Volume 10 Issue 03 Published, August 10, 2006 ISSN 1535-864X DOI: 10.1535/itj.1003

Intel® Technology Journal

Intel® Virtualization Technology

The broad availability of virtualization technology makes possible entirely new applications for virtualization in servers, clients and embedded systems, and provides new ways to improve system reliability, manageability, security and real-time quality of service. This issue of Intel Technology Journal (Volume 10, Issue 3), gives the reader an in-depth view into Intel's plans to support this emerging trend in systems based on Intel® architecture, and the vibrant ecosystem that is forming around it.

Inside you'll find the following articles:

More information, including current and past issues of Intel Technology Journal, can be found at:
http://developer.intel.com/technology/itj/index.htm

Volume 10 Issue 03 Published, August 10, 2006 ISSN 1535-864X DOI: 10.1535/itj.1003

Intel® Technology Journal

Intel® Virtualization Technology

Articles

THIS PAGE INTENTIONALLY LEFT BLANK

Preface

By Lin Chao
Publisher, *Intel Technology Journal*

Virtualization technology has a long history in computer science. It was Christopher Strachey who first published a paper entitled "Time Sharing in Large Fast Computers" in the International Conference on Information Processing at UNESCO, New York, in June 1959, and he commented that his paper, "...was mainly about multi-programming (to avoid waiting for peripherals)." Virtualization today is a methodology whereby the resources of a computer are divided into multiple execution environments, by applying one or more technologies such as hardware and software partitioning, time-sharing, partial or complete machine simulation, emulation, or quality of service. This issue of the *Intel Technology Journal* (Volume 10, Issue 3) reviews virtualization, especially Intel® Virtualization Technology (Intel® VT). These seven technical papers describe the key capabilities of virtualization on Intel's hardware and software platforms and the virtualization roadmap for both Intel® Architecture and Intel® Itanium® processors. Intel VT is part of a collection of premier Intel designed and manufactured silicon technologies that deliver new and improved computing benefits for home and business users, and IT managers.

There are many advantages to virtualization. Virtual machines can be used to run multiple operating systems simultaneously—different versions, or even entirely different systems. These machines can also be used to consolidate the workloads of several under-utilized servers so that fewer machines are used, thereby saving hardware resources and reducing related infrastructure costs. Legacy software can run on virtual machines, whereas an older application might not run on newer hardware and/or operating systems. They can also provide secure, isolated sandboxes for running un-trusted applications, making virtualization an important concept in building secure computing platforms. To the user, they can provide the illusion of hardware, or hardware configuration (such as SCSI devices, multiple processors, etc.) and they can even be used to simulate networks of independent computers. Finally, in this age of heightened awareness regarding security of information, they enable powerful debugging and performance-monitoring scenarios and moreover make software easier to migrate, thus aiding application and system mobility.

The first two papers provide an overview of Intel VT architectures both for hardware and for directed I/O. The first paper provides an overview of Intel VT for the IA-32 Intel® architecture and Intel VT for the Intel® Itanium® architecture. The second paper looks at I/O-device virtualization known as Intel VT for Directed I/O. This paper surveys a variety of established and emerging techniques for I/O virtualization and outlines their associated advantages and challenges. The paper details the architecture of Intel VT for Directed I/O and describes how it enables the industry to meet the future challenges of I/O virtualization.

The third paper looks at Virtual Machine Monitors and Xen*, an open source virtual machine monitor (VMM) developed at the University of Cambridge to support operating systems that have been modified to run on top of the monitor. Intel has extended the Xen VMM to use Intel VT to support unmodified guest operating systems.

The fourth, fifth, and sixth papers look at new and extended usage models. The fourth paper looks at how Intel VT enables a virtualized environment for a host of provisioning manageability and diagnostic applications for the IT professional. One of the usage models involves embedded IT through the use of Intel VT for Client Isolation and Recovery (CIR) that emphasizes isolating manageability and security services. The fifth paper describes the unique requirements that embedded systems and communications infrastructure place on virtualized environments and shows how Intel is working with a number of third parties in the embedded markets. Some features of the embedded applications include maintaining bounded real-time performance; increased system uptime, and easier software migration—without having to bring down the application. The requirement for embedded applications dictates that different architectural and design tradeoffs be made within the VMM and the guest operating systems executing within the virtual machines. The sixth paper presents how an enterprise IT organization sees virtualization in the enterprise and how it can be applied. Enterprise virtualization programs can help to achieve higher server utilization, make it easier to manage data center assets, and reduce the consumption of datacenter resources (floor space, power, etc.), as well as facilitate simpler server releases.

The seventh and final paper looks at performance benchmarking for virtualization. There are no established performance methodologies to measure virtualization performance. The vConsolidate benchmark is presented as an example implementation, highlighting the compromises required in workload selection, component definition, and metric aggregation.

These papers highlight new and innovative virtualization markets, open standards support, and hardware platforms with efficient processor virtualization. Best of all, most PC users today, though they may not know it, are already using virtualization in their off-the-shelf systems that are based on Intel Architecture platforms.

Note: Throughout this journal references to VT-x refer to Intel VT for the IA-32 Intel architecture, VT-i refers to the Intel VT for the Intel Itanium architecture, and VT-d refers to Intel VT for Directed I/O.

Foreword

By Rich Uhlig
Senior Principal Engineer
Intel VT Architect, Corporate Technology Group

For many years, virtualization was confined to proprietary mainframe systems. The ability to run multiple operating systems on the same physical platform was regarded as useful or feasible only in the largest server systems and there it remained—for decades. But the introduction of Intel® Virtualization Technology (Intel® VT), and the recent emergence of new virtualization solutions for Intel-based systems are changing all of that.

We began our work on Intel VT with a simple premise: the ever-increasing performance of platforms based on Intel® Architecture (IA) would overcome the traditional performance barriers to full-system virtualization, and eventually bring the capability to all classes of systems from servers to clients to embedded systems. The ubiquity of the capability, in turn, would spur innovation in new uses for the technology.

However, some barriers stood in the way of turning this opportunity into a reality: traditional IA wasn't designed to allow straightforward sharing of processor or other platform resources, and so software needed to resort to complex work-arounds to make virtualization work. To get over these barriers, we set out to identify the various aspects of IA-based systems that complicate virtualization, and then we extended the hardware architecture to fix them. The result was Intel VT, a multi-generational series of extensions to Intel® processor and platform architecture that provides a new hardware foundation for virtualization, establishing a common infrastructure for all classes of IA-based systems. The broad availability of Intel VT makes possible entirely new applications for virtualization in servers, clients, and embedded systems, providing new ways to improve system reliability, manageability, security, and real-time quality of service.

This issue of the Intel Technology Journal (ITJ), Volume 10, Issue 3, on virtualization begins with a look at Intel's VT hardware roadmap, which is rooted in new support for virtualizing IA-32 and Intel® Itanium® processors, and extends into the platform with new support for I/O device virtualization. You'll find papers that show how Intel VT hardware significantly simplifies the design and implementation of virtualization software, and how it provides a supportive new infrastructure for system and I/O-device vendors to build and extend virtualization solutions based on Intel® platforms.

The success of any new hardware architecture is highly dependent on the system software that puts its new features to use. In the case of virtualization technology, that support comes from the virtual machine monitor (VMM), a layer of software that controls the underlying physical platform resources, sharing them between multiple "guest" operating systems. As the first to market with hardware support for IA-32 virtualization, Intel VT is already incorporated into most commercial and open-source VMMs including those from VMware, Microsoft, XenSource, Parallels, Virtual Iron, Jaluna and TenAsys. Throughout this issue of the ITJ you'll see how Intel VT hardware is being used by VMM developers, from the open-source Xen* VMM, to a lightweight VMM design in support of client manageability, to enhanced embedded real-time system design.

The traditional use of virtualization in mainframe systems has been to simplify the provisioning and management of the physical resources of large server systems. But as virtualization technology becomes widespread in IA-based systems, the computing industry is witnessing a remarkable proliferation of new uses for virtualization, and a reinterpretation of old usages that extends well beyond the traditional mainframe. In this ITJ issue, you'll find many such examples, including a case study of server consolidation in a modern datacenter by Intel's Information Technology Group, improved client manageability and security enabled by Intel® vPro™ technology-based platforms with Intel VT support, and the marriage of real-time system design with virtualization to meet the needs of embedded and communications applications. A common theme throughout these case studies is that different usage models and market segments require different styles of VMM software design—but all share a need for the improved platform hardware support for virtualization.

It's gratifying to see that our original vision for Intel VT of providing broadly available hardware support to enable innovative new applications of virtualization is indeed becoming a reality. I hope that you enjoy this special ITJ issue on virtualization and the in-depth view that it provides into Intel's plans to support this emerging trend in IA-based systems, and the vibrant ecosystem that is forming around Intel VT.

Technical Reviewers

Tom Adelmeyer, Software and Solutions Group
Mark Brown, Software and Solutions Group
Kevin Cassidy, Digital Enterprise Group
Clem Cole, Software and Solutions Group
David Cowperthwaite, Corporate Technology Group
Aaron Holzer, Digital Enterprise Group
Gene Forte, Software and Solutions Group
Arun Krishnaswamy, Digital Enterprise Group
Michael Lewin, Digital Enterprise Group
Ed Lisle, Corporate Technology Group
Tony Luck, Software and Solutions Group
Asit Mallick, Software and Solutions Group
Bill Mello, Digital Enterprise Group
Ashok Mishra, Digital Enterprise Group
Alberto Munoz, Corporate Technology Group
Gil Neiger, Corporate Technology Group
Rolf Neugebauer, Corporate Technology Group
Dev Pillai, Technology Management Group
Yasser Rasheed, Digital Enterprise Group
Greg Regnier, Corporate Technology Group
Ken Rule, Software and Solutions Group
Vivekananthan Sanjeepan, Corporate Technology Group
Rajesh Sankaran, Corporate Technology Group
Ioan Scumpu, Digital Enterprise Group
Rich Uhlig, Corporate Technology Group
Fred Yang, Software and Solutions Group

THIS PAGE INTENTIONALLY LEFT BLANK

Intel® Virtualization Technology: Hardware Support for Efficient Processor Virtualization

Gil Neiger, Corporate Technology Group, Intel Corporation
Amy Santoni, Digital Enterprise Group, Intel Corporation
Felix Leung, Digital Enterprise Group, Intel Corporation
Dion Rodgers, Digital Enterprise Group, Intel Corporation
Rich Uhlig, Corporate Technology Group, Intel Corporation

Index words: virtualization, processor, VT-x, VT-i

ABSTRACT

Virtualizing the physical resources of a computing system to improve sharing and utilization has been done for decades. Virtualization had once been confined to specialized server and mainframe systems, but improvements in the performance of platforms based on Intel® technology now allow those platforms to efficiently support virtualization. However, the IA-32 and Itanium® processor architectures pose a number of significant challenges to virtualization.

The first generation of Intel® Virtualization Technology$^{\Delta}$ (VT) for IA-32 and Itanium processors provides hardware support that simplifies processor virtualization, enabling reductions in virtual machine monitor (VMM) software size and complexity. Resulting VMMs can support a wider range of legacy and future operating systems (OSs) on the same physical platform while maintaining high performance.

In this paper, we provide details of the virtualization challenges posed by IA-32 and Itanium processors; present an overview and furnish details of VT-x (Intel Virtualization Technology for the IA-32 architecture) and VT-i (Intel Virtualization Technology for the Itanium architecture); show how VT-x and VT-i address virtualization challenges; and finally provide examples of usage of the VT-x and VT-i architecture.

INTRODUCTION

Virtualizing the physical resources of a computing system to achieve improved degrees of sharing and utilization is a well-established concept that goes back decades [1]. Full virtualization of all system resources (including processors, memory and I/O devices) makes it possible to run multiple operating systems (OSs) on a single physical platform. In contrast to a non-virtualized system, in which a single OS is solely in control of all hardware platform resources, a virtualized system includes a new layer of software, called a virtual-machine monitor (VMM). The principal role of the VMM is to arbitrate access to the underlying physical host platform resources so that these resources can be shared among multiple OSs that are "guests" of the VMM. The VMM presents to each guest OS a set of virtual platform interfaces that constitute a virtual machine (VM).

Virtualization was once confined to specialized, proprietary, high-end server and mainframe systems. It is now becoming more broadly available and is supported in off-the-shelf IA-based systems—systems based on Intel architecture hardware. This development is due in part to the steady performance improvements of IA-based systems, which mitigate traditional virtualization performance overheads. Other factors include new creative software approaches addressing the difficulties inherent to IA virtualization [2–4] and the emergence of novel applications for virtualization in both industry and academia.

In the sections that follow, we examine some of the technical difficulties with bringing virtualization to IA-based systems and present an overview of Intel Virtualization Technology (VT), which provides hardware assists for overcoming these difficulties. The first generation of VT focuses on a set of hardware assists that facilitates the virtualization of IA processors. VT-x refers to new architectural extensions that aid in IA-32 processor virtualization, while VT-i refers to a set of assists for virtualizing Itanium processors. VT-x and VT-i eliminate many of the problems that make writing a VMM for IA-based systems a challenge and hence make possible the

broader availability of virtualization technology in both server and client systems.

SOFTWARE-ONLY VIRTUALIZATION WITH THE IA-32 AND ITANIUM® ARCHITECTURES

Established and emerging uses provide strong motivation for improving virtualization support in both server and client computing systems. Unfortunately, the IA-32 and Itanium architectures present many challenges to providing such support. Software techniques exist that address some of those challenges.

Challenges to Virtualizing Intel Architectures

Intel microprocessors (both IA-32 and Itanium architecture) provide protection based on the concept of a 2-bit privilege level, using 0 for most-privileged software and 3 for least-privileged. The privilege level determines whether privileged instructions, which control basic CPU functionality, can execute without fault. It also controls address-space accessibility based on the configuration of the processor's page tables and, for IA-32, segment registers. Most IA software uses only privilege levels 0 and 3.

For an OS to control the CPU, some of its components must run with privilege level 0. Because a VMM cannot allow a guest OS such control, a guest OS cannot execute at privilege level 0. Thus, VMMs running on either IA-32 or Itanium processors must use *ring deprivileging*, a technique that runs all guest software at a privilege level greater than 0. A guest OS could be deprivileged in two distinct ways: it could run either at privilege level 1 (the 0/1/3 model) or at privilege level 3 (the 0/3/3 model).

Although the 0/1/3 model supports simpler VMMs, it cannot be used for guests on IA-32 processors in 64-bit mode (more details in "ring compression" section). (64-bit mode is part of Intel® Extended Memory 64 Technology$^{\Phi}$—Intel® EM64T—the 64-bit extensions to IA-32.)

Ring Aliasing

Ring aliasing refers to problems that arise when software is run at a privilege level other than the privilege level for which it was written.

An example in IA-32 involves the CS segment register, which points to the code segment. If the *PUSH* instruction is executed with the CS segment register, the contents of that register (which include the current privilege level) is pushed on the stack. Similarly, the Itanium instruction *br.call* saves the current privilege level into the *ppl* field of the Previous Function State (PFS) register, which can be read at any privilege level. In either case, a guest OS could easily determine that it is not running at privilege level 0.

Address-Space Compression

OSs expect to have access to the processor's full virtual-address space (known as the linear-address space in IA-32). A VMM must reserve for itself some portion of the guest's virtual-address space. It could run entirely within the guest's virtual-address space, which allows it easy access to guest data, but the VMM's instructions and data structures would use a substantial amount of the guest's virtual-address space.

Alternatively, the VMM can run in a separate address space, but even in that case, the VMM must use a minimal amount of the guest's virtual-address space for the control structures that manage transitions between guest software and the VMM. For IA-32, these structures include the interrupt-descriptor table (IDT) and the global-descriptor table (GDT), which reside in the linear-address space. For the Itanium architecture, the structures include the interruption vector table (IVT), which resides in the virtual-address space.

The VMM must prevent guest access to those portions of the guest's virtual-address space that the VMM is using. Otherwise, the VMM's integrity could be compromised (if the guest can write to those portions) or the guest could detect that it is running in a VM (if it can read those portions). Guest attempts to access these portions of the address space must generate transitions to the VMM, which can emulate or otherwise support them. The term *address-space compression* refers to the challenges of protecting these portions of the virtual-address space and supporting guest accesses to them.

Non-Faulting Access to Privileged State

Privilege-based protection prevents unprivileged software from accessing certain components of CPU state. In most cases, attempted accesses result in faults, allowing a VMM to emulate the desired guest instruction. However, the IA-32 and Itanium architectures both include instructions that access privileged state and do not fault when executed with insufficient privilege. For example, the IA-32 registers GDTR, IDTR, LDTR, and TR contain pointers to data structures that control CPU operation. Software can execute the instructions that write to, or load, these registers (*LGDT*, *LIDT*, *LLDT*, and *LTR*) only at privilege level 0. However, software can execute the instructions that read, or store, from these registers (*SGDT*, *SIDT*, *SLDT*, and *STR*) at any privilege level. If the VMM maintains these registers with unexpected values, a guest OS using the latter instructions could determine that it does not have full control of the CPU.

Another example pertains to the page-table address (PTA) register of the Itanium architecture, a field that references

the base address of the virtual hash page table (VHPT). The instruction *mov to cr.PTA* is the normal way to access this register, and software can execute it only at privilege level 0. However, the *thash* instruction indirectly exposes all or part of the VHPT base address, and software can execute it at any privilege level. If the VMM maintains the VHPT at a different address than the guest OS expects, a guest OS using the *thash* instruction could determine that it does not have full control of the CPU.

Adverse Impact on Guest System Calls

Ring deprivileging can interfere with the effectiveness of facilities in the IA-32 architecture that accelerate the delivery and handling of transitions to OS software. The IA-32 *SYSENTER* and *SYSEXIT* instructions support low-latency system calls. *SYSENTER* always effects a transition to privilege level 0, and *SYSEXIT* faults if executed outside that ring. Ring deprivileging thus has the following implications:

- Executions of *SYSENTER* by a guest application cause transitions to the VMM and not to the guest OS. The VMM must emulate every guest execution of *SYSENTER*.
- Executions of *SYSEXIT* by a guest OS cause faults to the VMM. The VMM must emulate every guest execution of *SYSEXIT*.

Interrupt Virtualization

Providing support for external interrupts, especially regarding interrupt masking, presents some specific challenges to VMM design. Both the IA-32 and Itanium architectures provide mechanisms for masking external interrupts thus preventing their delivery when the OS is not ready for them. IA-32 uses the interrupt flag (IF) in the EFLAGS register to control interrupt masking; the Itanium architecture uses the *i* bit in the processor status register (PSR) to provide this function. In both cases, a value of 0 indicates that interrupts are masked.

A VMM will likely manage external interrupts and deny guest software the ability to control interrupt masking. Existing protection mechanisms allow such denial of control by ensuring that guest attempts to control interrupt masking fault in the context of ring deprivileging. Such faulting can cause problems because some OSs frequently mask and unmask interrupts. Intercepting every guest attempt to do so could significantly affect system performance.

Even if it were possible to prevent guest modifications of interrupt masking without intercepting each attempt, challenges would remain when a VMM has a "virtual interrupt" to deliver to a guest. A virtual interrupt should be delivered only when the guest has unmasked interrupts. To deliver virtual interrupts in a timely way, a VMM should intercept some but not all attempts by a guest to modify interrupt masking. Doing so could significantly complicate the design of a VMM.

Access to Hidden State

Some components of IA-32 and Itanium processor state are not represented in any software-accessible register. Examples for IA-32 include the hidden descriptor caches for the segment registers. A segment-register load copies the referenced descriptor (from the GDT or LDT) into this cache, which is not modified if software later writes to the descriptor tables. IA-32 does not provide a mechanism for saving and restoring hidden components of a guest context when changing VMs or for preserving them while the VMM is running.

In the Itanium architecture, there is a field in the Register Stack Engine (RSE) called the current frame load enable (CFLE). There is no direct way to write this value. There are cases where the VMM may take an external interrupt and wants to return to the guest OS with this value equal to zero. The return from interrupt (*rfi*) instruction forces this value to a one.

Ring Compression

Ring deprivileging uses privilege-based mechanisms to protect the VMM from guest software. IA-32 includes two such mechanisms: segment limits and paging. Because segment limits do not apply in 64-bit mode, paging must be used in this mode. Because IA-32 paging does not distinguish privilege levels 0–2, the guest OS must run at privilege level 3 (the 0/3/3 model). Thus, the guest OS runs at the same privilege level as guest applications and is not protected from them. This problem is called *ring compression*.

Frequent Access to Privileged Resources

A VMM may prevent guest access to privileged resources by forcing attempts at such accesses to fault. Even when this ensures correct behavior, performance may be compromised if the frequency of such faults is excessive.

In the IA-32 and Itanium architectures, an example involves the task-priority register (TPR). For the IA-32 architecture, this register is located in the advanced programmable interrupt controller (APIC), and for the Itanium architecture, it is one of the control registers. Because it controls interrupt prioritization, a VMM must not allow a guest OS access to the TPR. However, some OSs perform such accesses with very high frequency. These accesses require VMM intervention only if they cause the TPR to drop below a value determined by the VMM.

The Itanium architecture supports efficient interruption handlers by providing them with information about the interruption and the interrupted context. These data are

recorded, not in memory, but in a set of interruption-control registers. The processor protects system integrity by generating faults in response to accesses to those registers outside privilege level 0. Typically, every interruption handler reads these registers. If each such access generates a fault to the VMM, the performance of these handlers will be severely compromised.

ADDRESSING VIRTUALIZATION CHALLENGES IN SOFTWARE

To address the virtualization challenges that the IA-32 and Itanium architecture present, VMM designers have developed creative techniques for modifying guest software (source or binary). Denali [5] and Xen* [2] are examples of VMMs that use source-level modifications in a technique called *paravirtualization*. Developers of these VMMs modify the source code of a guest OS to create an interface that is easier to virtualize. Paravirtualization offers high performance and does not require changes to guest applications. A disadvantage of paravirtualization is that it limits the range of supported OSs; VMMs that rely on paravirtualization cannot support an OS whose source code the VMM's developers have not modified.

A VMM can support unmodified OSs by transforming guest-OS binaries on-the-fly to handle virtualization-sensitive operations. VMMs that use such binary-translation techniques include those developed by VMware [4] as well as Virtual PC* and Virtual Server* from Microsoft. [3]. Such VMMs support a broader range of OSs than VMMs that use paravirtualization.

A central design goal for Intel VT has been to eliminate the need for CPU paravirtualization and binary translation techniques, to simplify the implementation of robust VMMs that can support a broad range of unmodified guest OSs, and to maintain high levels of performance.

INTEL® VIRTUALIZATION ARCHITECTURE OVERVIEW

In this section, we discuss some of the details of Intel VT architecture. We first describe the VT-x support for IA-32 processor virtualization [6], and then we describe the VT-i support for Itanium processor virtualization [7].

VT-x Architecture Overview

VT-x augments IA-32 with two new forms of CPU operation: *VMX root operation* and *VMX non-root operation*. VMX root operation is intended for use by a VMM, and its behavior is very similar to that of IA-32 without VT-x. VMX non-root operation provides an alternative IA-32 environment controlled by a VMM and designed to support a VM. Both forms of operation support all four privilege levels, allowing guest software to run at its intended privilege level, and providing a VMM with the flexibility to use multiple privilege levels.

VT-x defines two new transitions: a transition from VMX root operation to VMX non-root operation is called a *VM entry*, and a transition from VMX non-root operation to VMX root operation is called a *VM exit*. VM entries and VM exits are managed by a new data structure called the virtual-machine control structure (*VMCS*). The VMCS includes a *guest-state area* and a *host-state area*, each of which contains 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 operation is changed substantially in VMX non-root operation. The most important change is that many instructions and events cause VM exits. Some instructions (e.g., INVD) cause VM exits unconditionally and thus can never be executed in VMX non-root operation. Other instructions (e.g., INVLPG) and all events can be configured to do so conditionally using *VM-execution control fields* in the VMCS.

Guest-State Area

The guest-state area of the VMCS is used to contain elements of the state of virtual CPU associated with that VMCS.

For proper VMM operation, certain registers must be loaded by every VM exit. These include those IA-32 registers that manage operation of the processor, such as the segment registers (to map from logical to linear addresses), CR3 (to map from linear to physical addresses), IDTR (for event delivery), and many others. The guest-state area contains fields for these registers so that their values can be saved as part of each VM exit.

In addition, the guest-state area contains fields corresponding to elements of processor state that are not held in any software-accessible register. One of these elements is the processor's *interruptibility state*, which indicates whether external interrupts are temporarily masked (e.g., due to execution of the MOV-SS instruction) and whether non-maskable interrupts (NMIs) are masked because software is handling an earlier NMI.

The guest-state area does not contain fields corresponding to registers that can be saved and loaded by the VMM itself (e.g., the general-purpose registers). Exclusion of such registers improves the performance of VM entries and VM exits. Software can manage these additional registers more efficiently as it knows better than the CPU when they need to be saved and loaded.

VM-Execution Control Fields

The VMCS contains a number of fields that control VMX non-root operation by specifying the instructions and events that cause VM exits. In this section, we present some of these controls.

The VMCS includes controls that support interrupt virtualization:

- *External-interrupt exiting*. When this control is set, all external interrupts cause VM exits; in addition, the guest is not able to mask these interrupts (e.g., interrupts are not masked if EFLAGS.IF=0).
- *Interrupt-window exiting*. When this control is set, a VM exit occurs whenever guest software is ready to receive interrupts (e.g., when EFLAGS.IF=1).
- *Use TPR shadow*. When this control is set, accesses to the APIC's TPR through control register CR8 (available only in 64-bit mode) are handled in a special way: executions of MOV CR8 access a *TPR shadow* referenced by a pointer in the VMCS. The VMCS also includes a *TPR threshold*; a VM exit occurs after any instruction that reduces the TPR shadow below the TPR threshold.

There are also VM-execution control fields that support efficient virtualization of the IA-32 control registers CR0 and CR4. These registers each comprise a set of bits controlling processor operation. A VMM may wish to retain control of some of these bits (e.g., those that manage paging) but not others (e.g., those that control floating-point instructions). The VMCS includes, for each of these registers, a *guest/host mask* that a VMM can use to indicate which bits it wants to protect. Guest writes can freely modify the unmasked bits, but an attempt to modify a masked bit causes a VM exit. The VMCS also includes, for each of these registers, a *read shadow* whose value is returned to guest reads of the register.

To support VMM flexibility, the VMCS includes bitmaps that allow a VMM selectivity regarding the causes of some VM exits. The following items detail three of these:

- *Exception bitmap*: This field contains 32 entries for the IA-32 exceptions. It allows a VMM to specify which exceptions should cause VM exits and which should not. For page faults, further selectivity is supported based on a fault's error code.
- *I/O bitmaps*: These bitmaps contain one entry for each port in the 16-bit I/O space. An I/O instruction (e.g., IN) causes a VM exit if it attempts to access a port whose entry is set in the I/O bitmaps.
- *MSR bitmaps*: These bitmaps contain two entries (one for read, one for write) for each model-specific register (*MSR*) currently in use. An execution of RDMSR (or WRMSR) causes a VM exit if it attempts to read (or write) an MSR whose read bit (or write bit) is set in the MSR bitmaps.

In addition to the controls mentioned above, there are VM-execution controls that support flexible VM exiting for a number of privileged instructions.

VMCS Details

Like the IA-32 page tables, each VMCS is referenced with a physical (not linear) address. This eliminates the need to locate the VMCS in the guest's linear-address space (which, as noted below, may be different from that of the VMM). The format and layout of the VMCS in memory is not architecturally defined, allowing implementation-specific optimizations to improve performance in VMX non-root operation and to reduce the latency of VM entries and VM exits. VT-x defines a set of new instructions that allows software to access the VMCS in an implementation-independent manner.

Details of VM Entries and VM Exits

As noted earlier, VM entries load processor state from the guest-state area of the VMCS. (Note that, because the state loaded includes CR3, the guest may run in a different linear-address space than the VMM.) In addition to loading guest state, VM entry can be optionally configured for *event injection*. The CPU effects this injection using the guest IDT to deliver an event (exception or interrupt) specified by the VMM, just as if it had actually occurred immediately after VM entry. This feature removes the need for a VMM to emulate delivery of these events.

As noted above, VM exits save processor state into the guest-state area and then load processor state from the host-state area. (Again, because the state loaded includes CR3, the VMM may run in a different linear-address space than the guest.) This implies that all VM exits use a common entry point in the VMM. To simplify the design of a VMM, VT-x specifies that each VM exit save into the VMCS detailed information on the cause of the VM exit. Every VM exit records an exit reason (specifying, for example, which instruction caused the VM exit); many also record an exit qualification, which provides further details. For example, if a VM exit is caused by the MOV CR instruction, the exit reason would indicate "control-register access" and the exit qualification would identify the following: (1) the specific control register (e.g., CR0); (2) whether the MOV was to or from the register; and (3) which other register was the source or destination of the instruction.

Each VM exit due to an IA-32 exception saves, in addition to information about the exception, information about any event (e.g., an external interrupt) that was being

delivered at the time the exception occurred. This allows a VMM to virtualize nested exceptions properly.

VT-i Architecture Overview

VT-i expands the Itanium architecture with extensions to the processor hardware and the Processor Abstraction Layer (*PAL*) firmware.

VT-i adds a new PSR bit (PSR.vm) that allows guest OSs to be run at the privilege level for which they were designed and creates *interceptions* to a VMM necessary for the creation of a complete VM. The VMM runs with this bit equal to zero and runs guest software with this bit equal to one.

The PSR.vm bit modifies the behavior of all privileged instructions as well as that of some non-privileged instructions that access state that a VMM may want to control (including the *thash*, *ttag*, and *mov cpuid* instructions). When a guest OS executes one of these instructions a virtualization intercept is caused which transfers control to the VMM with the PSR.vm bit set to zero.

PSR.vm is orthogonal to the privilege level. This fact allows guest software to run at its designated privilege level; if desired, a VMM can span multiple privilege levels.

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 unimplemented data/instruction address faults or unimplemented instruction address traps are created if this bit is used. This provides a VMM a dedicated address space that guest software cannot access.

VT-i also includes a number of additions to the PAL firmware layer. These additions provide a consistent programming interface to a VMM even if the hardware is not implemented identically across processor generations. These 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.

The PAL procedures are used for setting up and tearing down a VM environment; for setting global VMM configuration options; for initializing and terminating virtual processors; and for saving and restoring a subset of state of a virtual processor. These procedures follow the same calling convention as existing PAL procedures. In addition, a new PAL interface called a PAL service has been introduced for virtualization. PAL services reduce overhead through use of a new calling convention specifically targeted for use by a VMM. *PAL services* provide functionality to synchronize guest hardware registers and the VPD; to save and restore a subset of the state of a virtual processor; to resume execution of the guest software after a virtualization intercept; to calculate guest VHPT hashes and tags; and to set up pending interrupts for the guest.

The VPD table is located in memory selected by the VMM. It is usually located in the VMM's virtual-address space and is accessed by both the PAL firmware and the VMM. The VPD contains configuration settings for the virtual processor and a subset of the virtual processor's state that influences its execution characteristics. For example, the virtual processor's control-register values are located in the VPD but not its general registers. The layout of the VPD is architected to be 64K in size and includes reserved space for future usage.

The VPD contains two configuration fields that allow the VMM to customize the virtualization environment:

- *Virtualization-acceleration field*. This field allows the VMM to customize the virtualization of a particular resource or instruction, leading to a reduction in the number of virtualization intercepts that the VMM has to handle. It provides accelerations for external-interrupt handling as well as intercept control for reads and writes to interruption control registers (cr16-cr25), reads of the PSR, reads of CPUID, the *cover* instruction, and the bank-switch instruction (*bsw*).

  For example, a VMM could enable the bank-switch optimization. Guest execution of bsw would use values that the VMM had set up in the VPD for the guest OS and would never cause a virtualization intercept to the VMM.

- *Virtualization-disable field*. This field allows the VMM to disable virtualization of a particular resource or instruction, leading to a reduction in the number of virtualization intercepts the VMM handles. This field provides disables for virtualization of the external interrupt control registers (cr65–71), the performance monitoring registers, the debug registers, the PSR.i bit, and the interval timer match register.

To provide efficient handling of virtualization intercepts for a VMM, the architecture has added two new vectors into the IVT:

- *Virtualization vector*. This vector is used for all virtualization-related intercepts. To reduce decoding complexity, a VMM can configure the processor to provide the cause of the virtualization intercept (a bitmap field of intercepting instructions) as well as the faulting opcode in two of the processor banked registers. A VMM can relocate this handler to a

memory location outside the IVT as well through a PAL interface.

- *Virtual external interrupt vector*. The processor uses this vector when the guest unmasks a pending external interrupt. It would be used when the VMM has a virtual interrupt for the guest that it cannot deliver due to guest masking. When the guest performs an operation to unmask the highest pending interrupt, the guest state is updated and control is transferred to this new vector. This streamlines delivery of guest external interrupts for the VMM.

VT-i also provides global configuration options that a VMM can set that apply to all virtual processors activated by the VMM. These global configuration options determine whether the cause of a virtualization intercept is provided, if the opcode of the instruction causing the virtualization intercept is provided, if the performance counters are frozen for all virtualization intercepts, and the byte order (or endianness) of the date located in the VPD.

VT-i also includes the *vmsw* instruction. This instruction transitions the PSR.vm bit with minimum overhead. This can reduce transition overhead between guest software and a VMM in cooperative virtualization environments.

SOLVING VIRTUALIZATION CHALLENGES WITH VT-X AND VT-I

VT-x and VT-i allow guest software to run at its intended privilege level. Guest software is constrained, not by privilege level, but because for VT-x it runs in VMX non-root operation or for VT-i with PSR.vm = 1. These facts allow VMMs to avoid the virtualization challenges identified earlier.

Address-Space Compression

VT-x and VT-i provide two different techniques for solving address-space compression problems.

With VT-x, every transition between guest software and the VMM can change the linear-address space, allowing guest software full use of its own address space. The VMX transitions are managed by the VMCS, which resides in the physical-address space, not the linear-address space.

With VT-i, the VMM has a virtual-address bit that guest software cannot use. A VMM can conceal hardware support for this bit by intercepting guest calls to the PAL procedure that reports the number of implemented virtual-address bits. As a result, the guest will not expect to use this uppermost bit, and hardware will not allow it to do so, thus providing the VMM exclusive use of half of the virtual-address space.

Ring Aliasing and Ring Compression

VT-x and VT-i allow a VMM to run guest software at its intended privilege level. This fact eliminates ring aliasing problems because instructions such as *PUSH* (of CS) and *br.call* cannot reveal that software is running in a VM. It also eliminates ring compression problems that arise when a guest OS executes at the same privilege level as guest applications.

Nonfaulting Access to Privileged State

VT-x and VT-i avoid the problem of providing nonfaulting access to privileged state in two ways: by adding support that causes such accesses to transition to a VMM and by adding support that causes the state to become unimportant to a VMM.

A VMM based on VT-x does not require control of the guest privilege level, and the VMCS controls the disposition of interrupts and exceptions. Thus, it can allow its guest access to the GDT, IDT, LDT, and TSS. VT-x allows guest software running at privilege level 0 to use the instructions LGDT, LIDT, LLDT, LTR, SGDT, SIDT, SLDT, and STR.

With VT-i, the *thash* instruction causes virtualization faults, giving a VMM the opportunity to conceal any modifications it may have made to the VHPT base address.

Guest System Calls

Problems occur with the IA-32 instructions SYSENTER and SYSEXIT when a guest OS runs outside privilege level 0. With VT-x, a guest OS can run at privilege level 0, which eliminates problems associated with guest transitions.

Interrupt Virtualization

VT-x and VT-i both provide explicit support for interrupt virtualization.

VT-x includes an external-interrupt exiting VM-execution control. When this control is set to 1, a VMM prevents guest control of interrupt masking without gaining control of every guest attempt to modify EFLAGS.IF. Similarly, VT-i includes a virtualization-acceleration field that prevents guest software from affecting interrupt masking and avoids making transitions to the VMM on every access to the PSR.i bit.

VT-x also includes an interrupt-window exiting VM-execution control. When this control is set to 1, a VM exit occurs whenever guest software is ready to receive interrupts. A VMM can set this control when it has a virtual interrupt to deliver to a guest. Similarly, VT-i includes a PAL service that a VMM can use to register the

vector of the pending virtual interrupt. When guest software executes instructions to unmask the pending interrupt, control is transferred to the VMM via the new virtual external interrupt vector.

Access to Hidden State

VT-x and VT-i use different techniques to allow a VMM to manipulate components of guest state that are not represented in any software-accessible register.

VT-x includes, in the guest-state area of the VMCS, fields corresponding to CPU state not represented in any software-accessible register. The processor loads values from these VMCS fields on every VM entry and saves into them on every VM exit. This provides the support necessary for preserving this state while the VMM is running or when changing VMs.

VT-i provides a way for the VMM to set the RSE CFLE bit to the desired value via an argument value in the PAL service used to return to guest interruption handlers.

Frequent Access to Privileged Resources

VT-x and VT-i allow a VMM to avoid the overhead of high-frequency guest accesses to the TPR register. A VMM can configure the VMCS (for VT-x) or use an acceleration (for VT-i) so that the VMM is invoked only when required: For VT-x this occurs when the value of the TPR shadow associated with the VMCS drops below that of a TPR threshold in the VMCS. For VT-i this occurs only when the writing of the TPR unmasks a virtual pending external interrupt for the guest.

With VT-i, a VMM can use the virtualization-acceleration field in the VPD to indicate that guest software can read or write the interruption-control registers without invoking the VMM on each access. The VMM can establish the values of these registers before any virtual interruption is delivered and can revise them before the guest interruption handler returns.

USAGE OF THE INTEL VIRTUALIZATION ARCHITECTURE

We have described the basic architecture for VT-x and VT-i, and in the next section, we provide some usage examples of the architecture by a VMM. This is intended to highlight some usage models, but it is not a comprehensive set of all usage models.

