The Reality of Virtualization for Windows Servers

Mark Friedman
Demand Technology

Abstract.

This paper discusses the performance and capacity concerns that arise when Windows servers are run as virtual machine guests on current virtualization solutions. It reviews the advantages and disadvantages of virtualization as a server consolidation strategy. It describes the major sources of performance degradation that applications running on guest machines face today and discusses the prospects to resolve these problems as new hardware emerges in the near future.

1.0. Introduction.

This paper highlights several important capacity planning considerations that arise in the virtualization technologies that are currently available for the Windows server platform. It begins with a discussion of the virtualization technology available today. It discusses the appeal of virtualization to address the proliferation of under-utilized Windows servers, which is widely perceived as a significant system management problem. It takes the somewhat contrarian view that this problem is not the profligate waste of resources that the evangelists for virtualization suggest it is, nor is virtualization the most direct approach to its solution. After providing a realistic assessment of the virtues of current virtualization technology, the paper then tries to identify the situations where virtualization technology can be employed most effectively.

Capacity planning for virtualization is best described as an *n:1* folding problem. The capacity planner must assure that the guest workloads can fit into the one physical machine managed by the virtual machine Host. This is a multi-dimensional problem where the capacity planner must assure that the processor, disk, memory and network bandwidth of the combined guest machines – plus some allowance for virtual machine "overheads" – does not exceed the physical capabilities of the underlying hardware. To the detriment of capacity planning, there is limited measurement data the Windows Server environment that can be used reliably to estimate the amount of virtualization overhead to expect in advance for a given workload.

The discussion of the sources of various virtualization "overheads" inevitably leads to a consideration of the performance issues that currently arise when Windows runs as a guest machine. This article focuses on two significant problems that have not been discussed much by other commentators. One concern is the technique used to schedule virtual machines that has a serious performance impact on I/O bound workloads. A second problem area involves the measurement perturbations that occur on the Windows platform *after* virtualization technologies are introduced. Finally, the paper discusses the hardware improvements expected in the near future and their potential to resolve these problems in a satisfactory manner.

2.0. Reigning in the Server Farm.

Mainly out of concern for easing the burden of system administration, Windows servers tend to be configured to run an homogenous workload performing a single task. Some of the historical reasons for preferring isolated Windows servers running single applications are (1) increased stability, (2) better reliability, (3) simpler problem-solving, and (4) simpler capacity planning.

A powerful argument for running homogenous, single workload Windows machines was increased stability. Indeed, the present author has been one of the forceful public advocates for this strategy to deploy Windows servers successfully on a large scale. [1] The practices and procedures forged in the crucible of experience that led in the past to successful deployment of large numbers of Windows servers running mission critical applications have an understandable resilience in the face of change. At the present time, however, it is worth considering whether the historical conditions that favored configuring Windows machines to run a single, homogenous workload are still present today.

Current industry best practices recommend that the Technical Support group responsible for Windows servers and desktops adopts a stable *image* of the operating system and application software that it intends to maintain and support going forward after a lengthy period of concentrated acceptance testing. The image is then cloned every time there is an organizational requirement to support this application in a new location. The result is a proliferation of large numbers of Windows machines that is evident to almost every observer of the IT department, most of which by almost any common measure of capacity are severely underutilized.

This widespread practice of cloning operating system and application software images that are certified by sysadmins for stability and reliability does not alone lead to massive over-provisioning of Windows servers. The hardware to run Windows keeps getting more powerful in leaps and bounds. Current generation hardware from Intel and AMD offer 64-bit addressing, massive amounts of RAM, dual core processors, and hyper-threading. These machines offer more processing power than many single application servers will ever need. This, in turn, leads to opportunities to save administrative costs by reducing the total number of machines that need to be managed. Server consolidation makes evident good economic sense, and virtualization is one path to server consolidation.

In the case of a remote field office operation, for example, an accomplished system administrator might think it is necessary to supply a minimum of three separate OS images: one to provide the essential Messaging application like MS Exchange or Lotus Notes to tie employees at the remote office to the corporate e-mail network; one to provide Active Directory-based security and authentication services, and a third to supply data protection (i.e., back-up) and data recovery. Conventional practice would be to supply three separate dedicated machines to perform these functions, all of which would likely be severely under-utilized. Virtualization is an appealing option in this instance because it allows all three machine images to be run inside a single box.

We are not ready to concede that this over-provisioning is nearly as big a problem as IT professionals reared on the frugality required to manage expensive mainframe technology cost-effectively presume it to be. In the face of the changing economics of Information Technology, it is certainly worthwhile to examine the assumptions behind this presumption that delivery of service requires separate machines. But it is also fair to say that an application of rudimentary capacity planning practices and procedures could sharply reduce the degree of over-provisioning that occurs.

2.1. Why virtualization. This leads us to the heart of the matter. Given this widespread over-provisioning, how has virtualization come to be considered the leading solution to the problem. This section explores some of the more obvious benefits that accrue to server consolidation using virtualization. It is a short step from running single application servers to recognizing that the servers these applications are running on are often massively over-provisioned. Virtualization promises to enable current hardware to be used more efficiently. [1]

Many IT organizations view virtualization technology as a viable solution to the evident problem that the organization has too many machines to manage. In theory, at least, virtualization technology is positioned to address the inefficiencies of running machines that are severely over-provisioned. For the most part, any potential performance concerns with virtualization are deemphasized. It is assumed that the hardware used to consolidate these workloads is so much more powerful than what its OS guests demand as to minimize most practical performance concerns. But, as will be discussed below in more detail, this assumption is naive.

[1] Dysfunctional software licensing policies sometimes play a role in discouraging consolidating multiple workloads on larger machines running a single OS image or multiple databases, for example, under a single SQL Server license.

Virtualization does provide a mechanism that allows system administrators to utilize current hardware more effectively while retaining all the administrative advantages of isolating workloads on dedicated servers. Using virtualization, it is, for example, possible to configure and run two or more virtual machines – each devoted to running a single, isolated workload – on a single hardware platform. In the multiple machine scenario described in section 2.0, instead of provisioning three separate machines to run the mail server application, the domain controller, and the back-up server, all three server machines can be consolidated on a single hardware platform running virtualization software.

So virtualization allows the system administration to supply all three essential services discussed above on a single piece of hardware, which is certainly a simplification along that important dimension. Moreover, the virtualization solution has the additional benefit that it appears not to require major changes in current system administration machine configuration practice. (The recognition that virtualization itself might add significant complexity to the operation is something that usually only surfaces later with experience. See, for example [3].)

