Software and Hardware Techniques for x86 Virtualization

VMware® ESX

In the early days of x86 virtualization, uniformity ruled: all CPUs implemented essentially the same 32-bit architecture and the virtual machine monitor (VMM) always used software techniques to run guest operating systems. This uniformity no longer exists. CPUs today come in 32- and 64-bit variants. Some CPUs have hardware support for virtualization; others do not. Moreover, this hardware support comes in multiple forms for virtualizing different aspects of the x86 architecture.

This document describes the x86 architecture from a virtualization point of view, relating critical architectural features to the major releases of VMware ESX. The goal is to provide, for each version of VMware ESX, an understanding of:

- Which CPU features are required
- Which CPU features can be utilized (but are not required)
- Which CPU features can be virtualized—that is, made available to software running in the virtual machine

With a better understanding of how CPU features are required, used, and virtualized by VMware ESX, you can reason more precisely about what can be virtualized, what performance levels may result for a given combination of CPU, guest operating system, and version of VMware ESX, and how workloads may respond to adjusting configuration parameters both for software running in the virtual machine and at the VMware ESX level.

Much of the information in this guide also applies to VMware hosted products, such as VMware Workstation for Linux and Windows. For simplicity the guide focuses on ESX, but you can apply the following rule of thumb: any Workstation release generally behaves similarly to the immediately succeeding release of ESX.

This guide covers the following topics:

Brief History of the x86 Architecture

The x86 architecture has roots that reach back to 8-bit processors built by Intel in the late 1970s. As manufacturing capabilities improved and software demands increased, Intel extended the 8-bit architecture to 16 bits with the 8086 processor. Later still, with the arrival of the 80386 CPU in 1985, Intel extended the architecture to 32 bits. Intel calls this architecture IA-32, but the vendor-neutral term *x86* is also common.

Over the following two decades, the basic 32-bit architecture remained the same, although successive generations of CPUs added many new features, including an on-chip floating point unit, support for large physical memories through physical address extension (PAE), and vector instructions.

In 2003, AMD introduced a 64-bit extension to the x86 architecture, initially dubbed AMD64, and began shipping 64-bit Opteron CPUs in 2004. Later in 2004, Intel announced its own 64-bit architectural extension of IA-32, calling it IA-32e and later also EM64T.

The AMD and Intel 64-bit extensions are extremely similar, although they differ in some minor ways, one of which is crucial for virtualization (see the discussion of segment limits in "Software Technique: Binary Translation" on page 3). This guide uses the vendor-neutral term *x64* to refer to the 64-bit extensions to the x86 architecture.

Brief History of VMware ESX

In 1999, VMware released the first version of VMware Workstation. It ran on, and virtualized, 32-bit x86 CPUs. Soon after, VMware shipped the ESX Server product. In contrast to Workstation, which depended on either a Linux or Windows host, ESX Server deployed a custom-built kernel instead. This VMkernel is designed to scalably and efficiently run a workload that consists primarily of virtual machines while providing strong information and performance isolation among the virtual machines.

Table 1. Physical CPU Requirements and Virtual CPU Options

	ESX 1.0-2.5	**ESX 3.0**	**ESX 3.5**	**ESX 4.0**
VMkernel	32-bit	32-bit	32-bit	64-bit
Virtual CPU	32-bit	32- and 64-bit	32- and 64-bit	32- and 64-bit

The VMkernel row in Table 1 shows the architectural requirements for running the VMkernel itself in different versions of ESX. All versions of ESX before 4.0 can run on the 32-bit x86 architecture. They also can run on x64 CPUs but do not take advantage of the 64-bit architectural extensions. Beginning with VMware ESX 4.0, a 64-bit CPU is required to run the VMkernel. Although this requirement causes a slight loss of hardware compatibility, by 2009 the majority of server CPUs implement the x64 architecture, making it desirable to use the large 64-bit address space and other architectural advances to improve performance and scalability.

