vmware®

VMware ESX Server 2

Architecture and Performance Implications

ESX Server 2 is scalable, high-performance virtualization software that allows consolidation of multiple applications in virtual machines onto a single physical platform. While virtual machines are isolated from each other, the physical resources of the platform are shared between them based on the resource allocation policies set by the system administrator.

This white paper uncovers ESX Server software architecture and provides insights into the techniques used for efficient resource control. You will also learn about advanced scheduling configuration options and performance tools for measuring system resource utilization.

The topics covered in this white paper are the following:

This white paper is intended for partners, resellers, and advanced system administrators who wish to understand more about the ESX System architecture as well as the performance implications of configuration options that may affect their system deployment decisions.

Introduction

VMware ESX Server is the datacenter-class virtualization platform used by many enterprise customers for server consolidation. Virtual machines deployed and managed by VMware technology provide multiple isolated and protected computing environments that can run on the same server, at the same time. Virtual machines are ideal for server consolidation because the entire working environment of an application is encapsulated within the virtual machine. Different operating systems and configurations can run within virtual machines on a single physical machine. Also, if one virtual machine crashes, the others are isolated from the failure and continue running. ESX Server dynamically allocates system CPU, memory, disk, and networking resources to virtual machines based on need or guarantees set by an administrator, providing mainframe-class capacity utilization and control of server resources.

This paper describes VMware ESX Server capabilities and provides an overview of the various hardware, performance, and resource utilization issues to consider when deploying virtual machines. It also provides an overview of the ESX Server 2 software architecture and includes guidelines for making decisions and choosing between various options that may affect or determine virtual machine performance. This paper does not provide a complete list or description of ESX Server features or configuration parameters; the *ESX Server Administration Guide* is the best source for that information. The paper also does not focus on specific tuning tips or present specific performance data, but is intended instead to give readers the background necessary to understand the performance characteristics of applications running on virtual machines.

There are two main themes in this paper:

- Virtual machines share the physical resources of the underlying physical server. Each virtual machine does not necessarily have dedicated access to all of the physical server's CPU, memory, network, disk, or other devices. Rather, the server's resources are shared among different virtual machines on the same physical server, although the users or applications running on the different virtual machines are isolated from one another.
- Resource utilization is the key to performance. Making sure no resource is a bottleneck is the best way to get the highest performance out of ESX Server.

ESX Server guarantees each virtual machine its share of the underlying hardware resources based on the resource allocation policies set by the system administrator. ESX Server implements dynamic partitioning, where resources are de-allocated and re-allocated to virtual machines as needed by specific applications. Each virtual machine consumes some fraction of CPU, memory, network bandwidth, and storage resources of the underlying physical server. For example, if eight virtual machines were continuously running on a four-CPU SMP server with 8GB of memory and 160GB local storage, an equal allocation of the resources results in each virtual machine having available 50% of one CPU's processing capacity, 1GB of RAM, and 20GB of storage, minus any overhead required to perform and manage the virtualization.

ESX Server is designed for high performance, although as with any additional software layer, virtualization does add some overhead. The ESX Server software requires its own use of a certain amount of the physical machine resources. Virtual machines running applications with a varied mix of operations, such as operating system tasks, memory transactions, and I/O, experience different levels of virtualization overhead.

In many cases, the resources available on a physical machine exceed the resource requirements of the virtual machines and associated applications running on the same machine. (Most servers today are so lightly loaded that even combining multiple applications on a single server doesn't exceed the server's capacity.) So, even though the virtualization overhead may add to the overall latency (execution time) of applications running on a virtual machine, as long as there are sufficient physical resources available, the throughput (or data rate) can be comparable.

For example, consider a database where clients are generating a light load of 100 transactions per second, and each transaction takes an average of 1 millisecond to complete. If running the same database on a virtual machine causes each transaction to take 1.2 milliseconds to complete, the database is still be able to process the 100 transactions per second, since the total CPU resources consumed to perform the 100 transactions still does not fully utilize all of the available CPU.

However, as with any computer, if you keep increasing the load of applications running in virtual machines, it is possible to overload the system beyond what the physical resources can sustain, so that virtual machine performance is adversely affected.

ESX Server 2 Architecture

ESX Server runs directly on a physical server. In doing so, it has direct access to the physical hardware of the server, enabling high-speed I/O operations as well as having complete resource management control. The major conceptual components of ESX Server are:

- **Virtualization layer**—implements the idealized hardware environment and virtualizes the physical hardware devices including CPU, network and disk controllers, memory subsystem, and human interface devices. Each virtual machine sees its own set of virtual devices and is isolated from other virtual machines running on the same physical system.
- **Resource Manager**—partitions the physical resources of the underlying machine and allocates CPU, memory, disk, and network bandwidth to each virtual machine.
- **Hardware interface components**—enable hardware-specific service delivery while hiding hardware differences from other parts of the system. These components include the device drivers and the VMFS (VMware ESX Server File System).
- **Service Console**—boots the system, initiates execution of the virtualization layer and resource manager, and relinquishes control to those layers. It also runs applications that implement support, management, and administration functions.