VMM Usage of VT-x Architecture Features

Exception Handling

VT-x allows a VMM to configure any IA-32 exception to cause a VM exit based on its vector (for page faults, further selectivity is supported based on a fault's error code). When handling such VM exits, a VMM has access to complete information about the exception, including its error code and any other fault-specific information (e.g., the faulting linear address for a page fault).

The VMM may determine that the exception causing the VM exit should be handled by the guest OS. In these cases, the VMM can perform a VM entry to guest using event injection to deliver the exception.

Alternatively, a VMM may respond to such a VM exit by eliminating the cause of the exception (e.g., by modifying the page tables to mark present a page that had not been present). In these cases, the VMM can then perform a VM entry to the guest, which will resume execution at the point at which the exception occurred. If the VM exit was due to a nested fault, the VMM can use event injection to deliver to the guest that event whose delivery encountered that nested fault.

Interrupt Virtualization

When a VMM has an interrupt to deliver to a guest OS, it can do so using event injection with the next VM entry. If guest software is not ready for an interrupt (e.g., because EFLAGS.IF = 0), the VMM can instead re-enter the guest having set the interrupt-window exiting VM-execution control. A VM exit will occur the next time the guest is ready for an interrupt. A VMM can then use event injection as part of the next VM entry.

Lazy Floating-Point State Processing

The IA-32 architecture includes features by which an OS can avoid the time-consuming restoring the floating-point state when activating a user process that does not use the floating-point unit. It does this by setting the TS bit in control register CR0. If a user process then tries to use the floating-point unit, a device-not-available fault (exception 7 = #NM) occurs. The OS can respond to this by restoring the floating-point state and by clearing CR0.TS, which prevents the fault from recurring.

VT-x includes features by which a VMM can process floating-point state lazily, even when supporting a guest OS that does so also. We outline how this may be done.

Before entering a guest whose floating-point state has not been restored, a VMM can do the following:

- Set the TS bit in the CR0 field in the guest-state area; this ensures that any guest floating-point access causes a #NM.
- Set bit 7 (corresponding to #NM) in the exception bitmap; this ensures that any #NM causes a VM exit.
- Set the TS bit in the CR0 guest/host mask; this ensures that any guest attempt to modify CR0.TS causes a VM exit.

- Set the TS bit in the CR0 read shadow to the value expected by guest software (determined on VM exits caused by guest attempts to modify CR0.TS).

In response to a VM exit caused by a #NM, a VMM can check the value of the TS bit in the CR0 read shadow. If it is set, the guest would have incurred its own #NM; the VMM can use event injection to deliver it to the guest. Otherwise, the VMM can do the following:

- Restore the guest's floating-point state.
- Set the TS bit in the CR0 field in the guest-state area to the value expected by guest software.
- Clear bit 7 in the exception bitmap; this ensures that the guest OS will handle any subsequent #NM.
- Clear the TS bit in the CR0 guest/host mask; this allows the guest to modify CR0.TS freely.

VMM Usage of VT-i Architecture Features

Instruction Emulation

The VMM virtualization intercept handler is responsible for emulating certain instructions for a guest OS including side effects of successful emulation. One example of instruction emulation is the `MOV-from-PTA` instruction. The VMM emulates this instruction by placing the guest PTA value in the target register of the instruction. Since the VMM has successfully implemented the `MOV-from-PTA` instruction, it needs to implement the side effects of the instruction execution required by the Itanium architecture. In this example the VMM must also update the value in the cr.iipa register, which records the last successfully executed instruction with PSR.ic equal to 1.

Virtualization Configuration

VT-i is capable of providing a virtualization intercept on every access to privileged resources that may be required or desired for certain VMM implementations. VT-i also provides a way for a VMM to specify virtualization policies on certain resources in advance such that interceptions to the VMM can be reduced for high frequency operations. This functionality is provided through virtualization-accelerations, virtualization-disables, and new synchronization services. One example is the interruption control register reads. Guest OS interruption handlers read interruption control registers frequently and cause a lot of interceptions into the VMM. The *interruption control register read acceleration* allows VMM software to provide preset values for all interruption control registers in the VPD and invoke the PAL write synchronization service before returning to a guest handler. When this acceleration is enabled, guest reads of the interruption control registers are not intercepted to the VMM; instead the value preset by the VMM is returned to the guest. Similarly, the *interruption control register write acceleration* allows the guest to write to interruption control registers without VMM interceptions. VMM can invoke the PAL read synchronization service to obtain the latest values written by the guest and perform any virtualization functions required before emulating the return from interrupt (*rfi*) instruction of the guest handler. All other accelerations and disables in VT-i have the same goal—to allow the VMM to specify the virtualization policies of the privileged resources ahead of time such that guest instructions can execute without interceptions to the VMM.

External and PAL-Based Interruption Handling

In addition to implementing policies to virtualize accesses to privileged resources on the processor, VMM software also needs to virtualize external interruptions as well as accesses to platform resources that are considered privileged. For example, VMM software will continue to handle external interruptions or PAL-based interruptions even if the guest OS had masked these interruptions.

VMM software delivers guest external interrupts only when they are unmasked. When unmasked, the VMM delivers the interruption to the guest handler required by the architecture. For example, the VMM needs to set up the values of the guest interruption control registers, PSR fields, and register stack engine (RSE) state. Since some of the RSE state is not accessible by VMM software, VT-i provides PAL service to allow VMMs to invoke guest handlers correctly.

VMM software registers the corresponding handlers for PAL-based interruptions (e.g., initialization and machine check events) and provides the virtualization policies for these events. VT-i makes no changes to the handling of PAL-based interruptions. The handling and propagation of these events from the VMM to the guest OS is VMM design specific.

FUTURE OF INTEL VIRTUALIZATION ARCHITECTURE

The following features are anticipated for future processors supporting VT-x:

- *NMI-window exiting*. The interrupt-window exiting VM-execution control (described earlier) causes a VM exit when a guest is ready for maskable external interrupts, allowing a VMM to deliver such interrupts in a timely way. NMI-window exiting provides corresponding support for non-maskable interrupts (NMIs), which are blocked by other conditions than those that block maskable external interrupts.

- *Virtual-processor identifiers (VPIDs)*. This feature allows a VMM to assign a different non-zero VPID to each virtual processor (the zero VPID is reserved for the VMM). The CPU can use VPIDs to tag translations in the TLBs. This feature eliminates the need for TLB flushes on every VM entry and VM exit and eliminates the adverse impact of those flushes on performance.
- *Extended page tables (EPT)*. When this feature is active, the ordinary IA-32 page tables (referenced by control register CR3) translate from linear addresses to *guest-physical* addresses. A separate set of page tables (the *EPT tables*) translate form guest-physical addresses to the *host-physical* addresses that are used to access memory. As a result, guest software can be allowed to modify its own IA-32 page tables and directly handle page faults. This allows a VMM to avoid the VM exits associated with page-table virtualization, which are a major source of virtualization overhead without EPT.

CONCLUSION

While the use of virtualization was once confined to proprietary server and mainframe computing systems, established and emerging applications for virtualization in both server and client systems are moving it into the mainstream. 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.

VT-x and VT-i are the first components of Intel VT, a series of processor 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 and thus enable the development of simpler VMM software that supports a wider range of legacy and future OS's while maintaining high levels of performance.

ACKNOWLEDGMENTS

The authors thank the following for their contributions to the development of the VT-x and VT-i architectures: Andrew V. Anderson, Steven M. Bennett, Jason Brandt, Stephen Fischer, Gideon Gerzon, Gary Hammond, Stalinselvaraj Jeyasingh, Alain Kägi, Mike Kozuch, Tariq Masood, Sanjoy Mondal, Rajesh Parthasarathy, Rajesh Sankaran, Sebastian Schönberg, and Larry Smith. We also thank Fernando C. M. Martins for his many contributions to the development of this paper.

REFERENCES

[1] R.P. Goldberg, "Survey of Virtual Machine Research," *Computer*, June 1974, pp. 34–45.

[2] P. Barham et al., "Xen and the Art of Virtualization," in *Proceedings 19th ACM Symp. Operating Systems Principles*, ACM Press, 2003, pp. 164–177.

[3] Microsoft Corp., "Microsoft Virtual Server 2005 Technical Overview," 2004,; at http://download.microsoft.com/download/5/5/3/55321426-cb43-4672-9123-74ca3af6911d/VS2005TechWP.doc*

[4] C.A. Waldspurger, "Memory Resource Management in VMware ESX Server," in *Proceedings 5th Symp. Operating Systems Design and Implementation*, The Usenix Association, 2002, pp. 181–194.

[5] A. Whitaker, M. Shaw, and S. Gribble, "Scale and Performance in the Denali Isolation Kernel," in *Proceedings 5th Symp. Operating Systems Design and Implementation*, The Usenix Association, 2002, pp. 195–210.

[6] Intel Corp., "IA-32 Intel® Architecture Software Developer's Manuals;" at http://www.intel.com/design/pentium4/manuals/index_new.htm

[7] Intel Corp., "Intel® Itanium® Architecture Software Developer's Manual-Volume 2: System Architecture, Revision 2.2," document number 1805, 2006; at ftp://download.intel.com/design/Itanium/manuals/24531805.pdf

AUTHORS' BIOGRAPHIES

Gil Neiger is a principal engineer in Intel's Corporate Technology Group and leads development of the VT-x architecture. He received his Ph.D. degree in Computer Science from Cornell University.

Amy Santoni is a principal engineer in Intel's Digital Enterprise Group and leads the Itanium Architecture Team. She is one of the principal architects of VT-i. Amy has been with Intel for 13 years working in design, validation, firmware, and architecture positions. She received her B.S. degree in Computer Engineering from the University of Michigan.

Felix Leung is a staff engineer in Intel's Digital Enterprise Group and is responsible for the definition and development of VT-i architecture. He has been with Intel for 11 years and has held verification, design, and architecture positions in various IA-32 and Itanium processor design projects. He received his B.S. degree in Computer Science from the University of Wisconsin-Madison.

Dion Rodgers is a senior principal engineer in Intel's Digital Enterprise Group and is responsible for bringing multiple advanced technology initiatives such as VT-x to the IA-32 product line. He received his M.S. degree in Computer Engineering from Clemson University.

Rich Uhlig is a senior principal engineer in Intel's Corporate Technology Group and leads various aspects of Intel's overall virtualization effort including architecture definition, research prototyping, performance analysis, and software usage. He received his Ph.D. degree in Computer Science and Engineering from the University of Michigan.

[Δ] Intel® Virtualization Technology requires a computer system with an enabled Intel® processor, BIOS, virtual machine monitor (VMM) and, for some uses, certain platform software enabled for it. Functionality, performance or other benefits will vary depending on hardware and software configurations and may require a BIOS update. Software applications may not be compatible with all operating systems. Please check with your application vendor.

[Φ] Intel® EM64T requires a computer system with a processor, chipset, BIOS, operating system, device drivers and applications enabled for Intel EM64T. Processor will not operate (including 32-bit operation) without an Intel EM64T-enabled BIOS. Performance will vary depending on your hardware and software configurations. See www.intel.com/info/em64t for more information including details on which processors support Intel EM64T or consult with your system vendor for more information.

THIS PAGE INTENTIONALLY LEFT BLANK

Intel® Virtualization Technology for Directed I/O

Darren Abramson, Mobility Group, Intel Corporation
Jeff Jackson, Corporate Technology Group, Intel Corporation
Sridhar Muthrasanallur, Digital Enterprise Group, Intel Corporation
Gil Neiger, Corporate Technology Group, Intel Corporation
Greg Regnier, Corporate Technology Group, Intel Corporation
Rajesh Sankaran, Corporate Technology Group, Intel Corporation
Ioannis Schoinas, Corporate Technology Group, Intel Corporation
Rich Uhlig, Corporate Technology Group, Intel Corporation
Balaji Vembu, Digital Enterprise Group, Intel Corporation
John Wiegert, Corporate Technology Group, Intel Corporation

Index words: Virtualization, I/O, VMM, DMA, Interrupts

ABSTRACT

Intel® Virtualization Technology$^{\Delta}$ for Directed I/O (VT-d) is the next important step toward comprehensive hardware support for the virtualization of Intel® platforms. VT-d extends Intel's Virtualization Technology (VT) roadmap from existing support for IA-32 (VT-x) [1] and Itanium® processor (VT-i) [2] virtualization to include new support for I/O-device virtualization. This paper surveys a variety of established and emerging techniques for I/O virtualization and outlines their associated problems and challenges. We then detail the architecture of VT-d and describe how it enables the industry to meet the future challenges of I/O virtualization.

INTRODUCTION

There are a number of existing and emerging usage models where support for I/O virtualization is, or will become, increasingly important. Performance, scalability, cost, trust, reliability, and availability are all important considerations, and their relative importance can vary depending upon usage models and the market segment in which they are deployed.

There are two key requirements that are common across market segments and usage models. The first requirement is protected access to I/O resources from a given virtual machine (VM), such that it cannot interfere with the operation of another VM on the same platform. This isolation between VMs is essential for achieving availability, reliability, and trust. The second major requirement is the ability to share I/O resources among multiple VMs. In many cases, it is not practical or cost-effective to replicate I/O resources (such as storage or network controllers) for each VM on a given platform.

First we consider the importance of I/O virtualization in the data center. Many server applications are I/O intensive, especially for networking and storage. Key requirements within the data center include scalability and performance to enable server consolidation. Reliability and availability are important as mission-critical applications move onto virtualized data center servers and infrastructures.

In the case of server consolidation, virtualization is used to deploy multiple VMs (each containing an operating system (OS) and associated services and applications) onto a single server. This consolidation is done primarily to utilize the underlying server hardware more effectively. Many server applications require a significant amount of I/O performance, and so it follows that the consolidation of multiple server applications will need a scalable and high-performance solution for I/O virtualization. The scalability requirement comes from the fact that the total network and storage I/O required from a given server platform is the aggregate of the I/O requirements of the multiple consolidated applications. I/O performance is needed by each VM to satisfy a wide range of server applications with varied and demanding I/O performance requirements.

Next we look at the importance of I/O virtualization in client platforms. For most client platforms, I/O scalability and performance are relatively modest as compared to

servers, but tend to be more sensitive to cost and trust issues.

In the case of the enterprise client, virtualization can be used to create a self-contained operating environment, or "virtual appliance," that is dedicated to capabilities such as manageability or security. These capabilities generally need protected and secure access to a network device to communicate with down-the-wire management agents and to monitor network traffic for security threats. For example, a security agent within a VM requires protected access to the actual network controller hardware. This agent can then intelligently examine network traffic for malicious payloads or suspected intrusion attempts before the network packets are passed to the guest OS, where user applications might be affected.

This virtual-appliance model can be applied beyond the enterprise client. Workstations and home computers can use this technique for management, security, content protection, and a wide variety of other dedicated services. The type of service deployed may dictate that various types of I/O resources, graphics, network, and storage devices, be isolated from the OS where the user's applications are running.

In this paper we survey a variety of existing and emerging techniques for addressing the above requirements of I/O virtualization. We begin in the next section by studying different options for Virtual Machine Monitor (VMM) structuring and software architecture, and then we discuss various techniques for sharing I/O resources among multiple guest OSs. Our survey highlights various challenges faced by today's I/O-virtualization techniques, and it underscores the need for new forms of hardware support to facilitate I/O-resource assignment, protection, and sharing. We then detail the architecture of Intel's VT-d and explain how it helps to establish a new platform infrastructure for addressing the challenges of I/O virtualization in future platforms based on Intel® technology.

VMM SOFTWARE ARCHITECTURE OPTIONS

As background, we identify and compare three distinct types of virtualization layer (or VMM) software architectures in this section (see Figure 1):

- OS-hosted VMMs
- Stand-alone hypervisor VMMs
- Hybrid VMMs

Each of these styles of VMM software architecture has its pros and cons, and the choice often depends on the particular requirements of a given usage model or market segment.

OS-Hosted VMMs

One approach to VMM software architecture is to build on the infrastructure of an existing OS [3] [15]. Such *OS-hosted VMMs* consist of a privileged ring-0 component (shown as the "VMM kernel" in Figure 1) that runs alongside the kernel of the hosting OS, and that obtains control of system resources–such as CPUs and system memory – to create an execution environment for one or more guest OSs. The VMM kernel context switches between host-OS and guest-OS state at periodic intervals as dictated by scheduling policy, or whenever host-OS support is required (e.g., to service hardware interrupts from a physical I/O device that is programmed by a host-OS device driver). Although the guest OS is allowed to directly execute on a physical CPU and to directly access certain portions of host physical memory subject to the control of the VMM kernel, any accesses to I/O devices are typically intercepted by the VMM kernel and proxied to a second, user-level component of the VMM (shown in Figure 1 as a *User-Level Monitor* or *ULM*). The ULM runs as an ordinary process of the host OS, and it contains virtual I/O-device models that service I/O requests from guest OSs. Device models in the ULM call the facilities of the underlying host OS via its file system and networking and graphics APIs to handle I/O requests from guest OSs.

An OS-hosted VMM architecture offers several advantages: the VMM can leverage any I/O device drivers that have been developed for the hosting OS, which can significantly ease porting of the VMM to a range of different physical host platforms. Further, the VMM can leverage other facilities of the host OS, such as code for scanning I/O busses, to perform I/O resource discovery and to manage host platform power-management functions.

A disadvantage of an OS-hosted VMM is that it is only as reliable, available, and secure as the host OS upon which it depends: If the host OS fails or must be rebooted (e.g., to install a software security patch), then all other guest OSs must be taken out of service as well. An OS-hosted VMM is also subject to the CPU scheduling policies of the host OS, which serves not only the VMM and its guest OSs, but also other applications running above the host OS. Depending on the security, availability, or real-time quality-of-service requirements of a given usage model, these disadvantages may not be acceptable, and alternative VMM software architectures may be warranted.

Stand-Alone Hypervisor VMMs

One such alternative approach is to structure the VMM as a stand-alone *hypervisor* that does not depend on a hosting

OS [4, 10, 11]. A hypervisor-style VMM incorporates its own I/O device drivers, device models, and scheduler.

A hypervisor-style VMM can fully control provisioning of physical platform resources, enabling it to provide scheduling and quality-of-service guarantees to its guest OSs. An additional advantage of a hypervisor-based VMM is that the code paths from guest OSs requests for I/O services to the actual physical I/O device drivers are typically shorter than in an OS-hosted VMM, which requires I/O requests to traverse two I/O stacks, first that of the guest OS, and then that of the host OS. Further, by controlling and limiting the size of the hypervisor kernel, the VMM can provide enhanced security and reliability through a smaller trusted computing base (TCB) [5, 9].

The advantages of a hypervisor-style VMM come at the expense of limited portability, because the necessary I/O-device drivers for any given physical platform must be developed to run within the hypervisor. More advanced system functions, such as ACPI-based system power management–which are inherited from the host OS in a hosted VMM–must also be reimplemented in a hypervisor-based VMM. While not as complex as a full modern OS, a mature hypervisor-based VMM can grow to a significant size over time, gradually compromising some of the benefits noted earlier (e.g., improved security through limiting the size of the TCB).

Hybrid VMMs

In an effort to retain some of the security and reliability benefits of hypervisor-style VMM architecture, while at the same time leveraging the facilities of an existing OS and its associated device drivers as in an OS-hosted VMM, some VMMs adopt a *hybrid* approach [6, 7, 9].

In a hybrid VMM architecture, a small hypervisor kernel (shown in Figure 1 as a *µ-hypervisor*) controls CPU and memory resources, but I/O resources are programmed by device drivers that run in a deprivileged *service OS*. The service OS functions in a manner similar to that of a host OS in that the VMM is able to leverage its existing device drivers. However, because the service OS is deprivileged by the µ-hypervisor, and because it operates solely on behalf of the VMM (i.e., it does not support other, arbitrary user applications), it is possible to improve the overall security and reliability of the system.

While a hybrid VMM architecture offers the promise of retaining the best characteristics of hosted- and hypervisor-style VMMs, it does introduce new challenges, including new performance overheads, due to frequent privilege-level transitions between guest OS and service OS through the µ-hypervisor. Further, the full benefits of deprivileging a service OS are only possible with new hardware support for controlling device Direct Memory Access (DMA) via the µ-hypervisor. As we will see later, such hardware support is provided by VT-d.

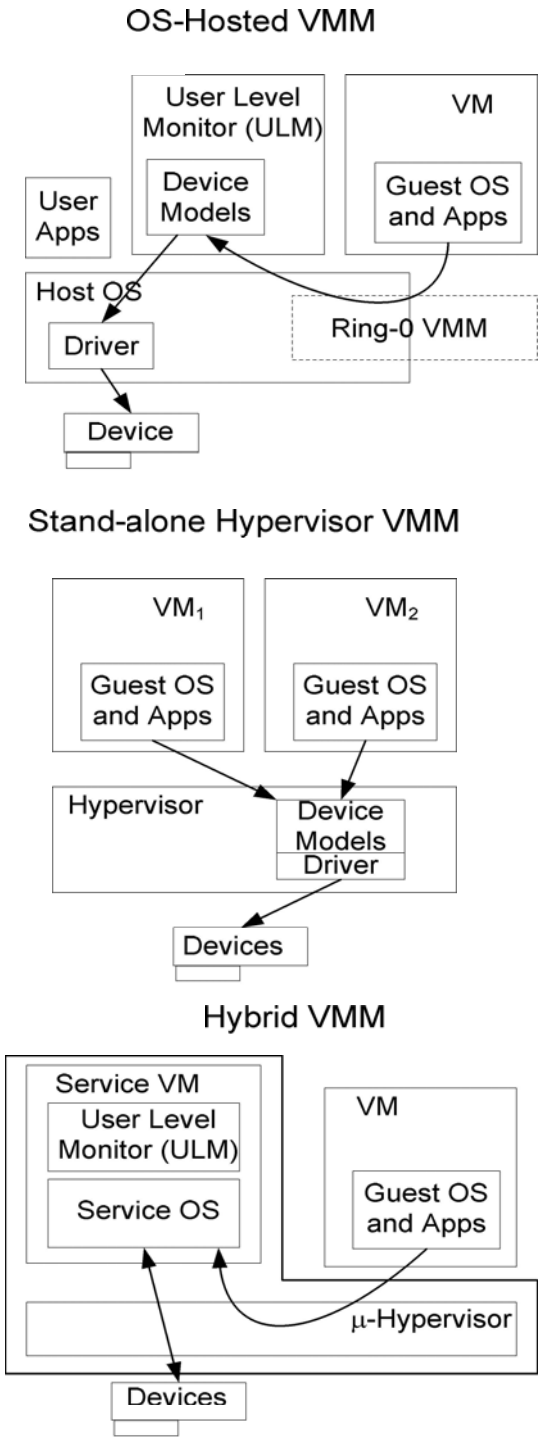

Figure 1: VMM software architectures

CURRENT I/O VIRTUALIZATION TECHNIQUES

When virtualizing an I/O device, it is necessary for the underlying virtualization software to service several types of operations for that device. Interactions between software and physical devices include the following:

- *Device discovery*: a mechanism for software to discover, query, and configure devices in the platform.
- *Device control*: a mechanism for software to communicate with the device and initiate I/O operations.
- *Data transfers*: a mechanism for the device to transfer data to and from system memory. Most devices support DMA in order to transfer data.
- *I/O interrupts*: a mechanism for hardware to be able to notify the software of events and state changes.

Each of these interactions is discussed, covering implementation, challenges, advantages, and disadvantages of each of the common virtualization techniques. The VMM could be a single monolithic software stack or could be a combination of a hypervisor and specialized guests (as shown in Figure 1). The type of VMM architecture used is independent of the concepts discussed in this section, but will become relevant later in our discussion.

Emulation

I/O mechanisms on native (non-virtualized) platforms are usually performed on some type of hardware device. The software stack, commonly a driver in an OS, will interface with the hardware through some type of memory-mapped (MMIO) mechanism, whereby the processor issues instructions to read and write specific memory (or port) address ranges. The values read and written correspond to direct functions in hardware.

Emulation refers to the implementation of real hardware completely in software. Its greatest advantage is that it does not require any changes to existing guest software. The software runs as it did in the native case, interacting with the VMM emulator just as though it would with real hardware. The software is unaware that it is really talking to a *virtualized* device. In order for emulation to work, several mechanisms are required.

The VMM must expose a device in a manner that it can be *discovered* by the guest software. An example is to present a device in a PCI configuration space so that the guest software can "*see*" the device and discover the memory addresses that it can use to interact with the device.

The VMM must also have some method for capturing reads and writes to the device's address range, as well as capturing accesses to the device-discovery space. This enables the VMM to *emulate* the real hardware with which the guest software believes it is interfacing.

The device (usually called a device model) is implemented by the VMM completely in software (see Figure 2). It may be accessing a real piece of hardware in the platform in some manner to service some I/O, but that hardware is independent of the device model. For example, a guest might see an Integrated Drive Electronics (IDE) hard disk model exposed by the VMM, while the real platform actually contains a Serial ATA (SATA) drive.

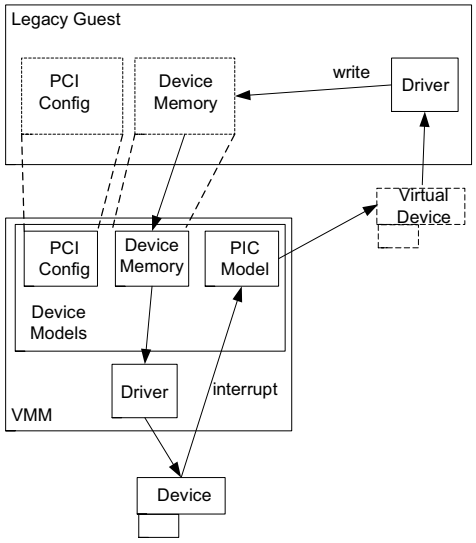

Figure 2: Device emulation model

The VMM must also have a mechanism for injecting interrupts into the guest at appropriate times on behalf of the emulated device. This is usually accomplished by emulating a Programmable Interrupt Controller (PIC). Once again, when the guest software exercises the PIC, these accesses must be trapped and the PIC device modeled appropriately by the VMM. While the PIC can be thought of as just another I/O device, it has to be there for any other interrupt-driven I/O devices to be emulated properly.

Emulation facilitates migration of VMs from one platform to another. Since the devices are purely emulated and have no ties to physical devices in the platform, it is easy to move a VM to another platform where the VMM can support the exact same emulated devices. If the guest VM did have some tie to any platform physical devices, those same physical devices would need to be present on any platform to which the VM was migrated.

Emulation also facilitates the sharing of platform physical devices of the same type, because there are instances of an emulation model exposed to potentially many guests. The VMM can use some type of sharing mechanism to allow all guest's emulation models access to the services of a single physical device. For example, the traffic from many guests with emulated network adapters could be bridged onto the platform's physical network adapter.

Since emulation presents to the guest software the exact interface of some existing physical hardware device, it can support a number of different guest OSs in an OS-independent manner. For example, if a particular storage device is emulated completely, then it will work with any software written for that device, independent of the guest OS, whether it be Windows*, Linux*, or some other IA-based OS. Since most modern OSs ship with drivers for many well-known devices, a particular device make and model can be selected for emulation such that it will be supported by these existing legacy environments.

While emulation's greatest advantage is that there are no requirements to modify guest device drivers, its largest detractor is low performance. Each interaction of the guest device driver with the emulated device hardware requires a transition to the VMM, where the device model performs the necessary emulation, and then a transition back to the guest with the appropriate results. Depending upon the type of I/O device that is being emulated, many of these transactions may be required to actually retrieve data from the device. These activities add considerable overhead compared to normal software-hardware interactions in a non-virtualized system. Most of this new overhead is compute-bound in nature and increases CPU utilization. The timing involved in each interaction can also accumulate to increase overall latency.

Another disadvantage of emulation is that the device model needs to emulate the hardware device very accurately, sometimes to the revision of the hardware, and must cover all corner cases. This can result in the need for "bug emulation" and problems arising with new revisions of hardware.

Paravirtualization

Another technique for virtualizing I/O is to modify the software within the guest, an approach that is commonly referred to as paravirtualization [4, 8]. The advantage of I/O paravirtualization is better performance. A disadvantage is that it requires modification of the guest software, in particular device drivers, which limits its applicability to legacy OS and device-driver binaries.

With paravirtualization (see Figure 3) the altered guest software interacts directly with the VMM, usually at a higher abstraction level than the normal hardware/software interface. The VMM exposes an I/O type-specific API, for example, to send and receive network packets–in the case of a network adaptor. The altered software in the guest then uses this VMM API instead of interacting directly with a hardware device interface.

Paravirtualization reduces the number of interactions between the guest OS and VMM, resulting in better performance (higher throughput, lower latency, reduced CPU utilization), compared to device emulation.

Instead of using an emulated interrupt mechanism, paravirtualization uses an eventing or callback mechanism. This again has the potential to deliver better performance, because interactions with a PIC hardware interface are eliminated, and because most OS's handle interrupts in a staged manner, adding overhead and latency. First, interrupts are fielded by a small Interrupt Service Routine (ISR). An ISR usually acknowledges the interrupt and schedules a corresponding worker task. The worker task is then run in a different context to handle the bulk of the work associated with the interrupt. With an event or callback being initiated directly in the guest software by the VMM, the work can be handled directly in the same context. With some implementations, when the VMM wishes to introduce an interrupt into the guest, it must force the running guest to exit to the VMM, where any pending interrupts can be picked up when the guest is reentered. To force a running guest to exit, a mechanism like IPI can be used. But this again adds overhead compared to a direct callback or event. Again, the largest detractor to this approach is that the interrupt handling mechanisms of the guest OS kernel must also be altered.

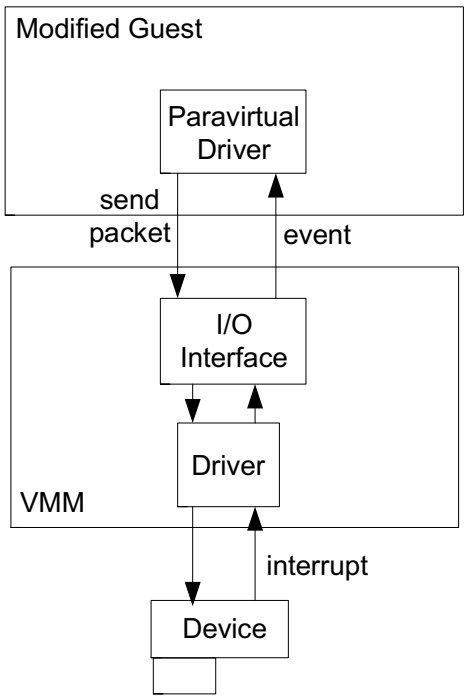

Figure 3: Device paravirtualization

Since paravirtualization involves changing guest software, usually the changed components are specific to the guest environment. For instance, a paravirtualized storage driver for Windows XP* will not work in a Linux environment. Therefore, a separate paravirtualized component must be developed and supported for each targeted guest environment. These changes require apriori knowledge of which guest environments will be supported by a particular VMM.

As with device emulation, paravirtualization is supportive of VM migration, provided that the VM is migrated to a platform that supports the same VMM APIs required by the guest software stack.

Sharing of any platform physical devices of the same type is supported in the same manner as emulation. For example, guests using a paravirtualized storage driver to read and write data could be backed by stores on the same physical storage device managed by the VMM.

Paravirtualization is increasingly deployed to satisfy the performance requirements of I/O-intensive applications. Paravirtualization of I/O classes that are performance sensitive, such as networking, storage, and high-performance graphics, appears to be the method of choice in modern VMM architecture. As described, para-virtualization of I/O decreases the number of transitions between the client VM and the VMM, as well as eliminates most of the processing associated with device emulation.

Paravirtualization leads to a higher level of abstraction for I/O interfaces within the guest OS. I/O-buffer allocation and management policies that are aware of the fact that they are virtualized can be used for more efficient use of the VT-d protection and translation facilities than would be possible with an unmodified driver that relies on full device emulation.

At least three of the major VMM vendors have adopted the capability to paravirtualize I/O in order to accomplish greater scaling and performance. Xen* and VMware already have the ability to run paravirtualized I/O drivers and Microsoft's plans include I/O paravirtualization in its next-generation VMM.

Direct Assignment

There are cases where it is desirable for a physical I/O device in the platform to be directly *owned* by a particular guest VM. Like emulation, direct assignment allows the owning guest VM to interface directly to a standard device hardware interface. Therefore, direct device assignment provides a native experience for the guest VM, because it can reuse existing drivers or other software to talk directly to the device.

Direct assignment improves performance over emulation because it allows the guest VM device driver to talk to the device in its native hardware command format eliminating the overhead of translating from the device command format of the virtual emulated device. More importantly, direct assignment increases VMM reliability and decreases VMM complexity since complex device drivers can be moved from the VMM to the guest.

Direct assignment, however, is not appropriate for all usages. First, a VMM can only allocate as many devices as are physically present in the platform. Second, direct assignment complicates VM migration in a number of ways. In order to migrate a VM between platforms, a similar device type, make, and model must be present and available on each platform. The VMM must also develop methods to extract any physical device state from the source platform, and to restore that state at the destination platform.

Moreover, in the absence of hardware support for direct assignment, direct assignment fails to reach its full potential in improving performance and enhancing reliability. First, platform interrupts may still need to be fielded by the VMM since it owns the rest of the physical platform. These interrupts must be routed to the appropriate guest–in this case the one that owns the physical device. Therefore, there is still some overhead in this relaying of interrupts. Second, existing platforms do not provide a mechanism for a device to directly perform data transfers to and from the system memory that belongs

to the guest VM in an efficient *and* secure manner. A guest VM is typically operating in a subset of the real physical address space. What the guest VM believes is its physical memory really is not; it is a subset of the system memory virtualized by the VMM for the guest. This addressing mismatch causes a problem for DMA-capable devices. Such devices place data directly into system memory without involving the CPU. When the guest device driver instructs the device to perform a transfer it is using guest physical addresses, while the hardware is accessing system memory using host physical addresses.

In order to deal with the address space mismatch, VMMs that support direct assignment may employ a pass-through driver that intercepts all communication between the guest VM device driver and the hardware device. The pass-through driver performs the translation between the guest physical and real physical address spaces of all command arguments that refer to physical addresses. Pass-through drivers are device-specific since they must decode the command format for a specific device to perform the necessary translations. Such drivers perform a simpler task than traditional device drivers; therefore, performance is improved over emulation. However, VMM complexity remains high, thereby impacting VMM reliability. Still, the performance benefits have proven sufficient to employ this method in VMMs targeted to the server space, where it is acceptable to support direct assignment for only a relatively small number of common devices.

VMM Software Architecture Implications

Different I/O virtualization methods are not equally applicable to all VMM software architecture options.

Emulation is the most general I/O virtualization method, able to expose standard I/O devices to an unmodified guest OS. Accordingly, it is widely employed in existing OS-hosted, stand-alone hypervisor or hybrid VMM implementations.

As already mentioned, paravirtualization is increasingly being deployed in many VMMs to improve performance for common guests. It is readily applicable to stand-alone hypervisor VMMs. It can also be used in the interaction between the guest OS and the ULM in an OS-hosted VMM or can be used in the guest OS and the service VM in a hybrid VMM.

Direct assignment is used in cases where the guest OS cannot be modified either because it is difficult to do so or the paravirtualized guest device drivers are not qualified for a specific application. However, it is difficult to introduce direct assignment in an OS-hosted VMM since in general, such VMMs do not own real platform devices and do not maintain device drivers for such devices. On the other hand, direct assignment naturally reduces complexity in stand-alone hypervisor and hybrid VMMs since device drivers can be moved to the guest OS or service OSs, respectively. This reduced complexity is not possible with either emulation or paravirtualization.

As our discussion suggests, it is quite likely that a VMM can employ many different techniques for I/O virtualization concurrently. For instance, in the context of hybrid VMM, direct assignment might be used to assign a platform physical device to a particular guest VM, whose responsibility it is to share that device with many guests. Depending upon the needs and requirements of the guest, it may offer both emulated device models, as well as paravirtualized solutions to the different guests. A common configuration is to provide paravirtualized solutions for the most common guest environments, while an emulation solution is offered to support all other legacy environments.

IOVM Architecture

A major emerging trend among developers of virtualization software, in particular for I/O processing and sharing, is the VMM system decomposition.

The trend for the software architecture of VMMs is to move from a monolithic hypervisor model towards a software architecture that decomposes the VMM into a very thin privileged "micro-hypervisor" that resides just above the physical hardware, and one or more special-purpose VMs that are de-privileged relative to the hypervisor, and are responsible for services and policy. With regard to I/O virtualization, these deprivileged components of the VMM can be responsible for I/O processing and I/O resource sharing. We call this general architecture the "IOVM" model (see Figure 4). The IOVM model is a generalization of the hybrid VMM architecture in that I/O devices can be allocated to different service VMs specialized for the specific I/O function (e.g., network VM, storage VM, etc.).

Two major benefits of the IOVM model are the ability to use unmodified device drivers within the IOVM and the isolation of the physical device and its driver(s) from the other guest OSs, applications, and hypervisor. The use of unmodified drivers is possible because these drivers can run in a separate OS environment, in contrast to a monolithic hypervisor where new drivers are often written for the VMM environment. The isolation of the device and its driver protect the guest VMs from driver crashes, that is, the IOVM may crash due to a driver failure without severely affecting the guest OSs. A disadvantage of the IOVM model is that there is additional overhead incurred, due to additional communication and data movement between the guest OS and the IOVM. This performance penalty can be offset by paravirtualizing the interface of the IOVM, thus minimizing the number of

interactions. The Xen VMM has implemented this architecture as "Isolated Driver Domains" [6], and Microsoft is in the process of developing a version of this architecture in their next generation of VMMs [7].

Direct assignment of I/O devices to IOVMs directly facilitates this usage model and is becoming increasingly important as VMMs are transitioning to this architecture. As we have seen, however, software by itself is not capable of fully protecting the system from errant DMA traffic between the I/O device and system memory while at the same time eliminating all device-specific functionality in the VMM. Hardware support on the platform closes this gap, by allowing the device to be safely assigned to an IOVM, thus allowing full protection from errant DMA transfers.