Curiously, the quaint possibility that multiple workloads can be readily consolidated today and delivered by a single operating system image does not seem to occur to most Windows Technical Support professionals. Yet while many of the system management deficiencies of early versions of the Windows NT platform have been addressed, the “best practices” associated with deploying single application servers has barely taken notice. (We will return to this topic in section 8 where some of the recent wide-ranging Microsoft system management initiatives are discussed.) In at least some instances, this is due to an incomplete understanding of the basic system management disciplines, including performance and capacity planning, and a woeful lack of fundamental technical skills, such as the ability to craft simple scripts to automate common administrative functions.[2]

3.0. Virtualization technology today

Software developers were among the earliest adopters of the virtualization technology that is available today for Windows. They faced a common problem, namely, the need to subject new releases of software to rigorous testing

[2] In fairness, it may well be that many quite capable system administrators are too busy responding to immediate and pressing problems in their massive server farms to ever have the time to acquire and hone these basic skills. Moreover, the mobile pension and benefits packages that most companies provide to their employees today are also a major disincentive for IT departments to nurture and train in-house Tech Support professionals for long-term service to the corporation. In today’s workplace, IT knowledge workers are obliged to acquire the repertoire of technical skills that ensure they can remain profitably employed over the course of a long career themselves.

on a wide variety of platforms. Virtualization software allows a single machine to be configured to run multiple operating system images that can then be used to ensure that the software being developed functions correctly in diverse configurations. Software development and Quality Assurance testing remains the one area where virtualization can be deployed with the greatest unqualified chance of success.[3]

Virtualization is accomplished as illustrated in Figure 1, by installing a virtual machine host on the bare metal that is then capable of running numerous virtual machines guest operating system images beneath it; in practice, as many guest machines as will fit.

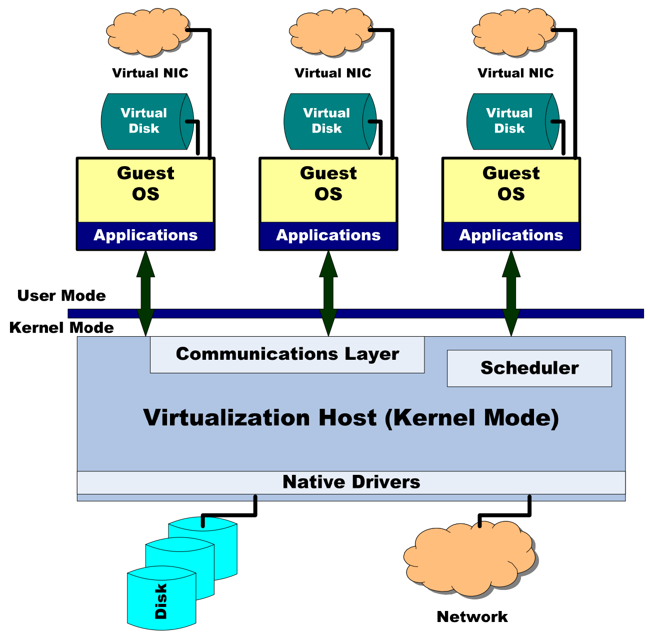

Figure 1. The architecture of VMware ESX version 2.

Figure 1 illustrates several key architectural features of the most popular virtualization solution for server consolidation, which is the VMware ESX Server product. (The discussion that follows is based on ESX version 2. Details about ESX version 3 were just starting to emerge as this paper was being written. Based on the limited information currently available, ESX version 3.0 does not make any major architectural changes to the software.) Note that the VMware ESX software functions as the primary operating system (OS) supervisor that interacts directly with the physical hardware – the processor, RAM, the disks, the network, the video display, etc. Due to its status as a base platform that runs the virtual machines, the VM Host software layer that you install the virtual machine OS on top of is sometimes called the hypervisor [5]. For an academic audience, the VM Host software is known as the Virtual Machine Monitor [6]. Both are terms for an operating system supervisor that is very limited in scope.

Unlike a more general purpose operating system, the functions that the VM Host software performs are narrowly delineated to those that are required to define the virtual machine guests and sustain them once they are activated. Since VM Host software for Windows originated with Independent Software Vendors (ISVs) who had no preferred access to Windows internals, an additional goal of the 3rd party developers who created the VM Host software was to run Windows guest machines transparently.

Note that in the ESX architecture the VM Host software is responsible for all native devices attached to the machine. The requirement that VMware be able to provide native device drivers is a major encumbrance, the burden of which VMware attempts to minimize by using a Linux-compatibility module. This Linux interoperability makes it relatively easy to adapt existing Linux device driver modules so that they can be re-compiled into the VMware Host kernel. (Due to the GNU Open Source licensing restrictions, VMware is careful to say that ESX was not derived directly from Linux, despite outward evidence of its family resemblance.) In practice, ESX supports a wide variety of disk, network, and SAN-attached devices (see http://www.vmware.com/pdf/esx_io_guide.pdf for reference), similar to the range of devices that can be attached to most Linux servers.

Relying on the VM Host software to provide native device drivers to support all attached physical devices is not the only way to achieve virtualization's goals. The VMware GSX and Workstation products, as well as the Microsoft Virtual Server 2005 product based on the software Microsoft originally acquired from Connectix in 2003, provide virtualization software that is installed on top of a standard Windows OS installation. This approach allows you to install and run native Windows device drivers, which are more widely available for some peripherals than Linux drivers. Native Windows device drivers typically exploit Windows Plug-and-Play technology during installation and set-up. They frequently also have more elaborate feature sets and user interfaces than their Linux counterparts. The ESX dedicated Virtual Machine Monitor approach permits a greater degree of vm guest isolation, such that a problem with a device driver on one virtual machine guest is less likely to impact other vm guests that offer shared access to the same device. The ESX Host software also supplies a dedicated Scheduler service, as illustrated, to dispatch VM guests, rather than rely on the standard Windows priority-based thread Scheduler which was never designed with virtualization in mind.

[3] Interestingly, the two major vendors of virtualization software for Windows development and application testing – Microsoft and and EMC's VMware subsidiary – both currently provide full-featured downloadable versions of their products for free. Free versions of both Virtual Server 2005 R2 and VMware GSX are currently available. Since the GSX product line no longer generates any licensing revenue to VMware, the company is forced to rely almost entirely on licensing revenues that the ESX product line generates. ESX, of course, is positioned squarely at the market for enterprise-class server consolidation solutions, not software QA.