Figure 1 shows the basic architecture of ESX Server 2. As shown in the figure, each virtual machine has its own guest operating system and applications. The VMkernel and Virtual Machine Monitor (VMM), shown in the middle band of the diagram, implement the virtualization layer. The VMkernel controls and manages the physical resources of the underlying server. The VMM implements the virtual CPUs for each virtual machine. The Resource Manager and hardware interface components are implemented in the VMkernel.

Finally, the Service Console component is shown on the far right in the diagram. The service console provides bootstrapping and other services to the ESX Server system.

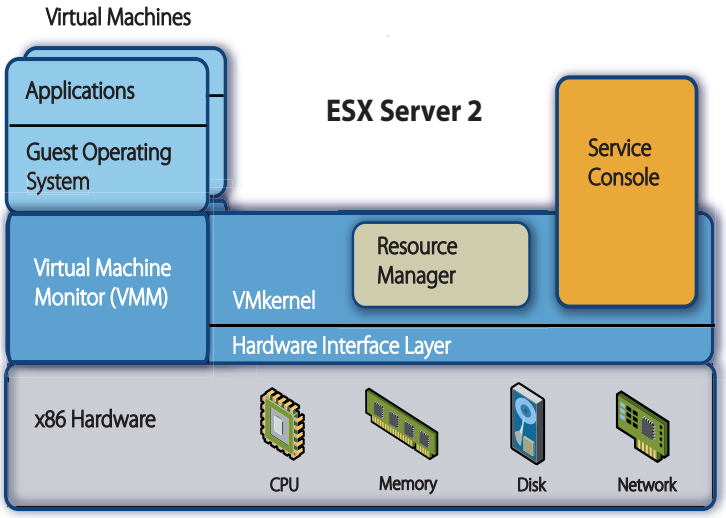

Figure 1: ESX Server architecture

Virtualization of Hardware Resources

VMware ESX Server virtualizes four key server resources: CPU, memory, disk, and network. The following section describes the implementation and performance characteristics for each of these resource types.

CPU Virtualization

Each virtual machine appears to run on a dedicated CPU, or set of CPUs, with each CPU having its own registers and control structures. VMware uses the terms virtual CPU and physical CPU to distinguish between the processors within the virtual machine and the underlying physical x86-based processors, respectively. ESX Server 2 supports virtual machines with one or two virtual CPUs. Virtual machines with more than one virtual CPUs are also called SMP (symmetric multi-processing) virtual machines. The virtual machine monitor (VMM) is responsible for virtualizing the CPUs. When a virtual machine starts running, control transfers to the VMM and it begins executing instructions from the virtual machine. The transfer of control to the VMM involves setting the system state so that the VMM runs on the bare hardware.

Traditional mainframe-based virtual machine monitors like those provided by the IBM System/370 ran a guest operating system directly on the underlying processor, but at a reduced privilege level. The VMM itself ran with supervisor privileges. When the virtual machine attempted to execute a privileged instruction, the CPU would trap or redirect control into the virtual machine monitor. The virtual machine monitor would then emulate the privileged operation in the virtual machine state.

The traditional trap-and-emulate approach is not sufficient, however, for current x86-based servers because their instruction set architecture is not strictly virtualizable—this means that there are instructions that can inspect or modify privileged machine state data without trapping, even when the instructions are executed from a non-privileged mode. Under certain conditions, the ESX Server VMM can run the virtual machine directly on the underlying processor like a traditional virtual machine monitor. This mode is called direct execution and it provides near-native performance in the execution of the virtual machine's CPU instructions.

Otherwise, the virtual machine CPU's instructions must be virtualized. This adds a varying amount of virtualization overhead depending on the specific operation being performed. In most cases, unprivileged, user-level application code runs in direct execution mode because, in most operating systems, user-level code does not access privileged state data. Operating system code does modify privileged state data and thus requires virtualization. As a result, a micro-benchmark that makes only system calls runs significantly more slowly in a virtual machine than on a native system. However, code that runs in direct execution mode incurs little extra performance costs and runs at near-native CPU instruction speed. Figure 2 provides a simplified view of the execution states of the VMware virtual machine monitor.