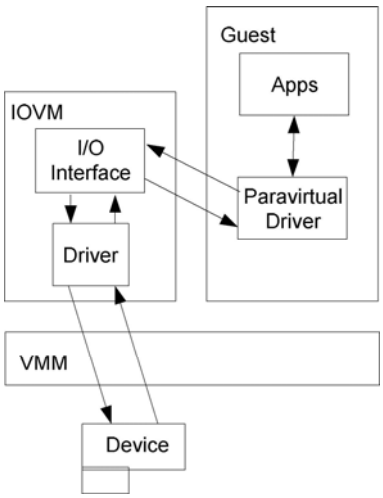

Figure 4: IOVM software architecture

PLATFORM HARDWARE SUPPORT FOR I/O VIRTUALIZATION

To enforce the isolation, security, reliability, and performance benefits of direct assignment, we need efficient hardware mechanisms to constrain the operation of I/O devices. The primary I/O device accesses that require this isolation are device transfers (DMAs) and interrupts. CPU virtualization mechanisms are sufficient to efficiently perform device discovery and schedule device operations.

Accordingly, VT-d [12] provides the platform hardware support for DMA and interrupt virtualization.

DMA Remapping

DMA remapping facilities have been implemented in a variety of contexts in the past to facilitate different usages. In workstations and server platforms, traditional I/O memory management units (IOMMUs) have been implemented in PCI root bridges to efficiently support scatter/gather operations or I/O devices with limited DMA addressability [17]. Other well-known examples of DMA remapping facilities include the AGP Graphics Aperture Remapping Table (GART) [18], the Translation and Protection Table (TPT) defined in the Virtual Interface Architecture [14], and subsequently influencing a similar capability in the InfiniBand Architecture [16] and Remote DMA (RDMA) over TCP/IP specifications [19]. DMA remapping facilities have also been explored in the context of NICs designed for low latency cluster interconnects [15].

Traditional IOMMUs typically support an aperture-based architecture. All DMA requests that target a programmed aperture address range in the system physical address space are translated irrespective of the source of the request. While this is useful for handling legacy device limitations (such as limited DMA addressability or scatter/gather capabilities), they are not adequate for I/O virtualization usages that require full DMA isolation.

The VT-d architecture is a generalized IOMMU architecture that enables system software to create multiple DMA protection domains. A protection domain is abstractly defined as an isolated environment to which a subset of the host physical memory is allocated. Depending on the software usage model, a DMA protection domain may represent memory allocated to a VM, or the DMA memory allocated by a guest-OS driver running in a VM or as part of the VMM itself. The VT-d architecture enables system software to assign one or more I/O devices to a protection domain. DMA isolation is achieved by restricting access to a protection domain's physical memory from I/O devices not assigned to it, through address-translation tables.

The I/O devices assigned to a protection domain can be provided a view of memory that may be different than the host view of physical memory. VT-d hardware treats the address specified in a DMA request as a DMA virtual address (DVA). Depending on the software usage model, a DVA may be the Guest Physical Address (GPA) of the VM to which the I/O device is assigned, or some software-abstracted virtual I/O address (similar to CPU linear addresses). VT-d hardware transforms the address in a DMA request issued by an I/O device to its corresponding Host Physical Address (HPA).

Figure 5 illustrates DMA address translation in a multi-domain usage. I/O devices 1 and 2 are assigned to protection domains 1 and 2, respectively, each with its on view of the DMA address space.

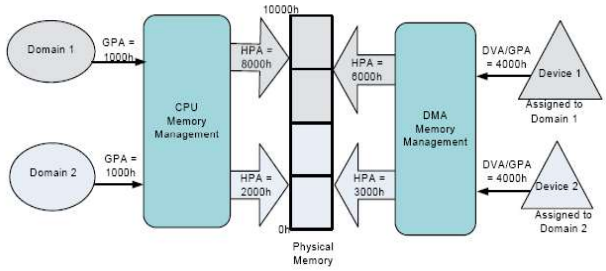

Figure 5: DMA remapping

Figure 6 illustrates a PC platform configuration with VT-d hardware implemented in the north-bridge component.

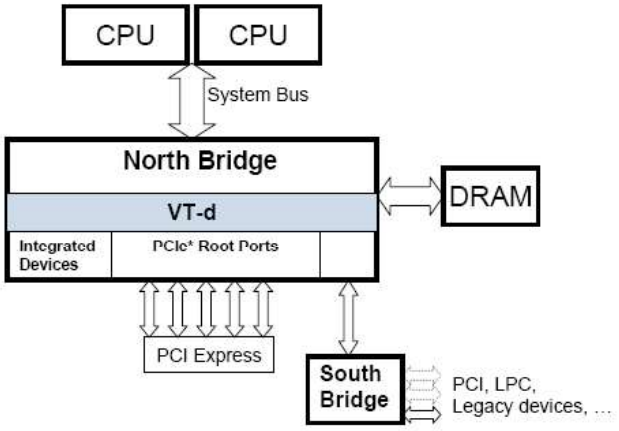

Figure 6: Platform configuration with VT-d

Mapping Devices to Protection Domains

To support multiple protection domains, the DMA remapping hardware must identify the device originating each DMA request. The requester identifier of a device is composed of its PCI Bus/Device/Function number assigned by PCI configuration software and uniquely identifies the hardware function that initiated the request. Figure 7 illustrates the requester-id as defined by the PCI specifications [20].

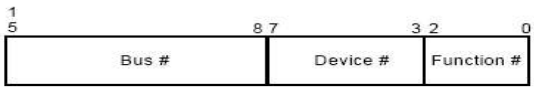

Figure 7: PCI requester identifier format

VT-d architecture defines the following data structures for mapping I/O devices to protection domains (see Figure 8):

- *Root-Entry Table*: Each entry in the root-entry table functions as the top-level structure to map devices for a specific PCI bus. The bus-number portion of the requester-id in DMA requests is used to index into the root-entry table. Each present root entry includes a pointer to a context-entry table.
- *Context-Entry Table*: Each entry in the context-entry table maps a specific I/O device on a bus to the protection domain to which it is assigned. The device and function-number portion of the requester-id is used to index into the context-entry table. Each present context entry includes a pointer to the address translation structures used to translate the address in the DMA request.

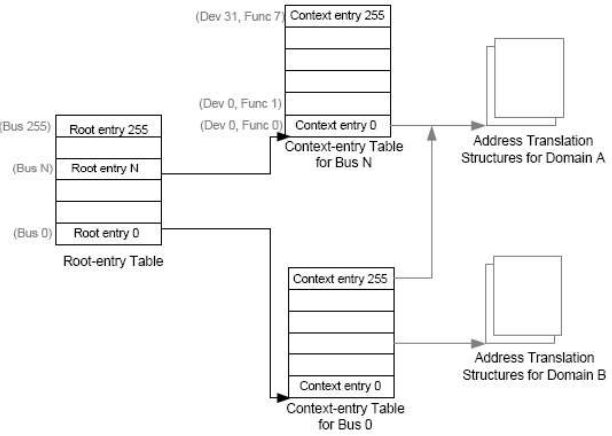

Figure 8: Device mapping structures

Address Translation

VT-d architecture defines a multi-level page-table structure for DMA address translation (see Figure 9). The multi-level page tables are similar to IA-32 processor page-tables, enabling software to manage memory at 4 KB or larger page granularity. Hardware implements the page-walk logic and traverses these structures using the address from the DMA request. The number of page-table levels that must be traversed is specified through the context-entry referencing the root of the page table. The page directory and page-table entries specify independent read and write permissions, and hardware computes the cumulative read and write permissions encountered in a page walk as the effective permissions for a DMA request. The page-table and page-directory structures are always 4 KB in size, and larger page sizes (2 MB, 1 GB, etc.) are enabled through super-page support.

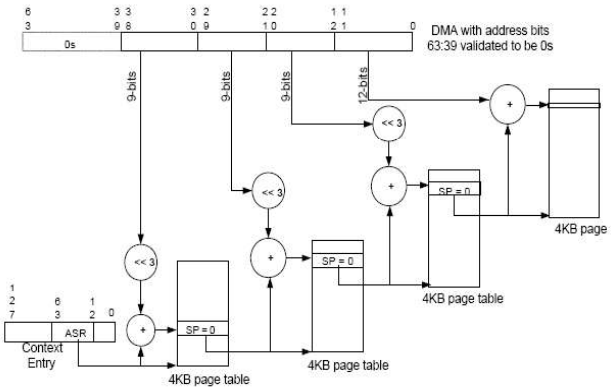

Figure 9: Example 3-level page table

Interrupt Remapping

For proper device isolation in a virtualized system, the interrupt requests generated by I/O devices must be controlled by the VMM. In the existing interrupt architecture for Intel platforms, a device may generate either a legacy interrupt (which is routed through I/O interrupt controllers) or may directly issue message signaled interrupts (MSIs) [20]. MSIs are issued as DMA write transactions to a pre-defined architectural address range, and the interrupt attributes (such as vector, destination processor, delivery mode, etc.) are encoded in the address and data of the write request. Since the interrupt attributes are encoded in the request issued by devices, the existing interrupt architecture does not offer interrupt isolation across protection domains.

The VT-d interrupt-remapping architecture addresses this problem by redefining the interrupt-message format. The new interrupt message continues to be a DMA write request, but the write request itself contains only a "message identifier" and not the actual interrupt attributes. The write request, like any DMA request, specifies the requester-id of the hardware function generating the interrupt.

DMA write requests identified as interrupt requests by the hardware are subject to interrupt remapping. The requestor-id of interrupt requests is remapped through the table structure. Each entry in the interrupt-remapping table corresponds to a unique interrupt message identifier from a device and includes all the necessary interrupt attributes (such as destination processor, vector, delivery mode, etc.). The architecture supports remapping interrupt messages from all sources including I/O interrupt controllers (IOAPICs), and all flavors of MSI and MSI-X interrupts defined in the PCI specifications.

Software Usages of DMA and Interrupt Remapping

The VT-d architecture enables DMA and interrupt requests from an I/O device to be isolated to its assigned protection domain. This capability makes possible a number of usages:

- *Remapping for legacy guests*: In this usage an I/O device is assigned directly to a VM running a legacy (virtualization unaware) environment. Since the guest OS has the guest-physical view of memory in this usage, the VMM programs the DMA remapping structures for the I/O device to support appropriate GPA to HPA mappings. Similarly, the VMM may program the interrupt-remapping structures to enable the interrupt requests from the I/O device to target the physical CPUs running the appropriate virtual CPUs of the legacy VM.
- *Remapping for IOMMU-aware guests*: An OS may be capable of using DMA and interrupt remapping hardware to improve its OS reliability or for handling specific I/O-device limitations. When such an OS is running within a VM, the VMM may expose virtual (emulated or paravirtualized) remapping hardware to the VM. The OS may create one or more protection domains each with its own DMA Virtual Address (DVA) space and program the virtual remapping hardware structures to support DVA to Guest Physical Address (GPA) mappings. The VMM must virtualize the remapping hardware by intercepting guest accesses to the virtual hardware and shadowing the virtual remapping structures to provide the physical hardware with structures for DVA to HPA mappings. Similar page table shadowing techniques are commonly used by the VMM for CPU MMU virtualization.

Hardware Caching and Invalidation Architecture

To improve DMA and interrupt-remapping performance, the VT-d architecture allows hardware implementations to cache frequently used remapping-structure entries. Specifically, the following architectural caching constructs are defined:

- *Context Cache*: Caches frequently used context entries that map devices to protection domains.
- *PDE (Page Directory Entry) Cache*: Caches frequently used page-directory entries encountered by hardware during page walks.
- *IOTLB (I/O Translation Look-aside Buffer)*: Caches frequently used effective translations (results of the page walk).

- *Interrupt Entry Cache*: Caches frequently used interrupt-remapping table entries.

These caching structures are fully managed by the hardware. When updating the remapping structures, the software is responsible for maintaining the consistency of these caches by invalidating any stale entries in the caches. VT-d architecture defines the following invalidation options:

- *Synchronous Invalidation*: The synchronous invalidation interface uses a set of memory-mapped registers for software to request invalidations and to poll for invalidation completions.
- *Queued Invalidation*: The queued-invalidation interface uses a memory-resident command queue for software to queue-invalidation requests. Software synchronizes invalidation completions with hardware by submitting an invalidation-wait command to the command queue. Hardware guarantees that all invalidation requests received before an invalidation-wait command are completed before completing the invalidation-wait command. Hardware signals the invalidation-wait command completion either through an interrupt or by coherently writing a software-specified memory location. The queued-invalidation interface enables usages where software can batch invalidation requests.

Scaling Address Translation Caches

Caching of the remapping structures enables hardware to minimize the DMA translation overhead that may otherwise be incurred when accessing the memory-resident translation structures. One of the challenges for DMA-remapping hardware implementations is to efficiently scale its hardware caching structures. Unlike CPU TLBs that support accesses from a CPU that is typically running one thread at a time, the DMA-remapping caches handle simultaneous DMA accesses from multiple devices, and often multiple DMA streams from a device.

This difference makes sizing the IOTLBs in DMA-remapping hardware implementations challenging, especially when the hardware design is re-used across a wide range of platform configurations. An approach to scaling the IOTLBs is to enable I/O devices to participate in DMA remapping by requesting translations for its own memory accesses from the DMA-remapping hardware and caching these translations locally on the I/O device in a Device-IOTLB.

To facilitate scaling of address translation caches, PCI Express* protocol extensions (referred to as Address Translation Services (ATS)) [22] are being standardized by the PCI Special Interest Group (PCI-SIG) [21]. ATS consist of a set of PCI transactions that allow the optimization of VT-d address translations. These extensions enable I/O devices to request translations from the root complex and for the root complex to return responses for each translation request. I/O devices may cache the returned translations in its local Device-IOTLBs and indicate if a DMA request is using un-translated address or translated address from its Device-IOTLB. To support usages where software may dynamically modify the translations, the ATS protocol extensions enable the root complex to request invalidations of translations cached in the Device-IOTLB of an I/O device, and for the I/O devices to return responses indicating when an invalidation request is completed.

VT-d architecture supports ATS protocol extensions and enables software to control (through the device-mapping structures) if an I/O device can issue these transactions. For DMA requests indicating translated addresses from allowed devices, VT-d hardware bypasses the DMA-address translation.

I/O devices may implement Device-IOTLBs and support these protocol extensions to minimize performance dependencies on the DMA-remapping caching resources in the platform. However, to preserve the security, isolation, and reliability benefits of DMA remapping, device implementations must ensure that only translation responses from the root complex cause entries to be inserted into the Device IOTLB.

Handling Remapping Errors

Any errors or permission violations detected as part of remapping a DMA request are treated as a remapping fault. Unlike CPU page faults, which are restart-able at instruction boundaries, DMA-remapping faults are not restart-able due to the posted nature of PCI transactions. Any DMA write request that generates a fault is blocked by the remapping hardware, and the DMA read requests return an error to the device in the read response. Hardware logs detail DMA requests that cause remapping faults and use a fault event (interrupt) to inform software about such faults. For devices that explicitly request translations, an error detected while processing the translation request is not treated as a DMA-remapping fault, but is merely conveyed to the device in the translation response. This enables such devices to support device-specific demand page faulting. Demand page faulting is beneficial for devices (such as graphics adapters) with large DMA footprints, enabling software to demand pin the DMA buffers.

FUTURE HARDWARE SUPPORT

While VT-d enables the direct assignment of devices to guest VMs, it does not directly facilitate the efficient

sharing of devices across multiple guest VMs. Such efficient sharing is not feasible without fundamental changes in the way that devices present their resources to the platform. Further work is being done in the PCI-SIG [21] [22] to enhance the PCI Express* specifications to enable devices to be shared.

Briefly, these extensions enable PCI Express devices to support multiple virtual functions, each of which can be discovered, configured, and managed. This allows the direct assignment of a virtual function to a VM using VT-d, thus allowing a single physical device to be sharable among multiple VMs.

The importance and applicability of these sharable PCI Express devices may be largely dependent upon the performance requirements, usage model, and market segment in which they may be deployed.

CONCLUSION

The virtualization of I/O resources is an important step toward enabling a significant set of emerging usage models in the data center, the enterprise, and the home. VT-d support on Intel platforms provides the capability to ensure improved isolation of I/O resources for greater reliability, security, and availability.

Specifically, VT-d supports the remapping of I/O DMA transfers and device-generated interrupts. The architecture of VT-d provides the flexibility to support multiple usage models that may run un-modified, special-purpose, or "virtualization aware" guest OSs. The VT-d hardware capabilities for I/O virtualization complement the existing Intel VT capability to virtualize processor and memory resources. Together, this roadmap of VT technologies offers a complete solution to provide full hardware support for the virtualization of Intel platforms.

Ongoing and future developments within the virtualization hardware and software communities will build upon VT-d to ensure that the requirements for sharing, security, performance, and scalability are being met. I/O devices will become more aware of the existence of VT-d to ensure efficient caching and consistency mechanisms to enhance their performance. Given the protection provided by VT-d, future I/O devices will emerge that are sharable among multiple guest OSs. With VT-d, software developers can develop and evolve their architectures that provide fully protected sharing of I/O resources that are highly available, provide high performance, and scale to increasing I/O demands.

REFERENCES

[1] Intel Corp., "Intel Virtualization Technology Specification for the IA-32 Architecture," at www.intel.com/technology/vt/.

[2] Intel Corp., "Intel Virtualization Technology Specification for the Intel Itanium Architecture;" at www.intel.com/technology/vt/.

[3] Sugarman, J., Venkitachalam, G., Lim, B., "Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor," in *Proceedings of 2002 USENIX Annual Technical Conference*, pp. 1–14, June 2001.

[4] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pp. 164–177, October 2003.

[5] T. Garfinkel, B. Pfaff, J. Chow, M., Rosenblum, and D. Boneh, "Terra: A virtual machine-based platform for trusted computing," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pp. 193–206, 2003.

[6] K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. Warfield, and M. Williams, "Safe hardware access with the Xen virtual machine monitor," in *Proceedings of the First Workshop on Operating System and Architectural Support for the on-demand IT Infrastructure (OASIS-2004)*, October 2004.

[7] M. Kieffer, "Windows Virtualization Architecture," *WinHEC 2005*, at http://download.microsoft.com/download/9/8/f/98f3fe47-dfc3-4e74-92a3-088782200fe7/TWAR05013_WinHEC05.ppt*.

[8] A. Whitaker, M. Shaw, and S. Gribble, "Scale and Performance in the Denali Isolation Kernel," in *System Design and Implementation (OSDI),* Boston, MA, December 2002.

[9] P. England, B. Lampson, J. Manferdelli, M. Peinado, B. Willman, "A Trusted Open Platform," *IEEE Computer*, pp. 55–62, July 2003.

[10] R. Goldberg, "Survey of Virtual Machine Research," *IEEE Computer*, pp. 34–45, June 1974.

[11] R. Creasy, "The Origin of the VM/370 Time-Sharing System," *IBM Journal of Research and Development*, pp. 483–490, September 1981.

[12] Intel Corp., "Intel Virtualization Technology Specification for Directed I/O Specification," at www.intel.com/technology/vt/.

[13] Microsoft Corp., "Microsoft Virtual Server 2005 Technical Overview," 2004, at http://download.microsoft.com/download/5/5/3/55321426-cb43-4672-9123-74ca3af6911d/VS2005TechWP.doc*.

[14] Dave Dunning, Greg Regnier, Don Cameron, Gary McAlpine, et al., "The Virtual Interface Architecture," *IEEE Micro, Volume 18, Issue 2*, pp. 66–76, March-April 1998.

[15] Ioannis Schoinas and Mark D. Hill, "Address Translation Mechanisms in Network Interfaces," in *Proceedings of the Fourth International Symposium on High-Performance Computer Architecture*, February 1998.

[16] InfiniBand Trade Association, http://www.infinibandta.org/specs*.

[17] Grant Grundler, "Porting Drivers to HP ZX1," in *Proceedings of the Ottawa Linux Symposium,* June 2002.

[18] Intel Corp., AGP V3.0 Interface Specification.

[19] RDMA Consortium, http://www.rdmaconsortium.org*

[20] PCI Express Base Specification 1.1, http://www.pcisig.com/specifications/pciexpress/base*

[21] http://www.pcisig.com*

[22] PCI Express Address Translation Services and I/O Virtualization, WinHEC 2006, http://www.microsoft.com/whdc/winhec/pres06.mspx*

AUTHORS BIOGRAPHIES

Darren Abramson is a principal engineer in Intel's Chipset Group. He received his B.S. degree from the University of Kansas. Darren joined Intel in 1991 and has worked primarily on client chipset development and architecture, bringing to market I/O initiatives such as PCI, USB, and more recently PCI Express. His e-mail is darren.abramson at intel.com.

Jeff Jackson is a senior architect in Intel's Corporate Technology Group. Recently his areas of interest have been around networking-related technologies in virtualized environments. Jeff received an M.S. degree in Computer Science from Purdue University in 1994. His e-mail is jeff.jackson at intel.com.

Sridhar Muthrasanallur is a senior I/O architect in Intel's Digital Enterprise Group. He has eight years of experience in architecting I/O solutions for the Intel® Server Chipsets. His e-mail is sridhar.muthrasanallur at intel.com.

Gil Neiger is a principal engineer in Intel's Corporate Technology Group and leads development of the VT-x architecture. He received his Ph.D. degree in Computer Science from Cornell University.

Greg Regnier is a principal engineer in Intel's Corporate Technology Group. He joined Intel in 1988 and his experiences include massively parallel supercomputers, cluster communications, and high-performance network architecture. Regnier has a B.S. degree in Computer Science from St. Cloud State University in Minnesota. His e-mail is greg.j.regnier at intel.com.

Rajesh Sankaran is a principal engineer in Intel's Corporate Technology Group and is involved in development of CPU and I/O virtualization architecture. He received his M.S. degree in Electrical Engineering from Portland State University. His e-mail is rajesh.sankaran at intel.com.

Ioannis (Yannis) Schoinas is a principal engineer in Intel's Corporate Technology Group. He received his B.S. and M.S. degrees from the University of Crete-Heraclion and his Ph.D. degree from the University of Wisconsin-Madison. Yannis joined Intel's Server Architecture Lab in 1998 and worked on coherence protocols for the i870 chipset and future Intel® platforms. He also worked on a wide range of platform architecture topics including memory RAS, system partitioning, configuration management, system security and VT-d architecture. He is currently focusing on Tera-Scale Computing architecture challenges. His e-mail is ioannis.t.schoinas at intel.com.

Rich Uhlig is a senior principal engineer in Intel's Corporate Technology Group and leads various aspects of Intel's Virtualization Technology program, including architecture definition, research prototyping, performance analysis and characterization of VMM software usage. Rich received a Ph.D. degree in Computer Science and Engineering from the University of Michigan in 1995.

Balaji Vembu is a client ingredient architect in Intel's DEG Architecture and Planning group. He received his Bachelor's degree in EE from Regional Engg College, Bhopal, India. He received his M.S. degree in Computer Science from the University of California, Santa Barbara. He joined Intel in 1993 and worked on graphics and video acceleration in the chipset group. He is currently focused on virtualization and security architecture definition for client platforms. His e-mail is balaji.vembu at intel.com.

John Wiegert is a senior software engineer in Intel's Corporate Technology Group. John received his B.S. degree in Computer Science from the Rochester Institute of Technology. His current research interests involve I/O virtualization. His e-mail is john.a.wiegert at intel.com.

[Δ] Intel® Virtualization Technology requires a computer system with an enabled Intel® processor, BIOS, virtual machine monitor (VMM) and, for some uses, certain platform software enabled for it. Functionality,

performance or other benefits will vary depending on hardware and software configurations and may require a BIOS update. Software applications may not be compatible with all operating systems. Please check with your application vendor.

This document contains information on products in the design phase of development. The information here is subject to change without notice. Do not finalize a design with this information. Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Extending Xen* with Intel® Virtualization Technology

Yaozu Dong, Core Software Division, Intel Corporation
Shaofan Li, Core Software Division, Intel Corporation
Asit Mallick, Core Software Division, Intel Corporation
Jun Nakajima, Core Software Division, Intel Corporation
Kun Tian, Core Software Division, Intel Corporation
Xuefei Xu, Core Software Division, Intel Corporation
Fred Yang, Core Software Division, Intel Corporation
Wilfred Yu, Core Software Division, Intel Corporation

Index words: Xen, Virtualization, Hypervisor, Intel® VT, virtual machine monitor

ABSTRACT

Xen* is an open source virtual machine monitor (VMM) developed at the University of Cambridge to support operating systems (OSs) that have been modified to run on top of the monitor. Intel has extended the Xen VMM to use Intel® Virtualization Technology$^{\Delta}$ (VT) to support unmodified guest OSs also. This was done for IA-32 Intel® Architecture processors as well as Itanium® architecture processors.

In this paper we describe the changes that have been made to Xen to enable this support. We also highlight the optimizations that have been made to date to deliver good virtualized performance.

INTRODUCTION

Xen is an open source virtual machine monitor (VMM) that allows the hardware resources of a machine to be virtualized and dynamically shared between OSs running on top of it [1]. Each virtual machine (VM) is called a Domain, in Xen terminology. Xen provides isolated execution for each domain, preventing failures or malicious activities in one domain from impacting another domain. The Xen hypervisor and Domain0 (Dom0) are a required part of any Xen-based server. Multiple user domains, called DomainU in Xen terminology, can be created to run guest OSs.

Unlike the full virtualization solutions offered by the IBM VM/370*, or VMware's ESX* and Microsoft's Virtual PC product*, Xen began life as a VMM for guest OSs that have been modified to run on the Xen hypervisor. User applications within these OSs run as is, i.e., unmodified. This technique is called "paravirtualization," and it delivers near native performance for the guest OS, only if the guest OSs source code can be modified.

Xen versions 1.0 and 2.0 use paravirtualization techniques to support 32-bit platforms and Linux* guests. They use the standard IA-32 protection and segmentation architecture for system resource virtualization. The hypervisor runs in the highest privilege level *ring 0* and has full access to all memory on the system. Guest OSs use privilege levels 1, 2, and 3 as they see fit. Segmentation is used to prevent the guest OS from accessing the Xen address space.

Xen 3.0 is the first open-source VMM that uses Intel Virtualization Technology (VT) to support unmodified guest OSs as well as paravirtualized guest OSs. Xen 3.0 also added support for 64-bit platforms and 64-bit guests [9]. Page-level protection is used to protect the 64-bit hypervisor from the guest.

In this paper, we begin with a brief overview of Intel VT and then we explain how we extended Xen to take advantage of VT. We highlight key virtualization issues for IA-32, Intel® EM64T$^{\Phi}$, and Itanium processors and explain how they are addressed in Xen 3.0. Finally, we highlight some of the changes that have been made to the hypervisor and the device models to improve performance.

INTEL® VIRTUALIZATION TECHNOLOGY

Intel VT is a collection of processor technologies that enables robust execution of unmodified guest OSs on Intel VT-enhanced VMMs [2]. VT-x defines the

extensions to the IA-32 Intel Architecture [3]. VT-i defines the extensions to the Intel Itanium architecture [4].

VT-x augments IA-32 with two new forms of CPU operation: virtual machine extensions (VMX) root operations and VMX non-root operations. The transition from VMX root operation to VMX non-root operation is called a VM entry. The transition from a VMX non-root operation to VMX root operation is called a VM exit.

A virtual-machine control structure (VMCS) is defined to manage VM entries and exits, and it controls the behavior of instructions in a non-root operation. 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.

The VMM runs in root operation while the guests run in VMX non-root operation. Both forms of operation support all four privilege levels (i.e., rings 0, 1, 2, and 3). The VM-execution control fields in the VMCS allow the VMM to control the behavior of some instructions in VMX non-root operation and the events that will cause VM exits. Instructions like CPUID, MOV from CR3, RDMSR, and WRMSR will trigger VM exits unconditionally to allow the VMM to control the behavior of the guest.

VT-i expands the Itanium processor family (IPF) to enable robust execution of VMs. A new processor status register bit (PSR.vm) has been added to define a new operating mode for the processor. The VMM runs with this bit cleared while the guest OS runs with it set. Privileged instructions, including non-privileged instructions like *thash, ttag* and *mov cupid* that may reveal the true operating state of the processor, trigger virtualization faults when operating in this mode.

The PSR.vm bit also controls the number of virtual-address bits that are available to software. When a VMM is running with PSR.vm = 0, all implemented virtual-address bits are available. When the guest OS is running with PSR.vm = 1, the uppermost implemented virtual-address bit is made unavailable to the guest. Instruction or data fetches with any of these address bits set will trigger unimplemented data/instruction address faults or unimplemented instruction address traps. This provides the VMM a dedicated address space that guest software cannot access.

VT-i also defines the processor abstraction layer (PAL) interfaces that can be used by the VMM to create and manage VMs. A Virtual Processor Descriptor (VPD) is defined to represent the resources of a virtual processor. PAL procedures are defined to allow the VMM to configure logical processors for virtualization operations and to suspend or resume virtual processors. PAL run-time services are defined to support performance-critical VMM operations.

EXTENDING XEN* WITH INTEL VT

Xen 3.0 architecture (Figure 1) has a small hypervisor kernel that deals with virtualizing the CPU, memory, and critical I/O resources, such as the interrupt controller. Dom0 is a paravirtualized Linux that has privileged access to all I/O devices in the platform and is an integral part of any Xen-based system. Xen 3.0 also includes a control panel that controls the sharing of the processor, memory, network, and block devices. Access to the control interface is limited to Dom0. Multiple user domains, called DomainU (DomU) can be created to run paravirtualized guest OSs. Dom0 and DomU OSs use hypercalls to request services from the Xen hypervisor.

When Intel VT is used, fully virtualized domains can be created to run unmodified guest OSs. These fully virtualized domains are given the special name of HVMs (hardware-based virtual machines). Xen presents to each HVM guest a virtualized platform that resembles a classic PC/server platform with a keyboard, mouse, graphics display, disk, floppy, CD-ROM, etc. This virtualized platform support is provided by the Virtual I/O Devices module.

In the following sections we describe the extensions to each of these Xen components.

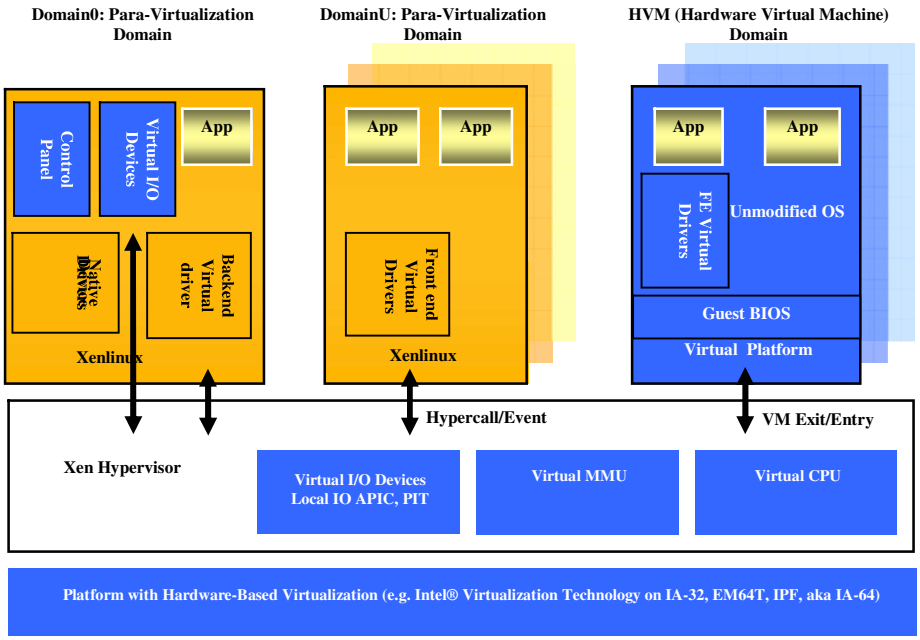

Figure 1: Xen 3.0 architecture

Control Panel

We have extended the control panel to support creating, controlling, and destroying HVM domains. The user can specify configuration parameters such as the guest memory map and size, the virtualized disk location, network configuration, etc.

The control panel loads the guest firmware into the HVM domain and creates the device model thread (explained later) that will run in Dom0 to service input/output (I/O) requests from the HVM guest. The control panel also configures the virtual devices seen by the HVM guest, such as the interrupt binding and the PCI configuration.

The HVM guest is then started, and control is passed to the first instruction in the guest firmware. The HVM guest executes at native speed until it encounters an event that requires special handling by Xen.

Guest Firmware

The guest firmware (BIOS) provides the boot services and run-time services required by the OS in the HVM. This guest firmware does not see any real physical devices. It operates on the virtual devices provided by the device models.

For VT-x, we are re-using the open source Bochs BIOS [5]. We extended the Bochs BIOS by adding Multi-Processor Specification (MPS) tables [6], Advanced Configuration and Power Interface (ACPI) tables [7], including the Multiple APIC Description Table (MADT). The BIOS and the early OS loader expect to run in real mode. To create the environment needed by these codes, we use VMXAssist to configure the VT-x guest to execute in virtual-8086 mode. Instructions that cannot be executed in this mode are intercepted and emulated with a software emulator.

For VT-i, we developed a guest firmware using the Intel® Platform Innovation Framework for Extensible Firmware Interface (EFI). This guest firmware provides all EFI boot services required by IPF guest OSs. It is compatible with the Developer's Interface Guide for 64-bit Intel® Architecture-based Servers (DIG64) and provides the System Abstraction Layer (SAL), ACPI 2.0, and EFI 1.10 tables required by IPF guest OSs.

Processor Virtualization

The Virtual CPU module in Xen provides the abstraction of a processor to the HVM guest. It manages the virtual processor(s) and associated virtualization events when the guest OS is executing. It saves the physical processor state when the guest gives up a physical CPU, and restores the guest state when it is rescheduled to run on a physical processor.

For the IA-32 architecture, a VMCS structure is created for each CPU in a HVM domain (Figure 2). The execution control of the CPU in VMX mode is configured as follows:

- Instructions such as CPUID, MOV from/to CR3, MOV to CR0/CR4, RDMSR, WRMSR, HLT,

INVLPG, MOV from CR8, MOV DR, and MWAIT are intercepted as VM exits.

- Exceptions/faults, such as page fault, are intercepted as VM exits, and virtualized exceptions/faults are injected on VM entry to guests.
- External interrupts unrelated to guests are intercepted as VM exits, and virtualized interrupts are injected on VM entry to the guests.
- Read shadows are created for the guest CR0, CR4, and time stamp counter (TSC). Read accesses to such registers will not cause VM exit, but will return the shadow values.

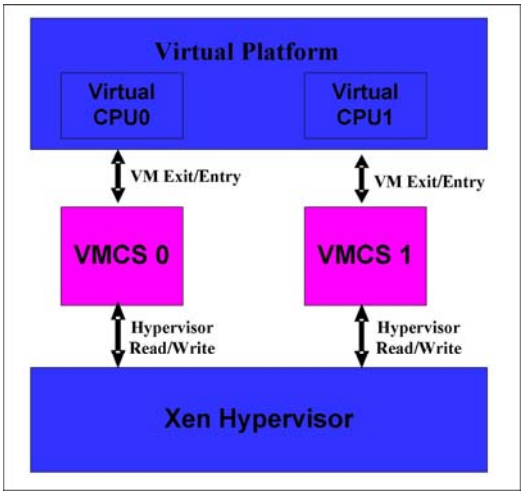

Figure 2: VMCS

For the Itanium architecture, a Virtual Processor Block (VPD) structure is created for each CPU in a HVM domain. The VPD has similar functionality as the VMCS in the IA-32 architecture. The virtualization control of the CPU is configured as follows:

- Instructions such as MOV from/to RR, MOV from/to CR, ITC/PTC, ITR/PTR, MOV from/to PKR, MOV from/to IBR/DBR are intercepted as virtualization faults.
- Instructions such as COVER, BSW are optimized to execute without virtualization faults.
- Exceptions/faults are intercepted by the VMM, and virtualized exceptions/faults are injected to the guest on a VM resume.
- External interrupts are intercepted by the VMM, and virtualized external interrupts are injected to the guest using the virtual external interrupt optimization.
- Read shadows are created for the guest interruption control registers, PSR, CPUID. Read accesses to such registers will not cause virtualization fault, but will return the shadow values.
- Write shadows are created for the guest interruption control registers. Write accesses to such registers will not cause virtualization fault, but will write to the shadow values.

An interesting question when designing Xen concerns the processor features that are exposed to HVM guests. Some VMMs present only a generic, minimally featured processor to the guest. This allows the guest to migrate easily to arbitrary platforms, but precludes the guest from using new instructions or processor features that may exist in the processor. For Xen, we are exporting most CPUID bits to the guest. We clearly need to clear the VMX bit [Leaf 1, ECX:5], or else the guest may bring up another level of virtualization. Other bits to be cleared include machine check architecture (MCA), because MCA issues are handled by the hypervisor. Today's OSs also use model-specific registers to detect the microcode version on the processor and to decide whether they need to perform a microcode update. For Xen, we decided to fake the update request, i.e., bump the microcode version number without changing the microcode itself.

Memory Virtualization

The virtual Memory Management Unit (MMU) module in the Xen hypervisor presents the abstraction of a hardware MMU to the HVM domain. HVM guests see guest physical addresses (GPAs), and this module translates GPAs to the appropriate machine physical addresses (MPAs).

IA-32 Memory Virtualization

The virtual MMU module supports all page table formats that can be used by the guest OS.

- For IA-32
    a. it supports 2-level page tables with 4 KB page size for 32-bit guests.
- For IA-32 Physical Address Extension (PAE)
    a. it supports 2-level page tables with 4 KB page sizes for 32-bit guests.
    b. it supports 3-level page tables with 4 KB and 2 MB page sizes for 32-bit PAE guests.
- For Intel EM64T
    a. it supports 2-level page tables with 4 KB page size for 32-bit guests.

b. it supports 3-level page tables with 4 KB and 2 MB page sizes for 32-bit PAE guests.

c. it supports 4-level page tables with 4 KB and 2 MB page sizes for 64-bit guests.