A third design alternative is the approach used in the Xen project where the VM Host software virtualizes its devices, which allows it to rely on native devices drivers installed on the guest OSes to communicate directly to attached peripherals. While Xen's approach allowing for native guest OS device drivers has many commendable virtues (see [7] for details), Xen is not transparent to the guest OS. In fact, in Xen the guest OS must be modified to run under Xen. Relaxing the transparency requirement means in practice that Xen cannot currently host Windows guest machines today, so any further discussion of its creative approach to virtualization is beyond the scope of the present essay.

When you set up a Windows server guest machine to run under either VMware ESX or MS Virtual Server 2005, the guest OS sees only the virtual disk, network, and video devices exposed by the VM Host software. The VM Host software exposes a limited set of generic devices that the guest OS detects, configures, and uses. In the case of ESX, it also limits the number of logical processors that the guest OS is able to detect. (This was recently raised to a maximum of four CPUs per guest machine in ESX version 3.)

Both the VMware Host and Virtual Server 2005 also inject one or more service processes into the guest Windows machine. These are used to facilitate communication between the Host and the Guest. In the case of ESX, the Windows guest OS runs VMWareService, plus the VMWareTray and VMWareUser processes. Communication between the VM Host and the Windows Guest OS occurs across a virtual NIC that simulates a generic AMD PCNET Family Ethernet adapter. (Host-guest machine communication in Virtual Server 2005 is similar.) From a performance monitoring perspective, when ever you can detect that the VMWareService process is active, then you can accurately deduce that the machine in question is a virtual machine guest.

4.0. Sizing virtualization environments today

So long as the virtualization technology being deployed was confined primarily to assisting with software development and testing, the technology raised few pressing capacity planning or performance concerns. (Running the sort of application stress-testing workload where performance actually mattered could always be diverted to a dedicated machine. See [4]). It was only when this same virtualization technology was re-positioned as a way to achieve server consolidation that serious capacity planning and performance considerations began to surface.

In principle, capacity planning to size the machine hosting two or more virtual machines should be quite simple. It corresponds to the problem of folding *n* virtual machines into one container, the machine that hosts the VM Monitor. It does require an optimal solution over time that factors in whether individual workload peaks overlap or not. (If you can consolidate non-overlapping workloads, you can achieve significantly more efficient operations.) Some care must be taken to ensure that the solution is optimal across multiple dimensions where each dimension corresponds to utilization of some physical resource that the host machine must apportion among the guest machines – the processor, RAM, the disks, and the network interfaces. To the degree that generously-sized current hardware capabilities often lead to machines that appear to be massively over-provisioned, sizing the host machine ought, in principle, to be relatively easy.

4.1. Sizing the processor. The processor, for example should be large enough to handle the sum of the processor demand from each configured virtual machine, plus some additional headroom to accommodate some amount of the inevitable virtual machine management "overhead" (to be dissected in somewhat more detail below):

$$\textit{Physical CPU Capacity} > \textit{VMM management overhead} + \sum(\textit{VM-Guest}_n\ \textit{CPU} + \textit{Overhead}_n)$$

In the case of sizing the processor, at least three major sources of VM overhead can be identified. In [8], Gunther identifies the VMM Scheduler that is responsible for either round-robin or weighted dispatching of virtual machine guests as one source of overhead. Gunther relies on a VMware-published ESX benchmark [9] that shows that this management overhead is minimal and well-behaved. It appears to scale linearly with the number of VM guests that are defined. The ESX benchmark data from [9] is summarized in Figure 2 for a four-way machine.

The benchmark workload in [9] is severely CPU-constrained. It is also designed to minimize the other two major sources of VMware management overhead. Notice that overall throughput tails off slightly as more virtual machines are configured to run than there are physical processors available to run them. Therefore, the difference between the dotted horizontal line at the top of the chart identified as "theoretical" and the actual Completion rate represents the processor scheduling overhead. The scheduling overhead per guest machine can be calculated as:

$$\textit{(Theoretical – Actual throughput) * 100 / Theoretical throughput / \# of VMs}$$

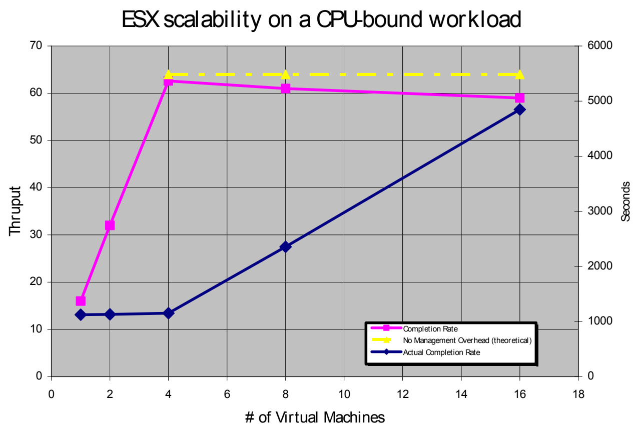

Figure 2. ESX scalability on a CPU-bound workload. Taken from benchmark results published by the VMware Corporation. [8]

In [10], Menascé identifies a second source of virtualization processor "overhead," which arises because the guest OS executes in User mode (Ring 3 on an Intel processors). Under VMware, every time an OS function inside the guest machine attempts to issue a privileged instruction, a hardware exception is raised. The VM Host interrupt handler has to trap this exception and recover from it. It does this by *emulating* the privileged instruction issued by the guest OS that failed. In practice, the emulation routine can be quite involved, depending on the function that VMware must mimic.

To illustrate this process, let's look, for example, at what happens when the guest OS needs to perform an I/O operation to disk. In the course of generating an I/O request, the Windows kernel-mode I/O Manager and the physical disk device's associated driver code normally operate exclusively in Privileged mode, or Ring 0. In the virtualization environment, all of this code is executed in User mode, or Ring 3. Whenever any kernel mode instruction that is only valid in Privileged mode is issued in User mode, the instruction fails. At this point, the VM Host software intervenes. After trapping the invalid instruction interrupt that occurs, the VM Monitor runs an emulation routine that mimics the original intent of the guest OS, and then returns control to the guest machine. Menascé characterizes this overhead in modeling terms as an execution delay, which it certainly is. In the virtualization environment, each attempt to execute a privileged instruction by the guest OS is replaced by an interrupt, the execution of the VM Host interrupt handler, and, finally, the execution of the emulation routine. The instruction path length associated with the function increases enormously. Unfortunately, the full extent of the associated delay is impossible to characterize accurately without measurements taken by the VM Host on the number of privileged instructions emulated, which are currently not forthcoming.