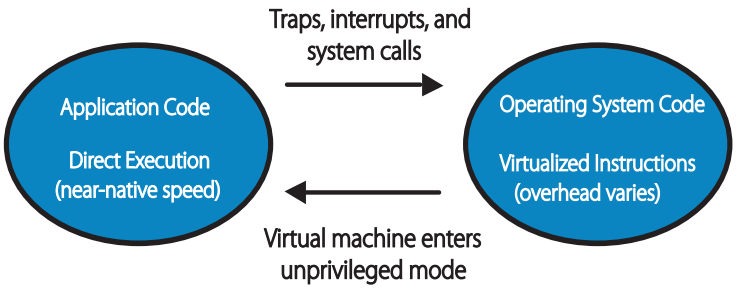

Figure 2: Execution states of the VMware virtual machine monitor

The virtual machine runs in direct-execution whenever possible. Traps, interrupts, system calls, and other events cause the virtual machine to enter the guest operating system, where instructions require virtualization by the VMM.

Note that there is a distinction between emulation and virtualization. With emulation, all operations are executed in software by an emulator. A software emulator allows programs to run on a computer system other than the one they were originally written for. The emulator does this by emulating, or reproducing, the original computer's behavior by accepting the same data or inputs and achieving the same results. With virtualization, the underlying physical resources are used whenever possible and the virtualization layer executes instructions only as needed to make the virtual machines operate as if they were running directly on a physical machine.

Emulation provides portability and is often used to run software designed for one platform across several different platforms. For example, there are a number of Atari 2600 emulators that can run Atari 2600 games on x86-based PCs. In contrast, virtualization emphasizes performance and runs directly on the processor whenever possible. Because VMware software virtualizes the CPU, the virtual machine is aware of the specific model of the processor on which it is running. Some operating systems install different kernel versions tuned for specific processor models, and these kernels are installed in virtual machines as well. That is why it isn't possible to migrate virtual machines installed on a machine running one processor model (say, an AMD processor) to another system (for example, running with Intel processors).

Performance Implications CPU virtualization adds varying amounts of overhead depending on the percentage of the virtual machine's workload that can run in direct execution, and the costs of virtualizing the remaining instructions that can't be executed directly. For applications that are CPU-bound (that is, most of the application's time is spent executing instructions rather than waiting for external events such as user interaction, device input, or data retrieval), any CPU virtualization overhead likely translates into a reduction in overall performance. In those cases, the CPU virtualization overhead requires additional instructions to be executed, which takes CPU processing time that could otherwise have been used by the application itself. For

applications that are not CPU-bound, any CPU virtualization likely translates into an increase in CPU utilization but since there's CPU available to absorb the overhead, it can still deliver comparable performance in terms of overall throughput.

ESX Server 2 supports one or two virtual processors or CPUs. Some applications are single-threaded and thus can take advantage only of a single CPU. Deploying such applications in dual-processor virtual machines won't speed up the application but it does cause the second virtual CPU to use physical resources that could otherwise be used by other virtual machines. As a result, single-threaded applications should be deployed in uniprocessor virtual machines for best performance and resource utilization.

Memory Virtualization

The VMkernel manages all machine memory, except for the memory that is allocated to the service console. (ESX Server 2 requires between 192 and 512MB depending on the number of virtual machines that are run.) The VMkernel dedicates part of this managed machine memory for its own use and the rest is available for use by virtual machines. Virtual machines use machine memory for two purposes: the VMM requires memory for its own code and data, and the virtual machines require their own memory.

All modern operating systems provide support for virtual memory. This gives software the ability to use more memory than the amount of physical memory the machine actually has. The virtual memory space is divided into smaller size blocks, typically 4 KB, called pages. The physical memory is also divided into blocks, also typically 4 KB in size. When physical memory is full, the data for virtual pages that are not present in physical memory are stored on disk. (The x86 architecture has an additional mapping from virtual memory to linear memory via segmentation, which is not described here.)

On native systems, page tables translate virtual memory addresses into physical memory addresses. Within a virtual machine, the guest operating system's page tables maintain the mapping from guest virtual pages to guest physical pages.

ESX Server virtualizes guest physical memory by adding an extra level of address translation. The VMM for each virtual machine maintains a mapping from the guest's physical memory pages to the physical memory pages on the underlying machine. VMware refers to the underlying physical pages as machine pages and the guest physical pages as physical pages. Each virtual machine sees a contiguous, zero-based, addressable physical memory space. The underlying machine memory on the server used by each virtual machine may not be contiguous.