For the IA-32 architecture, this module maintains a shadow page table for the guest (Figure 3). This is the actual page table used by the processor during VMX operation, containing page table entries (PTEs) with machine page-frame numbers. Every time the guest modifies its page mapping, either by changing the content of a translation, creating a new translation, or removing an existing translation, the virtual MMU module will capture the modification and adjust the shadow page tables accordingly. Since Xen already has shadow page table code for paravirtualized guests, we extended the code to support fully virtualization guests. The resultant code handles paravirtualized and unmodified guests in a unified fashion.

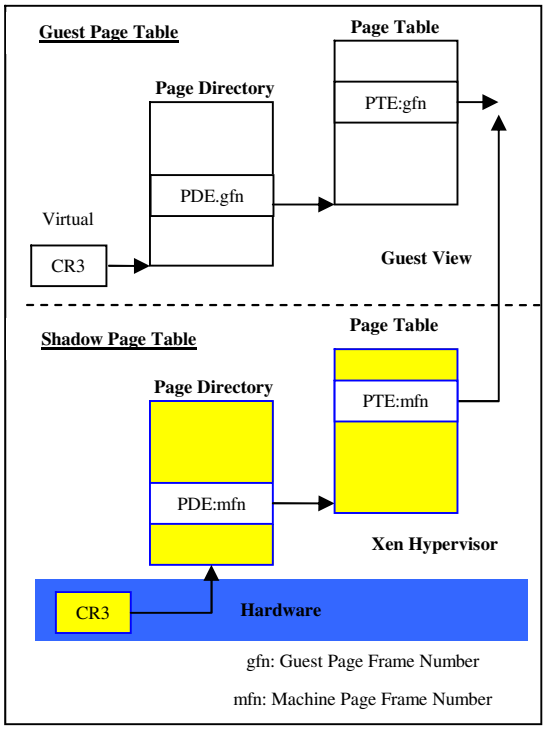

Figure 3: Shadow page table

From a performance point of view, the shadow page table code is the most critical for overall performance. The most rudimentary implementation includes the construction of shadow page tables from scratch every time the guest updates CR3 to request a TLB flush. This, however, will incur significant overhead. If we can tell which guest page table entries have been modified, we just need to clean up the affected shadow entries, allowing the existing shadow page tables to be reused.

The following algorithm is used to optimize shadow page table management:

- When allocating a shadow page upon page fault from the guest, write protect the corresponding *guest page table page*. This allows you to detect any attempt to modify the guest page table. For this to work, you need to find all translations that map the guest page table page. There are several optimizations for this as discussed below.
- Upon page fault against a guest page table page, save a "snapshot" of the page and give write permission to the page. The page is then added to an "out of sync" list with the information on such an attempt (i.e., which address, etc.). Now the guest can continue to update the page.
- When the guest executes an operation that results in the flush TLB operation, reflect all the entries on the "out of sync list" to the shadow page table. By comparing the snapshot and the current page in the guest page table, you can update the shadow page table efficiently by checking if the page frame numbers in the guest page tables are valid (i.e., contained in the domain).

Itanium Processor Architecture Memory Virtualization

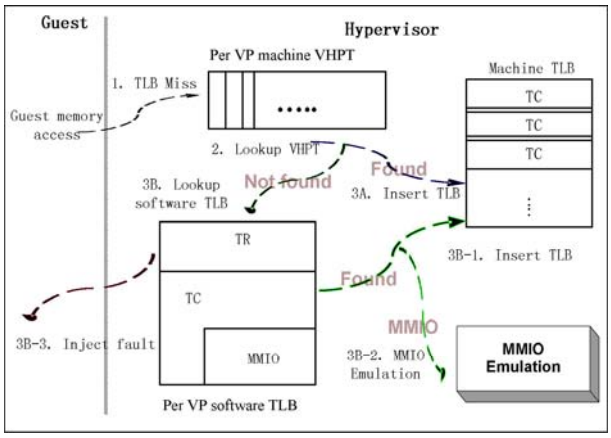

Figure 4: IPF TLB virtualization

The Itanium processor architecture defines Translation Register (TR) entries that can be used to statically map a range of virtual addresses to physical addresses. Translation Cache (TC) entries are used for dynamic mappings. Address translation entries can reside in either the TLB or in a Virtual Hash Page Table (VHPT). On a TLB miss, a hardware engine will walk the VHPT to extract the translation entry for the referenced address and insert the translation into the TLB.

Figure 4 illustrates the TLB virtualization logic in Xen. We extended the Xen hypervisor to capture all TLB

insertions and deletions initiated by a guest OS. This information is used to maintain the address translation for the guest. Two new data structures are added to Xen:

- The Machine VHPT is a per virtual CPU data structure. It is maintained by the hypervisor and tracks the translations for guest TR and TC entries mapping normal memory. It is walked by the hardware VHPT walker on a TLB miss.

  The Itanium processor architecture defines two formats for the VHPT. The short-format VHPT is meant to be used by an OS to implement linear page tables. The long-form VHPT has a larger foot print but supports protection keys and collision chains. We have extended the Xen hypervisor to use the long-form VHPT.

- The guest software TLB structure is used to track guest TRs and TCs mapping memory mapped I/O addresses or less than preferred page table entries. Access to these addresses must be intercepted and forwarded to the device model.

Region Identifier (RID) is an important component of the Itanium architecture virtual memory management system. It is used to uniquely identify a region of virtual address. Per Itanium architecture specifications, RID should have at least 18 bits and at most 24 bits. The exact number of RID bits implemented by a processor can be found by using the PAL_VM_SUMMARY call. An address lookup will require matching the RID as well as the virtual address.

Each IPF guest OS thinks it has unique ownership of the RIDs. If you allow two VT-i domains to run on the same processor with the same RID, you need to flush the machine TLB whenever a domain is switched out. This will have a significant negative impact on system performance.

The solution we used for Xen is to partition the RIDs between the domains. Specifically, we reserved several high-order bits from the RID as the guest identifier. The machine RID used for the guest is then a concatenation of the guest ID and the RID managed by the guest itself.

Machine_rid=guest_rid + (guest_id << 18)

As an illustration, if we have a CPU that support a 24-bit RID, the guest firmware inside the VT-i guest will report only 18-bit RID to the guest. The actual 24-bit RID installed into the machine will have the guest identifier in the upper 6-bit.

We also need two more RIDs per domain for guest physical mode emulation. The guest physical mode accesses are emulated by using a virtual address with special RIDs. This restricts the total number of IPF guests to 63.

This is a reasonable solution when the number of concurrent guests is limited and the guests are not running millions of processes concurrently. A more elaborate scheme is needed if this assumption is not true.

Device Virtualization

Figure 5 illustrates the device virtualization logic in Xen. The Virtual I/O devices (device models) in Dom0 provide the abstraction of a PC platform to the HVM domain. Each HVM domain sees an abstraction of a PC platform with a keyboard, mouse, real-time clock, 8259 programmable interrupt controller, 8254 programmable interval timer, CMOS, IDE disk, floppy, CDROM, and VGA/graphics.

To reduce the development effort, we reuse the device emulation module from the open source QEMU project [8]. Our basic design is to run an instance of the device models in Dom0 per HVM domain. Performance critical models like the Programmable Interrupt Timer (PIT) and the Programmable Interrupt Controller (PIC), are moved into the hypervisor.

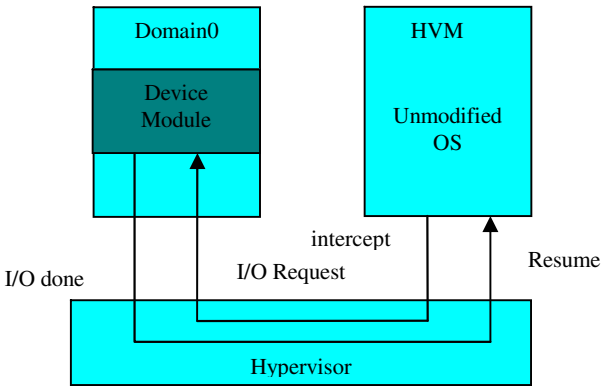

Figure 5: I/O Device virtualization

The primary function of the device model is to wait for an I/O event from the HVM guest and dispatch it to the appropriate device emulation model. Once the device emulation model completes the I/O request, it will respond back with the result. A shared memory between the device model and the Xen hypervisor is used for communication of I/O request and response.

The device model utilizes Xen's event channel mechanism and waits for events coming from the HVM domain via an event channel, with appropriate timeouts to support the internal timer mechanisms within these emulators.

I/O Port Accesses
We set up the I/O bitmap to intercept I/O port accesses by the guest. At each such VM exit, we collect exit qualification information such as port number, access size, direction, string or not, REP prefixed or not, etc. This information is packaged as an I/O request packet and sent to the device model in Dom0.

Following is an example of an I/O request handling from a HVM guest:

1. VM exit due to an I/O access.
2. Decode the instruction.
3. Make an I/O request packet (ioreq_t) describing the event.
4. Send the event to the device model in Dom0.
5. Wait for response for the I/O port and MMIO operation from the device model.
6. Unblock the HVM domain.
7. VMRESUME back to the guest OS.

Although this design significantly reduced our development efforts, almost all I/O operations require domain switches to Dom0 to run the device model, resulting in high CPU overhead and I/O latencies. To give HVM domains better I/O performance, we also ported Xen's Virtual Block Device (VBD) and Virtual Network Interface (VNIF) to HVM domains.

Memory-Mapped I/O Handling
Most devices require memory-mapped I/O to access the device registers. Critical interrupt controllers, such as I/O APIC, also require memory-mapped I/O access. We intercept these MMIO accesses as page faults.

On each VM exit due to page fault, you need to do the following:

- Check the PTE to see if the guest page-frame belongs to the MMIO range.
- If so, decode the instruction and send an I/O request packet to the device model in Dom0.
- Otherwise, hand the event to the shadow page code for handling.

The Itanium processor family supports memory-mapped I/O only. It implements the above logic in the page fault handler.

Interrupts Handling
The real local APICs and I/O APICs are owned and controlled by the Xen hypervisor. All external interrupts will cause VM exits. Interrupts owned by the hypervisor (e.g., the local APIC timer) are handled inside the hypervisor. Otherwise the handler in Dom0 is used if the interrupt is not used by the hypervisor. This way the HVM domain does not handle real external interrupts.

The HVM guests only see virtualized external interrupts. The device models can trigger a virtual external interrupt by sending an event to the interrupt controller (PIC or APIC) device model. The interrupt controller device model then injects a virtual external interrupt to the HVM guest on the next VM entry.

Virtual Device Drivers
The VBD and VNIF are based on a split driver pair where the front-end driver runs inside a guest domain while the backend driver runs inside Dom0 or an I/O VM. To port these drivers to HVM domains, we have to solve two major challenges:

1. Define a way to allow the hypervisor to access data inside the guest, based on a guest virtual address.

   We solved this problem by defining a copy_from_guest() hypercall that will walk the guest's page table and map the resulting physical pages into the hypervisor address space.

2. Define a way to signal Xen events to the virtual drivers. This must be done in a way that is consistent with the guest OSs device driver infrastructure.

   We solved this problem by implementing the driver as a fake PCI device driver with its own interrupt vector. This vector is communicated to the hypervisor via a hypercall. Subsequently, the hypervisor will use this vector to signal an event to the virtual device driver.

The send performance of the VNIF ported this way approximates that of the VNIF running in paravirtualized DomU. The receive throughput is lower. We are continuing our investigation.

PERFORMANCE TUNING VT-X GUESTS

In this section we describe the performance tuning exercise done to date for VT-x guests. The classic approach is to run a synthetic workload inside an HVM domain and compare the performance against the same workload running inside an identically configured paravirtualized domain. But to understand why the domain operates the way it does, we have to extend tools like Xentrace and Xenoprof to support HVM domains also.

Xentrace is a tool that can be used to trace events in the hypervisor. It can be used to count the occurrence of key

events and their handling time. We extended this tool to trace VT-x specific information such as VM exits, recording the exit cause and the handling time.

Xenoprof is a port of OProfile to the Xen environment. It is a tool that uses hardware performance counters to track clock cycle count, instruction retirements, TLB misses, and cache misses. Each time a counter fires, Xenoprof samples the program counter, thus allowing a profile to be built for the program hotspots. The original Xenoprof supports paravirtualized guests only. We extended this tool to support HVM domains.

A typical tuning experiment proceeds as follows:

1. Run a workload and use Xentrace to track the VM exit events occurring during the run.
2. Run a workload and use Xenoprof to profile the hotspots in the hypervisor.

We observed the bulk of the exits is caused by I/O instruction or shadow page table operations. I/O instructions have the longest handling time, requiring a context switch to Dom0. At one stage of our tuning experiment, 40% of the hypervisor time was spent in the shadow code.

Based on the above findings, we focused on tuning the I/O handler code and improving the shadow page handling.

- From the Xentrace result, we observed that the majority of the guest I/O accesses are to the PIC ports. This is because the guest timer handler needs regular access to PIC ports. By moving the PIC model to the hypervisor, we dramatically reduced the PIC handling time. Kernel build performance improved 14% and the CPU2k benchmark improved by 7%.
- The original QEMU IDE model handles IDE DMA operations in a synchronous fashion. When a guest starts an IDE DMA operation, the QEMU model will wait for the host to complete the DMA request. We added a new thread to handle DMA operations in an asynchronous fashion. This change increased guest kernel build performance by 8%.
- The original QEMU NIC model is implemented using a polling loop. We changed the code to an event driven design that will wait on the packet file descriptors. This change improved SCP performance by 10–40 times.
- The original QEMU VGA model emulated a graphics card. When the guest updates the screen, each video memory write causes a VM exit, and pixel data have to be forwarded to a VGA model in Dom0. To speed up graphics performance, we implemented a shared memory area between the QEMU model and the HVM guest. Guest video memory write will no longer cause a VM exit. The VGA model will update the screen periodically using data in the shared memory area. This improved XWindow performance dramatically by 5–1000 times.

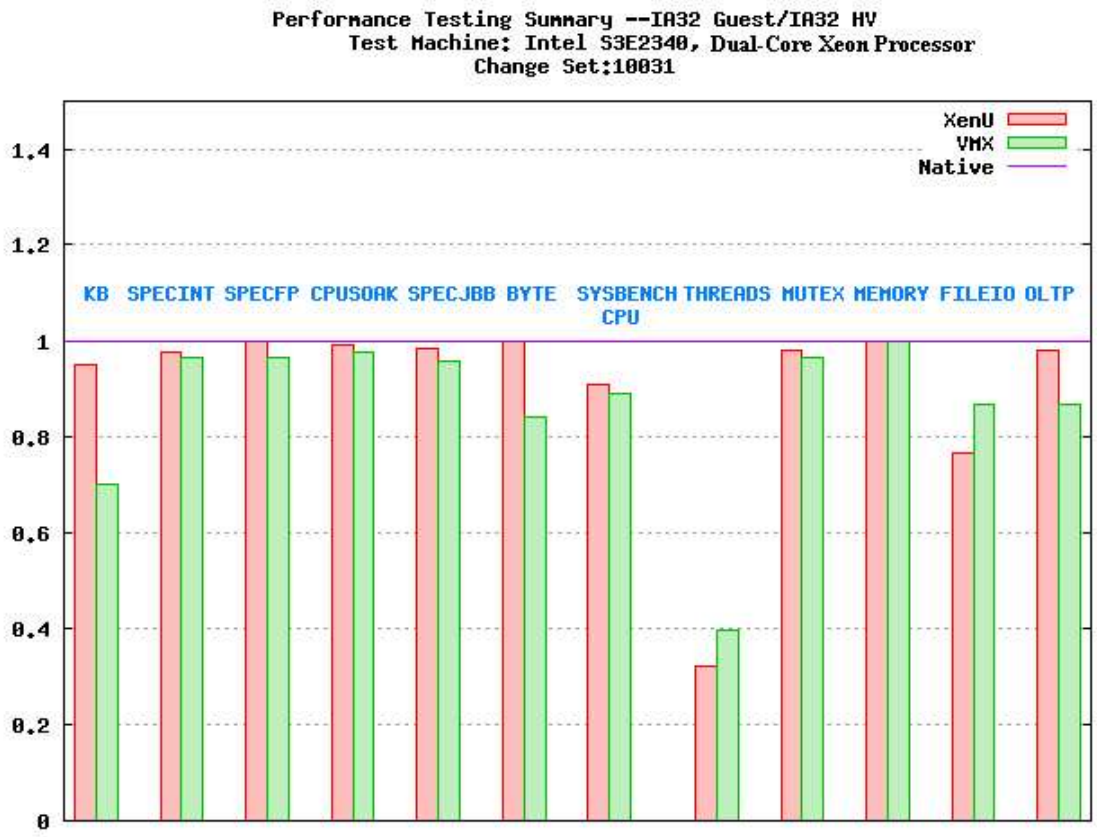

Figure 6: Performance comparison of paravirtualized vs. VT-x domain

BENCHMARK PERFORMANCE

Figure 6 compares the system performance results reported by various benchmarks when running in an identically configured paravirtualized domain and a VT-x domain. The performance of the same benchmark in a native environment is used as a reference. The data are collected on an Intel® S3E2340 platform, with 2.3 GHz/800 MHz FSB dual-core Intel® Xeon® processor, 4 GB of DDR2 533 MHz memory, a 160 GB Seagate SATA disk and an Intel® E100 Ethernet controller. RHEL4U1 is used as the OS in Dom0, DomU, and VT-x domains. Dom0 is configured with two virtual CPUs and 512 MB of memory. DomU and the VT-x domains are configured with a single virtual CPU with 512 M of memory and a 20 GB physical partition as its virtual disk.

CURRENT STATUS

As of this writing, Xen is under active development by Intel and various partners in the community. Readers interested in the latest status should consult the xen-devel* or xen-user* mailing list. Novell and RedHat are incorporating Xen into their upcoming releases. Virtual Iron and XenSource are developing products that will leverage Xen and Intel Virtualization Technology.

ACKNOWLEDGMENTS

The work described in this paper was made possible by many people. We thank our management for supporting this work. We acknowledge the many past and present members of the Xen team in the Intel® Open Source Technology Center for the many hours they spent developing and testing this code. Felix Leung, Alberto Munoz, and Mary Xie have provided immeasurable help for the VT-i project. A special thanks goes to Ian Pratt and Keir Fraser for working the full virtualization issues with us. Leendert van Doorn is the creator of the VMXAssist logic to execute real mode code. Their guidance and assistance throughout the course of this project has been invaluable.

REFERENCES

[1] Pratt, Ian; Fraser, Keir; Hand, Steve; Limpach, Christian; Warfield, Andrew; Magenheimer, Dan; Nakajima, Jun; Mallick, Asit; "Xen 3.0 and the Art of Virtualization," in *Proceedings of Linux Symposium, Volume Two*, 2005.

[2] Uhlig, R.; Neiger, G.; Rodgers, D.; Santoni, A.L.; Martins, F.C.M.; Anderson, A.V.; Bennett, S.M.; Kagi, A.; Leung, F.H.; Smith, L.; "Intel Virtualization Technology," *IEEE Computer Volume 38, Issue 5*, pp. 48–56, May 2005.

[3] Intel Virtualization Technology Specification for the IA-32 Architecture, at www.intel.com/technology/vt/, Intel Corp.

[4] Intel Virtualization Technology Specification for the Intel Itanium Architecture, at www.intel.com/technology/vt/, Intel Corp.

[5] Bochs IA-32 Emulator project, at http://bochs.sourceforge.net/*

[6] MultiProcessor Specification, at http://developer.intel.com/design/pentium/datashts/24201606.pdf, Intel Corp., Version 1.4, May 1997.

[7] Hewlett-Packard Corporation; Intel Corporation; Microsoft Corporation; Phoenix Technologies Ltd.; Toshiba Corporation; *Advanced Configuration and Power Interface Specification, Revision 3.0*, September 2, 2004.

[8] QEMU at http://fabrice.bellard.free.fr/qemu*

[9] Nakajima, Jun; Mallick, Asit; Pratt, Ian; Fraser, Keir, "X86-64 XenLinux: Architecture, Implementation, and Optimizations," *Ottawa Linux Symposium*, 2006.

AUTHORS' BIOGRAPHIES

Yaozu Dong is a technical lead in the Open Source Technology Center in Shanghai, PRC. He joined Intel in 1998 and had been involved in various embedded system projects from PalmOS* to Windows CE* to Linux, and several virtualization projects. He received his Bachelors and Masters degrees in Engineering from Shanghai Jiao Tong University, PRC. His e-mail address is eddie.dong at intel.com.

Shaofan Li is an engineering manager in the Open Source Technology Center in Shanghai, PRC. She joined Intel in 1999 and had been involved in IPMI, EFI, and several virtualization projects. She currently manages the Xen development team of PRC in the Intel Open Source Technology Center. Her team is focusing on enabling Intel Virtualization Technology in Xen for both IA-32 and Itanium architectures. She received her Bachelors and Masters degrees in Engineering from Shanghai Jiao Tong University, PRC. Her e-mail address is susie.li at intel.com.

Asit Mallick is a senior principal engineer leading the system software architecture in the Intel Open Source Technology Center. He joined Intel in 1992 and has worked on the development and porting of numerous operating systems to Intel architecture. Prior to joining Intel, he worked in Wipro Infotech, India on the development of networking software. Asit earned his Masters degree in Engineering from the Indian Institute of Science, India. His e-mail address is asit.k.mallick at intel.com.

Jun Nakajima is a principal engineer leading Linux and Xen projects at the Intel Open Source Technology Center. He is recognized as one of the key contributors to Xen, including Xen/XenLinux port to Intel EM64T, the major VT-x support codes, and the architecture. He has over 15 years of experience with operating system internals and an extensive background in processor architectures. Prior to joining Intel, he worked on various projects in the industry such as AT&T/USL Unix System V Releases (SVR) like the SVR4.2, and Chorus microkernel based fault-tolerant distributed SVR4. Jun earned his Bachelors of Engineering degree from the University of Tokyo in Japan. His e-mail is jun.nakajima at intel.com.

Kun Tian is a software engineer in the Open Source Technology Center in Shanghai, PRC. He joined Intel in 2003 and has been involved in Linux kernel development and virtualization-related projects. He is currently working on adding Intel Virtualization Technology to Xen for Itanium processor. He received his Masters degree in Engineering from the University of Electronic Science and Technology of China. His e-mail is kevin.tian at intel.com.

Xuefei Xu is a software engineer in the Open Source Technology Center in Shanghai, PRC. He joined Intel in 2003 and had been involved in several virtualization projects. He currently is working on adding Intel Virtualization Technology to Xen for Itanium. He received his Masters degree in Engineering from the Huazhong University of Science and Technology, PRC. His e-mail is anthony.xu at intel.com.

Fred Yang is a project lead in the Intel Open Source Technology Center in Santa Clara, California. He joined Intel in 1989 and had been involved in a series of operating system projects for Intel processors. He currently leads the team that is adding Intel Virtualization Technology to Xen for Itanium. He received his M.S. degree in Computer Science from the University of Texas at Arlington. His e-mail is fred.yang at intel.com.

Wilfred Yu is an engineering manager in the Intel Open Source Technology Center in Santa Clara, California. He joined Intel in 1983 and had been involved in a series of operating system projects for Intel processors. He currently manages the team that is adding Intel

Virtualization Technology to Xen. He received his Bachelors degree in Engineering from McGill University and his Masters of Applied Science and PhD degrees from the University of Waterloo, Canada. His e-mail is wilfred.yu at intel.com.

[Δ] Intel® Virtualization Technology requires a computer system with an enabled Intel® processor, BIOS, virtual machine monitor (VMM) and, for some uses, certain platform software enabled for it. Functionality, performance or other benefits will vary depending on hardware and software configurations and may require a BIOS update. Software applications may not be compatible with all operating systems. Please check with your application vendor.

[Φ] Intel® EM64T requires a computer system with a processor, chipset, BIOS, operating system, device drivers and applications enabled for Intel EM64T. Processor will not operate (including 32-bit operation) without an Intel EM64T-enabled BIOS. Performance will vary depending on your hardware and software configurations. See www.intel.com/info/em64t for more information including details on which processors support Intel EM64T or consult with your system vendor for more information.

THIS PAGE INTENTIONALLY LEFT BLANK

New Client Virtualization Usage Models Using Intel® Virtualization Technology

Mahendra Ramachandran, Digital Enterprise Group, Intel Corporation
Ned Smith, Digital Enterprise Group, Intel Corporation
Matthew Wood, Digital Home Group, Intel Corporation
Sharad Garg, Digital Home Group, Intel Corporation
Jim Stanley, Digital Home Group, Intel Corporation
Eswar Eduri, Software Solutions Group, Intel Corporation
Rinat Rappoport, Software Solutions Group, Intel Corporation
Arie Chobotaro, Software Solutions Group, Intel Corporation
Carl Klotz, Digital Enterprise Group, Intel Corporation
Lori Janz, Mobile Platforms Group, Intel Corporation

Index words: virtualization, operating systems, manageability, business computing, security

ABSTRACT

Intel's Embedded IT (EIT) strategy focuses on defining a set of usages aimed at benefiting IT departments and home PC customer support by providing advanced and remote capabilities for provisioning, manageability, diagnostics, and remediation of the client (desktop and mobile) platform. EIT leverages key platform technologies supported on Intel® vPro™ technology platforms and select digital home platforms, including Intel® Virtualization Technology$^{\Delta}$ (VT), Intel® Active Management Technology^{+} (AMT), and dual-core processors to deliver an innovative framework on which these capabilities may be implemented and enhanced.

One of these EIT usages enabled through the use of Intel VT is the Client Isolation and Recovery (CIR) usage model that emphasizes isolating manageability and security services in a virtual manageability appliance. IT departments benefit from this ability to isolate key services from end-user access while still maintaining the same level of flexibility and performance for end-user services. Additionally, the strategy anticipates that the manageability appliance will provide a rich environment for innovation for software vendors. The CIR usage model provides the ability to remotely manage the client PC during times when the primary operating environment is unavailable.

The other key usage models defined by EIT include Embedded PC Health, End-point Access Control, Outbreak Containment, and Agent Integrity and Assurance. The capabilities of these models are enhanced by the presence of Intel VT via isolation of the execution environments required by the IT manager compared to those exposed to the end user. In this paper we discuss how Intel VT enables a virtualized environment for a host of provisioning manageability and diagnostic applications for the IT professional.

INTRODUCTION

Intel® microprocessors and chipsets that support Intel Virtualization Technology (VT) make it feasible to run multiple operating systems (OSs) concurrently [1]. This enables the execution of multiple distinct protected execution environments that run in parallel. One such environment, the services or manageability partition, provides an isolated, controlled, and protected environment to support Embedded IT (EIT) on the platform.

EIT is Intel's strategy of embedding capabilities on the platform that enhance the overall manageability, security, and maintainability of the platform. The usages that define EIT in the business or office environment create a compelling value proposition for the use of virtualization technology on the platform. The challenges faced in the home computing environment present an opportunity to explore some of the key differentiators between the business and home computing environments.

The Intel® Lightweight Virtual Machine Monitor (LVMM) is a Virtual Machine Monitor (VMM) that partitions a client platform into two execution environments, using Intel VT, known as VT-x [2]. An execution environment is referred to as a virtual machine (VM) or a partition. One partition is the main user partition, and it can run a shrink-wrapped OS such as Windows XP*. The second partition is a services partition that runs a headless OS in an isolated execution environment. The user partition owns all the devices on the platform except for the network interface controllers. The latter are owned by the services partition, providing an ability to monitor and/or filter network traffic. Management applications that run in the services partition provide a remote console the ability to administer the client system.

In this paper we first present an overview of EIT and the capabilities that are enabled through the use of Intel VT. Next, we discuss the implications of using EIT in the home environment and follow that by an explanation of the VMM solution that we implemented for client virtualization. Finally, we conclude with a discussion of the implication of EIT on performance in the mobile environment.

EIT IN THE OFFICE

Enterprise IT departments are being asked to secure and manage an increasingly heterogeneous enterprise computing environment with less and less resources. IT departments face the need to satisfy multiple end-user client usage models and support requirements. Additionally, the IT manager faces a substantial increase in attacks to mission-critical applications and services with *for hire* attacks becoming more prevalent. As the enterprise increases in size, the scalability of existing manageability solutions is becoming a serious issue. Manageability solutions that require human intervention to discover, diagnose, and remediate system problems cannot scale to meet the requirements of large enterprise computing [4, 5]. One solution to these problems is to rely on the client platform's capability to secure, discover, diagnose, and remediate itself. In order for this to occur, manageability and security features need to be "embedded" into the client platform.

EIT Usage Models

Based on the issues IT departments face in managing their assets we came up with a set of usages that provide the capabilities required to address these issues. In this section we describe these usages and discuss how they address the challenges.

Client Isolation and Recovery

Among the challenges IT departments face today is the need to satisfy multiple end-user client usage models and support requirements. In response to these greatly varied requirements, the end user may even be granted "Administrator" rights on the client PC to install the custom software and hardware required to perform a specific job. Unfortunately, in this scenario, the end user leverages this access to install additional, non-IT validated software and hardware or disable IT security services. This results in unstable and unsecured PC configurations threatening the overall enterprise environment. Even though this additional un-validated software and hardware causes problems, end users still expect IT to support them when the client PC services and data stored on the PC become unreliable and unavailable, regardless of what the end-user Service Level of Agreement stipulates.

IT departments benefit from the ability to isolate key manageability and security services from end-user access while still maintaining the same level of flexibility and performance for end-user services. The Digital Office Embedded IT platform strategy emphasizes isolating manageability and security services to a virtual manageability appliance based on Intel VT via the CIR usage model. Additionally, the strategy anticipates that the manageability appliance will provide a rich environment for manageability and security vendors to innovate their product offerings. The CIR usage model provides the ability to remotely manage the client PC during times when the primary operating environment is unavailable. IT needs this "out-of-band" management capability in the client PCs to enable support when the end user most desires it.

The user of a CIR-enabled platform is a corporate user. The user is aware of the primary operating environment referred to as the User OS (the User Partition). IT management software runs isolated from the user's OS in its own appliance-like virtual Service Partition. In fact, the end user has no knowledge or awareness that the virtual Service Partition exists. Within the Service Partition a Service OS is used to provide an environment for IT manageability services to do the following:

- Disable malicious code or user actions.
- Prevent invalid/unsecured client configurations from adversely affecting resources on the production network.
- Patch or repair infected clients.
- Prevent situations when a worm or user deletes critical OS files.

Below are use cases of a CIR-enabled system. They are stated as problems from the customer's (either end-user or IT department) point of view.

- Virus detection and containment.
- Malicious code or the user can disable features such as Intrusion Detection, Firewall Capabilities, and Asset Management.
- Content access enforcement differs based on environment and location.
- Clients that have an invalid or unsecured configuration can adversely affect other clients on a production network.
- Infected clients cannot be patched or repaired.
- A worm or the user deleted critical OS files.

Endpoint Access Control

The Endpoint Access Control (EAC) usage, also known as Network Access Control, is a major feature of the Digital Office initiative. In the EAC, usage client access to an enterprise is contingent on the client platform being in an acceptable state. The enterprise determines the parameters of acceptability expressed in the form of an access policy. The policy is interpreted by a Policy Decision Point (PDP), which in response controls Policy Enforcement Points (PEPs) that respond by controlling access. Access controls can include any of the following:

- Unrestricted access.
- Conditional access based on traffic filtering.
- Restricted access where only specific resources are accessible.

EAC follows a methodology that can be broken down into the following general steps:

- *Collection*–monitoring, reading and storage of security measurements of the client system.
- *Reporting*–formatting collected measurements for consumption by a PDP.
- *Evaluation*–interpretation of reports and organizational policies.
- *Enforcement*–applies access control rules.
- *Remediation*–applies configuration rules designed to bring the platform into compliance.

The EIT strategy emphasizes distribution of PDP functionality to an Intel VT Service Partition through delegation. EAC policies relating to the evaluation of measurements is provisioned to a Service Partition-hosted PDP process. This process may evaluate measurements directly and forward a summary to the enterprise PDP, or measurements may be forwarded unmodified. The PDP response is interpreted by the Service Partition-hosted PDP process, and it maps the result to a format and structure that is meaningful to the client platform. See Figure 1 for the EAC architectural diagram.

The EIT strategy places a strong emphasis on locating enforcement mechanisms inside the client platform while continuing to extend control interfaces to the enterprise network. Protection of enforcement mechanisms from user applications is achieved through Intel VT to create a Service Partition. User applications and OSs function within a single User Partition. Partitioning of Host and Management environments provide isolation of EAC enforcement mechanisms and ensures that threats originating from the host environment will not defeat the goals of enterprise-controlled EAC.

The Service Partition is a *collection point* for host traffic destined for enterprise networks. Traffic filters that implement EAC enforcement policies are applied by a firewall contained in the Service Partition. Use of hardware-based filters, such as those implemented in the chipset, is under the control of the firewall process in the Service OS running in the Service Partition.

The Service Partition is an endpoint of communication for the platform. When connecting over an entrusted communication layer, a Virtual Private Network (VPN) must be constructed to establish a trustworthy connection to the enterprise network. VPN terminology broadly refers to any channel security protocol that provides data integrity or data confidentiality. A VPN therefore can be constructed at any layer in a network protocol stack. The client side of the VPN that is used for EAC originates within the Service Partition (and not the User Partition). The keys used to authenticate the endpoint and to protect channel data are managed by the Service Partition. Use of hardware-based encryption/decryption of network traffic is controlled by a VPN management process in the Service Partition. Even if packets are encrypted/decrypted in a hardware component, the logical endpoint of communication is still the VPN process.

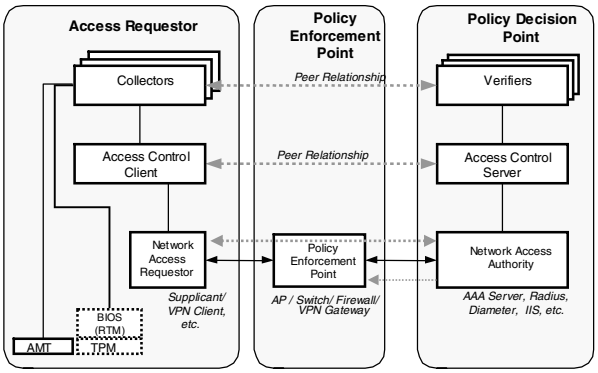

Figure 1: End point access control architecture

Outbreak Containment

Outbreak Containment (OC) provides the capability to contain the threat once an outbreak is detected. In a scenario where the client is infected, the client may be switched to a private network to enable remediation. In more serious threat scenarios, the client may be powered off to protect it from the network. In a known threat scenario, the client is updated with a patch to protect it against the outbreak.

The OC process starts when an outbreak is declared and enabled from the Management Console. The process is enabled by configuring OC filters that enable deep packet inspection for monitoring network traffic. The OC Filter Manager analyzes the collected data to assess the client health and generates a report for appropriate actions. The report can be either sent to a centralized Intrusion Prevention System (IPS), a decision-making system with a database for further analysis (Figure 2). The IPS will analyze the threat situation aggregating data from all clients. The Management Console gets the threat report from the database. If the report indicates a threat, an IT technician initiates steps to protect against the threat. In such a situation, the client is isolated from the network and a trusted out-of-band (OOB) channel is used to patch the client against the threat from network.

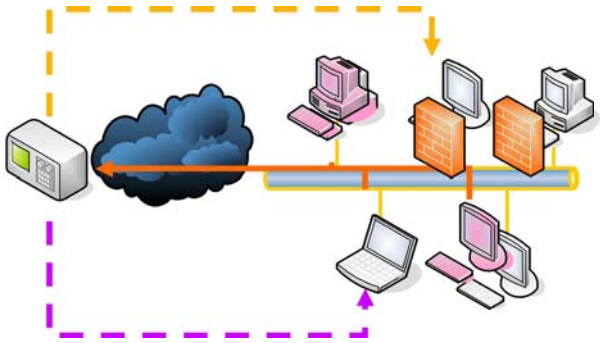

Figure 2: Outbreak containment applied at firewall

Embedded PC Health

Embedded PC Health (EPCH) usage reduces the client PC lifecycle costs by providing embedded asset management, provisioning, self-diagnostic, self-repair, and self-optimization capabilities within the Intel® platform. This OS-independent framework, based on AMT, utilizes platform-specific knowledge from Intel's processor, chipset and NIC. For details on AMT the reader is referred to [4].

This framework complements existing activities of system vendors services organizations, and manageability framework providers by adding persistent, secure, and reliable self-managing agents that support these autonomic frameworks. These programmable agents offer Independent Software Vendors and Original Equipment Manufacturers (OEMs) the ability to differentiate their offerings while benefiting from standardized capabilities and interfaces. They are accessible in an OOB mode, allowing for reconnaissance and management actions even if a PC has not yet been provisioned with an OS, or if the OS is dysfunctional.

The following is a summary of the main objectives of the EPCH:

- *Deployable*: Utilize currently deployed protocols and services in the IT environment. Minimize the need to develop and deploy new protocols and services.
- *Highly available*: Provide remote management capabilities regardless of the operational state of the PC hardware or OS.
- *OS-independence*: Provide a base set of platform management functions and interfaces regardless of the OS type or version installed on the PC.
- *Tamper-resistant*: Prevent the end user from removing or disabling the remote management service.

Security Implications

The addition of virtualization, service partition, and management processors to address security and reliability goals may seem at first counter productive due to increased overall complexity. Complexity implies greater opportunity for vulnerabilities to remain hidden and the threat of new attacks to continue.

The good news is EIT adds complexity where it is needed; it creates safer execution environments and improves the ability to detect and prevent attacks. Among these improvements is boot verification. A technique relied upon by malware is to silently install attack code into core OS files and in boot code. Each time the system boots, the malware is reinserted and

reinvoked. Anti-virus scanners are helpful, but can be spoofed by compromised OS code that lies about its existence.