Virtualizing 32- and 64-bit CPUs

The VMkernel does not run virtual machines directly. Instead, it runs a VMM that in turn is responsible for execution of the virtual machine. Each VMM is devoted to one virtual machine. To run multiple virtual machines, the VMkernel starts multiple VMM instances.

Because the VMM decouples the virtual machine from the VMkernel, it is possible to run 64-bit guest operating systems on a 32-bit VMkernel (and vice versa) as long as the underlying physical CPUs have all the required features. The logic to handle the subtleties of 32- and 64-bit guest operating systems is encapsulated within the VMM, which can take advantage of a 64-bit physical CPU to run a 64-bit guest operating system efficiently, even if the underlying VMkernel runs in 32-bit mode. The Virtual CPU row in Table 1 shows which versions of ESX can run just 32-bit virtual machines and which can run both 32- and 64-bit virtual machines.

Execution Modes

The VMM implements the virtual hardware on which the virtual machine runs. This hardware includes a virtual CPU, virtual I/O devices, timers, and other devices. The virtual CPU has three features of interest: the virtual instruction set, the virtual memory management unit (MMU), and the virtual interrupt controller (PIC or APIC). The VMM can implement each of these aspects using either software techniques or hardware techniques.

The combination of techniques used to virtualize the instruction set and memory determines an *execution mode*.

For diagrams illustrating the techniques described in this section, see "Performance Aspects of x86 Virtualization" (see "References" on page 9 for a link).

Instruction Set Virtualization

In order to run one or more virtual machines safely on a single host, ESX must isolate the virtual machines so that they cannot interfere with each other or with the VMkernel. In particular, it must prevent the virtual machines from directly executing privileged instructions that could affect the state of the physical machine as a whole. Instead, it must intercept such instructions and emulate them so their effect is applied to the virtual machine's hardware, not the physical machine's hardware. For example, issuing the reboot command in a virtual machine should reboot just that virtual machine, not the entire host.

This section compares binary translation, a software technique for running the virtual machine's instruction stream, with the hardware support for virtualization found in recent CPUs from Intel (the VT-x feature) and AMD (the AMD-V feature).

Software Technique: Binary Translation

The original approach to virtualizing the (32-bit) x86 instruction set is just-in-time binary translation (BT). This approach is implemented in all versions of VMware ESX, and it is the only approach used in VMware ESX 1.x and 2.x. For specificity, because this technique virtualizes the 32-bit architecture, we call it BT32.

When running a virtual machine's instruction stream using binary translation, the virtual machine instructions must be translated before they can be executed. More concretely, when a virtual machine is about to execute a block of code for the first time, ESX sends this code through a just-in-time binary translator, much like a Java virtual machine translates Java bytecode on the fly into native instructions. The translator in the VMM does not perform a mapping from one architecture to another, but instead translates from the full unrestricted x86 instruction set to a subset that is safe to execute. In particular, the binary translator replaces privileged instructions with sequences of instructions that perform the privileged operations in the virtual machine rather than on the physical machine. This translation enforces encapsulation of the virtual machine while preserving the x86 semantics as seen from the perspective of the virtual machine.

To keep translation overheads low, the VMM translates virtual machine instructions the first time they are about to execute, placing the resulting translated code in a translation cache. If the same virtual machine code executes again in the future, the VMM can reuse the translated code, thereby amortizing the translation costs over all future executions. To further reduce translation cost and to minimize memory usage by the translation cache, the VMM combines binary translation of kernel code running in the virtual machine with direct execution of user mode code running in the virtual machine. This is safe because user mode code cannot execute privileged instructions.

A BT-based VMM must enforce a strict boundary between the part of the address space that is used by the virtual machine and the part that is used by the VMM. The VMware VMM enforces this boundary using segmentation.