The VMM intercepts virtual machine instructions that manipulate guest operating system memory management structures so that the actual memory management unit on the processor is not updated directly by the virtual machine. ESX Server maintains the virtual-to-machine page mappings in a shadow page table that is kept up to date with the physical-to-machine mappings. The shadow page tables are then used directly by the processor's paging hardware. This allows normal memory accesses in the virtual machine to execute without adding address translation overhead, once the shadow page tables are set up. Since the translation look-aside buffer on the processor will cache direct virtual-to-machine mappings

read from the shadow page tables, no additional overhead is added by the VMM to access the memory. Figure 3 illustrates the ESX Server implementation of memory virtualization.

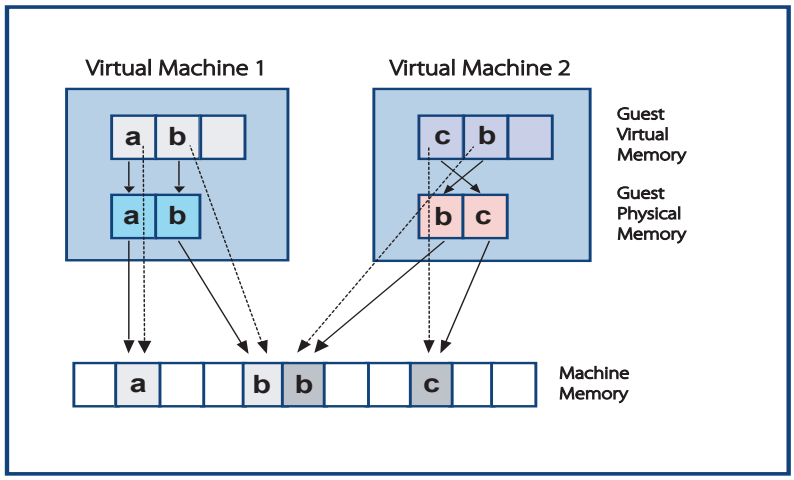

Figure 3: ESX Server Memory Virtualization.

The boxes shown in the figure represent pages and the arrows show the different memory mappings. The arrows from guest virtual memory to guest physical memory show the mapping maintained by the page tables in the guest operating system. (Note that the mapping from virtual memory to linear memory for x86-architecture processors is not shown in Figure 3). The arrows from guest physical memory to machine memory show the mapping maintained by the VMM. The dashed arrows in the illustration show the mapping from guest virtual memory to machine memory in the shadow page tables also maintained by the VMM. The underlying processor running the virtual machine uses the shadow page table mappings.

Because of the extra level of memory mapping introduced by virtualization, ESX Server can efficiently manage memory across all virtual machines. Some of the physical memory of a virtual machine may in fact be mapped to shared pages or pages that are unmapped, or swapped out. ESX Server performs its virtual memory management without the knowledge of the guest operating system and without interfering with the guest operating system's own memory management subsystem. (See Memory Management on page 14 for more information.)

Performance Implications There are two kinds of memory overhead that may be incurred by ESX Server virtual machines:

- the additional time to access memory within a virtual machine
- the extra space needed by ESX Server for its own code and data structures, beyond the memory allocated to each virtual machine.

ESX Server memory virtualization adds little time overhead to memory accesses. As previously discussed, because the shadow page tables are used directly by the processor's paging hardware, most memory accesses in the virtual machine can execute without address translation overhead. However, updates by the virtual machine to memory management data structures (for example, writing a page table entry to a page table, or changing page table root pointers) are intercepted and processed by the VMM, which adds some virtualization overhead. For example, page faults originating in the virtual machine causes control to switch into the VMM so that the VMM can update its data structures.

The memory space overhead is comprised of two components: a fixed system-wide overhead for the service console and the VMkernel, and an additional overhead for each virtual machine. For ESX Server 2.5, the service console typically uses from 192 to 512MB and the VMkernel uses a smaller amount of memory, about 24MB.

For virtual machines configured with 512MB or less, the overhead memory required for each virtual machine is typically 54MB for a uniprocessor virtual machine, and 64MB for a two-way SMP virtual machine. Virtual machines configured with more than 512MB require an additional 32MB of overhead memory per additional gigabyte of configured main memory. This amount of overhead memory per virtual machine is for data structures such as the virtual machine's SVGA frame buffer and memory mappings maintained by the VMM. These VMM memory mappings are why additional overhead memory is needed for each extra 1GB of virtual machine memory.

ESX Server also provides optimizations to reduce the amount of physical memory used on the underlying server, which can save more memory than is taken up by the overhead. For more information, see Memory Management on page 14.

Device Emulation

The VMkernel is responsible for managing virtual machine access of network and storage devices. The service console handles other devices such as floppy disks and CD-ROMs. The VMkernel handles networking and storage directly because of the performance-critical nature of these devices in server environments.