One idea is that the delay associated with emulating instructions that require privileged mode is proportional to the amount of time the guest OS spends in kernel-mode (% Privileged Time). This is a helpful suggestion, but hardly a precise way to proceed. Very few of the instructions executing in kernel-mode are privileged instructions, but, this depends on the OS function being executed.[4] Device-driver functions that issue instructions that reference physical addresses (not virtual ones) require Privileged mode to succeed. Major OS functions related to I/O processing in general, including the Cache Manager, the Workstation and Server services, processing within the TCP/IP stack, and the new kernel-mode http.sys driver in IIS 6.0, all make extensive use of physical addressing mode. The performance of all of these functions suffers in the virtualization environment, in some cases prohibitively so.[5]

A third source of virtual machine management overhead is a doubling of the number of instructions to initiate I/O operations and to service I/O completion interrupts. When a hardware-related device interrupt occurs, the native device driver code running in the VM Host software layer is driven initially. Once the native device driver services the interrupt, the VM Host software must determine which guest OS initiated the request and how to map the physical request into the appropriate virtualized context. Once this is accomplished, the VM Host software queues a virtual interrupt for the guest OS, which must then await dispatching by the VM Host guest machine Scheduler before the guest machine can detect that a device interrupt has occurred and process it. (This leads to delays in I/O interrupt servicing that are discussed in section 4.3.) When the device interrupt is received by the guest OS, its version of the interrupt handler is then dispatched to deal with it. Clearly, two similar sets of code are traversed, where only one set would be executed in a native run-time environment.

For Windows guest machines running under VMware, the sum of the amount of % Interrupt Time and % DPC Time recorded at the Processor level *when the system was*

[4] The implications of this last statement on secure, protected mode operations are wide-ranging [16]. Windows relies upon the privilege level to isolate processes from each other and from the protected mode operating system supervisor. For example, memory protection is used to ensure that a User process cannot access a memory location in the system range. On Intel hardware, even though there are four instruction execution privilege levels – or Rings – there is only a single bit privilege-level associated with a page table entry. (On recent Intel processors, the four privilege Rings have been collapsed into the two that Windows actually uses.) The normal protection memory mechanism used in Windows relies on the hardware to trap an instruction issued in User mode that references a memory location in the system address range. But in the virtualization environment, all kernel mode functions run in User mode. Unfortunately, VMware documentation is silent on the precise mechanisms used to maintain the security of the protected-mode Windows kernel by preventing User-mode instructions from accessing system addresses. Performance implications would likely preclude having every guest OS kernel-mode instruction fail whenever it attempted to access an address in the system range, so it is probably safe to assume that all memory locations associated with the guest OS are marked in the actual PTEs maintained by the VMware Host software as non-supervisor state allocations.

5 Menascé [8] suggests that the cycles lost to privileged instruction emulation are at least partially offset by running the virtual machine on a correspondingly faster processor. While this is a worthy suggestion, it is not the end of the story. If you are consolidating workloads running on previous generation hardware onto new machines where a virtualization engine is installed, the newer machines are likely to have a clock rate that is twice as fast. However, the virtualization cost of emulating guest OS privileged instructions is on the order of hundreds, if not thousands, of additional instructions to be executed, compared to one.

running in native mode provides a good estimate of the amount of additional CPU time that the VM Host software will require to process device interrupts on behalf of the guest machine.

4.2. Sizing RAM. In the case of sizing RAM, the memory requirements of a Windows guest can usually be reliably estimated by subtracting the Memory/Available Bytes counter from the size of RAM when the machine runs natively. VMware advises allowing for an additional 32 MB of RAM per virtual machine, plus the VM Host software itself, which requires about 400 MB of RAM. VMware must also provide a shadow copy of every page table entry (PTE) that is present on each guest OS (note that page tables are built per process). The VM Host software must intervene on every context switch to override the attempt by the guest OS to establish a new virtual addressing context.

VMware exploits the shadow PTE mechanism to wring some counter-balancing efficiencies from the virtual memory management process. Guest machines that have memory resident pages that are identical are able to share a single, memory-resident copy of the page. This effort is more noteworthy for the effort involved in identifying pages that are eligible to be shared, than the result, which is limited when very heterogeneous servers are consolidated. In the case of homogenous workloads, Waldpurger in [17] reports an impressive level of savings, nearly 40% in the case where 10 guest machines all running Windows NT 4.0 were defined. Nevertheless, consolidating the workloads under a single OS image where possible remains the superior approach. In Virtual Server 2005, a full complement of RAM on the host machine must be provided to the guest OS or else the guest machine simply will not boot.

The VMM maintains a single set of page tables that the virtual address translation mechanism in the hardware recognizes, which essentially duplicates the virtual address mapping information that each guest OS must itself maintain in virtual memory. It should go without saying that you would not want to configure a memory-constrained guest OS that had high paging rates. VMware uses a technique called ballooning [17] where it injects a device driver into a guest OS that ties up large amounts of physical memory in order to force the guest OS to trim unused virtual memory. Ballooning thus allows VMware to defer making page replacement decisions to the guest OS, which is in a far better position to make them intelligently. The ability of Windows Server 2003 to communicate directly with server applications that perform their own memory management (predominantly in support of I/O buffering) [13] suggests ballooning could be quite effective in this environment. The VMware Host software can determine which memory locations the OS has freed up by examining the guest OS page tables. Then VMware can re-distribute these available pages to other guest machines.

4.3. I/O interrupt delays. While not necessarily a capacity issue *per see*, the foregoing catalog of the performance issues impacting the scalability of virtualization technology would not be complete without some mention of significant interrupt processing delays that can easily arise. Significant interrupt delays are likely whenever there are more virtual machines defined than there are physical processors to run them. The problem can grow acute when some of the virtual machine workloads are I/O bound.

In a virtualization environment, servicing a high priority device interrupt becomes a two stage process. The VM Host driver software services the native device interrupt on demand as a necessarily high priority operation that preempts any lower priority task that is currently dispatched. One effect of this is that a guest machine that is currently dispatched can be interrupted by an I/O completion that was initiated by a different virtual machine. After the VM Host software services the interrupt, it queues a virtual interrupt to be processed by the initiating virtual machine. The virtual machine waiting on the interrupt may also be waiting to be dispatched on the VM Host queue. The guest machine cannot service the interrupt until the next VMware Host Scheduler interval in which it is scheduled to run. Any dispatchability delays effectively increase the time it takes the virtual machine to process device interrupts. This interrupt processing delay can be considerable.