In EIT systems, only code that is approved by IT or its manufacturer can be loaded. If attack code is successful in inserting itself into the system, the EIT verified boot procedure will detect the modification and apply an appropriate remediation action that can include failing to load the attack code or booting a safe-mode environment that hasn't been compromised.

Even legitimate code contains vulnerabilities that can be exploited by attackers. For example, a network driver is always subject to attacks on the networking protocols. If an attacker is successful in exploiting a vulnerability, the executable code in memory could become compromised. As systems become more reliable, they reboot less often making active attacks to memory more profitable to attackers. EIT is countering this threat by monitoring memory pages that should not change or should change in a prescribed way. Monitoring agents serve as integrity sentinels that notify the VMM whenever an invalid page access is attempted. The VMM can respond by blocking such accesses. Integrity Agents are themselves protected by a VM boundary where direct access between partitions is not allowed.

Should an attack be successful resulting in compromised EIT services, the VMM can respond by placing the platform into a more secure state. This can be achieved by alerting a management console, blocking I/O, and causing one or more VMs to cease operating. The latter is usually applied as a last resort if other corrective action fails and when a convenient time (for the user) can be identified. Automated and semi-automated dismantling of execution environments is analogous to boot verification; the core principle being that the system is always able to operate securely.

A fundamental tenet of EIT security mechanisms is the ability to create isolated execution environments that are less susceptible to attack. Intel VT and LaGrande Technology (LT) are instrumental in creating such environments. LaGrande Technology can be used to create a trusted environment even when most other parts of the system become compromised including memory, disk, and I/O. From this vantage point, it is possible to construct an environment from any remaining uncompromised components. By incorporating remediation capabilities into each primitive environment, actions can be taken that are most appropriate to the severity of the attack or failure [3].

EIT IN THE HOME ENVIRONMENT

Home IT, EIT for the home space, shares many common traits with the office EIT challenges and solutions. Like office EIT users, home users are experiencing an increasing number of attacks and spyware that degrade their experience or compromise their personal information. Home PC support organizations also spend millions of dollars, and users spend hours on the phone to diagnose and correct issues with their platforms. In many cases, both the time and monetary costs of support can be reduced using the same basic architecture as the office EIT solution, which allows the support organization to connect to the platform and diagnose problem even if the primary operating environment is unavailable. They both share a common VMM infrastructure, network isolation, and application model. The major differences are found in the connectivity model, privacy requirements, and absence of AMT.

The connectivity model differs from the enterprise model in that the managed platform is always located across the open Internet from the service provider and is highly likely to be located behind a NAT firewall. The NAT firewall presents challenges that are overcome in a couple of different ways, which are described later.

Privacy requirements in the home differ considerably from the enterprise environments. In the United States, corporate IT has complete access to a platform and all of the data stored within. This means that no permission from the user is required before IT administrators are able to access and maintain the system, access stored data, etc. In the home space, it is especially important for users to maintain control over how and when their platform and data are accessed by the IT service provider. The Home IT architecture ensures that the user is always the initiator of all service provider accesses, has visibility into all actions performed by the remote administrator, and has the right to restrict access to sensitive data.

Architectural Implications

The Home IT architecture is similar to that of Office EIT. The platform is virtualized using the LVMM and split into two VMs: the User OS (UOS) and Service OS (SOS). The SOS owns all PCI-based network devices, such as integrated Ethernet adapters, and virtualizes the devices into the UOS. The UOS owns all other platform devices, such as USB controllers, video chipsets, audio chipsets, and storage devices. A description of the LVMM is provided in the Client VMM section of this paper.

The SOS contains the Home IT control applications. The control applications work with agents in the UOS to

carry out manageability activities. The control applications use heartbeats to monitor the operation of agents in the UOS. If agents are not found to be running, a control application will signal the UOS to restart them.

The control applications communicate with the IT service provider using Web services. To request services, the control applications use Simple Object Access Protocol (SOAP) messages to the service provider. The applications accept commands from the service provider via a Web Services for Management (WS-MAN) interface. Each command is authenticated and checked for proper authentication before being routed to the proper control application for processing. Commands that require data from, or actions to be carried out in the UOS, are proxied via VMM channels to agents in the UOS, which report their status back to the SOS control applications. Figure 3 describes the architecture of Home IT systems.

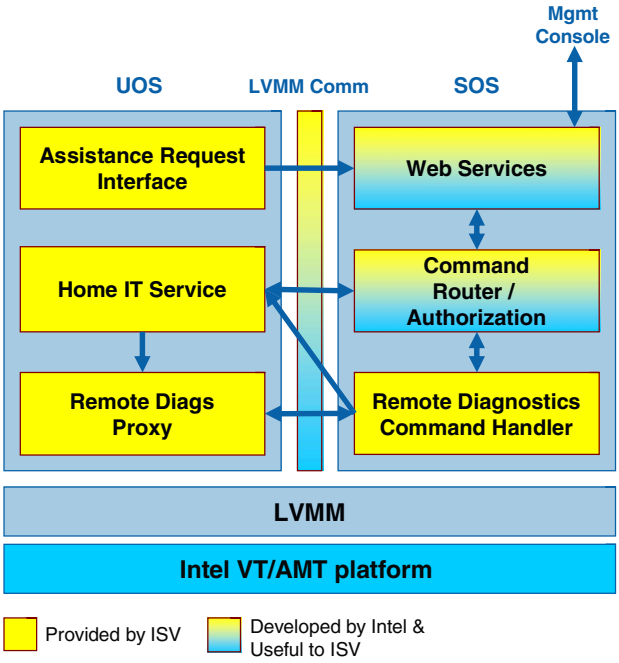

Figure 3: Home IT systems architecture

Connectivity Model

The primary Home IT connectivity model assumes that the platform will be located behind a broadband gateway router that implements a NAT firewall. In this configuration, the service provider is unable to directly connect to the SOS control applications as would be the case in the enterprise LAN configuration. The Home IT architecture includes two connection methods that can accommodate SP protocol requirements.

The first connection method is compatible with all broadband routers with NAT functionality. With this method, the service provider maintains a rendezvous server to receive incoming requests from customers. When the platform user requests assistance, the SOS control application connects to the rendezvous server and establishes a mutually authenticated Transport Layer Security (TLS) channel. Once the user's request has been sent to the service provider, the SOS processes WS-MAN commands over the channel. Service providers connect to the home platform via the rendezvous server, which proxies the commands and results between the management console and the SOS control application.

The second connection method requires the user's broadband router to support the UPnP* Forum's Internet Gateway Device (IGD) interface, but adds the flexibility of service-provider-initiated connections if necessary. For this method, the service provider maintains a registration server to receive requests from customers. When the platform user requests assistance, the SOS control application selects a random TCP port and uses the UPnP IGD interface to create a port mapping in the broadband router. The port mapping allows a connection from a specific external source to pass through the broadband router to a specific address and port on the internal network. The SOS control application then makes a Web services call to the registration server over a mutually authenticated TLS connection. The call submits the home network's WAN IP and mapped port, along with the user's problem statement and basic system configuration information, to the service provider. When a technician becomes available to assist the user, the management console obtains a waiting help request from the registration server and makes a mutually authenticated TLS connection to the user with the supplied WAN parameters. At this point, the administrator has the same capabilities as with the previous connection method. The difference is that multiple connections from the service provider to the SOS control apps can be created if necessary without involving the rendezvous server.

Privacy Considerations

In most of the world, the home environment requires a higher level of user privacy than the corporate environment. Where the corporate IT typically has complete access to a platform and its data for any reason, the home environment should afford the service provider the minimum necessary access to process user requests. This can be accomplished via a combination of policy and technical means.

Policy methods of protection require the service provider to create the Home IT software in such a way that it does not violate the user's rights. For instance, the software

should not communicate information about the user or platform to the service provider without the consent of the user. This includes user activity information and personal files. Personal files include user-created documents and anything in regions of the file system marked private by the user. Application information in the registry not directly connected to the proper configuration and operation of the program should also be out of reach of the service provider.

Technical methods of protection include encrypted or removable media for personal data, and user-specified file system filters enforced by the SOS control applications. Encrypted media is a simple solution that prevents service providers from gaining access to data by making it unintelligible, even if it is accessed by the service provider. Unfortunately, it has a number of key management complexities that could make it frustrating to the user. For instance, if the user forgets the encryption password or loses the encryption token, such as a smartcard, then the data become unreadable for the user unless there is an offline backup.

User-specified file system filters are a simple mechanism that can be enforced by the SOS control application's WS-MAN implementation. For this method, the user creates a list of file system locations that are off-limits to the Home IT agents in the UOS. The WS-MAN implementation then applies that filter to incoming commands from the service provider that either deny requests that include those file system locations, or filters the results to exclude those areas.

Authentication and Authorization

It is important to make sure that all parties and commands are properly authenticated and have the appropriate authorizations. The Home IT reference implementation uses mutually authenticated TLS and WS-security mechanisms to authenticate users. It employs access control lists in conjunction with WS-security mechanisms to enforce authorization constraints.

Authentication of communicating parties is accomplished via mutually authenticated TLS. When the SOS control applications for a platform are provisioned, they are given a unique TLS credential and one or more TLS root certificates to which the service provider components must be associated. On connection, the SOS control agent confirms that the remote party's certificate is issued by a valid root, and the service provider confirms that the SOS credential is valid and likely confirms that the user's account is in good standing. Since both parties possess a TLS credential, either one may act in the role of TLS "server" for establishing connections. If the only connection method supported by a service provider uses the rendezvous server, as described above, an alternative method of SOS authentication to the service provider exists. In this case, unilateral TLS authentication of the rendezvous server by the SOS control application can be used with a unique access token in the SOS. The SOS control application is authenticated via a challenge response exchange within the TLS tunnel. This reduces the Public Key Infrastructure (PKI) requirements of the service provider at the cost of connection flexibility.

The Home IT reference implementation uses an authorization mechanism based on WS-security with public key cryptography. For this mechanism, each administrator holds an authentication key pair used to sign WS-MAN commands. The SOS control application confirms the signature on the command and then checks an Access Control List (ACL), which includes the file system privacy filters described previously, that determines whether or not the command is permitted for that administrator. When the authorization model is fully implemented, the administrator does not have to sign each message, as the public key in the TLS client certificate shows the identity of the command issuer.

To make administration of permissions for a large number of administrators fairly easy, the Home IT authorization mechanism supports administrator groups and delegations. Administration groups allow the service provider to define administrator roles and permissions for each. The authorization mechanism then assigns one or more roles to the individual administrators. Administrators include WS-security tokens in the WS-MAN commands to assert their roles and associated permissions.

Delegation is supported in the authorization Home IT reference implementation to allow for situations where administrators may need to issue one or more commands that would normally be outside their permissions. In this case, a second administrator with the necessary permission issues a security token to the administrator handling the customer issue that grants the specific permission for a limited time. The administrator can then use that token to perform the additional action. It is up to the service provider policy to ensure that only appropriate delegations are used.

The combination of roles and delegation enable a number of useful scenarios for the service provider. For instance, the service provider can create a role for an automated triage component on the rendezvous server. The triage component would be capable of performing system inventory queries and possibly triggering routine operations such as virus or spyware scans, but would not be able to perform actions that modify the platform. Once the triage operations are complete, the case would

be passed to an administrator with the appropriate permissions to perform further diagnosis and repairs if necessary.

Another scenario involves multiple classes of administrator. In this case, the typical administrator has the permissions necessary to carry out common fixes, but not actions that are more "dangerous," including certain registry or driver modifications. If a more restricted action is necessary, the administrator must obtain a delegation of permission from a higher class of administrator, usually after the second administrator has reviewed the action for correctness.

CLIENT VIRTUAL MACHINE MONITORS

The Intel® LVMM architecture was designed with several goals in mind: maximize performance, have low complexity, maintain user experience, and provide an isolated execution environment for management applications that are always accessible and active.

High Performance

The Intel LVMM architecture was designed to maximize performance. As a result, the VMM itself virtualizes only the minimal set of devices required for allowing two distinct execution environments to execute concurrently, e.g., interrupt controllers and system timers. The LVMM allows the user partition direct access to most of the devices and therefore does not intercept I/O accesses made to those devices. This minimizes the overhead incurred by the LVMM on the user partition.

The network traffic of the user partition is handled by the services partition. The architecture depicted in Figure 4 shows that the network traffic flows through a physical NIC driver in the services partition, a bridge driver that routes the packets between the services partition network stack and the user partition network stack. In the user partition, a virtual NIC driver is responsible for sending all outgoing packets from the user partition to the bridge driver. The bridge driver forwards them to the physical NIC driver which in turn sends on the wire. Incoming packets are forwarded by the physical NIC driver to the bridge driver. The bridge driver forwards the incoming packets to the virtual NIC driver which in turn forwards them up the user partition network stack. This networking architecture provides a higher virtualization abstraction level. It performs better than a virtualization scheme that exposes a NIC device model to the user partition. In this scheme all the user partition accesses to the NIC device need to be intercepted and emulated.

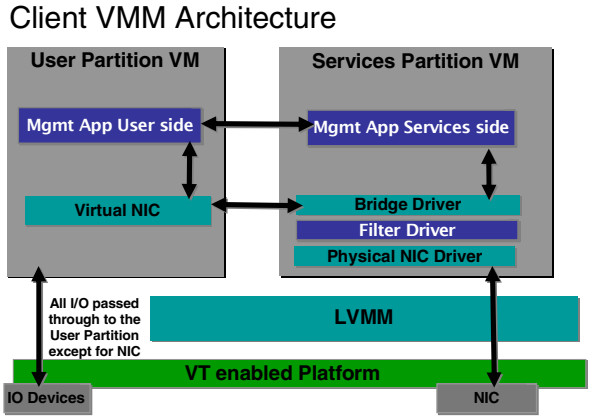

Figure 4: Client VMM architecture

The services partition has been modified to be aware of virtualization (paravirtualized). For example, the services partition interrupt handling is performed using a generic interrupt controller that is greatly simplified compared to a standard interrupt controller (e.g., Programmable Interrupt Controller a.k.a. 8259). The paravirtualization of the services partition simplifies the interaction with LVMM and saves unnecessary context transitions between them.

Unmodified User Experience

An important goal for the LVMM was to preserve the same user experience as that of a platform without the LVMM. The following are design decisions that were made in order to achieve this goal.

The design of the networking architecture guarantees that it is transparent to the end user and that the NICs are controlled by the services partition. All the functionality of the NICs controlled by the services partition, are exposed to the user partition. The virtual NIC driver in the user partition acts as a proxy for the physical NIC driver executing in the services partition. The bridge driver in the services partition provides the relay for packet data and control information between the virtual NIC driver and the physical NIC driver.

The ACPI policy decisions for configuration and power management operations of the platform are owned by the user partition. Any sleep state transition requested by the user partition is honored by the LVMM and services partition. For example, if the end user wants to transition the platform into "stand by" mode (sleep state S3) to preserve battery life, then the LVMM will eventually forward the request to the underlying platform. The user experience is preserved with respect to system power management, system thermal management, battery life, and sleep state usage models.

Isolated Execution Environment

In order to keep the partitions isolated from each other, there is a need to protect their physical memory from being tampered with by another partition. Memory accesses are performed by the CPU (e.g., any move to memory instruction) and are performed by devices through Direct Memory Access (DMA) operations. DMA allows a device with appropriate hardware to directly access system memory for data transfer without the intervention of the CPU.

The LVMM and the services partition have to be protected from memory accesses performed by user partition code. The LVMM needs to retain control over the physical memory, and thus over the processor's address-translation mechanism. We employ VT-x to prevent intentional or unintentional memory accesses from the user partition that may compromise the services partition or the LVMM.

The LVMM maintains an alternative page-table hierarchy that effectively caches translations derived from the hierarchy maintained by the OSs running in the user and services partitions. VT-x provides the necessary hooks for the LVMM to keep the alternative page-table hierarchy consistent with the OSs original page-table hierarchy. Such a hook is the trap on CR3 change. CR3 points to the base of the page-table hierarchy. Each time the OS switches to a different page-table hierarchy (i.e., changes the CR3 value), then the LVMM gets notified and switches to an alternative page-table hierarchy that matches the new OS page-table hierarchy. Since the LVMM controls the actual page tables, it can prevent a situation in which one partition has access to another partition's or the LVMM's physical memory. The LVMM prevents the existence of virtual to physical translations that map physical pages that do not belong to the partition.

The LVMM and the services partition have to be protected also from DMA bus mastering devices mapped to the user partition. These DMA-capable devices can access the entire system memory and can intentionally or unintentionally access (read/write) memory pages hosting the LVMM and services partition code and data structures. Such accesses could compromise IT secrets or render the platform useless by memory corruption. We employ Intel VT for Directed I/O (VT-d) to prevent such DMA-based attacks.

VT-d allows two views of the system memory: Guest Physical Address (GPA) and Host Physical Address (HPA). The LVMM keeps the HPA view which is the same as the system physical address space. The user and services partitions are provided their respective GPA views. The LVMM maintains shadow page tables to translate GPA to HPA for accesses from the CPU. Similarly, using VT-d DMA remapping engines and corresponding translation tables, the LVMM maintains GPA-to-HPA mapping for all DMA-capable I/O devices. Figure 5 illustrates this usage model.

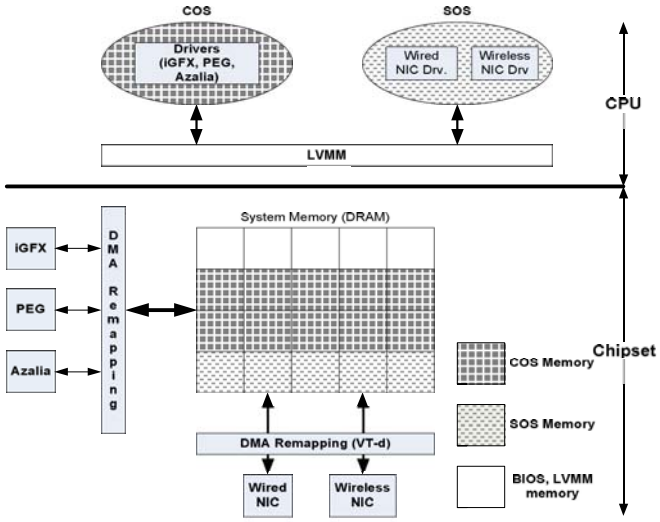

Figure 5: VT-d usage model in the client VMM

The mapping is performed as follows:

- All services partition memory pages are added to one domain such that only DMA devices mapped to services partition (NICs) can access these pages.
- All remaining pages (except LVMM and BIOS reserved) are added to the user partition domain, and all devices except those mapped to services partition can access these pages (e.g., iGFX, PCI/PCIe add-on cards etc.).
- The LVMM and BIOS reserved regions are protected from DMA accesses by virtue of being absent from the VT-d translation page tables.

The aforementioned device-to-domain mapping has the following benefits:

- I/O devices mapped to one domain can't access the memory of another domain. For example PCI/PCIe add-on cards in user partitions can't access the LVMM or the services partition.
- Device drivers in the services and user partitions run without any changes to comprehend GPA-to-HPA mapping. This translation is transparently performed by VT-d hardware when the device issues an I/O request using GPA.

If a device misbehaves by trying to access an address outside of the mapped domain, the VT-d hardware generates a fault. This fault is captured by LVMM and is indicated to the services partition. An optional management application in the services partition can

process these faults by taking appropriate actions such as displaying an error message or initiating a platform reboot, depending on the severity of the fault.

Always Accessible and Active
Management applications in the services partition are guaranteed connectivity with the external network allowing the platform to be managed even when the user partition has been isolated. The NICs are controlled by the services partition, and any action that the user partition attempts to make that can compromise the connectivity of the management applications is blocked. For example, if an action on the user partition disables the NIC, then it will get an indication that the NIC is disabled, although the real NIC remains enabled for use by the management applications.

This allows the services partition to be always accessible and reachable by a remote management console, so that management actions can be initiated.

Moreover, VT-x allows the services partition to run in parallel to the user partition. This means that the services partition is always active, and any diagnostics it runs can always be made available.

DISCUSSION

Intel LVMM vs. General-Purpose VMM

The LVMM is a custom VMM that was tailored for specific EIT usage models for the enterprise. General-purpose VMMs allow the user to create VMs that all run on the same platform. Examples of such general-purpose VMMs are VMware GSX* and Microsoft's Virtual PC*.

The main differences between the LVMM and a general-purpose VMM are as follows:

- *Number of partitions*: With a general-purpose VMM, the user can create and run multiple user partitions. In the case of the LVMM, only one user partition and a services partition are supported.
- *Partition configuration*: With a general-purpose VMM, the user can configure the execution environment of the partition. For example, the user can determine how much physical memory is allocated for the partition, and which OS to install on it. In the case of the LVMM, the execution environment of the user partition is the same as before the LVMM was installed on the system.
- *Performance*: It can be expected that with a general-purpose VMM, the performance overheads will be higher than with the LVMM. This is due to the fact that devices might not be directly assigned to the partition, and the VMM might need to emulate some devices (for use by multiple partitions).
- *User experience*: With a general-purpose VMM, the user experience is not preserved as it is with the LVMM. For example, if the end user wants to transition the platform into "stand by" mode (sleep state S3) to preserve battery life, then the LVMM will eventually forward the request to the underlying platform. A general-purpose VMM does not allow the user to directly control the platform. The power management actions are contained within the partition itself.

Mobile Performance Implications
The Intel LVMM architecture minimizes the impact on mobile battery life because, except for the network interface, the user partition owns all of the devices on the platform. A native system is a PC not virtualized, and running a single OS. Power tests executed in Intel's lab show the LVMM battery life within 95% of a native system when running local applications on the mobile machine. The LVMM achieves this by allowing the UOS to dictate the power management state of the platform.

The user partition runs an OS that complies with the Advanced Configuration and Power Interface (ACPI) specification. When the UOS requests the system to go to a certain S-state, the following occurs:

1. The LVMM intercepts the command.
2. The LVLMM sends the appropriate command back to the UOS so the UOS thinks the request has been executed.
3. The LVMM checks the SOS to see if there is any activity occurring.
    a. If yes, then the LVMM delays forwarding the S-state command to the platform.
    b. If no, then the S-state is forwarded to the platform, and the platform is placed in a lower power management state.
    c. If the UOS "wakes up" before the S-state command is executed at the platform level, then the LVMM will not change the platform power setting.

The SOS partition to support CIR, EAC, Embedded PC Health, OC, Security, and EIT in the home are all valuable usage models; however, when executed on a mobile machine, these usage models increase the number of interrupts on the system and impact battery life. Worse case, a mobile system running LVMM can use 60% of the battery life compared to a native system. In

this scenario intensive network traffic is occurring non-stop over a few hours.

Intel's LVMM architects and developers are working to improve the battery life by minimizing polling and interrupts, and optimizing memory footprint and caching. The mobile architecture team has also been educating ISVs on how mobile power management works and what they can do to ensure their software runs more efficiently to achieve better battery life.

CONCLUSION

As the usage models for virtualizations evolve, and platforms are enhanced with more capabilities, the LVMM may extend its capabilities in the following directions:

- *Multiple services partitions*–The current services partition suits the EIT manageability requirements. Other services partitions may offer other usage models such as VOIP.
- *Standard VMM API*–As more VMMs emerge in the market, there will be a need to standardize the API provided by the VMMs, so that various applications will be able to run on different VMMs.

Intel VT capabilities embedded in Intel's microprocessors and chipsets has enabled new capabilities in client systems (both desktop and mobile). In this paper we discussed how Intel VT working in concert with AMT and LT enables the design of novel solutions for embedding IT capabilities on the client platform.

ACKNOWLEDGMENTS

The authors would like to acknowledge the contributions of a number of people in Digital Office and Digital Home and Software Solutions Groups who were instrumental in the realization of the work described in this paper. In particular the following people have contributed to various parts of this work: Eugene Yarmosh, Ioan Scumpu, Yasser Rasheed, Abdul Bailey, Alok Kumar, Ved Shanbhogue, Romil Shah, Dhanu Agnihotri, Uttara Korad, Rao Pitla, and Shanyu Zhao.

REFERENCES

[1] Rich Uhlig, et al., "Intel Virtualization Technology," *IEEE Computer*, May 2005, pp. 48–56.

[2] Intel Corp., "Intel Virtualization Technology Specification for IA-32 Architecture," at www.intel.com/technology/vt.

[3] Trusted Computing Group, "TCG specification architecture overview," at https://www.trustedcomputinggroup.org/*

[4] Intel IT, "Reducing Enterprise Management Costs with Intel® Active Management Technology," at ftp://download.intel.com/it/digital-office/active-management-technology.pdf

[5] D. Busch, G. Bryant, B. Sayles, T. Swinford, "The Digital Office: Cross-Platform Embedded IT for Manageability, Security, and Connectivity," *Technology@Intel Magazine*, September 2004.

AUTHORS' BIOGRAPHIES

Mahendra Ramachandran is a staff architect in Intel's Digital Enterprise Group. He is responsible for driving the Embedded IT architecture definition for the Digital Office Platforms Division. Mahendra has been at Intel for eight years focusing on architecting manageability solutions for both client and server platforms. He is involved in several working groups of the DMTF. Mahendra earned his Ph.D. degree in Computer Science from Ohio State University in Columbus, Ohio. His e-mail is mahendra.a.ramachandran at intel.com.

Ned Smith is a senior security architect in Intel's Digital Enterprise Group. He is responsible for architecting security solutions for Digital Office Platforms. Ned is the co-chair of the Infrastructure Working Group in the Trusted Computing Group. He earned his M.S. degree in Computer Science from Portland State University, Portland Oregon and his B.S. degree in Computer Science from Brigham Young University, Provo, Utah. His e-mail is ned.smith at intel.com.

Matthew Wood is a security architect within the Digital Home Group, and he has extensive experience implementing authorization systems, cryptographic middleware, firewalls, and platform security features in hardware. His e-mail is matthew.d.wood at intel.com.

Sharad Garg is an architect and engineering manager within the Digital Home Group. His research interests include parallel programming, distributed file systems and storage, and manageability. His e-mail is sharad.garg at intel.com.

Jim Stanley is a senior software engineer within the Digital Home Group, and he has extensive networking, systems, and middleware programming and embedded systems experience. His e-mail is jim.stanley at intel.com.

Eswar Eduri is a software architect in SSG/IPSD. His technical interests are data communication protocols and networks, operating systems, and real-time embedded systems. Eswar has a Masters of Technology in

Electrical Engineering from IIT-Chennai, India. His e-mail is eswar.m.eduri at intel.com.

Rinat Rappoport is a software architect in SSG/IPSD. Her technical interests are processor architecture, operating systems, and real-time embedded systems. Rinat has a Masters of Science degree in Computer Science from Technion, Israel. Her e-mail is rinat.rappoport at intel.com.

Arie Chobotaro is a software engineer in SSG/IPSD. His technical interests are processor architecture, operating systems, and real-time embedded systems. Arie has a Bachelor of Science degree in Computer Science from Technion, Israel. His e-mail is arie.chobotaro at intel.com.

Carl Klotz, Jr. manages the Digital Office Platform and Solutions Development Group and he is an architect of the digital office Embedded IT appliance. He and his team have defined the solution architecture for Intel's digital office Embedded IT program. Carl is knowledgeable in various client manageability and security standards and various Intel platform technologies such as Intel Virtualization Technology, Intel Active Management Technology and LaGrande technology. Carl joined Intel in 1995 and for the past 11 years has focused his efforts on evolving Intel's platforms to be increasingly more manageable and secure. In his spare time he enjoys snowboarding, running, and memorizing long strings of random numbers. His e-mail is carl.klotz at intel.com.

Lori Janz is a program manager within the Mobile Platforms Group. She is responsible for managing the software program management team that leads all of Mobile's software platform programs as well as Mobile initiatives and technologies. Her e-mail is lori.a.janz at intel.com

[Δ] Intel® Virtualization Technology requires a computer system with an enabled Intel® processor, BIOS, virtual machine monitor (VMM) and, for some uses, certain platform software enabled for it. Functionality, performance or other benefits will vary depending on hardware and software configurations and may require a BIOS update. Software applications may not be compatible with all operating systems. Please check with your application vendor.

[+] Intel® Active Management Technology requires the computer to have additional hardware and software, connection with a power source, and a network connection. Check with your PC manufacturer for details.

Intel® Virtualization Technology in Embedded and Communications Infrastructure Applications

Dean Neumann, Communication Infrastructure Architecture & Planning, Intel Corporation
Dileep Kulkarni, Communication Infrastructure Architecture & Planning, Intel Corporation
Aaron Kunze, Corporate Technology Group, Intel Corporation
Gerald Rogers, Infrastructure Processor Division, Intel Corporation
Edwin Verplanke, Infrastructure Processor Division, Intel Corporation

Index words: virtualization, virtual machine monitor (VMM), hypervisor, real-time operating system (RTOS), multi-core, isolation, consolidation, embedded, communications, industrial control, aerospace

ABSTRACT

Intel® Virtualization Technology$^{\Delta}$ delivers improved computing benefits for home users, business users, and IT managers alike. This paper describes the unique requirements that embedded systems and communications infrastructure equipment place on virtualized environments and shows how Intel is working with a number of third parties to extend the benefits of Intel Virtualization Technology to these market segments. Bounded real-time performance can be maintained while using virtualization to consolidate systems; system uptime can be increased by enabling software failover without redundant hardware; and software migration can be performed without bringing down the application. Virtualization also allows legacy applications to co-exist with new applications by executing both software environments in parallel, and it provides the means for applications to take advantage of multi-core processors without re-architecting for multi-threaded execution.

INTRODUCTION

As in the desktop, mobile, or IT server domains, embedded systems and communications infrastructure equipment can also realize several benefits from virtualizing the hardware execution environment, so that multiple operating systems (OSs) can share the common resources of the hardware platform. These benefits may be realized in terms of cost reduction (either by reducing capital costs or operational costs), in terms of increased performance and functionality, or in terms of increased system reliability and security. There are also several different "usage models" (mechanisms by which virtualization can be used) which provide these benefits. In this paper we survey several of these models within embedded and communication systems.

Regardless of the mechanism by which virtualization is used, one commonality between usage models is that it is always necessary for an additional layer of software to exist that schedules the operating systems which share the hardware platform, manages the resources assigned to each OS, and saves/restores state when context switching between the OSs. In this way each OS executes within a "virtual machine" (VM) rather than on a physical machine. This additional layer of software, the Virtual Machine Monitor (VMM), manages the execution of OSs in much the same way that OSs manage the execution of applications.

Although the existence of a VMM is common to all usage models, we shall see that the architecture of these VMMs is tailored to the constraints of the market segments they address, and that different design decisions must be made to optimize the VMM for the specific requirements of each market segment. It is not only the VMM that is tailored to different market segments, but also the OSs that execute within the VMs that must be tailored to the requirements of these market segments. Whereas a General Purpose-Operating System (GPOS) such as Linux*, Microsoft Windows* or Microsoft Windows Server* addresses the requirements of desktop or IT server environments, a different class of OS—the Real-Time Operating System (RTOS)—is required to address the requirements of embedded and communications systems. As we shall see, providing specific real-time behaviors in

a virtualized environment becomes an overriding factor in embedded system design.

EMBEDDED VMM DESIGN CONSIDERATIONS

Embedded systems have requirements that make software design for these systems different than software design for a server or desktop environment. These requirements come from many factors, including the closed nature of the systems and the real-time workloads that often run on these systems. The design requirements of these devices impact the design of VMMs that are targeted to this environment. Although there are some similarities in the requirements of many embedded systems, even within the embedded systems market segments, different vertical market segments have different requirements. Before Intel released processors with Intel Virtualization Technology (Intel VT), the complexity and costs involved in developing a VMM for different embedded systems market segments was extremely high. With Intel VT, however, the unique requirements of embedded systems can be inexpensively met by targeted products like those highlighted in this paper.

In this paper, we highlight three design considerations for VMMs for an embedded system. These design considerations include the unique isolation requirements, the prevalence of static VMs, and support for real-time workloads.

Unique Isolation Requirements

One difference between the requirements of an embedded VMM and a general-purpose VMM affects how well VMs are isolated from one another. Most embedded systems are closed systems, where all of the software is either written by or installed by a single vendor, and where end users do not have the flexibility to run their own software. In some cases this can reduce the isolation requirements that exist in general-purpose VMMs. In embedded systems, this reduced isolation often increases performance or increases the predictability of performance. In other cases such as security-critical applications, the isolation requirements are increased rather than reduced, often to the point of requiring formal analysis and certification.

There are two examples of optimizations supported by embedded VMMs that capitalize on reduced isolation requirements. First, many real-time OSs run without paging [1]. Page walks make it difficult for real-time systems to predict the performance of code and meet real-time guarantees efficiently. They can also result in a reduction in overall performance. In an Intel VT-enabled system, running guests with paging disabled reduces the isolation of that guest—the guest can read and write all of physical memory, including that of devices or other guests. This is a tradeoff that would not be acceptable in a general-purpose environment, but may be preferred in some embedded environments. Second, I/O performance can be critical to the success of an embedded system. This has caused some embedded VMM vendors to allow VMs to have direct access to Direct Memory Access (DMA)-capable devices. This increases performance, but would allow a malicious application or device driver to use the device to read and write the memory of the other guest OSs on the system. Again, in environments where the software environment is fixed, this tradeoff might be acceptable.

Even though some embedded environments will accept reduced isolation between guests, there are exceptions with other embedded environments. Specifically, security-critical and safety-critical environments like those supported by LynuxWorks would not allow these types of tradeoffs [2]. Again, Intel VT reduces the cost of producing a VMM so that the development of targeted VMMs with different design tradeoffs is possible.

Static Virtual Machines

Many embedded systems are designed with the knowledge of exactly what workloads will be run at all times. They are also designed knowing exactly what hardware will be available on the system. This allows designers to make design-time choices about how to allocate hardware to the tasks running in the workload. As such, when using an embedded OS it is typical for the designer to statically allocate specific cores and specific regions of memory to processing tasks. This reduces the emphasis on dynamic scheduling and dynamic memory management that is often important in a GPOS. Although scheduling multiple tasks that are running on the same core is still important, especially in an RTOS, deciding which cores will run which particular tasks is less important.

In a virtualized environment, the VMM is responsible for performing processor scheduling and memory management for the guest OSs just as the OS is responsible for processor scheduling and memory management for applications. Therefore, many of the requirements that apply to OSs in an embedded environment also apply to VMMs in an embedded environment. With this in mind, VMMs can be designed so that system designers can make static configuration choices at design time with respect to how resources are allocated to VMs. This changes the design of VMMs for embedded environments in two ways. First, the design of the scheduler is simplified since the designer will likely statically map tasks to processor cores manually. The scheduler must still schedule between VMs running on the same core, and must do so using different scheduling

policies than would be used in desktop or IT data center domains, but scheduling between cores is simplified. Second, memory management is designed more for configurability than for dynamic reallocation. VMMs rely on the designer to specify which memory ranges should be given to each VM. This simplifies the design of memory managers within the VMM, but increases the complexity of configuring the VMM.

Real-time System Support

Many embedded systems are required to support real-time workloads. One of the reasons why GPOSs such as Microsoft Windows cannot provide real-time control is due to the limitations of the scheduler. The scheduler in this case is the entity which determines how much time a specific process or thread is allowed to spend on a given processor. Microsoft Windows uses a quantum-based, preemptive priority scheduling algorithm. This basically means that threads with equal priority are scheduled round robin and threads with higher priority are serviced above threads with a lower priority. Linux uses a priority-based scheduling algorithm as well. An RTOS, however, often schedules tasks using a strict priority scheduler [3]. A priority scheduler will let higher priority tasks run for as long as they have work to do, whereas a general-purpose scheduler tries to guarantee that no task can prevent another from progressing.

Supporting real-time workloads in a virtualized environment is a challenge. A VMM does not typically have visibility into the individual tasks running in an RTOS, nor does it have knowledge about their priority. The solutions that embedded VMM vendors are using today is to isolate RTOSs alone on a processor core, or to only allow the RTOS to share a core with a GPOS and give the RTOS strict priority over the GPOS.

There are also differences between a real-time environment and a general-purpose environment in the way interrupts are handled. In many embedded applications, interrupts generated by external hardware have to be serviced within a specific time frame. GPOSs do not necessarily have this capability built in as they can frequently turn off interrupt processing for an unbounded amount of time. This kind of behavior is unacceptable for applications that require real-time control.

For real-time interrupt handling, Intel VT plays a key role. With Intel VT, the system can be configured such that when a guest disables interrupts, only the delivery of interrupts to itself is disabled and not the delivery to other guests. Therefore, if a GPOS guest were to disable interrupts, and an interrupt for a real-time device were to be asserted, the VMM could still decide to interrupt an RTOS that required small, predictable interrupt latencies. On the other hand, if an interrupt targeted for the GPOS would occur while the RTOS is executing a critical task, the VMM should delay delivery of the interrupt until the RTOS has finished execution of the time-critical task to maintain determinism.

Many embedded systems also require strict prioritization of I/O traffic between code running at different priority levels [4]. For example, if multiple VMs are to access a network device, the application may require some quality of service (QoS) guarantees be enforced between the guests. These guarantees may come in the form of bandwidth or latency guarantees. Such requirements are not typically found in a general-purpose system. This complicates the design of the infrastructure used to share devices between VMs.

Embedded systems have unique design criteria that have a large impact on how VMMs are designed for such systems. Intel VT reduces the cost of producing a VMM such that targeting an individual embedded environment is a possibility. It allows VMM vendors to focus on how they meet the requirements of their market segment without dealing as much with the complexities of virtualizing system hardware.

VIRTUALIZATION IN INDUSTRIAL CONTROL

Many industrial control systems feature highly visual human interfaces that depict the process under control, which may be a medical device, an industrial plant, an assembly line, etc. These displays may also involve rapidly changing data, or they may include interfaces to network-accessible databases to access schematic diagrams or diagnostic and maintenance procedures. Such systems therefore benefit from the ubiquity and richness of GPOSs such as Microsoft Windows and the computational performance of powerful CPUs. Yet they also require strict real-time control to ensure that robotic machines assemble parts with exacting precision, move gantries and X-ray machines to exact locations, operate switches and actuators at precise times, or perform functions for exact durations. Many require closed-loop feedback control systems: for example, if a sensor detects that a machine has reached a specific point, that sensor sends a signal that must be acted upon by the control software within a timeframe that is measured in microseconds in order that the machine can be halted in that exact position without variance.