Physical network and storage devices are presented as virtual devices to the virtual machines running on ESX Server. Each storage device appears as a SCSI drive connected to a SCSI adapter within the virtual machine, regardless of the specific SCSI, RAID, and Fibre Channel adapters that might actually be installed on the underlying server. For example, the server may be attached to a SAN (Storage Area Network) using a QLogic or Emulex HBA (host bus adapter), but the guest operating system sees it as a SCSI adapter. ESX Server emulates either a BusLogic or an LSI Logic SCSI adapter, so either a BusLogic or an LSI Logic driver gets loaded in the guest operating system. The virtual machine's virtual disks are implemented as files on the underlying storage.

For network I/O, VMware emulates either an AMD Lance/PCNet adapter or uses a custom interface called `vmxnet`. For the Lance adapter, the guest device driver is the AMD PCnet driver.

The `vmxnet` guest device driver is available as part of the VMware Tools and works with the corresponding emulated vmxnet networking adapter.

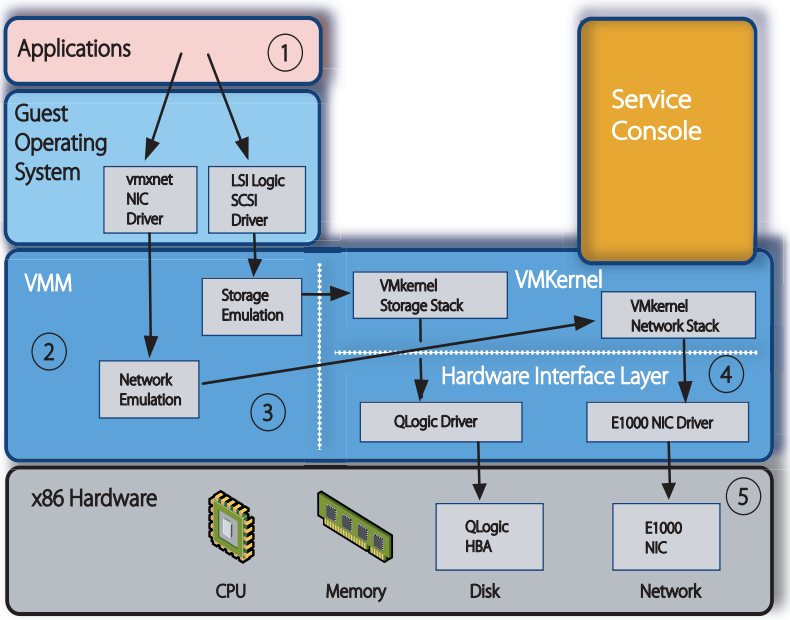

Figure 4: ESX Server I/O path

Figure 4 shows the ESX Server architecture from Figure 1 modified to illustrate the paths taken when the virtual machine issues network or storage I/O requests. In this example, the virtual machine has a `vmxnet` virtual network adapter and an LSI Logic SCSI adapter. The underlying physical server has a QLogic Fibre Channel HBA attached to a SAN and an Intel e1000 network adapter.

The application within the virtual machine first makes an I/O request to the guest operating system—for example, to send data over the network, or to read or write from a file on disk (step 1 in the figure). The guest operating system processes the request and calls the guest device driver (step 2). When the guest device driver attempts to access the device, the VMM intercepts the I/O request and then transfers control to the VMkernel (step 3). The device-independent networking or storage layer in the VMkernel then processes the I/O request (step 4).

The VMkernel manages requests from multiple virtual machines and takes into account any administrator-specific resource management controls. The VMkernel then sends the I/O request to the hardware interface layer, which contains the VMkernel device driver for the specific device on the physical machine (step 5). Note that this is a simplified view and does not cover some of the more advanced ESX Server features such as storage multipathing or virtual local area networks.

When an I/O request completion interrupt arrives, processing of the completion takes the reverse path of an I/O request. The VMkernel device driver fields the interrupt and calls the VMkernel to process the interrupt. The VMkernel then notifies the VMM of the target virtual machine, which raises the interrupt to the guest operating system in the virtual machine. The VMkernel ensures that data intended for each virtual machine is isolated from other virtual machines. Since ESX Server 2.1, interrupt handling has been improved for most cases to handle interrupts on idle, physical CPUs whenever possible. This allows the VMkernel to utilize CPU

resources of the server more efficiently. Once the VMkernel handles the interrupt, the VMkernel then delivers it to the virtual CPU that can most efficiently process the interrupt. Note that fielding interrupts on idle processors has scalability implications. For example, when you run a single virtual machine, that virtual machine may consume a portion of another otherwise idle processor to handle interrupts. This additional utilization should be taken into account when you estimate the total number of virtual machines that can be run on the server. This effect is noticeable only for extremely disk- and network-intensive workloads.