Past experience with virtualization technology in the mainframe world (see, for example [11]) shows that I/O bound virtual machine workloads are prone to a secondary effect if they suffer extended periods when they are ineligible to be dispatched to service the interrupts they are waiting for. I/O interrupts that are queued to be serviced can only be processed when the virtual machine is finally dispatched. The effect is to stagger interrupt processing at the guest machine in a manner that leads to a skewed arrival rate distribution, a worst case that maximizes the queuing delays that are experienced. Ultimately, this secondary effect proved so powerful that the dispatcher mechanisms in mainframe partitioning schemes had to be modified to counteract it. The same behavior is currently evident in the virtualization solutions available for the Windows platform.

The performance of virtual machines during I/O intensive operations like back-up illustrates the problem. Suppose the virtual machine running the back-up task is one of two virtual machines vying for a processor. When the vm is able to execute, it usually has a backlog of I/O interrupts to service. After the interrupts are serviced and the next I/Os in the sequence are initiated, there may be little or no other processor-oriented work that needs to be done. So the virtual machine idles its way through the remainder of the time slice that it is eligible to run. When its time slice expires, the virtual machine waits. Meanwhile, some of the I/Os that were initiated during its last cycle of activity complete. But they cannot be serviced because the vm is not eligible to be dispatched. At the next interval where the vm is dispatched, the cycle repeats itself. Compared to running native, the I/O throughput of the virtual machine is slashed by 50% or more.

Currently, the only way to minimize the impact of this scheduling delay is to configure an I/O bound workload so it has access to at least one dedicated physical processor.

5.0. Performance expectations.

The previous section discusses some of the important architectural features of the virtualization software available for the Windows platform that have a major performance impact. Focused primarily on the popular VMware ESX package, it identified virtualization overheads that need to be factored into an initial server sizing effort. In this section we discuss the performance impact of the architectural features that were described in section 4. It is important to have some reasonable expectations about the performance of your applications once you start running them under virtualization.

In the area of processor utilization, three main issues were raised in section 4 that impact capacity. This section focuses on their performance impact.

- Virtual machine guests are subject to either round-robin or time-weighted dispatching by the VM Host software on the physical processors assigned for their use. (In VMware ESX version 3, a guest machine can be assigned to use from 1-4 physical processors.) The VMware Scheduler overhead used to switch the processor between virtual machines is minimal, but scales linearly with the number of virtual machines that are defined.

For any given processor workload, instruction execution throughput is degraded in proportion to its scheduling weight and the contention for the physical processors from virtual machines. VMware claims that ESX can detect when a virtual machine is idle, and will dispatch a different eligible virtual machine when it does so. This is critical to the performance of Windows guests because Windows does not issue a processor HLT instruction when it has no work to do [13].[6] Using a periodic timer check, the interval of which is currently set to 1.5 milliseconds by default, the ESX Host Scheduler would have an opportunity to preempt an idle virtual machine, assuming it can detect the system state reliably.

Using a series of benchmarks, Shelden tried to characterize ESX processor dispatcher queuing delay in [14], which he discovered could be substantial. He found, for example, that when a virtual machine was eligible for multiple processors, VMware dispatched the guest OS on both processors (ESX version 2) simultaneously. This technique, which VMware calls *co-scheduling*, provides a consistent multiprocessor environment for the guest OS, but it also means potentially significant dispatching delays when there is contention among virtual machines for the processor.

Shelden's results suggest that co-scheduling virtual machines causes significant performance degradation when there is CPU contention among the OS guests. Performance analysts need to be on the lookout for runaway application threads that are looping, something that sites can blithely ignore in many dedicated server application environments. A guest machine running a thread stuck in a loop has the potential to create significant contention for any processors it shares with other guest machines.

- The guest OS is run in User mode. All attempts by the guest OS code to issue privileged instructions are trapped by the VM Host software and emulated. The additional interrupts that are generated (due to the failed instructions) and the emulation code that substitutes for the original instruction add significant path length to routine attempts by the guest OS to execute privileged instructions.

Server applications that rely on physical addresses are likely to be the most vulnerable to this delay. The general mechanism used in Windows to make physical addresses available to device driver and other system modules is for them to allocate memory in the system's Nonpaged Pool. This should make it relatively easy to identify applications that are subject to this delay. Any Windows server application that makes extensive use of physical addresses are potentially vulnerable.

When issued by a virtual machine guest running in User mode, the privileged instruction to disable paging in order to utilize physical addresses or to re-enable paging when it is safe to do so fails. The failed instruction must then be emulated by the VM Host software. Extensive VMware Host intervention is required to perform these functions, which are associated with high performance DMA device controllers. Windows has a number of subsystems that remain in kernel-mode so that they can work directly with DMA controllers and utilize physical addresses. These include the MDL Cache interface (MDL stands for Memory Descriptor List) used by the file Server service and IIS. High performance Fibre Channel and SCSI device drivers also make extensive use of physical addressing and MDLs.

For example, the http.sys driver module introduced in IIS 6.0 can process Get Requests for static html objects entirely in kernel-mode. In IIS 6.0, TCP/IP, also running in kernel mode, queues an HTTP Get Request object to an http.sys worker thread. The worker thread has access to a physical-memory resident look-aside buffer (known as the Web Service Cache) where fully rendered HTTP Response objects are cached. If a cache hit occurs, the HTTP Response object can then be transferred from the cache to a physical memory-resident MDL output buffer for transmission by the NIC back to the requestors IP address

[6] It is not clear how this Idle Loop detection algorithm works or how effective it is in ensuring that idle Windows guests do not waste processor cycles. As reported in [13], the Windows Idle loop is a succession of no-op instructions issued from the HAL. On machines that support APCI power management, the HAL transfers control from the Idle loop to the processr.sys driver module to take appropriate action, which may include slowing down or powering off the processor.

without leaving kernel mode. For the sake of efficiency, this process relies on instructions that are accessing physical memory addresses directly.[7]

- The guest OS code to initiate and service I/Os is executed twice, once by the guest OS and a second time by the native device drivers running on the VM Host. This slows down the execution of both disk and network operations.

This strongly suggests that I/O bound workloads with critical performance requirements – and that includes network I/O of all types – should be run in native mode, not as virtual machine guests. Workloads that are not primarily I/O bound may still have periods where I/O performance is critical. It may be necessary, for example, to migrate guest machine back-up operations to a VM Host native back-up operation if back-ups cannot complete in the window that is available. The performance trade-off here is that VM Host native back-up operations view the guest machine's disk storage as a container file that must be backed up or restored in its entirety. Application-level file back-ups, like those performed on SQL Server or Exchange databases, must be run on the platform that runs the application. Of course, this type of application server may fall into the category of I/O bound workloads that need to run natively anyway.

