. Rodgers, A. L. Santoni, F. C. M. Martins, A. V. Anderson, S. M. Bennett, A. Kagi, F. H. Leung and L. Smith, "Intel virtualization technology," *Computer,* vol. 38, no. 5, pp. 48-56, 2005.

[3] A. J. Younge, R. Henschel, J. T. Brown, G. von Laszewski, J. Qiu and G. C. Fox, "Analysis of Virtualization Technologies for High Performance Computing Environments," in *IEEE 4th International Conference on Cloud Computing,* Washington, DC, USA, 2011.