The networking and storage I/O path from guest operating system to hardware (and vice versa) is relatively straightforward and requires no involvement of the service console. However, it does involve a transition from the virtual machine context to the VMkernel context. In the case of network I/O, the latency of physical network devices is low so these devices can send and receive network packets at a high rate. To avoid the overhead of the transitions between the virtual machine and VMkernel contexts, ESX Server with `vmxnet` collects a cluster of network packets as it sends or receives them, before making the transition. The clustering occurs only if the rate of packets sent or received is high so that packets are not delayed unnecessarily. ESX Server with `vmxnet` can also take advantage of advanced features available on some network cards, such as TCP checksum and TCP segmentation offloading.

Performance Implications for Storage ESX Server emulates either a BusLogic or LSI Logic SCSI adapter, which is likely to be different from the adapter installed on the physical server. The specifics of the implementation of the SCSI driver loaded into the guest operating system can affect disk I/O throughput. For example, the depth of the queue of outstanding commands in a driver can significantly impact disk performance. A queue depth that is too small limits the disk bandwidth that can be pushed through the virtual machine. For the BusLogic adapter, VMware provides a custom BusLogic driver for Windows guest operating systems that is recommended for applications requiring high performance. (The BusLogic driver is part of VMware Tools and can be installed when you install VMware Tools.) The guest driver queues may also be tuned. (See the driver-specific documentation for more information on how to do this.)

The driver queue depth can also be set for some VMkernel drivers. For example, the default queue depth of the QLogic driver is 16; however, specifying larger queue depths may yield higher performance. You can also adjust the number of outstanding disk requests per virtual machine in the VMkernel through the ESX Server Management User Interface. Setting this parameter can help equalize disk bandwidth across virtual machines. (See knowledge base article 1269 for details.)

Different storage systems may also have their own tuning and configuration parameters. For example, issues such as the RAID level you choose for disk arrays, the distribution of LUNs across RAID sets, and whether or not caching is enabled on the storage device, can all drastically impact storage performance both natively and in virtual machines. Storage performance issues you encounter are most often the result of configuration issues with underlying storage devices and are not specific to ESX Server. If you suspect a storage performance problem, you may want to try running applications natively, if possible, to determine if the problem is due to a storage configuration issue. If storage is centrally managed or configured in your environment, contact your storage administrator.

Performance Implications for Networking ESX Server emulates either an AMD Lance/PCnet network adapter (called `vlance`) or the custom `vmxnet` network adapter. The `vlance` adapter is the default, although `vmxnet` is recommended for higher performance. To use the `vmxnet` network adapter, configure virtual machines to use `vmxnet` through the ESX Server Management User Interface and install the vmxnet driver (available in the VMware tools) on the virtual machines. Note that the network speeds reported by the AMD PCnet and `vmxnet` guest

drivers on the virtual machine do not necessarily reflect the actual capacity of the underlying physical network interface card. For example, the AMD PCnet guest driver on a virtual machine reports a speed of 10Mbps, even if the physical card on the server is 100Mbps or 1Gbps, because the AMD PCnet cards that ESX Server emulates are 10Mbps, by definition. However, ESX Server is not limited to 10Mbps and transfers network packets as fast as the resources on the physical server machines allow.

As is the case with native networking performance, gigabit networking requires more CPU resources than 100Megabit networking. As a result, available CPU resources may limit the networking bandwidth you can obtain. For example, a 500MHz server with a gigabit network card is likely to run out of CPU cycles before saturating the gigabit link. As is the case for disk performance, native system network parameter tuning can also provide significant improvements in virtual machine performance.

Resource Management

ESX Server installations commonly have many virtual machines running on the same physical machine. This section describes how ESX Server manages and allocates the physical resources across multiple virtual machines.

CPU Scheduling

ESX Server gives administrators dynamic control over the CPU utilization and processor assignments of virtual machines. ESX Server implements proportional-share processor scheduling with optional rate guarantees to control the amount of CPU time allocated to each virtual machine. Each scheduled virtual machine is allocated a number of shares. The amount of CPU time given to each virtual machine is based on its fractional share of the total number of shares in the whole system. For example, if a virtual machine has three times as many shares as another virtual machine, it is entitled to three times as much CPU time. Note that changing the number of shares allocated to one virtual machine changes the total number of shares and thus can impact how CPU time is allocated across all virtual machines.

Shares are not the same as hard partitions of CPU time. If a virtual machine does not use its full allocation of shares, for example, because it is currently halted, the unused shares are partitioned among the remaining active virtual machines. That way, unused shares are not wasted and the active virtual machines benefit from the extra resources available. Similarly, if a virtual machine is idle, ESX Server detects the inactivity and treats the virtual machine as halted.