The likely bigger concern for I/O bound workloads is where there is processor contention among virtual machine guests and I/Os to the guest OS remain pending during relatively long periods when the virtual machine is not eligible to be dispatched, as discussed in section 4.3.

- The balloon technique which forces page replacement back to the OS in various guest machines, along with the ability to share common pages among homogenous guests, makes memory management relatively efficient in the virtualization environment. Whatever additional memory that is required in the VM Host software and on VM guests to make virtualization work can be offset by this bag of clever memory management tricks.
- The simple virtual machine processor dispatching method used in VMware, along with the co-scheduling of processors to virtual machines defined to run on multiprocessors, leads to long delays in I/O interrupts processed by the guest OS. This, in turn, can lead to an artificial staggering in the rate of I/O initiation and completion. This secondary effect of the virualization engine's Scheduler can have a large negative impact on even modestly I/O bound workloads.

The cumulative effect of these negative performance impacts can be substantial. Shelden's benchmark testing [14], which was designed to isolate the performance impact of the VMware ESX processor scheduler showed significant degradation due to co-scheduling. In a test case where two virtual machines were active, each with two active threads, each capable of consuming approximately 55% of a processor in a conventional environment, overall utilization of each physical processor in the 2-way VM Host machine managed to reach only about 65% busy, instead of an expected 100% busy. One explanation is that the VM Host software was unable to detect correctly when a Windows processor is idling. Another possibility that Shelden explores is that processor co-scheduling requires that physical processors be assigned to virtual machines in tandem. When Shelden changed his simulation to use a Dispatch in Pairs scheduling algorithm, instead of the expected Dispatch When Ready, his simulation results were a significantly better fit to the actual data.

In [8], the principal researchers behind the development of Xen compare and contrast the performance of native Linux, Xen, and VMware Workstation on a series of benchmarks. Figure 3 below summarizes three of the comparisons: a CPU stress test running specint; a database-oriented benchmark; and the SpecWeb99 test suite that is a very comprehensive test of web server capabilities (it stresses the CPU, the network and the disks). On the CPU-bound workload, VMware results keep pace. But on the database and web server workloads, VMware lags native performance considerably. On the web server tests, the Xen developers observed, "VMware suffers badly due to the increased proportion of time spent emulating ring 0 code while executing the guest OS kernel."[8]

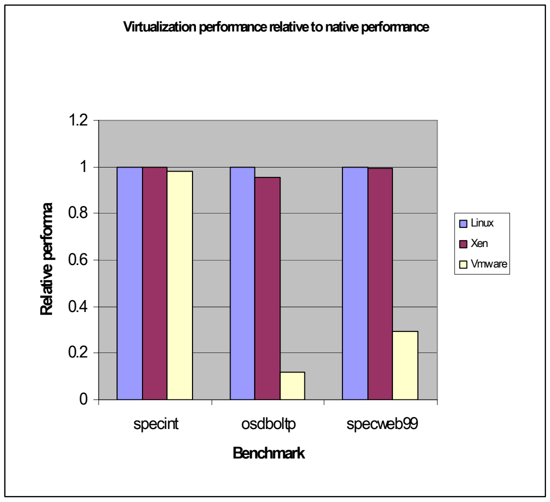

Figure 3. Benchmark results comparing native Linux, Xen, and VMware on three different workloads. From a

[7] The popular Apache web server application that runs primarily on Linux has comparable facilities.

[8] The Xen developers were prohibited from reporting comparable results with the ESX Server software due to the vendor's licensing restrictions. Perhaps we can assume that ESX performed better, but not so much better that VMware was willing to grant permission to publish the ESX test results. To date, VMware has only made public benchmark results for ESX running a CPU-bound workload similar to the one showed in Figure 3.

paper written by the principal developers of Xen at Cambridge University. [7]

The benchmark results presented by VMware in [8] on a CPU-bound workload are consistent with the specint workload shown in Figure 3. It tells only part of the story. The memory-resident, compute-bound workload discussed in [8] ignores significant sources of delay that typically accompany virtualization for Windows guest machines. Performance of workloads where there is a significant amount of thread and process context switching, interrupt handling, and I/O in general suffer from significant performance degradation. Section 9 will look briefly at the coming generation of processor hardware that has built-in virtualization capabilities. These are designed to significantly reduce these sources of performance degradation that are evident today. Until that hardware is available and a new generation of virtualization software is developed to exploit it, you need to have realistic expectations for the performance of virtualization solutions for Windows.

7.0. Monitoring the virtualization environment.

Performance monitoring for virtual machines running under VMware ESX is complicated today because many of the familiar guest OS measurements become difficult to interpret and cannot be relied upon. It is necessary to augment guest OS monitoring procedures with measurements from the VM Host software, especially regarding the use of the processor by various guest machines. Fortunately, VMware ESX does provide some necessary processor utilization statistics, but it lacks measurements that can give you a clear picture of what is going wrong when performance issues surface.

One side-effect of the two-step process in which device interrupts are handled by the VM Host software is that the timing mechanisms on a Windows guest machine are not reliable. This affects all timer-based measurements that any performance monitoring software running inside the Windows guest makes, including the % Processor Time utilization measurements at the Thread, Process, and Processor level and the Avg. Disk Secs/Transfer response time measurements for both Logical and Physical Disk.

The Processor utilization measurements in Windows are derived from samples taken once every clock interval. (See [13] for details.) The precision of the system clock is maintained as if ticks occur every 100 nanoseconds. In reality, the system clock time is updated much more slowly, normally about once every 5-10 milliseconds, depending on the machine. In official Microsoft documentation, this duration between clock ticks is sometimes referred to as the *periodic interval*. The mechanism used to maintain the system time is a timer interrupt that is set to fire regularly every periodic interval.[9] When the timer interrupt occurs, the Windows OS advances the system clock. It also samples the state of the machine at the time of the interrupt and determines what thread from what process was running when the clock interrupt occurred. All of the processor utilization measurements at the thread, process, and processor level are based on this sampling technique.