Segmentation is a hardware feature of the x86 CPU that dates back to its 16-bit ancestors. A segment is a consecutive range of memory, identified by a base (the starting address) and a limit (the length of the segment). Whenever an x86 instruction accesses memory, it does so with respect to a particular segment. The segmentation hardware checks the memory address against the segment limit. If it is within the limit, the base address is added and the access is permitted to proceed. If the address exceeds the limit, the memory access is aborted and the processor raises a protection fault.

Because most modern operating systems, including Windows, Linux, and Solaris, make only light use of segmentation, it is possible for the VMM to use segmentation to enforce the boundary between virtual machine and VMM. In rare cases when the uses of segmentation by the virtual machine and the VMM conflict, the VMM can resort to software segmentation checks, albeit at a slight loss of performance.

In 2003, when AMD extended the x86 architecture from 32 to 64 bits, it eliminated segment limit checks for 64-bit code (32-bit code still retained segment limit checks for backwards compatibility). This change meant that a BT-based VMM could not use segmentation to protect the VMM from a 64-bit virtual machine. In other words, BT32 could virtualize the 32-bit x86 architecture efficiently, but BT64 could not virtualize the 64-bit architecture efficiently.

When this architectural deficiency became apparent, AMD added segment limits back into 64-bit code. This addition missed the initial Opteron Rev C processor (which shipped in limited quantities only) but was present beginning with the next revision, Rev D, and has remained present in AMD 64-bit CPUs ever since. Thus, all 64-bit AMD CPUs, with the exception of the original Opteron Rev C, can run virtual machines with BT64.

The Intel 64-bit extensions to the x86 architecture also omitted support for segment limit checks for 64-bit code. Unlike AMD, however, Intel has not added support for segment limit checks in subsequent processors. This limitation makes it inefficient to run 64-bit virtual machines using BT64 on Intel CPUs.

Table 2 summarizes the forms of binary translation that different versions of ESX can use on Intel and AMD CPUs.

Table 2. Supported Uses of Binary Translation

	ESX 1.0-2.5	ESX 3.0	ESX 3.5	ESX 4.0
AMD	BT32	BT32, BT64	BT32, BT64	BT32, BT64
Intel	BT32	BT32	BT32	BT32

Hardware Technique: VT-x and AMD-V

About the same time they were making the transition from 32-bit to 64-bit hardware, both Intel and AMD recognized the importance of virtualization. Both companies began designing hardware that made it easier for a VMM to run virtual machines. The first hardware designs focused on the subproblem of how to virtualize the 32- and 64-bit x86 instruction set. The Intel design, called VT-x, took on particular importance because it provided a way to virtualize 64-bit virtual machines efficiently. (BT64 is not efficient because of the lack of segment limit checks in 64-bit mode on Intel CPUs.) AMD subsequently introduced AMD-V to provide hardware support for instruction set virtualization but virtualization of 64-bit virtual machines using BT64 was possible already for AMD CPUs.

VT-x and AMD-V are similar in aim but different in detail. Both designs allow a VMM to do away with binary translation while still being able to fully control the execution of a virtual machine by restricting which kinds of (privileged) instructions the virtual machine can execute without intervention by the VMM.

In spirit, the two hardware mechanisms reach back to the classic days of virtualization on IBM 360 mainframes. VT-x and AMD-V both allow a VMM to give the CPU to a virtual machine for direct execution (an action called a *VM entry*) up until the point when the virtual machine tries to execute a privileged instruction. At that point, the virtual machine execution is suspended and the CPU is given back to the VMM (an action called a *VM exit*). The VMM then follows the classic approach from mainframe days, inspecting the virtual machine instruction that caused the exit as well as other information provided by the hardware in response to the exit. With the relevant information collected, the VMM emulates the virtual machine instruction against the virtual machine state and then resumes execution of the virtual machine with another VM entry.

Intel CPUs acquired VT-x support in later Pentium 4 CPUs (starting with Prescott). AMD introduced AMD-V in AMD Opteron Rev F CPUs.