As we noted above, GPOSs such as Microsoft Windows are not suitable for performing this level of real-time control because they are designed to share processor resources fairly between running processes, thereby preventing “starvation” of some processes. Traditional industrial control systems therefore typically separate their

processing and control functions into a Master Station component that implements the human computer interface, database interface, and other non-real-time management functions, and separate Remote Terminal Units (RTUs) which are small, rugged computers that implement the real-time control functions, traditionally using separate Programmable Logic Controllers for that purpose. These RTUs contain the sensors and actuators to detect and control the operating environment, along with sufficient computing performance to react to environmental changes and control commands extremely quickly, and to execute the communications protocol stack to exchange data with the Master Station. RTUs frequently must also operate within extreme temperature, humidity, vibration, or other environmental conditions that are more challenging for the electronics than for the mechanical components. These environmental conditions restrict the choice of computing components that can be employed, or they increase the cost of those components, particularly if ample performance headroom is desired to support future functionality. Therefore, for reasons of cost reduction and the desire for a common, scalable infrastructure upon which greater functionality can be layered, trends in industrial control are toward a consolidated platform that can provide the required separation of the real-time control functionality from the non-real-time functions [5]. In this way RTUs can be simplified, while the Master Station provides real-time control that can be given absolute priority so that signals can be operated upon within strictly defined bounded timeframes, while still providing the rich graphical environment, databases, and device support of a GPOS.

Scheduler and Interrupt Latencies for Virtualized Solutions in Industrial Control

Although the graphical user interface of a GPOS such as Microsoft Windows or Linux combined with XWindows meets the requirements for control of medical equipment or industrial equipment, the applications' response times are often not acceptable. The *average* interrupt latency provided by a GPOS may indeed be within acceptable limits (on the order of 5-20 microseconds), but the *worst-case* latency must be designed for, and this may be orders of magnitude too long in a GPOS.

The main reasons why GPOSs are not suitable are due to the policies of the scheduler and the design of the kernel. The scheduler is the entity that determines how much time a specific task/process/thread is allowed to spend on a given processor. Modern GPOS schedulers allow users to provide an element of application scheduling control. Both Windows and Linux OSs provide such features. Still, even when equipped with these kinds of features, the scheduling algorithms utilized by GPOSs do not provide sufficient real-time control.

The scheduler behavior has an immediate effect on interrupt handling. When controlling a critical process, interrupts generated by the equipment under control must be serviced within a very specific and bounded time frame. GPOSs do not have this capability, as high priority tasks/processes or threads can take priority. This behavior is unacceptable for applications that require real-time control. The design of the OS kernel may also include critical sections of code that must execute atomically, and therefore interrupts must be disabled during these critical sections, thereby causing the worst-case latency to increase by the length of the longest critical section in the OS.

Today there are different solutions available to provide this element of real-time control. Some solutions provide a real-time kernel and run the GPOS and graphical user interface within a complete thread. An API is defined that allows the GPOS to interact with the real-time kernel. If the requirement for thread scheduling exists, an application within the GPOS would utilize the thread mechanism offered by the real-time kernel.

Another approach is to run a small real-time kernel along with the GPOS. Both OSs share the CPU, memory, and interrupt controller. However, each version of the kernel has its own context (descriptor tables, memory management etc.). The real-time kernel determines when specific processes have to be executed to maintain determinism, and interrupt delivery is also controlled so that it does not affect the behavior of the system. If an interrupt occurs during the execution of a real-time task, the real-time kernel will not necessarily execute the interrupt service routine if it is associated with the GPOS. Instead it will continue execution of the real-time task and on completion it will hand over control to the GPOS, which will then deal with the interrupt.

Designing in determinism in a GPOS this way can be very complex. The OS source may not be available: with this in mind, some assumptions must be made in order to predict the behavior of the general-purpose kernel.

Commercial Virtualization Solutions for Industrial Control Applications

Commercial solutions exist that meet the stringent latency constraints of real-time control while still providing the rich and ubiquitous development framework of Microsoft Windows. Products such as TenAsys Corporation's INtime* employ the hardware capabilities of Intel's processors with Intel VT to provide extremely low latencies (worst-case measured as low as 3 microseconds) and real-time capabilities for Microsoft Windows, while allowing Visual Studio* to be used for development in both the Windows environment and the INtime environment. This enables developers to create and

deploy sophisticated real-time applications without trying to force a GPOS or device driver to achieve real-time performance.

VIRTUALIZATION IN COMMUNICATIONS NETWORKS

Communication Usage Models

As Intel Corporation and other vendors migrate towards multi-core processors, communications equipment manufacturers are changing their programming paradigms to take advantage of these additional cores. Communications equipment tends to utilize highly specialized software that has been optimized and validated to execute as sequential logic. Thus, it is not easily ported to a multi-core platform. By eliminating the need for equipment manufacturers to refactor their software for multi-threaded execution, Intel VT makes this migration simpler. Equipment manufacturers can instead execute multiple instances of their single-threaded software, each within a separate VM, each processing a portion of the total workload. A suitably architected VMM provides the software infrastructure necessary to distribute the workload between VMs. Examples of multi-core migration include multiple Home Location Registers in a cellular network; or splitting workloads between intrusion detection systems.

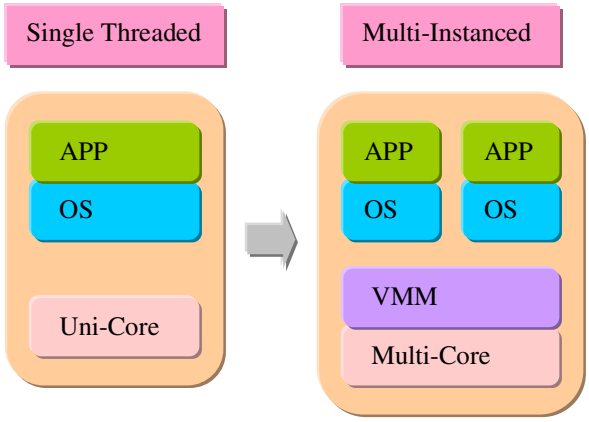

Figure 1: Virtualized vs. non-virtualized environment

Consolidation is common across all market segments, but offers unique benefits in communication market segments. Telecommunications Equipment Manufacturers could utilize a VMM to consolidate multiple instances of an older legacy single threaded application on a multi-core platform, avoiding the need to spend expensive R&D cycles on modifying legacy code to take advantage of multi-core architectures (see Figure 1). Much of the communication equipment processing is split between Data Plane, Control Plane, and Management Plane processing. Each plane has different processing requirements, memory latency and bandwidth requirements, and network I/O requirements. By using Intel VT and a real-time VMM, a manufacturer can consolidate these different planes onto fewer processing elements. This reduces equipment and operational costs, and these savings allow the equipment manufacturers as well as their customers (the service providers) to remain competitive. An example of such a consolidation is in the Mobile Wireless business where a system for determining the current location of a mobile unit, called a Home Location Register (HLR), exists. Many of these systems are proprietary in nature, and restricted to 32-bit addressing. Using Intel VT, more than one HLR can be collocated onto a single system. The VMM allows for the splitting of workloads to multiple HLRs, and allows for a HLR database to be greater than 4 GB in size.

A unique requirement of communication systems is their extremely high reliability. Communication systems may be required to be available to process calls 99.999% of the time. This corresponds to less than five minutes *per year* of downtime, which includes all scheduled maintenance, software and hardware upgrades, and system corrective actions. In comparison, we may spend five minutes *per day* brushing our teeth, so communication systems permit approximately 1/300th the maintenance that we perform on our teeth. Due to the implications on software design, today only high-end communication systems can provide this level of reliability. With Intel VT, communication systems can provide greater availability without the traditional software infrastructure costs. Many of these reliability issues arise from the customized nature of the communication software. Intel VT provides for software fault isolation on all levels of communication systems. This is achieved by allowing Active and Standby instances of the executing software, each within its own VM. In the event of a software failure, the Standby instance will continue execution and assume Active status, while the failed instance is restarted by the VMM. With this capability, the cost of a software fault, which has traditionally been protected against via redundant hardware, is eliminated.

In addition to redundancy, the ability to perform live upgrades of software is accomplished by providing redundant hardware components. As indicated in Figure 2, a Standby partition could be used for either hot upgrades or fault tolerance. With Intel VT, the need for redundant hardware is eliminated. Now simply upgrading the standby instance, restarting it, and designating it the Active instance accomplishes the software upgrade. In the event the new software fails, the previous software version is still available to fall back on.

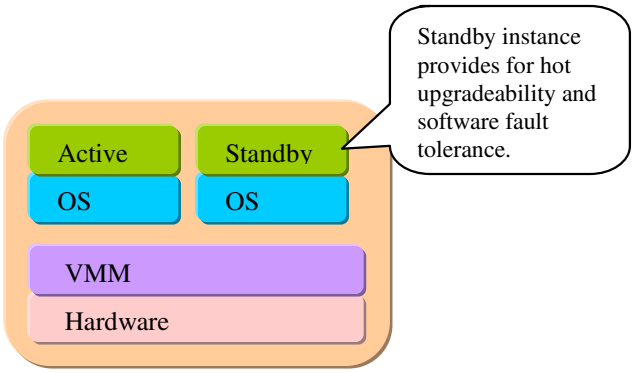

Figure 2: Hot upgradeability and fault tolerance

Workload migration is a more common feature of virtualized enterprise servers; however, it also has applicability to the communications market segment. For instance, in many Voice over Internet Protocol (VoIP) implementations, there is a device called a Soft Switch. This switch handles all aspects of call establishment and management. This switch has a set level of capacity, and once exceeded must be replaced or augmented with a new switch. The process of configuring the new switch is very time consuming due to its manual nature. Intel VT simplifies this process by allowing for the migration of a complete switch instance from one hardware platform to another. In addition, expanding a network can be simplified by first performing all configuration in a controlled lab environment and then pushing that configuration to the live switch, thus reducing the risks associated with expansion. Using a test harness and traffic patterns from the live environment, an expansion switch can be fully configured and tested in the lab prior to deployment in the field. Once the expansion switch configuration has been tested, and a migration strategy put into place, the live upgrade can proceed. This migration is shown in Figure 3, where the expansion switch has been added into the network, and a Region, from the installed switch, is being migrated to the expansion switch. This makes for a simpler management model than existing solutions.

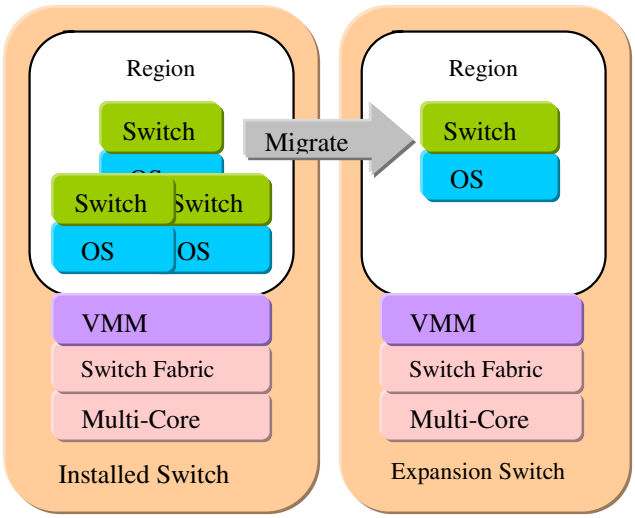

Figure 3: Virtual machine migration

Communication-Oriented Operating Systems

More so than any other market segment, the communications market segment contains many customized home grown OSs. Many times these systems are developed with a specific product in mind and don't lend themselves well to maintainability either due to complexity or lack of original knowledge. Virtualization allows a company to take advantage of this valuable intellectual property while still moving forward with new technology. By providing an environment within which the proprietary OS can operate, Intel VT allows new development to occur on general-purpose or modern OSs, while providing a link back to the proprietary OS. Intel VT offers the first step in providing support for these legacy OSs. It provides migration to advanced hardware technologies such as multi-core, without requiring multi-processor support within the OS. It also eliminates the need for modification of the OS, and it improves performance by eliminating the need for binary translation. With this capability, the proprietary technology is utilized for the purpose it was intended, and it is saved from costly revalidation and software development efforts.

Sharing vs. Assigning I/O Devices

The communications market segment demands high I/O performance from the hardware/software solution. Cost is always a factor in the design, and obtaining the most performance per watt is a driving fact for every design. In virtualized solutions, two methods exist for providing access to high-performance I/O, namely Shared I/O and Direct Assignment models (i.e., driver domains).

In Shared I/O the VMM (or its host OS) provides access to an I/O device by multiplexing that access through emulation. The guest OSs are presented with a virtual

device through which they communicate. The VMM then multiplexes the access from those virtual devices to the real I/O device below. The Shared I/O mechanism results in a performance loss due to the introduction of a multiplexing and emulation layer; yet provides for the most flexibility in migration. Due to this performance impact, shared I/O in communication systems is limited to non-performance critical tasks, such as the management plane.

In Direct I/O Assignment, the VM is assigned an I/O device exclusively. Intel VT for Directed I/O (Intel VT-d) addresses this requirement, and today this assignment occurs on the PCI bus within commercial VMMs architected to address this need. The VMM hides access to PCI devices that are not assigned to a particular guest OS.

Technical challenges exist for Direct I/O Assignment. The biggest challenge comes with those devices that perform DMA operations. Since a guest OS is unaware that it has been moved to a location in memory above its known starting point, it will provide addresses to DMA devices that may reside outside its memory range. To overcome this problem, it is necessary for either the VMM to remap these memory accesses or for hardware to dynamically do so. In the case where the VMM remaps addresses, this either will require that the guest OS be aware of the fact that it will be relocated into a new memory location, or that the VMM restrict the relocation accordingly. In the case where the hardware remaps DMA addresses (as with Intel VT-d), it is necessary that the VMM program the hardware with the VM base address, and that VM's device assignments. Direct I/O Assignment provides an order of magnitude performance improvement over Shared I/O, at the expense of VM dynamic migration ability. This performance improvement is mandatory for all high throughput interfaces in communications equipment and thus the tradeoff is warranted.

Partitioning the Platform for Better Communication Performance

When designing for general-purpose architectures, communication systems designers are often forced into a paradox: They want to leverage GPOSs, various operator interface options, and other general-purpose software, but the networking performance provided by GPOSs is less than acceptable. Virtualization can be used to solve this paradox by creating one partition that executes a minimal OS containing just what is needed to run the performance-critical parts of the application and provide direct access to networking devices, while another partition runs a GPOS that executes those parts of the system that are not performance-critical, such as operator interfaces or management agents for configuration, monitoring, and statistics and alarm reporting. Intel has prototyped an application running on such a system and found that it outperforms the same application running on a GPOS on the same hardware by 24%.

Commercial Virtualization Solutions for Communication Networks

Commercial products such as Jaluna OSware* offer solutions that are optimized to meet the stringent demands of communications equipment. OSware provides a robust platform that offers the key ingredients: Direct and Shared I/O, hard real-time guarantees, bounded interrupt latencies (measured at 21 microseconds), efficient memory virtualization, and the ability to execute both commercial as well as proprietary OSs without requiring them to be modified. Figure 4 shows that OSware provides identical network I/O performance of benchmark applications on RedHat Enterprise Linux* when executing in virtualized and non-virtualized environments.

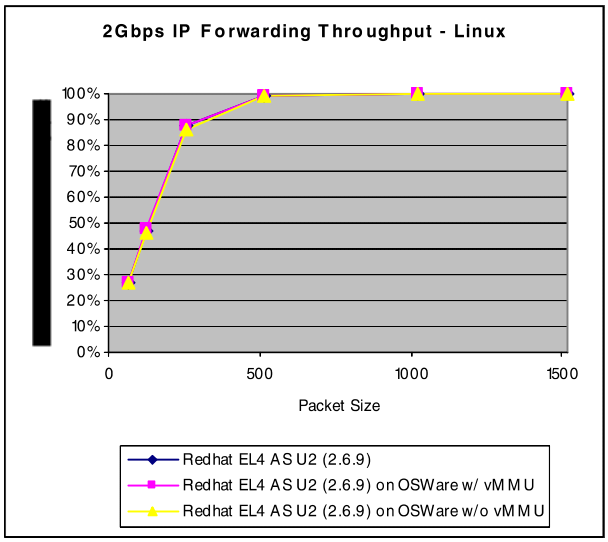

Figure 4: Virtualized vs. native OS networking performance

VIRTUALIZATION IN SAFETY-CRITICAL APPLICATIONS

The US government is migrating its Department of Defense, Department of Energy, and Homeland Security infrastructures from proprietary systems developed solely for government specifications to commercial off-the-shelf (COTS)-based systems with incremental security and reliability requirements. It is easy to imagine that the efficiencies and cost savings resulting from migrating to COTS systems would easily run into the billions of dollars, but the real benefits lie beyond that. Rapid deployment of new technologies allows the US armed

forces to retain the technological superiority so vital to their military and intelligence actions. Modern COTS-based systems permit increasingly sophisticated security methods to be employed to safeguard data while permitting the sharing of data that has proven very difficult across different proprietary architectures of the past. Safety-critical systems are also found in many other non-governmental applications where human life is at stake, such as aerospace (flight control systems).

A major challenge in migrating to COTS architectures is ensuring the security of both the hardware and software elements. The Federal Aviation Administration (FAA) has established criteria for certifying software for safety-critical aviation systems, and likewise the National Institute of Standards and Technology (NIST) and the National Security Agency (NSA) have established a common criteria for evaluation of technology products for security-critical systems. An enabling architecture known as Multiple Independent Levels of Security (MILS) is in the process of dramatically reducing the size and complexity of security-critical code, thus allowing faster and more cost-effective development and evaluation.

The MILS architecture defines four conceptual layers of separation:

- separation kernel and hardware
- middleware services
- trusted applications
- distributed communications

Our focus in this discussion is mainly on the MILS separation kernel. The separation kernel must be mathematically verified and evaluated. This practically limits kernel size to less than 5,000 lines of code. Also, the separation kernel must be completely isolated from other layers of software including OS services, which themselves must also be separated from other middleware components.

Intel VT is ideally suited to meet these separation kernel requirements. Figure 5 illustrates how Intel's family of virtualization technologies provides the foundation for an implementation of the MILS architecture.

Benefits of Intel Virtualization Technology

In summary, the benefits of Intel VT are these:

- It provides the separate root ring structure necessary for isolation of separation kernel from non-separation kernel services.
- Just as we would not expect a minivan to do the same job as a pickup truck, we cannot expect a desktop-oriented OS or a desktop-oriented VMM to operate within the constraints of embedded, communications or safety-critical environments, and still provide the functionality, configurability, separation, or performance of solutions that have been architected specifically for those attributes.
- It simplifies VMM design keeping the separation kernel code very small and thus making it possible to build a mathematically verifiable separation kernel.
- It simplifies the migration of single-threaded legacy software to multi-core processors by allowing virtualization of unmodified OSs. This gives end customers an option to simultaneously run multiple instances of non-SMP OSs.
- Intel VT-d allows for direct access to assigned devices. Separation of network interfaces is an essential component of system security. Intel's family of virtualization technologies will be extended to allow efficient sharing of physical I/O devices among VMs without requiring a "service" partition that has access to all network traffic, thus allowing the directing of network traffic to the specific guest OS and application for which it is intended.
- Intel VT also supports the use of a Trusted Platform Module (TPM) to provide the ability to authenticate both the VMM and the guest OSs and applications, to ensure that their image on disk has not been tampered with between reboots. The TPM is a microcontroller that stores keys, passwords, and digital certificates. Microcontrollers that adhere to the TPM specification as defined by the Trusted Computing Group [6] are available from a number of manufacturers.

Commercial Virtualization Solutions for Safety-Critical Applications

Safety-critical systems and security-critical systems are being developed using Intel VT by companies such as LynuxWorks, which provides its LynxOS* RTOS and LynxOS-178* safety-critical RTOS and corresponding development tools. Intel and LynuxWorks are working together to demonstrate the MILS architecture shown in Figure 5 using Intel® Core™ Duo processors. The LynuxWorks separation kernel has been developed to be mathematically verifiable, and it utilizes Intel VT and Intel® EM64T$^{\Phi}$ technologies to support virtualization and both 32-bit and 64-bit operating modes. It provides SMP support and is architected to take full advantage of Intel®

multi-core processors and their various platform-enhancing technologies.

CONCLUSION

We have shown that virtualization has many varied uses in embedded systems, communications infrastructure, and safety- and security-critical environments. The requirements of these domains are however quite different from one another and from the more familiar desktop, mobile, and IT data center environments. These requirements dictate that different architectural and design tradeoffs be made within the VMM and the guest OSs executing within the VMs.

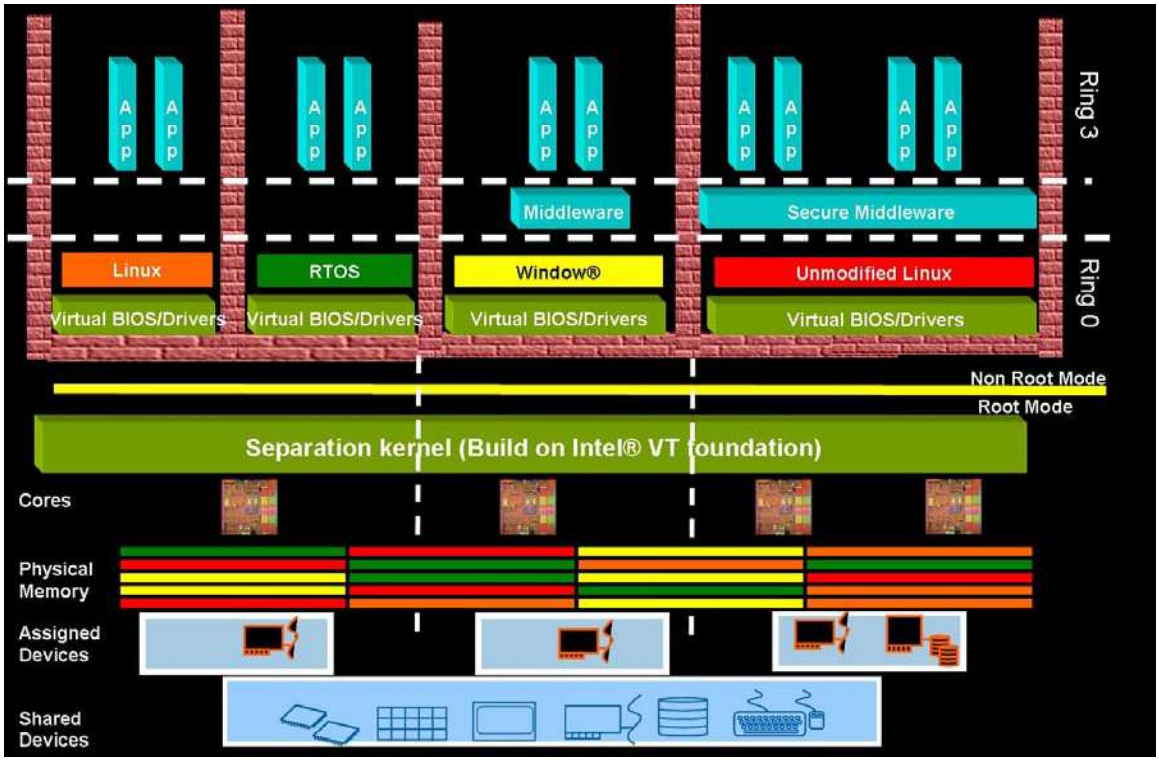

Figure 5: Example of MILS architecture with Intel Virtualization Technology

REFERENCES

[1] Abrossimov, E., Rozier, M., and Shapiro, M., "Generic virtual memory management for operating system kernels," in *Proceedings of the twelfth ACM symposium on Operating systems principles*, 1989, pp. 123–136.

[2] United States Department of Defense, *Department Of Defense Trusted Computer System Evaluation Criteria*, 1985.

[3] Ramamritham, K., Stankovic, J.A., "Scheduling algorithms and operating systems support for real-time systems," in *Proceedings of the IEEE,* January 1994, pp. 55–67.

[4] Kuhns, F., Schmidt, D., Levine, D., "The Design and Performance of a Real-time I/O Subsystem," in *Proceedings of the Fifth IEEE Real-Time Technology and Applications Symposium*, June 1999, pp. 154–163.

[5] Falco, J, Gilsinn, J., and, Stouffer, K., "IT Security for Industrial Control Systems: Requirements Specification and Performance Testing," *2004 NDIA Homeland Security Symposium & Exhibition.*

[6] Trusted Computing Group, at http://www.trustedcomputinggroup.org*.

AUTHORS' BIOGRAPHIES

Dean Neumann specializes in strategic planning of advanced technologies for Intel's Communication Infrastructure Group. He holds Bachelor's and Master's degrees in Computer Science, and has 20 years of experience in engineering fault tolerant and highly available systems for the communications and financial industries. His e-mail is dean.neumann at intel.com.

Dileep Kulkarni specializes in strategic planning of advanced technologies for Intel's Communication Infrastructure Group. He holds an Engineering degree from the Indian Institute of Technology and an M.B.A.

degree from the University of Michigan He has broad experience in planning, marketing, sales, and finance. His e-mail is dileep.kulkarni at intel.com.

Aaron Kunze is a network software engineer in Intel's Corporate Technology Group. He has a Bachelor's degree from Purdue University and a Master's degree from Oregon Graduate Institute. Aaron's area of research is in the design of communication systems, and he co-wrote two books about Intel's IXP Network Processors. His e-mail is aaron.kunze at intel.com.

Gerald Rogers is a software engineer specializing in Performance Analysis of Intel® Architecture for Communications, and he works for Intel's Infrastructure Processor Division. He holds a Bachelors degree in Electrical Engineering and a Masters degree in Computer Science. He has 16 years of embedded software development experience in the Telecommunications and Networking industry. His e-mail is gerald.rogers at intel.com.

Edwin Verplanke is a platform solution architect and is part of Intel's Infrastructure Processor Division. He holds a Bachelor's and Master's degree in Computer Science and Electrical Engineering, respectively. For the past 12 years Edwin has focused on communications board design, participated in various standards development covering high-speed interconnects, and more recently, researching multi-core architectures for the embedded market. His e-mail is edwin.verplanke at intel.com.

[Δ] Intel® Virtualization Technology requires a computer system with an enabled Intel® processor, BIOS, virtual machine monitor (VMM) and, for some uses, certain platform software enabled for it. Functionality, performance or other benefits will vary depending on hardware and software configurations and may require a BIOS update. Software applications may not be compatible with all operating systems. Please check with your application vendor.

[Φ] Intel® EM64T requires a computer system with a processor, chipset, BIOS, operating system, device drivers and applications enabled for Intel EM64T. Processor will not operate (including 32-bit operation) without an Intel EM64T-enabled BIOS. Performance will vary depending on your hardware and software configurations. See www.intel.com/info/em64t for more information including details on which processors support Intel EM64T or consult with your system vendor for more information.

Virtualization in the Enterprise

Patrick Fabian, Technology and Manufacturing Group, Intel Corporation
Julia Palmer, Information Technology, Intel Corporation
Justin Richardson, Information Technology, Intel Corporation
Mic Bowman, Corporate Technology Group, Intel Corporation
Paul Brett, Corporate Technology Group, Intel Corporation
Rob Knauerhase, Corporate Technology Group, Intel Corporation
Jeff Sedayao, Information Technology, Intel Corporation
John Vicente, Information Technology, Intel Corporation
Cheng-Chee Koh, Information Technology, Intel Corporation
Sanjay Rungta, Information Technology, Intel Corporation

Index words: virtualization, use cases, case studies, enterprise IT

ABSTRACT

We present how an enterprise IT organization sees virtualization in the enterprise and how it can be applied. We look at key enterprise services and applications used within Intel's IT department and examine the issues associated with virtualizing servers within the context of those services. We demonstrate that virtual machine (VM) isolation does not extend to performance isolation as we show how applications running in separate VMs can significantly interfere with each other. Enterprise services depend on host characteristics like available cycles, platform configurations, and on proximity to other services. We define a taxonomy of these dependencies derived from our study. Next, we describe uses of Intel® Virtualization Technology$^{\Delta}$ (Intel® VT) that we are investigating. The ability to run multiple operating systems (OSs) is of great interest in our design environment where highly specialized tools are tied closely to OS versions. The ability to checkpoint, suspend, resume, and migrate VMs is very useful when we run long simulations. The ability to allocate VMs at the location of choice opens up other possible use cases, such as network monitoring, security monitoring, and content distribution. We see this capability also enabling safe yet realistic experimentation, as a way to extend virtualization into clients. Finally, we present a real case study applying virtualization to enterprise IT problems. This virtualization program achieved higher server utilization, made it easier to manage datacenter assets, and reduced the consumption of datacenter resources (floor space, power, etc.), as well as simplified server releases through standardization.

INTRODUCTION

Virtualization is touted as a new and upcoming trend in computing. Simply stated, virtualization is a technology to run multiple independent virtual operating systems (OSs) on a single physical computer. It is not a particularly new idea in the enterprise, having been implemented in the 1960s on IBM mainframes [1].

A number of characteristics of virtualization make it a much discussed topic of conversation today. One is the potential to better use compute resources, allowing an enterprise to maximize its investment in hardware. In an average datacenter, the majority of the infrastructure resources are used about 25% of the time. Virtualizing a large deployment of older systems on fewer highly scalable, highly reliable, modern, enterprise-class servers significantly reduces hardware costs for infrastructure services. Multiple hardware and software solutions are available on the market and ready to provide a secure, easily managed platform to deploy, manage, and remotely control VMs.

Virtualization offers so much more than just server consolidation. Intel's IT organization has been studying other uses of virtualization that can add tremendous value to an enterprise. Virtualization features such as the ability to suspend, resume, checkpoint, and migrate running VMs is extremely useful in dealing with long running jobs. If a VM with a long running job checkpoints its state and then the physical machine it is on fails for some reason, the job can be restarted from where it left off, along with the VM, rather than being restarted from the beginning.

A key difference of virtualization today and the mainframe age is the ability to allocate a VM at the location of a service's choice. This notion of Distributed Virtual Machines (DVMs) opens a whole host of possible uses, such as network monitoring, security policy validation, and content distribution. It enables enterprises to create such things as virtual secure enclaves and do safe yet realistic testing of large scale, even planetary scale, services. This idea is useful and compelling enough to power the PlanetLab testbed [2], which is slated to become a core part of a next-generation Internet project called GENI [3].

Virtualization, while a viable technology today, is not without issues. Allocating VMs for enterprise services is not as simple as finding the first available host. Services have dependencies on network topology and other services. Also, VMs, while offering many types of isolation, do not offer complete performance isolation. VMs can interfere with each other.

This paper examines the virtualization of physical host machines, enterprise services, and multi-site instantiation of virtual environments. First, we introduce the difficulty of virtualizing enterprise service host machines. Second, we discuss use cases that can give IT organizations many new options in supporting their company's business units. Third, we review a case study of server virtualization for a business group at Intel and the process they followed from project inception through implementation. We conclude the paper with a discussion of our results and a description of future work.

CHALLENGES OF VIRTUALIZATION IN THE ENTERPRISE

For batch-oriented tasks, provisioning VMs and getting predictable performance appears to be relatively straightforward. This seemly simple task can be difficult if VMs interfere significantly with each other. When we introduce virtualization with enterprise services like the Domain Name System (DNS) [4], VM provisioning becomes more complex, especially as the location of the physical machine hosting the VM becomes important. In this section, we describe the challenges of server virtualization in an enterprise context.

Studying the Issues of Virtualization in the Enterprise

Our approach to studying the issues of virtualization in the enterprise had two parts. First, we looked at how VMs running on the same physical host could affect each other, particularly when different applications were running on the VMs. Second, we looked at key enterprise services and investigated how these services would fare in a virtual environment.

This is how we studied virtual machine interference:

- We obtained baseline performance (a control) of an application running on one VM.
- We attempted to optimally degrade the performance of one of two VMs running on the host, typically by attempting to use some shared resource.
- We documented and analyzed the results.

Once we had studied how individual VMs interact on one physical host, we looked at how VMs would interact in an enterprise. We first looked at the services that are the most critical and generate the most volume on the Intel Wide Area Network (WAN). Our goal was to examine those applications for performance bottlenecks and platform dependencies that would be problematic when servers for those applications and services would be virtualized. In addition, we also looked for five additional applications commonly used within Intel. For all of these services and applications, we searched for information on the Web and talked to IT personnel who are expert at running them in Intel's IT environment, looking for the issues mentioned previously.

VM Interference

VMs, as a technology, offer many advantages to users and administrators. Security isolation prevents a malicious application from accessing data or altering running code. Fault isolation prevents one misbehaving application from bringing down the whole system—rather than rebooting the box, one can simply reboot the VM. Environment isolation allows multiple OSs to run on the same machine, accommodating legacy applications and cutting-edge software alike.

While VMs offer these forms of isolation, we have observed that modern VM environments [4, 5] do not really provide performance isolation. While in theory, the virtual machine manager (VMM, also known as the hypervisor) "slices" resources and allocates shares to different VMs, there are still ways in which the behavior of one VM can adversely affect the performance of another. Furthermore, the isolation that VMs provide limits visibility of an application in a VM into the cause(s) of performance anomalies that occur in a virtualized environment. Contemporary platforms with Intel VT, however, provide mechanisms that we can use to detect and classify performance interference, which can then be used for a number of purposes:

- As input to the local scheduler, which can alter its behavior (e.g., change quanta or ordering) to ameliorate the effects of the interference.

- As input to a global scheduler, or orchestration engine, which uses the information to rearrange the placement of VMs to minimize interference and improve performance.
- As "metering" data, so that systems that charge for usage (e.g., free-market allocation systems such as HP Labs' *Tycoon*[*] [6], grid computing pay-per-cycle or "cycle-rental" schemes, etc.) can more accurately charge/credit users for the resources they consume/provide.

Our research to date has shown that shared state in resources under contention can indeed dramatically affect the performance of a VM, beyond the expected performance degradation that is due to simple time-slicing of the resource.

The first type of interference we studied was the interference within the processor's L2 cache and to server state (disk head position, cache state, etc.). We designed experiments to quantify these types of interference by running "benchmark" workloads against other VMs with code designed to be explicitly pessimistic in terms of interference to that particular benchmark. Our results show that in each case, there can be a non-negligible effect on the benchmark's performance.

For the cache experiment, we wrote code to continuously write to a large (bigger than the L2 cache) array in one VM to show how this would interfere with a memory-intensive application in another VM. We looked at the Freebench test suite [7] because it was freely available and had been used in other VMM performance testing [8]. We selected Freebench's analyzer benchmark as an application. The analyzer's performance is limited by the memory subsystem, making it a good candidate for cache interference. It runs a deterministic computation, so we used time-to-completion as our measurement of performance.

Our experiments compared the runtime of the program versus another VM executing a simple spinloop. Because of this, the slowdown seen can be attributed directly to cache interference (i.e., its degradation is due to more than simply sharing half the CPU with another VM). We ran our experiment on several types of Intel® platforms, with varying configurations. A typical run is shown in Figure 1. As the amount of cache used by our interference-generating application increases, the slowdown in application performance increases. That a dirty cache slows down an application's execution is not surprising; however, the application and its OS are completely unaware that the cache is being dirtied, and because they are running on a VM, typical techniques (e.g., OS task scheduling) are unavailable. With Intel VT features, we are able to determine the interference and provide that information to higher-level constructs (e.g., the hypervisor scheduler, or a global orchestration system) as mentioned above.

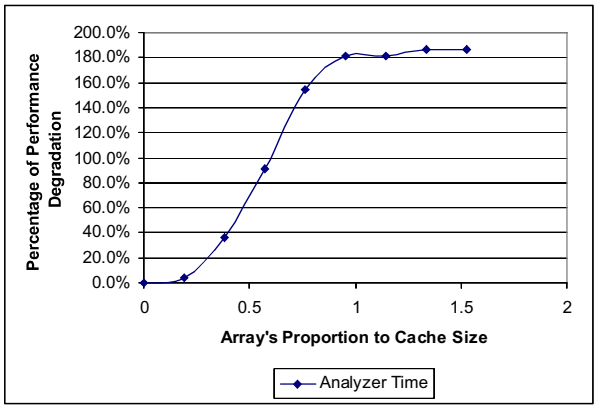

Figure 1: Performance degradation as cache is increasingly dirtied

We also ran tests for storage interference. The simplest example entailed two VMs accessing the same disk device, to most easily demonstrate head-position and disk-cache state (outside the VM). Our results (as in the CPU cache case that we compared against a simple spinloop VM) showed similar amounts of additional degradation—between 50% and 90% depending on the nature of the disk access (sequential/random and character/block).

Virtualization and Service Dependencies

To get the list of critical network services, we consulted with Intel IT's WAN engineers. They reported the following are the five most critical network services:

- Exchange*
- DNS
- Active Directory*
- Chip design associated file transfers
- Web proxy traffic

In addition, we studied the following internal applications:

- Internet Information Server (IIS[*])
- Apache Web server[*]
- SQL server[*]
- Oracle
- Sendmail[*]

We looked at a number of service orchestrators also. We looked at how Oracle orchestrates its operations, as well as the IBM/Auremia director[*], the HP Workload Manager[*], the 3-DNS[*] and Big IP[*] load balancers from F5, and Microsoft's Visual Studio 2005[*]. To do this, we

looked at documentation as well as talked to operational experts within Intel's IT organization. In addition, we found that some services and applications like DNS, Sendmail, and Active Directory, have some mechanisms that perform orchestration functions.

We found that service platform dependencies fall into the following categories:

- Network
- Host/System
- Storage
- Services

We next discuss these dependencies and emphasize the aspects that are typically not covered by service orchestrators. At the end of each section, we describe constraints that need to be specified by orchestrators to deal with these dependencies, since these would be additional concerns with provisioning VMs for those services.