The high priority clock interrupt that Windows relies upon to keep time internally and for all its measurements of processor utilization is subject to interrupt pending delays in a virtualization environment. The periodic interval that Windows relies upon to keep accurate time is subject to erratic behavior, as documented in [15]. Consequently, the processor utilization samples that Windows gathers are no longer uniformly distributed in time. This should not invalidate the measurements completely, especially when you are looking at long enough intervals with sufficient samples to minimize normal sampling error. However, Windows performance monitoring software does not have direct access to the number of processor utilization samples. The utilization metrics the software calculates are based on the assumption that the intervals between samples are uniform. Obviously, this is not correct in the virtualization environment. The resulting calculation of % Processor Time is subject to major error because VMware does not deliver virtual clock interrupts at uniform time intervals.

A number of knowledgeable observers have puzzled over the interpretation of the processor utilization measurements that rely on this timing mechanism when they are taken in a virtualization environment. They have had little success trying to make sense of the normally reliable processor utilization measurements provided by the Windows guest machine and and even less correlating them with measurements taken by the ESX host software. See for example, [3] and [14]. The problem is that it is difficult to know what to make of any of the Windows % Processor Time measurements (all counters of type PERF_100NSEC_TIMER are impacted) as calculated by the Windows guest OS. The way interrupts in general, the clock interrupt included, are stacked up waiting for service when the virtual machine is finally dispatched undermines the uniform sampling methodology that Windows relies on in its % Processor Time calculations.

[9] The timer is a peripheral device on the Intel platform – it normally resides on one of the supporting chip sets external to the processor. Because the clock timer is a peripheral, it requires an I/O operation to access the current clock value. Intel processors beginning with the Pentium do contain a Timestamp Counter (TSC) that reflects the internal frequency of the microprocessor and the TSC can be used very efficiently to time the duration of any event. VMware ESX, in fact, uses the TSC for its measurements of processor utilization. But the TSC can only be used to tell time, it cannot be used to generate an interrupt at some future point in time. On Intel microprocessors with power-saving features, the TSC does not maintain a constant uniform clock rate, which is a function performed instead by the ACPI Timer.

Direct measurements taken by VMware ESX are necessary to augment the guest OS measurements. (Generous portions of ESX version 3 and its Infrastructure add-ons appear to be devoted to performance monitoring.) ESX version 2 currently provides an interface to allow programs running on the VM guest to pull performance metrics, including the Total CPU Seconds used by the VM (in milliseconds). At least one third party performance monitoring product currently supplies measurements of VM Guest CPU % Used and % Ready (purportedly, the percentage of time a guest OS was Ready to run on the VM Host, but was unable to). Consolidating this information with data from various guest machines is always a challenge, of course.

Even though Windows presents system timer values in ticks of a 100 nanosecond clock, the system time value is actually only adjusted every periodic interval. The actual granularity of clock intervals is in the range of 5-10 milliseconds, as discussed above. This granularity proved too coarse for many performance-oriented measurements. So, beginning in Windows 2000, a new timer facility based on the TSC was introduced. This newer timing facility, also known as a high precision clock, is accessed using the QueryPerformanceCounter Win32 API call, somewhat inappropriately named because it is a general purpose interface that issues the RDTSC instruction.

In VMware, the TSC is virtualized and the RDTSC instruction, which can only be issued in privileged mode, is emulated. [15]. In [15], VMware warns,

> Reading the TSC takes a single instruction (rdtsc) and is fast on real hardware, but in a virtual machine this instruction incurs substantial virtualization overhead. Thus, software that reads the TSC very frequently may run more slowly in a virtual machine. Also, some software uses the TSC to measure performance, and such measurements are less accurate using apparent time than using real time. In a virtualization environment, there is no guarantee that the I/O interrupt from the clock will be issued or serviced by the guest OS in a timely manner.

The Windows performance measurements that are impacted by this anomaly are the Logical and Physical Disk % Idle Time counters that measure disk utilization, one of several disk performance counters of type PERF_PRECISION_100NS_TIMER. The accuracy of the Avg. Disk secs/Transfer counter that furnishes disk response time measurements is also suspect in a VMware environment. Both the Logical and Physical Disk measurement layers in the I/O Manager stack issue a call to QueryPerformanceCounter at the start and end of every I/O operation. Unfortunately, native ESX only measures disk and network throughput per virtual guest, which does not close the gap that not having these important guest OS disk performance statistics leaves.

8.0. Alternatives to virtualization.

Given these drawbacks in current virtualization technology, it is reasonable to ask why more IT departments do not embrace consolidating single application server workloads under a single OS image. If the problem is indeed too many OS machines (and machine images) to manage, why not consider consolidating multiple workloads under a single, native OS image. All in all, we see little reason today to prefer server consolidation using virtualization, which does not actually reduce the number of machine images that need to be managed, to workload consolidation under a single OS image, which does.

One reason for not consolidating workloads on a single image of the Windows OS is that the practice cuts across the grain of the institutionalized procedures that have grown up for testing deploying stable OS images, as discussed at the outset. For capacity planning, performance, and tuning issues, running isolated Windows machines running single applications was preferred in the past. The present author made a strong case for this administrative practice in [1] as far back as 1996 and more recently in [2]. The case for deploying single server application machines is substantially weaker today due to (1) major improvements in Windows Server and server application reliability, (2) increased flexibility to configure and manage heterogeneous workloads in IIS and SQL Server, (3) the capability to manage the performance of heterogeneous workloads using WSRM, and (4) the relief from virtual memory constraints that the 64-bit version of the OS grants. Let's consider each of these areas of improvement in more detail below.

For the Microsoft Windows server platform, there are two web-based services designed to improve reliability that are very visible. These are (1) the Automated Update feature that automatically delivers maintenance to the OS and its server applications, and (2) the facility to gather information about application program failures automatically and send data on these failures to Microsoft for analysis. The failure analysis program allows Microsoft to gather information on what problems are occurring and allocate resources appropriately to fix the more common and serious ones. As fixes to the software are developed, the Automatic Update service can deliver them to the machines that are experiencing the problems.

Another set of reliability improvements that is not as visible to system administrators, but is no less important to the cause of system stability is a series of initiatives to improve device driver reliability. Bugs in device driver code have historically been one of the most common causes of serious reliability problems. Microsoft has several programs to improve the quality of device driver code. It has made significant enhancements to the OS so that it can detect malfunctioning device drivers and recover better from these malfunctions. The result of all these initiatives is to improve the quality of the OS and driver code, which leads to significantly better reliability.