ESX Server supports running more virtual machines than there are physical processors on the server, in which case, the virtual machines share the underlying CPUs. The ESX Server scheduler gives each virtual machine the portion of CPU time to which it is entitled based on its shares. The scheduler may migrate virtual machines to different processors to ensure that each virtual machine gets the proper amount of CPU time. Note that one, and only one, virtual processor can run on a physical processor or CPU at any given instant.

For two-way SMP virtual machines, the virtual processors from the same virtual machine are co-scheduled, that is, the virtual processors are mapped one-to-one onto the underlying processors and are run simultaneously, if possible. Co-scheduling ensures that the guest operating system behaves as if it were running on a dedicated server with the same number of processors as the virtual machine. In some cases, guest operating systems have timing constraints that require operations on other processors to complete within a certain amount of time. Co-scheduling also improves performance by facilitating prompt communication and synchronization between the processors. For example, consider the case where one virtual CPU is spinning on a lock held by the other virtual CPU. Without co-scheduling, if the virtual CPU holding the lock gets de-scheduled, the other virtual CPU wastes CPU cycles spinning on the

lock. It will continue to spin until the virtual CPU holding the lock is finally scheduled again and releases the lock. Co-scheduling avoids this situation by running both virtual CPUs at the same time.

The ESX Server scheduler does allow the execution of virtual CPUs within a virtual machine to become slightly skewed, where one virtual CPU runs without the other for a limited amount of time. A periodic timer checks the status of the virtual CPUs, and de-schedules the virtual machine only if the skew between the virtual CPUs of a virtual machine is greater than a fixed threshold (currently 1.5 milliseconds by default).

Figure 5 shows a simple scheduling situation in which five virtual machines contend for CPU resources. In this figure, the virtual machines having a single square represent uniprocessor virtual machines. Virtual machines with two squares represent two-way SMP virtual machines. The bottom portion of the figure shows a possible scheduling scenario with these five virtual machines scheduled onto a system with four physical processors at three different points in time.

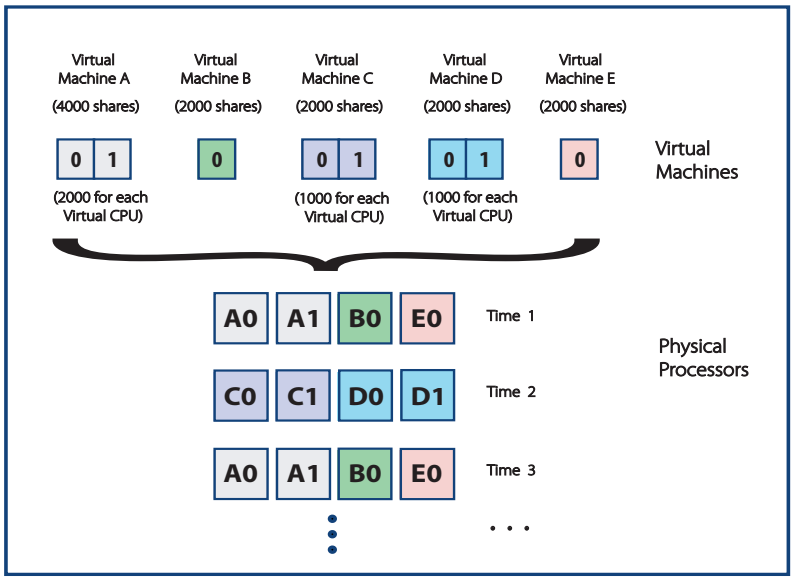

Figure 5: ESX Server 2 scheduling

The ESX Server scheduler runs the virtual machines and co-schedules virtual CPUs from the SMP virtual machines according to the shares each virtual machine has been allocated.

Administrators can also specify absolute guarantees for minimum and maximum CPU utilization for each virtual machine. The minimum and maximum utilizations are expressed as a percentage of a physical CPU's entire time availability. A minimum CPU reservation guarantees that a virtual machine is always able to use this minimum percentage of a physical CPU's time, regardless of the total number of shares. A maximum CPU limit ensures that the virtual machine never uses more than this maximum percentage of a physical CPU's time, even if extra idle time is available. ESX Server also has an admission control policy that does not allow you to power on a virtual machine if ESX Server cannot guarantee the minimum CPU reservation time specified for that virtual machine. After the minimum and maximum allocations have been determined, the shares are used to allocate any contended CPU resources. Shares have no effect if the maximum limit is set to match the minimum limit.