Network

We found that services had a number of network dependencies that are not typically dealt with by service orchestrators. Often these dependencies rely on network topology specifics. One example is the Web proxy service offered by Intel IT. This service proxies Web traffic between systems on the internal Intel network and the Internet and reduces network traffic by caching Web pages already fetched. The proxy service maintainers require that a directly connected proxy (with direct access to the Internet) has a high bandwidth network path to the Internet. For fairly large Intel sites with low bandwidth links to the Intel® WAN, Intel® IT deploys a proxy server locally and chains this proxy server to another. While some orchestration specification languages like JSDL [9] allow conditions to be set on network bandwidth, they do not address network topology.

Intel's DNS service relies on multiple network connections from a site for deployment. Intel sites with only one connection to the Intel® internal WAN have DNS servers deployed to them. Multiple and reliable network connectivity is a dependency for the DNS service.

DNS also monitors query latencies and uses them to generate basic orchestration functionality. It records the time it takes to process queries for a domain and uses the name servers for that domain that respond the fastest. In this case, network latency is a significant contributor to how the DNS service behaves and partitions work.

Some services want to see a specific number of network interfaces on the platform. Some deployments of the Oracle database system require three network interfaces. One of the network interfaces is used for heartbeat information between servers and requires low latency. Other services assume that they have a network interface (or at a minimum, an IP address on an interface) that can be directly reached at a particular port. This applies to services that use well known ports, such as Web servers like Apache or Microsoft IIS. While Web servers can do virtual hosting, they assume that a standard HTTP port is directly reachable, and in a virtualized environment, this implies that there is an IP address per each VM that runs a Web server. For each IP address, it is assumed that there is a MAC address associated with that interface, and that there is a way to route packets to each VM.

We define network constraints that we need to manage as follows:

- Minimum bandwidth between server hosting service and particular points.
- Topology and availability requirements, in particular minimum availability and/or a minimum number of paths from a server's location to other locations.
- Minimum or a specific number of network interfaces.
- Maximum latency between server and other servers.

Host

We found two notable host dependencies that are not covered by orchestration service specifications. The first is a dependency on a fixed amount of CPU resources. A commonly used mail forwarding program called Sendmail depends on the notion of load average to decide whether to queue up mail or whether to reject mail connections. In a virtual environment where resources are shared equally among VMs, an application cannot be certain whether it will receive the same amount of CPU resources since other VMs may be assigned to the same physical host it runs on. Thus, using load average is not an accurate indicator of the available resources.

The second host dependency is non-pageable memory. The Exchange mail service relies on having a significant amount of memory that cannot be paged out for good performance. While orchestrators allow you to specify how much memory a job or service may require, there do not seem to be options for non-pageable memory.

Service Affinity/Proximity Requirements

One key service dependency that is not always captured in orchestrators is affinity or proximity to other services. A good example is Exchange and Active Directory. Exchange requires fast responses from Active Directory. Operationally, an Active Directory server should be on the same segment as an Exchange server. Deviations from this configuration have proven disastrous operationally.

An additional aspect of service dependency is the need for a maximum time to complete a service's basic transactions. DNS operations personnel recommend that DNS queries must be resolved within one second in order to prevent applications that rely on DNS from hanging.

Storage

Some applications rely on specific platform features. For example, some versions of Oracle require that Oracle write directly to disk blocks. Other applications, such as Active Directory, require large disk-write caches.

OTHER USE CASES FOR VIRTUAL MACHINES IN THE ENTERPRISE

Virtualization is typically discussed in the context of datacenters, where multiple VMs are loaded onto a single host to increase server utilization or reduce the cost of buying new hardware. We cover this use case extensively in an upcoming section. Virtualization enables other capabilities that can be extremely useful to enterprises. We now discuss enterprise use cases for virtualization that go beyond the usual examples of increased utilization, which are being investigated by Intel's IT organization. We start with ways to enhance the operation and efficiency of large-scale computation. We then talk about distributing virtualization, and how the ability to allocate VMS in the location of choice opens up new applications and paradigms for service deployment.

Enhancing Standardization

While a completely homogeneous computing environment would yield obvious efficiencies, it is generally not realistic. Intel's design environment supports a huge variety of software tools for a diverse roster of design teams, some of whom joined Intel as a result of acquisitions. The design environment employs laptop PCs, dedicated compute servers, and everything in between.

In this complex environment, virtualization could achieve many of the efficiencies of homogeneity. A software tool could be bundled with its own specialized virtual computing environment. When a new version of this bundle is standardized, it could be quickly pushed out to all sites on top of VMMs without requiring expensive and time-consuming OS upgrades. This is especially helpful to small sites which often lack sufficient staff, and sometimes lack even all the required computing platforms, to implement a never-ending stream of company-wide directives.

Making it easy and inexpensive to push out new standard images to all sites not only reduces costs, but accelerates innovation, because it frees a small team to develop specialized expertise in a product and to support it worldwide instead of relying on generalists dispersed across many teams. It enables the leveraging of good ideas and fixes from any of these specialized groups, because these fixes and ideas can be quickly and easily applied everywhere.

Legacy Operation

Closely related to the standardization issue is an inevitable heterogeneity across time. OSs and microprocessor architectures evolve, and they sometimes even die. Yet important legacy software tools that depend on particular OS versions are often useful far into the future. It is expensive and sometimes insecure to maintain special-purpose "classic" configurations. In addition, tools that run on them can't ride Moore's Law to ever better performance.

If snapshots of such classic configurations were encapsulated as VMs, then they could be inexpensively "revived" whenever and wherever needed and on enhanced hardware, resulting in the associated tool being executed faster.

This strategy would also be useful for any application that is run only occasionally, especially if it needs to or is expected to run on a dedicated server. For example, during a downtime, whether planned or unplanned, a temporary mail forwarding server could be activated at some unaffected site. This approach reduces not only costs, but also the risk that an infrequently used application has fallen behind and is now incompatible with changes in the computing environment. Such incompatibilities are typically discovered at the worst possible time, that is, at the exact moment when the application is needed.

Checkpointing

A grand reliability goal is to guarantee to users that no job will ever fail to complete for external reasons, such as a machine reboot, a power outage, a disk crash, or even a catastrophic failure such as an earthquake. An important enabling technology that would be immediately useful is the automatic checkpointing of a VM.

It should be possible to schedule a periodic saving of the VM state that could be used to go back in time and replay history from that archived moment, but without the externally induced failure. Less frequently, redundant copies of the state could be stored away remotely, at a rate correlated with the probability of various risks which grow with the duration of the task. For example, a simulation of the earth's climate that required a year of calendar time to complete would almost certainly be disrupted during that year by some external event. It could be argued that the more sophisticated the application, the more likely its developers are to have already built in checkpointing mechanisms. But even if we ignore the fact that many real

applications disregard checkpointing altogether, application-level checkpointing mechanisms can't reasonably be expected to cope with catastrophic failures. Where should the application save its own state to protect against fire and flood?

Checkpointing could obviate the need for engineers to submit redundant jobs as insurance. Today, the longer or more critical a task, the more likely it is to run multiple instances of the same job, causing the actual resource utilization for a computing task to be much larger than the resource requirements for an individual job might indicate. With VMs and checkpointing, a single job would only need to be run once because it could be resumed elsewhere, even if there were an external failure. Likewise, if a machine must be rebooted for OS patches, a planned site-wide downtime, or simply as an attempt to put it back into a good state, the tasks running on it could be terminated easily enough and resumed on some other machine. This eliminates the need to prevent long jobs from being submitted days in advance of such maintenance and the temptation to postpone prudent maintenance because of the speed bump it throws into user schedules.

In the long-term, a VM could support some analog of apoptosis (programmed cell death), killing itself when it detects errors. A daemon could automatically roll back (terminate and then reincarnate elsewhere) any VM that hasn't recently enough provided proof that it is healthy.

An issue that needs to be investigated is how to deal with external (non-virtual) state elements, such as the actual current calendar time and ongoing network communications, that can't be checkpointed. Another key issue concerns licensing. Some software applications require a license for physical CPU, while others require a license per actively running copy. The issues of program state and licensing need to be answered when deploying VM checkpointing in the enterprise.

Performance Isolation

When choosing a shared computing resource, such as a server on which to run a VNC [10] session, it's difficult to predict the impact of contention. The longer the task, the greater the chance that some other user may consume an unfair share of the resource and degrade one's own effectiveness. Although using VM checkpointing could enable the victims to pick up and move to "greener pastures," it would be better to prevent a "tragedy of the commons." If we can ration real-world computing resources by configuring the parameters (memory size, processor speed, disk access speed, etc.) of each VM assigned to a task, then limits on consumption would be inherent to the resource capacity of the VM that task is running in. A number of VMMs are capable of allocating computing resources and of performing a measure of performance isolation. While we have shown in a previous section that VMs can significantly interfere with each other, particularly through the interactions of shared resources like the cache, there are other resources like CPU cycles and memory that can be allocated in a way that significantly isolates the performance effects of tasks from each other.

Distributed Virtual Machines

The ability to allocate VMs at the location of choice is a capability we call Distributed Virtual Machines (DVM). DVMs enable a whole host of possible applications and servers. Throughout this section, we use the terms distributed virtualization, overlays, and distributed virtual machines, interchangeably. We view DVM as the methodology of choice for realizing robust, computationally rich, networked virtualization and for implementing overlays.

The Origins and Impact of DVM

One of the earliest and most successful implementations of DVMs is PlanetLab [2]. PlanetLab is a world-wide overlay network with over 689 nodes at 334 locations around the world. Designed to be a testbed and deployment platform for researchers to study planetary-scale distributed systems and services, PlanetLab has distributed virtualization at its core. Researchers allocate a "slice," a set of VMs, at the locations of their choice. Using VMs allows the researchers to develop and deploy innovative new services that do not interfere with each other on the same physical hosts. Using this model of computing, several innovative services with content distribution [11] and network measurement [12] were developed and deployed. These types of applications, and the way that PlanetLab was designed for the safe development and deployment of services, have implications for the way that DVM can be used by enterprises.

Network Monitoring

A global organization has many Internet users scattered across the planet. Some are Intel customers, some are suppliers, and some are employees. Employees can be within Intel's firewalls or working remotely from home or from customer sites located anywhere on the globe. Services that are utilized include Web sites such as Intel's corporate presence at www.intel.com, various e-commerce applications, and VPN connectivity back into Intel. This requirement for global access can result in Intel's Network Operation Center (NOC) receiving complaints about performance from any spot on the planet to any one of Intel's many DMZ zones. For example, the NOC might get a call from a user in China saying that the response for

an e-commerce application is very poor. Is the problem local to China? Is the problem local to the Internet connection in question? Is the problem Internet-wide? The NOC needs tools to be able to answer those questions.

A key question is where to monitor. The typical DMZ firewall model lends itself to monitoring the DMZ systems from within the DMZ. This ends up creating a monitoring model with limited scope that does not address problems with transit from anywhere in the world to the DMZ. An alternative would be an approach that examined Web logs for performance problems [13] or looked at traffic flow data using Cisco NetFlow* [14]. Because of our traffic volume and the fact that we didn't have Web servers at all of our Internet DMZs, we ruled out this option. It would be extremely useful to be able to proactively monitor for performance problems all around the world using active measurements. Active measurements from regions in the world could be taken from commercial services like Keynote* [15] or from hosts in datacenters strategically placed around the world. Using commercial services would limit the kind of applications we could run to monitor the DMZs and it could be fairly expensive. Deploying our own hosts in the locations around the world from where we want to monitor would permit much more flexibility, but would be even more expensive.

DVM presents a relatively inexpensive and flexible platform for global-scale monitoring, but poses challenges with software distribution and application management.

Security Monitoring

The traditional, closed network control model has disadvantages in protecting the enterprise networks from distributed network attacks because of data inaccuracy, inability to perform overall impact analysis, and lack of data correlation from distributed sources in large networks. As more and more enterprises move towards relatively "open" perimeters (sometimes without realizing it as through unauthorized wireless access points and VPN connectivity) and distributed network environments in order to meet business demands, the associated provisioning and management cost will consequently increase, as will the complexity. The IT infrastructure needs to be able to provision security requests quickly and be pre-positioned and ready for such requests. The notion of trusted and un-trusted network zones is fast changing in today's enterprise network. Enterprise networks are no longer a simple 2-trust level like they were a few years ago with "internal trusted" and "external un-trusted" zones. The requirement for protecting the resources at the service level is becoming more a reality, and the infrastructure to support this is at best expensive and difficult to justify from an IT security standpoint. Also, simply implementing network and service-level security such as firewalls, IPS, anti-virus, and a whole slew of defenses is not sufficient. In order to ensure these complex network and service-level security enforcements are functioning as desired, an automated and proactive security monitoring system is becoming more essential for enterprises. Proactive network security monitoring is required to validate the security implementations, patching, and provisioning of software to ensure it is not vulnerable to the most recent threats and to avoid costly network downtimes, security incidents, denial of service attacks, and worm and malware attacks, all of which impact productivity and service availability. In addition, regulatory and legal compliance requirements, such as the U.S. HIPAA, Sarbanes Oxley regulations and European privacy laws, are getting more strict for all types of enterprises to ensure they are following the rules to protect their assets, resources, and information.

Vulnerability scanning for the enterprise network to ensure compliance to minimum security specifications and auditing of network security policy to ensure it is implemented per the documented enterprise security policy are examples of add-on security monitoring that the enterprise IT would like to deploy extensively but which is limited due to the static nature of deploying these applications. Using the DVM approach, the ability to create instances/clones of systems that would be able to generate the required security monitoring functions would be extremely simplified. In addition, it would help create multiple views for network security assessments and monitoring. For example, in order to assess the effectiveness of network security implementations, such as firewalls, IPS, authentication/authorization, and other security enforcements, enterprises would have to perform the network security assessments/audits/scans from various parts of the network, such as from within the DMZ/internal network and from the external connected network. This would not only help validate the overall picture of the security posture for the network but also ascertain whether the implemented controls are sufficient. With DVM and the ability to "suspend, copy and resume" a VM, network security becomes relatively simple and cost effective. Another advantage of being able to inexpensively create multiple instances of the network security monitoring system would be to increase the speeds and parallelism of the results. Network security monitoring is then transformed from an infrequent and expensive annual or quarterly audit to a proactive one that can identify and fix security vulnerabilities as soon as they appear on the network.

The DVM approach to network security monitoring as discussed above would help reduce the cost of provisioning these relatively complex auditing/monitoring/scanning applications as compared to the traditional method of static provisioning of standalone security monitoring systems. Using the DVM approach

would reduce the capital costs of the hardware and the cost of the provisioning tasks required to maintain physical systems for these functions. Operational costs for network security operational staff would also be reduced as network staff would be able to leverage the network for simplicity of "on-demand" VMs for the network security monitoring functions. Using the classical server/operating system/application model, and not the DVM model, it is almost impossible to monitor to the level required to be proactive enough to identify security gaps before they are widely exploited.

Content Delivery

A content delivery overlay provides a common service to various applications such as distributed file storage and sharing. Each overlay node maintains a small overlay routing table for finding the destination with the shortest path length of complexity O(logn), where n is the network size. But these overlay search algorithms make the underlying network transparent to the overlay and only find the shortest search path in terms of the number of virtual hops in the overlay.

Safe Yet Realistic Experimentation

A challenging aspect of enterprise environments is the difficulty of testing and introducing new or innovative services into an established infrastructure. Changes are strictly controlled because changes in the computing environment can negatively affect critical enterprise services. This is particularly true when introducing new services to an already running physical host. The new service or application may require system libraries and other software that could potentially break existing services if introduced. Moreover, usage loads introduced by new services on existing infrastructure (both network and CPU) can potentially starve existing services. Thus, the traditional enterprise approach is to bundle new hardware with each new service. Deploying new hardware for each additional service severely slows the introduction of new services, adds to the Total Cost of Ownership (TCO), and further complicates change control. Testing of new services is often done in isolated lab environments, where realistic conditions are difficult, if not impossible, to replicate.

Alternatively, the ability to create VMs that are effectively isolated from each other and share resources fairly resolves these problems. The fundamental idea here is to decouple the introduction of new services from the deployment of new hardware. New services can be deployed on existing hardware by allocating VMs in the preferred service locations. The VM isolation shields existing services from library conflicts with new services, which are sequestered in their own individual VMs. Deploying new services on existing servers also speeds the development and testing of new services, in a realistic, closer-to-production environment without impacting existing services and without requiring installation of new hardware.

Virtual Enclaves

Within large and complex enterprises, there is a need to separate mission-critical environments from the rest of the organization. Critical areas like manufacturing should be immune to worms and malware that might proliferate in the rest of the organization, and access to these critical areas needs to be restricted to those individuals who need it. Fundamentally, these critical areas require their own separate enclave. The traditional approach to building these enclaves is to use dedicated hardware, as shown in Figure 2. This approach has several drawbacks. Deploying the entire infrastructure needed to make the enclaves self-sustaining (such as DNS servers) is time-consuming and expensive. If the infrastructure in one of the enclaves goes down, there is no easy way of getting more resources, short of either repairing the down nodes or installing new equipment.

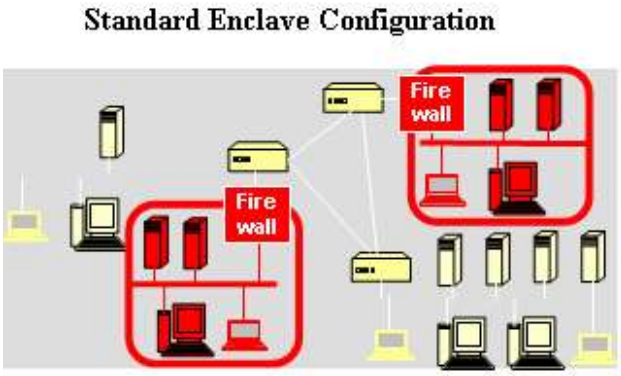

Figure 2: Enclaves currently need to be implemented with physical partitioning and hardware firewalls

The use of DVMs combined with overlay routing technology provides an innovative new way of deploying these enclaves. The VMs required by each service can be joined together with a secure overlay. The overlay isolates and controls access to the VMs as shown in Figures 2 and 3.

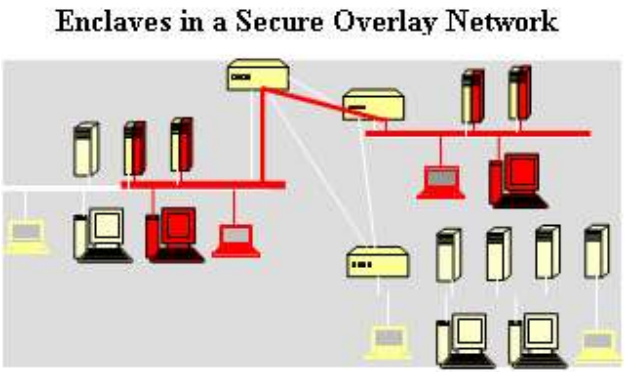

Figure 3: A secure overlay network connects distributed virtual machines

This approach has several benefits. If there is sufficient capacity, no new machines need be deployed. These new virtual enclaves can be deployed in a dynamic manner at a greatly reduced cost. If network segments go down, overlays can route around the problems. If hosts go down, VMs can be moved or allocated on other physical hosts.

Extending Virtualization into Clients

The computational, network, and storage resources of mobile devices (laptops and handheld devices) in an enterprise typically have low utilization and are not available for use by enterprise applications or services that could best utilize them. We envision an environment where the OS with which a mobile user interacts, is one of many OSs that run over VMMs. While the mobile user is interacting with the device, a VM dispatch service can request that the device's VMM create VMs for a variety of tasks, as displayed in Figure 4. These tasks can range from doing computations to running services like file systems, content distribution, and other services like Voice over IP (VoIP). This work can be transparent to the end user and done in the background.

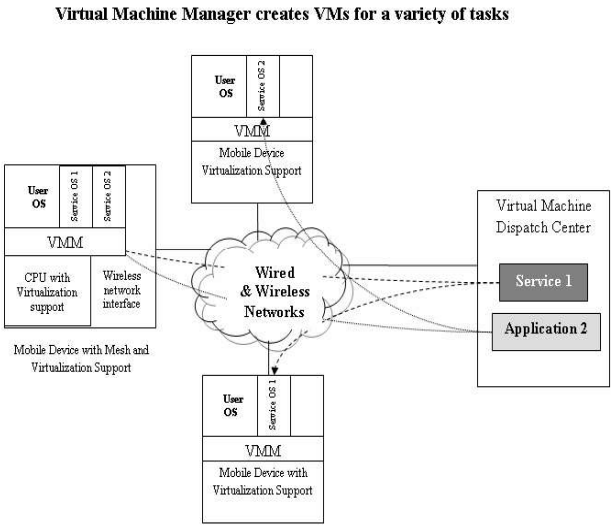

Figure 4: Dispatching DVMs to mobile devices

This architecture extending virtualization into clients and dispatching work via VMs to mobile devices has significant advantages over the current situation in most enterprises. Enterprises typically have low utilization of their mobile resources. Our proposed architecture enables better utilization and can potentially add enormous amounts of shared resources to an enterprise. It also has advantages when it comes to management of systems and services. Having a VMM underneath the OS visible to a user makes it easier to restart or rebuild the users' OS. Services can take advantage of the location of mobile devices and dispatch service instances in VMs that are close to their designated clients. This frees a service from having to manage network parameters such as delay and throughput to a central site. The service is also easier to maintain in the face of node outages because work can be moved between mobile devices.

The similarities between overlay networks and ad hoc networks, along with the technical merits that each introduce through their integration, motivated our interest to investigate and implement an alternative architecture of overlays on wireless mesh networks, called OverMesh [16]. Integrating overlays and wireless mesh enables OverMesh to be flexible enough to serve many networking purposes.

While OverMesh is similar to current ad hoc, sensor networking, and peer to peer computing systems, it is also architecturally distinct from these systems. These are the differentiating properties of OverMesh:

- *Infrastructure-free*: a peer-to-peer edge/access system is suggested over current hierarchical physical formations.

- *Network virtualization*: based principally on a distributed virtual machine overlay strategy.
- *Emergent control and manageability*: utilize learning and statistical inference techniques to off-load human-dependency on operational management and provisioning.
- *Cooperative and adaptive end-to-end control*: tighter layer integration and automation of application-to-network control and management through cross-layer facilities.

OverMesh can be applied to a variety of wireless mesh networks. At its current stage, we chose to realize it on one of the mesh networks that is being actively standardized—the IEEE 802.11s WLAN mesh network [17]. The PlanetLab service architecture [18] was customized and integrated with the WLAN mesh network to manage the DMV-based overlays. We believe that the implemented OverMesh platform will provide a unique testbed for developing a wide variety of services and applications on wireless mesh networks.

An IT Overlay

To experiment with, test, and deploy services using DVMs, Intel's IT department has created the IT Overlay. We envision it as an overlay network that will include hosts within Intel and eventually extend to hosts residing outside of Intel's firewalled perimeter, as shown in Figure 5. Systems hosting VMs have been deployed at five sites within Intel, with more to be added as use of the IT Overlay increases. Intel is also part of the PlanetLab consortium, and Intel IT hosts two PlanetLab sites. Intel has deployed a monitoring service that takes advantage of the distributed nature of PlanetLab.

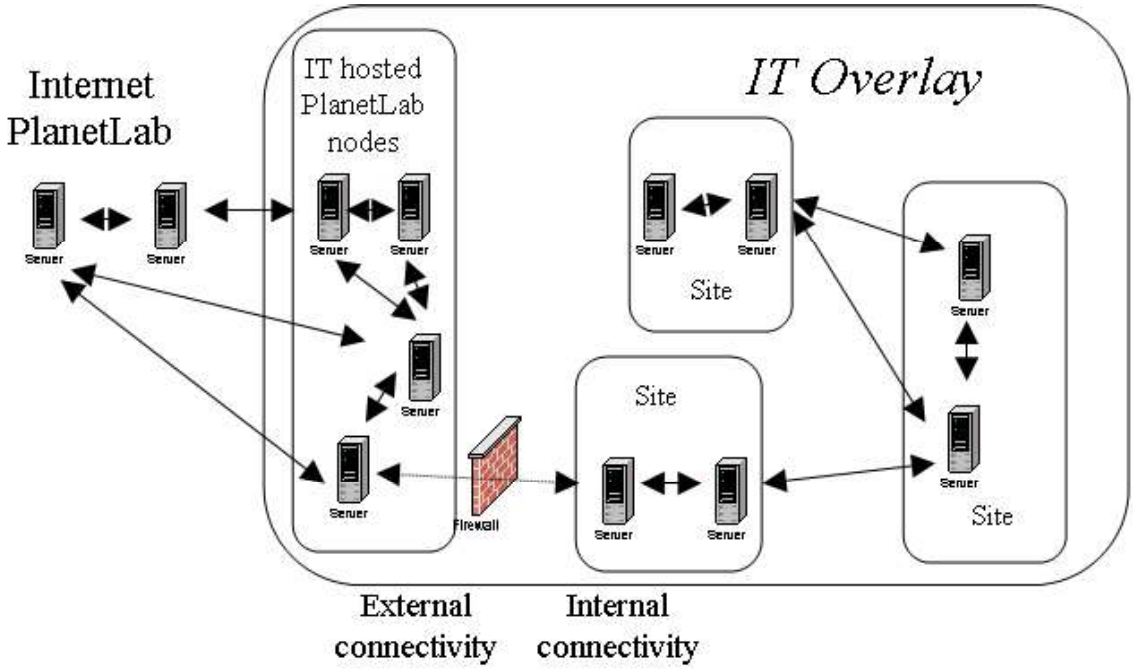

Figure 5: The IT Overlay inside and outside of Intel's firewalls

The internal portion of the IT Overlay will be modeled after PlanetLab. Services will be able to make requests to a central authority that will dispatch VMs to run applications and experiments. We intend to use the interfaces and APIs created by PlanetLab to dispatch VMs, although the Overlay uses Xen* [5] domains for VMs rather than the VServers [19] implementation. We envision running security, network monitoring, and content distribution applications on the IT Overlay and opening it up as a testbed and deployment vehicle for DVM-enabled services.

A CASE STUDY OF SERVER VIRTUALIZATION USING VMWARE

Here is a brief history leading up to the discussion and decision to implement server virtualization for a manufacturing support group at Intel. This organization's server population grew 65% over the last three years with

2006 projections meeting or exceeding this trend. As this organization grew and acquired servers, many of these acquisitions were waterfalled servers being released by other Intel business units. The initial costs made this type of acquisition financially attractive but as we move forward four years, most of these servers have reached their end of life. Another factor is that the primary datacenter for this group is projected to reach complete build out in 12-18 months with no plans to expand. The challenge for the organization was to continue supporting the server growth and replace aging hardware with limited datacenter space while maintaining the same high level of customer support.

This group partnered with a forward-thinking IT group to evaluate, plan, and implement a virtual server environment. In this case study, we walk you through the steps, lessons, hurdles, and successes of this effort. The covered topics include software evaluation, candidate evaluation, hardware design, host hardware setup, virtual server setup, server testing procedures, and initial results.

There are multiple factors to consider when evaluating and selecting server virtualization software. Our team carefully reviewed leading technology products and evaluated different system design options. The two most popular virtualization architectures were host-based virtualization (Microsoft Virtual Server 2005*; VMware GSX 3.1*; Microsoft Virtual PC 2004*; VMware Workstation 5.0*) and full virtualization (VMware ESX 2.5*).

Host-based virtualization requires the installation of a base OS first and then a VMM to be responsible for the execution of all VMs. In addition to the VMM application, the OS can execute other applications (e.g., Anti-Virus, Backup). The downsides to this type of architecture are a heavy performance penalty, high system resources utilization by host system management, additional work to support host maintenance and management, and the upkeep of host security.

The full virtualization design starts off with the installation of a mini kernel (a hypervisor optimized for virtualization) on the physical server. This kernel uses minimal system resources since it focuses only on tasks required for virtualization, and it does not run unnecessary processes or applications. The hypervisor provides full hardware virtualization and distributes the necessary system resources to all VMs. Each VM contains its own OS and cannot distinguish it is running on virtualized hardware. This architecture is ideal for consolidating high-end datacenter solutions.

The decision process to determine the proper virtualization architecture is a critical and time-consuming task. Our team researched benchmarking results of multiple virtualization products and analyzed the cost and supportability options. We prioritized our list of requirements and rated the various software options. We evaluated four products against our requirements and scored their performance. Our requirements included performance, manageability, supportability, stability, security, and a wide range of capabilities. Table 1 is an example of how we did our comparison: (utilizing fictitious data).

Table 1: Product evaluation scorecard example

Product requirements	Product A	Product B	Product C	Product D
1. Performance	10	8	10	7
2. Manageability	6	10	3	8
3. Supportability	8	10	4	10
4. Stability (VM uptime)	7	4	8	3
5. Capabilities	9	8	7	4
6. Security	10	5	7	4
Total:	**50**	**45**	**39**	**36**

After evaluating the scores, we selected a full virtualization software solution for our virtual server environment.

When virtualizing a datacenter, the project's success is directly dependent on choosing the appropriate candidates. We approached this step by defining our virtualization strategy for this business unit. First, we divided their server environment into four categories:

- Ideal candidates
- Candidates
- Potential candidates
- Not a candidate

To categorize each server, we started by collecting data on performance, system utilization, end-of-service timelines, business area, and application specifics. Once the selection criteria data were collected, we mapped our servers against the selection criteria to determine in which virtualization category a server belonged. Once categorized, our team focused on 75 candidates and worked with the business unit to evaluate application specifics and machine load analysis. With our performance evaluations and customer input, we assembled the server requirements:

- CPU consumption
- Required memory
- Disk I/O intensity

- Network requirements
- OS configuration

We used these data when evaluating different hardware platforms for our virtualization environment.

To maximize Return on Investment (ROI), number of virtual systems, and performance, this team's final choice for the virtual host servers was the 4-way Dual-Core Intel® Xeon® processor 7040$^{\Phi}$ 3.0 GHz-2 MB L2 cache system with 16 G of RAM, 2 x 2 Gb, 64-bit/133 MHz PCI-X-to-Fiber Channel Host Bus Adapters and three Network Dual-Port PCI-X 1000T Gigabit Server Adapters.

The hardware selected for this virtual environment is based on an Intel IT standardized platform. The team focused on designing a robust virtual infrastructure without introducing single points of failure. This design would address our customer's primary concern with consolidation of multiple applications to a single physical machine.

The team agreed on an environment that would be immune to hardware failure and power interruptions while possessing the ability to load-balance. The consolidated applications would reside on host servers containing dual power supplies, mirrored hard drives, and teamed network interface cards. The centralized storage solution selected is a multi-terabyte storage area network (SAN) with full fault-tolerant capabilities. Connections to the host servers were made possible through two 2 Gb fiber channel switches configured with redundant paths. This design enables load-balancing, as all VM files reside in a central location and access is possible by each host. Figure 6 shows the details of this design.

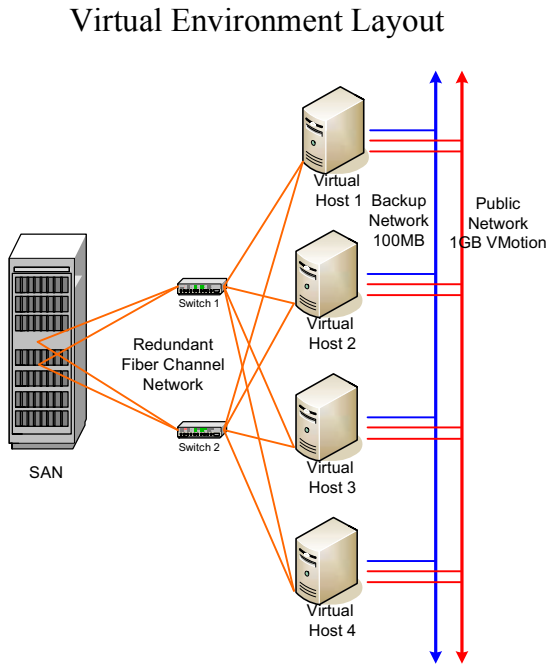

Figure 6: Layout of virtual environment detailing the built-in redundancy

Utilizing an available software feature, VMs can be migrated to another physical host. This migration is done in an active state and causes no server downtime while applications continue to operate uninterrupted. End users are unaware of such migrations. We use this tool to aid in managing downtimes, load-balancing, and other resource alignment needs.

After reviewing multiple virtualization case studies, the team agreed on a 20:1 consolidation ratio limit of VMs to a single physical system. Our initial design consists of 4 physical machines with 15 virtual guests configured on each. This will incorporate 60 ideal candidates targeted for consolidation while reserving resources for potential migrations. In case of physical server failure, the VMs on the failed host would migrate to the 3 remaining hosts as seen in Figure 7. This will permit 5 additional VMs to migrate to each host, respectively, maintaining the 20:1 consolidation and 100% availability.

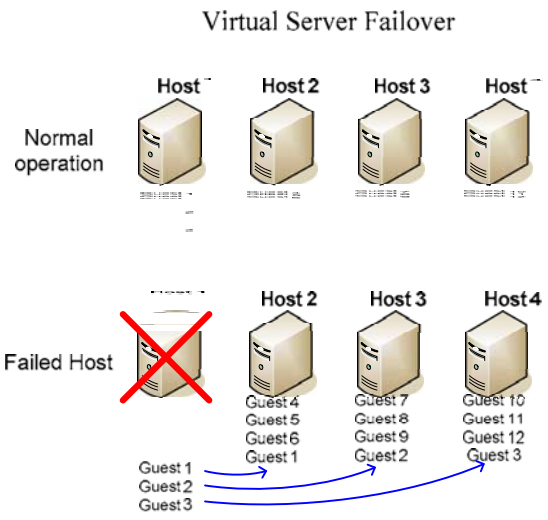

Figure 7: Demonstration of failover when a host system fails

It was easy to justify this project because we were up against several looming obstacles. First, the hardware in use was aged and being purged from our current supportability model. Replacement of this hardware on a one-for-one basis was very costly. Second, the datacenter is constrained by space and power. We needed a solution that would free up physical space in the computing environment. By replacing out-dated servers with virtual servers, we not only saved ~40% on hardware upgrade costs but more importantly extended the capacity of our datacenter. This basic ROI did not investigate costs associated with power, network, AC, etc. Figure 8 shows our first-year ROI.

Return on Investment Worksheet

Component	Qty	Hardware Costs	Software Costs	Rack & Storage	Total Cost
80 servers	80	$9,000.00	$0.00	$6,000.00	$726,000.00
Total to replace aging hardware					**$726,000.00**
Virtualization Software (approximate cost)	1		$50,000.00		$50,000.00
Hardware - (servers, SAN, switches)	1	$243,000.00		$500.00	$244,000.00
Total (Virtualization hardware)					**$294,000.00**
First Year Return on Investment					**$432,000.00**

Figure 8: Our first-year ROI calculation (software costs are approximations)

As the approval, purchasing, and installation of the actual virtualization island was in process, the team utilized a validation environment to begin building server configurations and testing potential candidate servers. To do this, we established an overall test, validation, and implementation plan for our "Ideal Candidate" servers. We notified the owners of these machines of the timeline for testing and identified our criteria for a successful test.

The technical team defined and created a "gold build" server definition (based on the data collected during server classification).

As the testing timeline progressed, server owners were notified three weeks prior to their servers being created. This notification included a detailed timeline for the next five weeks and the requirements for completing a virtualization test. Two weeks before testing began, the server owners met with the virtualization team to discuss special requests, variations from the gold build configuration, and to approve VM resource allocation. After this meeting, the technical team provisioned the new servers and kept them in a "power off" status. The server owners then had to prepare their test plan, success criteria, and migration strategy during these two weeks. The test plan had to include a regression test for any application installed on the server to ensure it executed properly, along with the server functions. Two days prior to the start of virtual server testing, test plans, success criteria, and migration plans were reviewed and approved. Once all requirements were met, the servers were released to the testing team to build their applications, copy data, and configure the server with all required software and application information. The test team did all OS and application testing in a two-week period and met with the virtualization team at the end of the two weeks. When all success criteria had been met, the server was shut down. Once the final hardware landed in the datacenter, the server configurations were moved to the production hardware, restarted, and each test group did a quick validation to ensure the server was in the state it was shut down in.

That was when the virtualization team turned over the "keys" to the server owner and the owners executed their migration plan and moved the physical machine contents to the VM. When the final migration was complete, the physical machines were powered off and removed from the datacenter within 30 days.

RESULTS

When this Intel® business group set out to upgrade its computing environment, the expected results were to have an equal environment to the existing one. After the server virtualization was completed, the expected ROI was realized along with additional benefits such as datacenter floor space, power, cooling, and network relief as well as easier manageability for the IT support team. This installation also built a solid production platform to begin

deploying enterprise services and monitoring capabilities through provisioning of virtual servers for these purposes.

CONCLUSION

In this paper, we explored the issues of implementing virtualization in the enterprise. We analyzed IT services and looked at how those services would be impacted in a virtualized environment.

We looked at several use cases being currently investigated by Intel's IT department. The DVM concept provides a new way of looking at deploying services, and it enables a further use case that we are exploring with our OverMesh implementation and the IT Overlay.

We presented a case study of virtualization of a datacenter in which the VMware ESX Server was used that allowed us to consolidate 20 or even more servers onto a single physical server reducing hardware, electrical, cooling, and administrative costs. Our solution provides robust resource controls for different types of applications, and we can control the levels and limits of CPU, networking, memory, and disk I/O allocated to and used by each virtual system.

Utilizing the virtual environment, IT can quickly create new servers; and virtual servers can be deployed in 30 minutes vs. 60 days to purchase and deploy a physical server. We achieved our goals to minimize our physical footprint in the datacenter, lower our administrative costs, improve our network uptime, and deploy new servers and applications faster. According to Thomas Bittman, research vice president and distinguished analyst at Gartner Inc., "integration of virtualization technology with the operating system is a natural evolutionary step for the x86 platform."