Table 3. Supported Uses of Hardware Instruction Set Virtualization

	ESX 1.0-2.5	ESX 3.0	ESX 3.5	ESX 4.0
AMD			AMD-V32, AMD-V64	AMD-V32, AMD-V64
Intel		VT-x64	VT-x64	VT-x32, VT-x64

Table 3 shows which versions of ESX can use VT-x or AMD-V to run 32- and 64-bit virtual machines. The table assumes an AMD "Barcelona" CPU or newer, because the initial AMD-V implementation in Opteron Rev F is affected by an issue that VMware has chosen not to work around. In ESX 3.5, AMD-V can be used only in combination with rapid virtualization indexing (RVI) (see "Memory and MMU Virtualization" on page 5), whereas in ESX 4.0, AMD-V can be used with or without RVI.

As shown in Table 2 and Table 3, the number of supported ways to virtualize the x86 instruction set has increased over time, as new versions of ESX and new CPUs have become available.

Memory and MMU Virtualization

All modern x86 CPUs implement virtual memory, which is a technique for flexibly mapping multiple virtual address spaces (typically one per process) into a possibly smaller amount of physical memory. The details of how an operating system manages this mapping is beyond the scope of this guide. However, for the x86 architecture, the mapping is specified using a set of memory-resident hierarchical 4KB page tables. A tree of such page tables, identified by a root page table, specifies the entire mapping of a virtual address space into physical memory.

The x86 MMU contains two main structures: a page table walker and a content-addressable memory called a translation lookaside buffer (TLB) to accelerate address translation lookups. When an instruction accesses a virtual address (VA), segmentation hardware converts the virtual address to a linear address (LA) by adding the segment base. Then the page table walker receives the LA and traverses the page table tree to produce the corresponding physical address (PA). When the page table walk completes, the (LA,PA) pair is inserted into the TLB to accelerate future accesses to the same address.

Accordingly, the task of the VMM is not only to virtualize memory but to virtualize virtual memory so that the guest operating system can use virtual memory. To accomplish this task, the VMM must virtualize the x86 MMU. It does so by having the VMM remap addresses a second time, below the virtual machine, from PA to machine address (MA), to confine the virtual machine to the machine memory that the VMM and VMkernel have allowed it to use.

The following sections compare shadow page tables, a software technique for memory virtualization, with the hardware support found in recent CPUs from AMD (RVI) and Intel (extended page tables, or EPT).

Software Technique: Shadow Page Tables

To virtualize memory without special hardware support, the VMM creates a shadow page table for each primary page table that the virtual machine is using. The VMM populates the shadow page table with the composition of two mappings:

- The $LA \rightarrow PA$ mapping specified by the guest operating system, obtained from the primary page tables
- The $PA \rightarrow MA$ mapping defined by the VMM and VMkernel

By building shadow page tables that capture this composite $LA \rightarrow MA$ mapping, the VMM can point the hardware MMU directly at the shadows, allowing the virtual machine's memory accesses to run at native speed while being assured that the virtual machine cannot access machine memory that does not belong to it.

However, shadow page tables incur overheads in other situations.

- When the virtual machine updates a primary page table, the VMM must trap the update and propagate the change into the corresponding shadow page table or tables. This slows down memory mapping operations as well as creation of new processes in virtual machines.

- When the virtual machine touches memory for the first time, the shadow page table entry mapping this memory must be created on demand, slowing down the first access to memory. (The native equivalent is a TLB miss.)
- When the virtual machine switches context from one process to another, the VMM must intervene to switch the physical MMU to the new process' shadow page table root.
- Shadow page tables consume additional memory.

Hardware Technique: RVI and EPT

To address the overheads inherent in shadow page tables, both AMD and Intel now build special-purpose hardware to support MMU virtualization. AMD introduced support for MMU virtualization, called RVI, in the quad-core Opteron ("Barcelona") CPU. Intel introduced similar functionality, called EPT, in its "Nehalem" generation of CPUs.