With the Web Gardens architecture developed for IIS version 6 and the multiple instance feature of SQL Server 2000, it is now possible to configure and manage multiple web sites and application databases under a single OS image. Using the Web Gardens feature of IIS 6.0, you can define multiple Application Pools. Then you can assign specific web sites to a specific Application Pool. Each Application Pool consists of one or more worker processes that run in isolation from each other. The health and performance of each Application Pool can be managed separately as well. With both SQL Server 2000 and 2005, it is possible to execute multiple instances of the sqlservr.exe process. Each copy of SQL Server can be managed separately, using processor affinity and memory working set settings to keep them from contending with each other for these system resources. The multiple address spaces available to web server applications and the DBMS also assist in scalability of these server applications on virtual memory-constrained 32-bit machines. [13]

Initial versions of the Windows server platform lacked any sort of built-in, system-wide tuning apparatus designed to help system administrators automatically balance resource usage among competing workloads according to some established set of management goals or priorities. A performance management framework that is capable of adjusting the priorities of various workloads automatically during times of stress makes it easier to configure and manage environments where heterogeneous workloads must co-exist on the same machine. In such an environment, whenever key shared computer resources such as the processor, memory, the disks, or the network become over-committed, the performance management framework can detect these conditions and intervene automatically, favoring higher priority workloads at the expense of lower priority ones.

As discussed long ago in [2], what performance management tuning knobs, options, and controls that did exist in Windows were initially available mainly at the application level. Moreover, the default values for the configuration parameters available for server applications like SQL Server, Exchange, the built-in Internet Information Server usually assumed that the application was being run on a dedicated server. Over-provisioning these machines to run a single workload or service is one handy way to deal with otherwise thorny capacity planning and performance and tuning issues because it ensures they surface only rarely.

In Windows Server 2003, Microsoft introduced the Windows Server Resource Manager (WSRM) utility that does provide system-wide, policy-based performance management capabilities. Using WSRM, the system administration can define workload categories. Based on the current resource utilization of the defined workloads, WSRM can dynamically adjust their dispatching priority when there is contention. WSRM controls can also be used to manage physical memory allocations and processor affinity settings. WSRM controls do not supersede the application-level settings that can be used to control SQL Server or IIS. But they do make it possible to manage any other servers applications you need to run from a system-wide perspective.

Finally, it is important to remark on the relief from 32-bit virtual memory constraints that the availability of 64-bit machines and a 64-bit OS provides. Even 32-bit applications gain some relief from virtual memory constraints in 64-bit Windows – they can expand to use a full 4 GB User private area. Native 64-bit applications like SQL Server 2005 currently have a huge 16 TB virtual addressing range available to them. Running the 64-bit OS is another way to consolidate workloads from multiple servers and run them effectively.

The same system administration practices and procedures that are adequate for running isolated, single application servers fall short of what is required for multiple workloads consolidated under a single OS image. In particular, both proactive performance management of the sort discussed in [13] and capacity planning and prediction are necessary to ensure that such machines run smoothly. Arguably, however, this is no more effort than the one required to manage a complex virtualization environment effectively. In fact, the performance and capacity planning effort necessary to assure stability in both environments appears to this observer to be very similar.

9.0 The Future of Virtualization.

Both Intel and AMD have announced strikingly similar new hardware designed with virtualization in mind. The Intel initiaive is called VT, while the AMD product is called Pacifica. This new hardware, scheduled to be available in 2006, adds a new privilege level where the virtualization monitor will run. That will allow the virtualization monitor to run a guest OS at its normal ring 0 privilege level, rendering current privileged instruction emulation techniques obsolete. The new hardware will also include new new instructions to allow the Virtual Machine Monitor to launch a guest virtual machine and facilitate communication between a guest OS and the VM Host software layer.

The new hardware appears to address the performance problems discussed here that plague the current virtualization solutions. Of course, a new generation of virtualization software will be required to exploit the new hardware. Both of the leading vendors of virtualization software report that they are actively working on new versions that will support the new hardware. With the right software support, for example, the new hardware environment could allow preferred guest machines to access native devices for better performance, although today we can only speculate about when virtualization software to permit that will be available.

References.

[1] Friedman, M., “Can Windows NT be tuned?” CMG *Proceedings* 1996.

[2] Friedman, M. and Pentakalos, O., *Windows 2000 Performance Guide*, Boston, MA: O’Reilly Associates, 2002.

[3] Fernando, G., “To V or not to V: a practical guide to virtualization,” CMG *Proceedings* 2005.

[4] Friedman, E., “Tales from the lab: Best Practices in application performance testing,” CMG *MeasureIT*, November 2005, available at http://www.cmg.org/measureit/issues/mit27/m_27_10.html.

[5] Seawright, L., and MacKinnon, R., “VM/370 – a study of multiplicity and usefulness, IBM Systems Journal, 1979, p. 4-17.

[6] *Intel Virtualization Technology Specification for the IA-32 Intel Architecture*, ftp://download.intel.com/technology/computing/vptech/C97063-002.pdf.

[7] Barham, et.al., “Xen and the Art of Virtualization,” University of Cambridge Computer Laboratory, published at *SOSP 2003*, available at http://www.cl.cam.ac.uk/Research/SRG/netos/papers/2003-xensosp.pdf.

[8] “ESX server performance and resource-management for CPU-intensive workloads.” Available at www.vmware.com.

[9] Gunther, N., “The Virtualization Spectrum from Hyperthreads to Grids,” CMG *Proceedings* 2006.

[10] Menasce, D., “Virtualization: concepts, applications and performance modeling,” CMG *Proceedings* 2005.

[11] Young, D., “Partitioning Large Processors,” CMG *Proceedings*, 1988, p. 67-74.

[12] “VMware ESX Server 2: Architecture and performance implications,” Available at www.vmware.com.

[13] Friedman, M., *Windows Server 2003 Performance Guide*, a volume in the Microsoft Windows Server 2003 Resource Kit, Microsoft Press, 2005.

[14] Shelden, W., “Modeling VMware ESX performance,” CMG Proceedings 2005.

[15] “Time-keeping in VMware virtual machines,” Available at www.vmware.com.

[16] Robin, J.S., and Irvine, C.E., “Analysis of the Intel Pentium’s Ability to Support a Secure Virtual Machine Monitor,” *Proceeding 9th USENIX Security Symposium*, 2000. Available at http://www.cs.nps.navy.mil/people/faculty/irvine/publications/2000/VMM-usenix00-0611.pdf.

[17] Waldspurger, C., “Memory Resource Management in VMware ESX Server,” *Proc. Fifth Symposium on Operating Systems Design and Implementation* (OSDI ’02), Dec. 2002. Available at http://www.waldspurger.org/carl/papers/esx-mem-osdi02.pdf.