ACKNOWLEDGMENTS

We acknowledge our reviewers Dev Pillai, Mani Janakiram, Greg Priem, Joe Whittle, Nicolas Robins, Vivekananthan Sanjeepan, Robert Adams, George Clement, Raju Nallapa. We also acknowledge the work of Rita Wouhaybi, Gang Ding, Winson Chan, Hong Li, Manish Dave, Claris Castillo, and Stacy Purcell in investigating enterprise uses of Distributed Virtual Machine Technology.

REFERENCES

[1] Goldberg, R., "Survey of virtual machine research," *IEEE Computer Magazine*, 7:34–45, June 1974.

[2] L. Peterson, T. Anderson, D. Culler, and T. Roscoe, "A Blueprint for Introducing Disruptive Technology into the Internet," in *Proceedings of HotNets I,* Princeton, NJ, October 2002.

[3] http://www.geni.org*

[4] Waldspurger, C. "Memory Resource management in VMware ESX Server," in *Proceedings of the Symposium on Operating Systems Design and Implementation* (OSDI '02), December 2002.

[5] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Hegebar, I. Pratt, A. Warfield, "Xen and the Art of Virtualization," in *Proceedings of the ACM Symposium on Operating Systems Principles* (SOSP). October 2003.

[6] Lai, K., Huberman, B., and Fine, L., "Tycoon: A Distributed Market-based Resource Allocation System," in *CoRR: Distributed, Parallel, and Cluster Computing*, 2004.

[7] P. Rundberg and Fredrik Warg, "Freebench v1.03," at http://www.freebench.org/*

[8] Clark, B., T. Deshane, E. Dow, S. Evanchik, M. Finlayson, J. Herne, and J. Matthews, "Xen and the Art of Repeated Research," in *Proceedings of the USENIX Annual Technical Conference*, Boston, July 2004.

[9] JSDL Version 1.0 recommendation, at http://www.ggf.org/documents/GFD.56.pdf*

[10] J. Dilley, et al., "Globally Distributed Content Delivery," in *IEEE Internet Computing*, September/October 2002, pp. 50–58.

[11] Tristan Richardson, Quentin Stafford-Fraser, Kenneth R. Wood, and Andy Hopper, "Virtual Network Computing," *IEEE Internet Computing*, Vol.2 No.1, Jan/Feb 1998, pp. 33–38.

[12] N. Spring, D. Wetherall, T. Anderson, "Scriptroute: A Public Internet Measurement Facility," *USENIX Symposium on Internet Technologies and Systems*, 2003.

[13] Cindy Bickerstaff, Ken True, Charles Smothers, Tod Oace, Jeff Sedayao, and Clinton Wong, "Don't Just Talk About the Weather–Manage It! A System for Measuring, Monitoring, and Managing Internet Performance and Connectivity," in *First Conference on Network Administration* (NETA '99), Santa Clara, 1999.

[14] Cisco Corporation. Netflow, at http://www.cisco.com/warp/public/732/Tech/nmp/netflow/index.shtml*

[15] Keynote. http://www.keynote.com/*

[16] J. Vicente, S. Rungta, G. Ding, D. Krishnaswamy, W. Chan, and K. Miao, "OverMesh: Network Centric Computing," under submission to *IEEE*

Communications Magazine, Emerging Technologies Series, 2006.

[17] IEEE 802.11s ESS Mesh Network working group at http://grouper.ieee.org/groups/802/11*

[18] Andy Bavier, Mic Bowman, Brent Chun, Scott Karlin, Steve Muir, Larry Petersen, Timothy Roscoe, Tammo Spalink, Make Wawrzoniak, "Operation System Support for Planetary-Scale Network Service," in *Proceedings of NSDI '04: First Symposium on Networked Systems Design and Implementation*, San Francisco, March 2004.

[19] Linux VServers at http://www.linux-VServer.org*

AUTHORS' BIOGRAPHIES

Patrick W. Fabian is the business operations manager for Intel's Enabling Technologies and Solutions group in the Technology Manufacturing Engineering organization. He works closely with IT to support his organization's infrastructure and server environment. Patrick holds a B.S. degree in Industrial Management and Computer Science from California University of Pennsylvania. He has over 25 years of IT experience, joining Intel in 1996 as an SAP developer. Along with developing key enterprise applications, his career at Intel includes managing SAP, Web, Teradata ETL, Microstrategy development teams and the infrastructure team responsible for Intel's enterprise data warehouse. His e-mail is patrick.fabian at Intel.com.

Julia Palmer is a senior systems engineer with Information Technology. She joined Intel in 1997 as an Automation Engineer for Fab 18 in Israel. Currently, Julia is leading multiple infrastructure projects for Storage, UNIX*, and Virtualization in the Manufacturing Computing organization. She holds an M.S. degree in Computer Science from Belarusian State University of Informatics and Radioelectronics. Her e-mail is julia.palmer at intel.com.

Justin B. Richardson is a Microsoft Certified Systems Engineer currently working for IT Manufacturing Computing Global Solutions. He joined Intel in 1996 and has used his expertise of server infrastructure and mass-storage solutions to support several manufacturing environments. His current virtualization projects are enabling server consolidation in a large-scale datacenter. His e-mail is justin.b.richardson at intel.com.

Mic Bowman is a principal engineer within Intel's System Technology Laboratory and principal investigator for the Distributed Virtual Machines Strategic Research Project. Bowman received his B.S. degree from the University of Montana, and his M.S. and Ph.D degrees in Computer Science from the University of Arizona. He joined Intel's Personal Information Management group in 1999. While at Intel, he developed personal information retrieval applications, context-based communication systems, and middleware services for mobile applications. Prior to joining Intel he worked at Transarc Corp. where he led research teams that developed distributed search services for the Web, distributed file systems, and naming systems. His e-mail is mic.bowman at intel.com.

Paul Brett joined Intel UK in 2000 as part of Intel´s Online Services group. He is currently working in Hillsboro, Oregon, focusing on Distributed Systems Management tools for developing, deploying and accessing planetary-scale services. From 1988 to 2000, Brett worked on the design and implementation of dependable systems for air traffic control. He is a graduate of the UK's Open University, where he earned a First Class Honours degree in Systems Engineering of software-based systems. His e-mail is paul.brett at intel.com.

Rob Knauerhase is a staff research engineer with Intel Labs. His professional interests include machine virtualization, Internet technologies, distributed systems, system software, and information privacy in the digital world. Knauerhase received an M.S. degree in Computer Science from the University of Illinois at Urbana-Champaign, and a B.S. degree in Engineering from Case Western Reserve University. He holds 14 issued patents, with more than 60 patents pending. He is a senior member of the IEEE and the IEEE Computer Society. His e-mail is knauer at jf.intel.com.

Jeff Sedayao is a staff research engineer in Intel's IT Research Group. He focuses on IT uses of virtualization, including applying PlanetLab and PlanetLab developed technologies to enterprise IT problems. Jeff has participated in IETF working groups, published papers on policy, network measurement, network and system administration, and authored the O'Reilly and Associates book, *Cisco IOS Access Lists*. His e-mail is jeff.sedayao at intel.com.

John Vicente, a senior principal engineer, is the director of Information Technology Research and chair of the IT Research Committee. John joined Intel in 1993 and has 22 years of experience spanning R&D, architecture, and engineering in the field of networking and distributed systems. John has co-authored numerous publications in the field of networking and has patent applications filed in internetworking and software systems. He is currently a Ph.D. candidate at Columbia University's COMET Group in New York City. John received his M.S.E.E. degree from the University of Southern California, Los Angeles, CA in 1991 and his B.S.E.E. degree from Northeastern University, Boston, MA in 1986. His e-mail is john.vicente at intel.com.

Cheng-Chee Koh is a senior systems engineer in Intel's Engineering Computing Group at Santa Clara, California. She received her B.A. degree in Mathematics and her M.S. degree in Computer Science from Indiana University, Bloomington. Her current interests include messaging, interactive computing, information security, and virtualization. Cheng-Chee has been with Intel for 13 years. Her e-mail address is cheng-chee.koh at intel.com.

Sanjay Rungta is a principal engineer with Intel's Information Services and Technology group. He received his B.S.E.E. degree from Western New England College and his M.S. degree from Purdue University in 1991 and 1993, respectively. He is the lead architect and designer for the Local Area Network for Intel. He has over 13 years of network engineering experience with three years of experience in Internet Web hosting. He holds one United States patent and three pending in the area of Network Engineering. His e-mail is sanjay.rungta at intel.com.

[Δ] Intel® Virtualization Technology requires a computer system with an enabled Intel® processor, BIOS, virtual machine monitor (VMM) and, for some uses, certain platform software enabled for it. Functionality, performance or other benefits will vary depending on hardware and software configurations and may require a BIOS update. Software applications may not be compatible with all operating systems. Please check with your application vendor.

[Φ] Intel® processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. See www.intel.com/products/processor_number for details.

Redefining Server Performance Characterization for Virtualization Benchmarking

Jeffrey P. Casazza, Digital Enterprise Group, Intel Corporation
Michael Greenfield, Software and Solutions Group, Intel Corporation
Kan Shi, Digital Enterprise Group, Intel Corporation

Index words: virtualization, benchmark, server consolidation

ABSTRACT

Virtualization will dramatically change enterprise system deployments and is being driven by innovation across a broad set of platform technologies. All of this attention has created a broad interest in comparing different products. One challenge is that there are no established performance methodologies to measure virtualization performance. Today's existing server benchmarks cannot be easily used as-is by an end user to generate clear and relevant results. This paper presents a workload methodology to help the reader characterize the performance of servers exploiting virtualization technologies to consolidate multiple physical servers. It presents an example benchmark using applications often found in typical virtualization deployments. The existence and exploitation of a standard methodology is a key to accelerating continued improvement of virtualization technologies.

INTRODUCTION

Virtualization has been part of the datacenter since the 1960s when it was exploited across mainframe systems [1]. Virtualization is experiencing a renaissance as this technology is finding its way to high-volume servers. To the IT technologist, virtualization brings the promise of solving several datacenter problems. Virtualization can reduce costs by consolidating older servers. It helps organizations become more nimble through fast provisioning of virtual servers. It improves equipment utilization and the end-user experience by enabling dynamic load balancing and improved disaster recovery capabilities. These benefits provide a strong motivation for accelerating server virtualization deployments.

Virtualization provides an abstraction of hardware resources enabling multiple instantiations of operating systems (OSs) to run simultaneously on a single physical platform. The abstraction provides isolation between the separate running partitions to prevent individual faults from affecting the entire system. The virtualization of the hardware also means that different OSs can be supported on a single platform simultaneously—even older OSs. Consolidating several physical servers that have workloads with non-overlapping peak utilization requirements over time allows better hardware utilization than if these were carried out on separate systems. These benefits are attractive in environments with legacy servers that, though important to the business, cannot justify the porting and validation activity to a newer OS [2].

In this paper we present a methodology and an example for characterizing the performance of servers using virtualization to consolidate multiple physical servers. We provide a general overview of two key virtualization usage models. We also briefly look at how contemporary methods can be applied to virtualization. We discuss the challenges generated by the virtualization abstraction layer and consolidation, and we present a systematic approach to performance measurement. Finally. an example workload, called vConsolidate, is defined to further clarify the methodology.

ENTERPRISE VIRTUALIZATION USAGE MODELS

The value virtualization brings to datacenters depends on what problems it can solve for the IT technologist. These are defined to be virtualization's usage models. Virtualization usage models today are focused on legacy consolidation, flexible provisioning, test/development, dynamic load balancing, and disaster recovery. As the technology penetrates the mainstream datacenter, new usage models will likely emerge.

The performance discipline [1] presented in this paper provides a basic framework fitting multiple usage models. However, we focus specifically on two usage models due to their historic appeal to datacenter managers: legacy consolidation and flexible provisioning

Legacy consolidation refers to transferring application and OS stacks from multiple servers to a fewer number of (typically more powerful) servers. The old "legacy" servers are often running older OSs because upgrading to a newer version has some manpower impacts. For example, a legacy application may need some porting work, but the resources to do this cannot be justified. In this paper an old (legacy) server is classified to be 3+ years old. The utilization level of these legacy systems is often quite low, 5-15%. Such low utilization of an older platform means that a contemporary server could provide the processing capability for running many multiples of the legacy applications simultaneously. Market research indicates that legacy consolidation projects with virtualization tend to mix different application types on the new consolidated server; however, IT managers are avoiding mixing OS types [3]. The final consolidated server typically has some mixture of Web servers, e-mail servers, database servers, and/or other types of applications running simultaneously, but all are running the same type of OS (e.g., Linux* or Windows*) and often the same version of the OS [3].

Flexible provisioning is essentially forward-looking consolidation. As new servers are requested from the IT department, rather than deploying a new physical server for each request, the deployment is a virtual slice of an existing server. For example, if a two-processor server is needed to support a new Web server application, perhaps two virtual processors of a multi-processor server are deployed in support of the request. This drives up the utilization of the servers, thereby improving efficiency. The virtual provisioning is often much faster, as hardware may already exist to support the request, eliminating the procurement cycle.

These two usage models, legacy consolidation and flexible provisioning, constitute the main driver behind the adoption of virtualization in mainstream datacenters. The focus of this methodology is to measure the performance of virtual servers in a way that is related to these two usage models. Other usage models, like test and development, are also popular but not addressed in this paper.

[1] A structured and consistent set of methods and processes that are accurate and repeatable.

Having the usage model defined is important for developing a relevant performance benchmark. The usage model provides boundary conditions to the testing. For example, the usage model indicates what types of applications should be used, whether the application and OS stacks are heterogeneous or homogeneous, what is the configuration of a typical system-under-test (SUT), and what is the appropriate size and number of the virtual machines (VM).

VIRTUALIZATION PERFORMANCE CHARACTERIZATION CHALLENGES

A question that may come to mind is "Why can't existing performance characterization methods be exploited?" Users already have many accepted methods and tools to characterize servers. Some of these include load generators (e.g., LoadRunner*) and a myriad of industry standard (e.g., SPEC*, TPC*) and proprietary workloads (e.g., SAP-SD 2 Tier*, MMB3*, R6iNotes*). There are several challenges presented in virtualization performance characterization including consolidation, virtualization, and implementation considerations. These limit the use of existing methods.

Consolidation Characterization Challenges

We need to differentiate between consolidation and virtualization challenges, as both introduce complexity into performance measurement and tuning. Virtualization facilitates creating multiple VMs on one physical machine. Consolidation relates to running multiple workloads on the system at the same time.

A challenge with consolidation characterization is the mixture of different workloads. If you consolidate a set of heterogeneous workload environments, consider that each will have a different set of requirements and metrics and that depending upon the users' specific requirements, the relative priority of each will vary across users, time, and other dimensions.

Another consolidation challenge relates to resource profiles. The non-steady state resource profile of the individual servers will look quite different from that of the consolidated system [4]. It is simplest to measure performance when all measurements are conducted in a time window after all workloads are in a steady state. While this may be nice for a benchmark, it fails to represent many real-world usage models. Consider the following examples:

- Most e-mail servers have distinct periods where the demands upon them vary a great deal. For example, the system may be idle until a wave of people arrive at work and log in, download their e-mail, and make other demands on the server. Conversely, the

demands on the server would decrease as people finish up for the day.

- A Web store that supported a worldwide customer base could be busy 24x7 and reach a steady state as opposed to some service that was provided to people in one locale.
- Some workloads have seasonal variations, end-of-month closings, holiday duty cycles, and other modifications that may differ greatly from normal operations.

Consider two examples of a consolidated e-mail server: Web store server, and a customer relationship management (CRM) server. In the first scenario, Figure 1, we see that none of these is ever run in a steady state. If these are the actual profiles of the consolidated server, it would be prudent to examine peak resource requirements when superimposed at one instant of time to determine how well the overall system is performing.

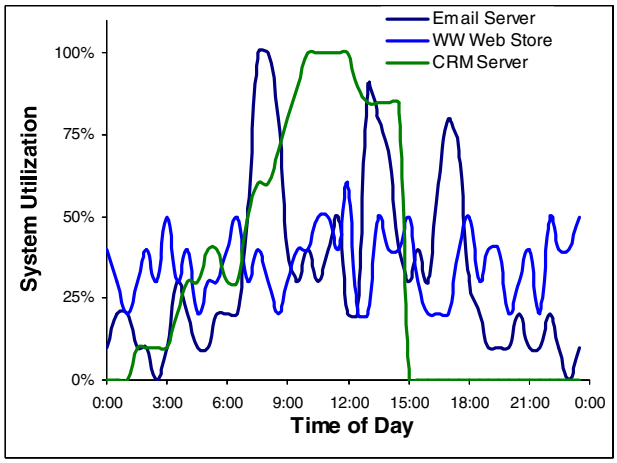

Figure 1: System resource profile for workloads that are not operating in a steady state

In the second example, Figure 2, we see three server utilization profiles that all reach a steady state. If we were to examine the performance of the consolidated system and did our study at some point after 15 hours of running, we would see a much simpler profile of the workloads in a steady state. While the second workload is easier to test and tune, it may not reflect the actual end-user resource profile.

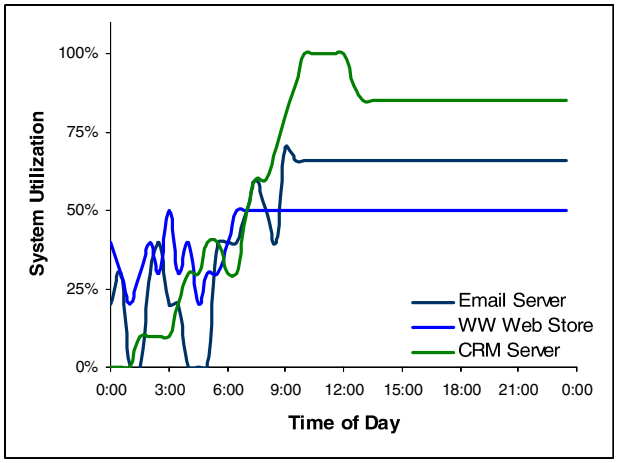

Figure 2: System profile for workloads that are operating in a steady state

Virtualization Characterization Challenges

Whatever performance tools, methods, and processes are used for the characterization, tuning and simulation of server-based workloads they are likely to continue to be relevant in a virtualized environment. As much as we would like to have a single benchmark (or a small set of benchmarks) to describe server performance, there is nothing as good as the actual end-user workload (what they do today and how that will change over time) to employ in developing a performance and projection discipline. This will also be true for virtualization performance, since no single workload will characterize all user requirements. Consider some different user requirements which may include the following:

- A threshold minimum throughput must be maintained over time.
- Some margin must be available for peak workload requirements or for future expansion.
- The server provides some service and the response time to any specific request or set of requests cannot exceed some specified quality-of-service threshold.

We can better understand some of the new challenges that are introduced in the context of virtualization when we consider the requirements for and how a system will be used. For our example, server environments are associated with the consolidation of existing (often legacy) systems and the virtual partitioning of an existing platform for new server deployments. The diversity of what is being consolidated requires that no one workload or environment can be used as a general proxy for (most) others. Consider the diversity of essential components (and how poorly one workload would serve as a proxy for another):

- One OS as a proxy for all others (e.g., Windows* to represent Linux*)
- One usage model or vertical to represent another (Linpack to represent MMB3)

Listed below are some of the other new challenges virtualization adds:

- There are many different options on how a platform will be partitioned and how resources can be allocated, each dramatically affecting the performance of each and how they interact with each other. These are further compounded depending upon what the goals are: for example, absolute performance, minimum performance thresholds, power consumption, TCO, or other optimization criteria.
- There are different strategies that can be used to evaluate a system, including response time, throughput, percentage utilization, and others. These may be exploited simultaneously across separate workloads running across different VMs in a single performance discipline.

Implementation Challenges

As different software stacks are combined inside a set of VMs, there are considerations that may affect precision and repeatability of the results. Often these are tied to the specific implementation of the virtualization abstraction layer and underlying platform. Though by no means an exhaustive list, such issues could include the following:

- *VM clock accuracy/precision*: Since there are several VMs running on a single platform, there is a variety of approaches to how the virtual clock is mapped to the physical platforms' clock, and any of these can cause clock skew. Since most benchmarks will compute a performance metric based upon the always assumed correct system clock, any changes in the clock behavior could lead to errors in computing the delivered performance. Such issues, as well as ways to minimize this possibility, are further explored in VMware [5].
- While an extensive set of system performance monitors are available under most native operating systems (OSs), most virtualization monitors provide only the most basic performance monitoring capabilities. This is sure to improve over time, but the combination of the environment getting more complicated from both consolidation and virtualization and the nascent state of performance monitoring conspire to increase the difficulty to comprehend and productively tune the system.
- All virtualization implementations introduce an additional level of abstraction and not unexpectedly, additional overhead. This makes appropriate system configuration even more important than it is for unvirtualized environments, since resource limitations usually drive up the context switching rates, perhaps at multiple levels of abstraction. Being more generous with memory and I/O capacity when setting up the system initial configuration in a virtualized environment can offer an even larger return in performance and price/performance than non-virtualized environments. As a simple example, a reduction in page fault activity after adding some RAM in a virtualized environment is likely to pay an even larger dividend than in the pre-virtualized environment.
- Many unvirtualized server benchmarks will have a range of observed performance. When multiple workloads are consolidated on a platform and hosted in VMs, this likely adds more variation, particularly if any of the constituent workloads can impact each other or are tested before they are running in a steady state. Readers are encouraged to run their experiments as many times as is necessary to understand the performance profile and variation from run to run.
- Obtaining consistent and predictable performance results assumes that scheduling across VMs is equitable and consistent. It is possible in a virtualization benchmark that the scheduler is not providing what appears to be an equitable distribution of compute and I/O resources across the VMs. For example, if you had N identical copies of a particular workload with the same virtualization monitor configurations, you would expect each to get 1/Nth of the resources available on the system. It is suggested that performance analysts inspect the system during benchmarking to ensure that expected resource profiles are observed.
- Some virtualization monitors will give you various options to map physical CPUs to virtual CPUs and to create affinity between certain sets or to allow a more general pool of resources to be shared amongst all VMs. Virtualization monitors may also permit the setting of weights or CPU percentage to each workload. The higher the workload's weight, the more it will be scheduled to use CPU resources. How to set these depends upon user requirements. For example, is it desirable to ensure that some CPUs are dedicated to certain workloads, or do you want the flexibility for the VM to allocate CPUs based upon dynamic workload changes in real-time? Is one of the

workloads more important than others and therefore should a bigger weight be assigned to it?

When consolidating multiple workloads on a single physical platform, a number of physical devices need to be shared between VMs. Some platforms and virtualization monitors provide different options on how to map the physical devices to virtualized devices. Some physical devices can be assigned solely to a specific workload or just shared between a set of VMs. It depends on customer requirements to set the options. For example, customers can decide to assign a NIC to a Web-bound workload exclusively, and all other more compute bound workloads will share another NIC.

VIRTUALIZATION PERFORMANCE DISCIPLINE

The following is a performance discipline to develop a consolidated virtualization benchmark. The steps are serial in nature and provide a framework to help ensure a reasonable result.

Select workloads, performance metrics, thresholds, and weight factors.

We start with a complete definition of what work needs to be serviced, how we will measure how well the system is meeting the requirement, and if there are any minimum requirements. Weight factors are developed to ensure that resources are allocated in a manner that favors the most demanding workloads.

Sizing the platform and selecting the initial configuration.

Once the requirements are comprehended, initial selection of what platform, configuration, and virtualization technologies (hardware and/or software virtualization, choice of VM implementation, etc.) will be made and how the system will be set up will be decided. The initial platform capabilities can be approached in (at least) two ways. One approach is to start with a platform with clearly more resources (CPU, RAM, network, disk) in every dimension and iteratively reduce them until the right balance is achieved. Alternatively, you can start with a configuration that is too low and add resources until the desired balance is achieved. The former is much quicker and the latter is less expensive.

Define aggregation strategy (how the performance of the constituent workloads will map to an overall performance metric).

This step allows the user to assign and refine the priorities of the different workloads and results that the system-under-test (SUT) will deliver. By assigning weights that are most representative of the user-specific requirements, user perceived improvements can be achieved as opposed to random shifts of resources and workload impact that cannot be comprehended and exploited. It is also during this step that you must consider the repeatability for each workload and whether you will observe and measure it in a transient or a steady state and compensate for interactions between the constituent workloads.

Develop and iteratively refine the mapping of workloads-to-VMs and VMs-to-platform resources.

There are many resource decisions that will dramatically affect how well the system responds by workload. A baseline mapping of these resource decisions would include how to map workloads to VMs and how to relate VMs to system resources (cores, I/O, etc.). Some adjustments to the workloads may be motivated by some of the resource decisions made here.

Tune each workload as if it was not virtualized and establish a baseline, then iteratively refine it (optional step).

It is assumed that each workload is well understood in terms of having a performance discipline and that all tuning steps are fully exploited. This almost always is the best starting point for characterization and optimization of the consolidated virtualized environment. Where the two workloads that are being consolidated have mutually exclusive configuration requirements, it may be best to use either some blending of both or use the one that affects the most heavily weighted workload.

Measure the performance and aggregate it into a performance result.

A comparison to defined thresholds to see if the overall result is valid is conducted, and then the performance of each workload is assessed and aggregated into a single metric.

Identify bottlenecks and attempt to shift resources, augment configuration, and optimize components as needed to drive continuous improvement.

This step focuses on identifying bottlenecks, making configuration adjustments, and making other modifications that help drive improvements in the aggregated performance result for the entire system. It should be noted that there may be limitations in the performance monitoring tools available in a virtualized environment.

Iterate as needed.

This better reflects the tuning and optimization that are usually part of any measurement process, and the focus here is influenced by the difference between the requirement and measured result and the resources available.

VCONSOLIDATE EXAMPLE

While there is no one proxy benchmark for all virtualization deployments, it is highly desirable to have one since it provides a basis of comparison and a basis for a consistent approach to measure and continuously improve. We will arbitrarily define a benchmark called vConsolidate. There are as many possible workloads as there are servers. No two users (or usages) are exactly alike, so we need to make reasonable representative approximations. The purpose of this section is to illustrate how to build up a performance discipline in a virtualized environment with one viable example; and to show the reader how they can do this with component choices that are aligned with their respective environment. The reader should look at this as a basis that can be progressively refined.

In Figure 3, we present a high-level view of the system running vConsolidate. At the base, we have the physical platform, then a virtualization monitor, and some multiple number of VMs, each running a designated workload. We have also defined an aggregation strategy that helps us consolidate workloads, define the performance metric(s) and tell how measurements will be taken.

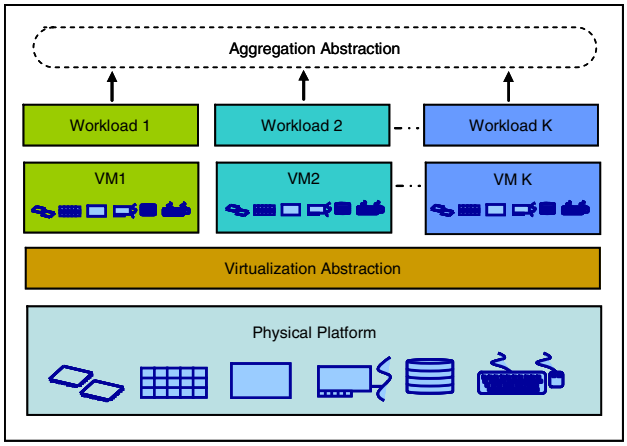

Figure 3: vConsolidate concept

We will construct a hypothetical example to illustrate the performance discipline discussed above. In this example, we select the following environments for consolidation: Web servers, e-mail servers, and database servers. For this example, we model the system by having some multiple of each running and designate the weights of each as Weight[i]. All the weights are fixed numbers pre-defined by the workload. Each workload can be run on the system before and after any virtualization layer is included, and we can compare the virtualized performance of each workload with a baseline measured without virtualization on a pre-defined standard machine. This serves as a useful tool for calibrating subsequent results. We replicate these workloads as needed based upon a set of usage model requirements. Then, we calculate the ratio of virtualized/baseline as the relative performance of each workload. The performance of the virtualized environment would be

$$\sum_{I=1}^{N} \text{Weight}[i]*\text{WorkloadPerf}[i]$$

where WorkloadPerf[i] is the relative performance of the i'th workload

Here is an example:

- The consolidation workload consists of one Web server, one e-mail server, and one data base server.
- To mimic the real-world scenario, one idle VM is also running on the physical system. In that VM, no real workload is running. We do not take any score from this idle VM.
- The weight factors are 35% for the one Web server, 20% for the e-mail server, and 45% for the database server. This would correspond to a weight vector of (0.35, 0.20, 0.45).
- After testing each workload individually in a non-virtualized pre-defined standard machine with a specific configuration, we can get the baseline of each workload.
- At this stage, we define how the workloads were mapped into VMs, how the VMs were mapped to the underlying physical platform, and how resources were allocated amongst each. This is not defined by the workload. The user chooses the best VM configuration settings to do the measurements.
- Each of the workload components has a well-defined performance metric and a known, unvirtualized baseline result. The observed result for the virtualization of each of the component workloads is normalized to this known baseline. The resulting benchmark metric is the combination of all of the normalized workload component results. As an example, the performance results ratio vector WorkloadPerf[i] could look like (1.8, 1.5, 2.3)[2]. The rollup result for the performance of the virtualized system would be

[2] The reader is reminded that these are ratios to some predefined (different and likely older) standard system and are likely to be greater than 1.0.

$$\sum_{I=1}^{N} \text{Weight}[i]*\text{WorkloadPerf}[i]$$

or

(0.35, 0.20, 0.45) * (1.8, 1.5, 2.3) = 1.965

This calculation is illustrated in Figure 4. The result becomes most useful when comparing different configurations. For example, the same consolidated set of workloads can be compared across different platforms or against two different virtualization monitors, or it can be used to compare two different sets of configuration settings.

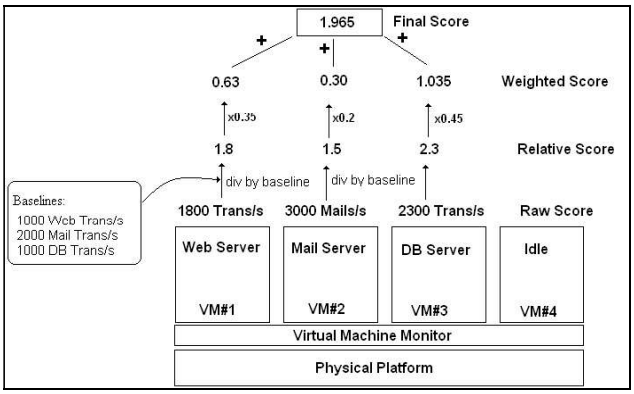

Figure 4: vConsolidate example virtualization benchmark results calculation

INDUSTRY-STANDARD VIRTUALIZATION BENCHMARKS

We have established a general methodology for developing a virtualization benchmark. The value of the benchmark will be amplified to the degree that its component workloads map to the actual work and metrics of the particular usage model. There is great value in having a proxy industry-standard benchmark since it permits the community to focus on some standard repeatable test that facilitates comparisons between configurations and platform technologies. Virtualization opportunities require that some abstraction on top of a set of workloads is developed, for example a throughput methodology (SPECthruput* and later SPECrate*) [6] applied on top of the well-known SPEC* CPU benchmarks.

Despite virtualization technologies and implementations dating back more than 30 years, no de-facto or industry-standard performance metric exists. As virtualization is exploited on commodity high-volume platforms across a range of server workloads, the community needs standards to compare virtualized performance (e.g., [7]). These are some of the motivations for an industry-standard benchmark:

- Motivate the industry to add a performance and optimization discipline to virtualization platforms, monitors, and tools.
- Help users considering virtualization to compare alternatives, particularly where they were unwilling or unable to develop, execute, and maintain their own benchmarks.
- Accelerate both the development and application of virtualization technologies.

What is a server virtualization performance benchmark and how could it be used? It is a performance discipline that includes consolidated server workloads running in virtualized partitions. It is most likely to be useful to compare platforms and virtualization technologies. Some examples on how it would be used are given:

- How does the performance, power efficiency, or price/performance of Platform A compare with Platform B?
- How does VM monitor A compare with VM monitor B (an upgrade, a competitive comparison, comparison of software and hardware accelerated implementations)?
- What is the effect of a configuration modification or a component substitution?

What is a good benchmark and how is this question modified by virtualization? We will first list what we believe is required of benchmarks and then what is desirable.

First, required attributes of benchmarks:

- They should be relevant to some well-defined user constituency like Web servers, application servers, database servers, and e-mail servers, that would be likely considered for consolidation.
- They should be timeless, in the sense they do not specify any platforms, technologies, or usage models that will not make sense over some reasonably long time.
- They should provide repeatable results from run to run and provide a means to ensure that valid results were obtained.
- They should be agnostic (in terms of platforms, virtualization monitors, and as many other parts of the stack) to the extent possible.

- They should be able to automate, repeat, scale, consolidate, and compare performance observations across a wide range of systems, times, and components.
- They should use a simple, single unit of measure for performance results (and associated component scores for those preferring more detail).
- They should be citable, which depends upon components that are freely accessible to the community.

Second, desirable attributes of benchmarks:

- They should not require infrastructure that will block most organizations from putting the system together. This rules out workloads that need a huge number of disks, a load-generation infrastructure, or an infrastructure that is otherwise encumbered.
- They should be easy to test. For example, load generation is considered a minor performance consumer and can be run within the test environment (no external load generators needed).
- They should be easy to obtain, set up, and test. For example, they should use relatively low-cost, easy-to-access and set up applications, tools, and utilities.

There are many practical challenges in constructing such a specification. Here are some examples.

- If you were to start with existing workloads, some of the most visible contemporary industry-standard server workloads may limit other components from participating in the benchmark or introduce complexities that cannot be easily managed across a range of systems and timeframes. One solution would be to develop new server workloads that were totally portable, but this would introduce a new challenge in that these would be unproven and unknown.
- Many benchmarks artificially attempt to saturate the CPU and marginally reflect best configuration practices.
- Some virtualization technologies may have different feature sets and in order to maximize those that could participate, we would have to limit the features that could be exercised, for example migration of VMs.
- Perhaps the most difficult set of decisions is to pick the constituent workloads and define an aggregation process—since there is no single right set. This will require compromises before it can be accepted by the community.

Driving a standardized benchmark is the best way to create a performance discipline, achieve continuous improvement at every layer of the platform stack, and ultimately develop an industry standard for measuring and optimizing virtualization performance.

CONCLUSION

In order to be agnostic to the underlying operating environments and implementation, we have taken a high-level approach that assumes a coherent performance discipline on each constituent workload component and have presented an approach on how to aggregate these into a unified metric for comparing systems. Since there are many different user requirements for assessing and projecting performance, there is no one right answer for what workloads should be included and how they should be combined, so we have provided readers with considerations for doing this in a manner that best addresses their needs. The vConsolidate benchmark was presented as an example implementation, highlighting the compromises required in workload selection, component definition, and metric aggregation. Industry standards for measuring the performance of virtualization environments will help accelerate the performance and deployment of same. The performance discipline presented outlined a basic framework that could be used to create industry-standard virtualization benchmarks.

ACKNOWLEDGMENTS

We thank Tom Adelmeyer, Andrew Anderson, Mark Brown, Clement Cole, Aaron Holzer, Ken Rule, and Jeff Smits for their technical contribution and review comments. Thanks to the paper's advisory sponsors: Naresh Sehgal, Paul Barr, Rich Uhlig, and Fernando Martins. A special thanks to the performance engineering team including Nan Bo, Vivi Huang, Steven Thomsen, Steve Yang, Mark Brown, Aamir Yunus, Ashok Emani, Anthony P. Bertapelli, and Priyalal Kulasinghe for evaluating many consolidation and virtualization test methodologies.

REFERENCES

[1] Creasy, R.J., "The Origin of the VM/370 Time-Sharing System," *IBM Journal of Research and Development*, vol. 25, no. 5, p. 483.

[2] Uhlig, R.; Neiger, G.; Rodgers, D.; Santoni, A.; Martins, F.; Anderson, A.; Bennett, S.; Kagi, A.; Leung, F.; and Smith, L., "Intel Virtualization Technology," *Computer*, vol. 38, no. 5, pp. 48–56.

[3] IDC. Server Virtualization 2005, June 2005, pp. 1–96.

[4] Parthasarathy Ranganathan and Norman Jouppi, "Enterprise IT Trends and Implications for

Architectural Research (2005)," in *Proceedings of the 11th International Symposium on High Performance Computer Architecture* (HPCA-11 2005), at http://www.hpcaconf.org/hpca11/papers/24_x_ranganathan-enterpriseittrends.pdf*

[5] VMware Whitepaper (2005), Timekeeping in VMware VMs, at http://www.vmware.com/pdf/vmware_timekeeping.pdf*

[6] Greenfield, Mike; Rayhona, Paul; and Bhandarkar, Dileep, "SPEC adopts new methodology for measuring system throughput. SPEC Newsletter," 2 (2), Spring 1990.

[7] Green, Diane (2006), Web Blog at http://www.vmware.com/vmtn/blog/diane/* advocating consistency and ease of making performance comparisons.

AUTHORS' BIOGRAPHIES

Jeffrey P. Casazza joined Intel in 1988 and has contributed to various projects including the MCS®96, i860®, Itanium® and Xeon® processor families. He holds a B.S.E.E. degree from Arizona State University. His e-mail is jeffrey.p.casazza at intel.com.

Michael Greenfield is a principal engineer at Intel. He joined Intel in 1997 and has held a variety of management and development roles in Workstations, High Performance Computing, and System Engineering. He holds an M.S.M.E. degree from Cornell University. His e-mail is michael.greenfield at intel.com.

Kan Shi obtained his B.S.E.E. degree from Shanhai Jiaotong University. He has been with Intel since 2003. Kan is currently a marketing engineer with Intel's Digital Enterprise Group responsible for server virtualization performance testing. His e-mail is kan.shi at intel.com.

THIS PAGE INTENTIONALLY LEFT BLANK

For further information visit:

developer.intel.com/technology/itj/index.htm