Just as AMD-V and VT-x are similar in spirit but different in detail, so are RVI and EPT. Both designs permit the two levels of address mapping to be performed in hardware by pointing the physical MMU at two distinct sets of page tables. The first is defined by the virtual machine: $LA \to PA$. The second, invisible to the virtual machine, is controlled by the VMM: $PA \to MA$. Given these two mappings, the physical CPU's page walker can walk the two sets of page tables to produce (LA,MA) pairs that are cached in the TLB.

This arrangement does away with shadow page tables at the cost of a single set of nested or extended page tables that map from *PA* to *MA*. Because the nested or extended page tables are largely static and need no update whenever the virtual machine creates or modifies page tables, the VMM need not intervene when virtual machine page tables are updated. Moreover, the VMM does not need to be involved in virtual machine context switches. The virtual machine can change the $LA \to PA$ page table root on its own.

Table 4. Support for RVI and EPT in ESX

	ESX 1.0-2.5	ESX 3.0	ESX 3.5	ESX 4.0
AMD RVI			yes	yes
Intel EPT				yes

Although RVI and EPT have compelling advantages, there is one potential downside: a TLB miss is now more expensive because it must be serviced by a two-level page walker.

For most workloads, RVI or EPT provides an overall performance win over shadow page tables, but there are exceptions to this rule: workloads that suffer frequent TLB misses or that perform few context switches or page table updates. For more information, see the papers "Performance Evaluation of AMD RVI Hardware Assist" and "Performance Evaluation of Intel EPT Hardware Assist" (see "References" on page 9 for links).

Table 4 shows which versions of ESX can use RVI or EPT. When such support is not present, either in the version of ESX or in the physical CPU, the fallback is shadow page tables.

Monitor Modes

This guide describes a two-way choice between software and hardware techniques for instruction set virtualization (BT on one hand and AMD-V or VT-x on the other hand). Similarly, it describes a two-way choice between software and hardware techniques for memory virtualization (shadow page tables on one hand and RVI or EPT on the other hand).

Unfortunately, the two forms of hardware support are not orthogonal features. RVI is inseparable from AMD-V and EPT is inseparable from VT-x. This leaves only three valid combinations:

- BT-swMMU—binary translation and shadow page tables
- HV-swMMU—AMD-V or VT-x and shadow page tables
- HV-hwMMU—AMD-V with RVI or VT-x with EPT

In the list above, HV stands for hardware support for instruction virtualization. It is a convenient common term for either AMD-V or VT-x. We refer to the above three options as *monitor modes* because they describe the way the VMM runs a particular virtual machine on a given physical CPU.

Choice of Monitor Mode

When a virtual machine is powering on, the VMM inspects the physical CPU's features and the guest operating system type to determine the set of possible execution modes. On ESX 3.0 and earlier, the set of candidate execution modes has exactly one choice, as you can determine from preceding tables.

However, beginning with ESX 3.5, and especially with ESX 4.0, there are cases in which more than one execution mode is possible. In such a case, how does the VMM make a choice? It first finds the set of modes allowed. Then it restricts the allowed modes by configuration file settings. Finally, among the remaining candidates, it chooses the "preferred" mode. The following examples illustrate the process:

- ESX 3.5 on an AMD Shanghai CPU and a 64-bit virtual machine—The allowed modes are BT-swMMU and HV-hwMMU (AMD-V with RVI). The preferred option for a 64-bit virtual machine is HV-hwMMU, so the VMM chooses this mode at power-on time.
- ESX 3.5 on an Intel Nehalem CPU and a 64-bit virtual machine— Run with HV-swMMU (because ESX 3.5 does not support EPT).
- ESX 4.0 on an AMD Shanghai CPU and a 64-bit virtual machine—The choice is among BT-swMMU, HV-swMMU, and HV-hwMMU. HV-hwMMU wins.
- ESX 4.0 on an older Opteron Rev F CPU and a 64-bit virtual machine—Only one option is available: BT-swMMU (because ESX cannot use AMD-V on this CPU).
- ESX 4.0 on an Intel Nehalem CPU and a 64-bit virtual machine—The allowed modes are HVswMMU and HV-hwMMU (BT is not allowed for 64-bit virtual machines on Intel CPUs because segment limit checks are missing). The VMM chooses HV-hwMMU.

Certain features may restrict the available modes. For example, VMware Fault Tolerance cannot use RVI or EPT because of their lack of determinism, and it avoids BT, narrowing the choice to HV-swMMU.

When multiple choices remain, a prioritization algorithm runs to choose the best mode:

- For ESX 3.5, the only case in which there is a choice is on AMD CPUs on which BT-swMMU and HV-hwMMU might both be available. The default choice for 32-bit virtual machines is BT-swMMU. For 64-bit virtual machines, it is HW-hwMMU, although OS/2, UnixWare, and OpenServer default to HW-hwMMU. (The choices for 32- and 64-bit virtual machines differ because many 32-bit guest operating systems suffer high overheads on APIC virtualization when running with hardware virtualization because they make frequent access to a memory-mapped TPR register.)
- For ESX 4.0, many more situations can result in multiple allowable execution modes. The general priority order for CPUs that have hardware support for APIC virtualization (most newer Intel CPUs from "Penryn" onwards) is: HV-hwMMU, followed by HV-swMMU, followed by BT-swMMU. For CPUs without hardware support for APIC virtualization, the order for 32-bit Windows guest operating systems is: HV-hwMMU, followed by BT-swMMU, followed by HV-swMMU. For certain unusual guest operating systems, including OS/2, UnixWare, and OpenServer, BT is discouraged or disallowed.

Specifying the Preferred Monitor Mode

In some cases, an explicit specification of monitor mode preference may be desirable. Although this situation is rare, the complexity of workloads and virtual machine configurations makes a manual override desirable in cases in which the default choice leads to less than optimal performance.

In virtual machine configuration files, you can restrict the set of modes by setting one or both of the following options:

```
monitor.virtual_mmu   =  software | hardware | automatic
monitor.virtual_exec  =  software | hardware | automatic
```

Choose one of `software`, `hardware`, or `automatic` for each variable. Both ESX 3.5 and ESX 4.0 recognize the `monitor.virtual_mmu` setting. Only ESX 4.0 recognizes `monitor.virtual_exec`. You can express all possible ESX 3.5 mode choices with the `monitor.virtual_mmu` option alone.

If a setting is not specified, the effect is the same as `automatic`. If it is set to `hardware`, it forces the use of the given form of hardware support if the feature is available and supported. Likewise, if the setting is `software`, the VMM attempts to run the virtual machine without the given form of hardware support, if allowed.

Although the configuration file settings are flexible enough to express all of the 2×2 possible combinations, only three of the four combinations are valid. Valid combinations are used to select one of the three execution modes. If the CPU does not support the requested execution mode, the settings are ignored. In addition, the settings are ignored if the CPU implements the execution mode but the version of ESX does not support it.

The vSphere Client user interface presents a three-way choice. The choice you make in the vSphere Client is mapped to the above (`monitor.virtual_mmu` and `monitor.virtual_exec`) settings before the virtual machine powers on.

The settings are always treated as hints. If you use VMotion to move a virtual machine from one host where the settings had an effect to another host where they are at odds with the physical CPU or the version of ESX, the settings are ignored. If you use VMotion to move the virtual machine back to the first host, the settings take effect again.

Determining Active Monitor Mode

Given the complexity of the mode determination logic, the dependency on the version of ESX, and the diversity of CPUs, it is not feasible to provide an exhaustive list of all possible outcomes. Instead, this guide describes how to determine from a log file which monitor mode is used.

For ESX 3.5 on an Intel CPU, if the virtual machine is 32-bit, the mode is BT-swMMU. If the virtual machine is 64-bit, the mode is HV-swMMU (that is, using VT-x without EPT). For ESX 3.5 on an AMD CPU, look for the string `hv-svm` or `hv-none` in `vmware.log`. If the string `hv-svm` is present, the monitor mode is HV-hwMMU, otherwise it is BT-swMMU.

For ESX 4.0, look for lines that contain `MONITOR MODE` in `vmware.log`. These lines list the allowed modes, the filtering steps, and the final chosen mode. For example:

```
greenland 1034> grep MONITOR vmware.log
MONITOR MODE: allowed modes : BT HV HWMMU
MONITOR MODE: user requested modes : BT
MONITOR MODE: guestOS preferred modes: BT HWMMU HV
MONITOR MODE: filtered list : BT
```

On Intel CPUs you may see BT32 listed instead of BT. This reflects the fact that the ability to use software virtualization on these CPUs is limited to 32-bit virtual machines. When using BT32, the monitor runs the virtual machine with software virtualization for as long as possible, but it dynamically switches to hardware virtualization if the virtual machine enters 64-bit mode.

VMware expects the ESX 4.0 approach to mode determination to remain valid in future releases of ESX, although the exact format of the configuration file lines might change as ESX supports new hardware features in future CPUs or as VMware finds ways to improve the VMM itself.

Conclusions

Virtualization used to be simple: all CPUs were 32-bit, and the VMM always used binary translation and shadow page tables to run the virtual machine. The landscape today is irregular and changing. Although this might seem like an unwelcome difficulty, VMware prefers to view it as an opportunity: having hardware vendors compete to optimize their hardware for virtualization might create diversity, but it also delivers efficiency gains. At the same time, the complexity can be difficult for users of virtualization software, and it is not viable for everyone to spend the time to become experts.

This guide, although it is richer in detail than the typical fact sheet, makes an effort to strike a useful balance, allowing you to understand the state of virtualization with current hardware and software and also to understand where virtualization might be going.

References

- K. Adams and O. Agesen, "A comparison of software and hardware techniques for x86 virtualization" in ASPLOS-XII: Proceedings of the 12th international conference on Architectural support for programming languages and operating systems (San Jose, California, USA, 2006), ACM, pp. 2-13.
  http://www.vmware.com/pdf/asplos235_adams.pdf

- R. Bhargava, B. Serebrin, F. Spadini, and S. Manne, "Accelerating two-dimensional page walks for virtualized systems" in ASPLOS XIII: Proceedings of the 13th international conference on Architectural support for programming languages and operating systems (Seattle, Washington, USA, 2008), ACM, pp. 26-35.
  http://www.amd64.org/fileadmin/user_upload/pub/p26-bhargava.pdf

T. Cramer, R. Friedman, T. Miller, D. Seberger, R. Wilson, and M. Wolczko, "Compiling Java just in time." IEEE Micro 17, 3 (1997), 36-43.

- Ole Agesen, "Performance Aspects of x86 Virtualization"
  http://www.vmworld.com/docs/DOC-2476

- Nikhil Bhatia, "Performance Evaluation of AMD RVI Hardware Assist"
  http://www.vmware.com/resources/techresources/1079

- Nikhil Bhatia, "Performance Evaluation of Intel EPT Hardware Assist"
  http://www.vmware.com/resources/techresources/10006

About the Author

Ole Agesen has worked in the VMware monitor group since 1999.

Acknowledgments

The author thanks those who contributed to this guide. Henrik Mønster first pointed out the need for such a guide. Jim Mattson and Jeffrey Sheldon provided many comments that improved the technical accuracy. Sean Borman assisted to greatly improve the clarity and focus.

If you have comments about this documentation, submit your feedback to: docfeedback@vmware.com

VMware, Inc. 3401 Hillview Ave., Palo Alto, CA 94304 www.vmware.com

Revision: 20090622 Item: EN-000240-00