ESX Server lets users set affinity for virtual machines to restrict the virtual machines so they run on only a subset of the physical processors available. This can be useful for creating a fixed partitioning of the CPU resources. However, in doing so, the ESX Server scheduler attempts to maintain fairness across all virtual machines, but affinity constraints may make this impossible, so performance may be affected. For example, if several virtual machines are all designated to run on one processor, then the scheduler must run them on that processor even if other processors are available.

Rather than using affinity settings, you should set minimum CPU reservation rates to ensure the most important virtual machines get the resources they need while still giving the ESX Server scheduler flexibility to make the best use of the physical processors. If you must specify affinity constraints, note that the service console always runs on CPU 0 (the first physical processor). As a result, it is worthwhile to keep CPU 0 as free as possible. Since virtual machines rely on the service console for a variety of functions (including display, keyboard, and low-speed I/O devices), making the service console wait can slow down the overall operation of virtual machines.

See the *ESX Server 2 Administration Guide* for more information on CPU share, affinity, and scheduler-related configuration options. The ESX Server scheduler is quite effective at making the best use of an underlying server's CPU resources. If all the virtual machines running on your ESX Server have equal importance, there is no need to explicitly specify min, max, or shares, and you can just use the defaults. But if you do need to preferentially allocate resources to virtual machines, the min, max, and shares options should be used.

Scheduling for NUMA Architectures ESX Server 2 provides memory access optimizations for both Intel processors and AMD Opteron processors in server architectures that support NUMA (non-uniform memory access). NUMA machines are made up of multiple nodes in which each node typically has a small number of processors plus main memory. The memory on all the nodes forms a global address space and all processors can access memory transparently from any node; however, access to local memory is faster than access to memory on remote nodes. The ESX Server scheduler attempts to balance virtual machines across the NUMA nodes to maximize the use of local memory while maintaining even CPU utilization among the nodes. See the *ESX Server 2 NUMA Support* white paper for more information. Also see the *ESX Server Administration Guide* and the `numa(8) man` page.

Hyperthreading Support ESX Server 2.1 and later versions support hyperthreading on processors that have this feature. Hyperthreaded processors can run multiple independent threads simultaneously. Intel recently added hyperthreading to the Intel Pentium IV and Xeon processors, and these processors support two independent threads. In Intel's terminology, the single processor is called a package and each thread is called a logical processor. On a hyperthreaded processor, most processor resources are shared between the threads, including the second- and third-level caches and all other functional units of the CPU. This differs from a true dual-core processor where the on-chip caches and CPU functional units are replicated. As a result, hyperthreading does not provide the same performance as a dual-core processor, but does allow better utilization of the processor resources for certain workload mixes.

Support for hyperthreading in ESX Server is turned on by default. The number of processors listed in the ESX Server management user interface reflects the number of logical processors available. (The number displayed is reduced by half if hyperthreading is turned off). Within the virtual machine, however, the guest operating system does not detect the virtual processors as being hyperthreaded.

The ESX Server scheduler schedules virtual machines onto the logical processors. The additional scheduling flexibility provided by using logical processors can lead to significant improvements in overall resource utilization. For example, when running a uniprocessor virtual machine and an SMP virtual machine on a two-way server, ESX Server co-schedules both virtual CPUs in the SMP virtual machine, which means that both virtual CPUs must be running at the same time or are idle. Without hyperthreading, if a uniprocessor virtual machine were running, one processor is idle but ESX Server is not able to run the SMP virtual machine, because two available processors are required. With hyperthreading, ESX Server can schedule the two virtual CPUs of the SMP virtual machine onto two logical processors in the same package, and then put the uniprocessor virtual machine onto the other package by itself.

The ESX Server scheduler accounts for CPU time in terms of package seconds, rather than logical processor seconds. Thus, a virtual CPU running on a package alongside another virtual CPU gets charged half the CPU time of a virtual CPU running on a package by itself. The ESX Server scheduler resource controls are integrated with hyperthreading. Virtual machines still receive CPU time proportional to their share allocation, but the time is capped by user-specified minimum and maximum values. While shares specify a relative allocation of resources, the minimum and maximum value settings guarantee that virtual machines get an absolute percentage of a package's resources. For example, a virtual machine with a minimum value set at 75% is guaranteed at least 75% of the entire package's resources, not just 75% of a logical processor's resources.

To meet the guarantees specified by the administrator's resource controls, the ESX Server scheduler may expand a high-priority virtual machine to use an entire package by not scheduling anything on its partner logical processor. This works even if there are other virtual machines ready to run in the system. This does not waste resources, but rather allows the virtual machine to use a full physical package. Similarly, the scheduler may expand a high-priority SMP virtual machine to use two entire packages, with one virtual CPU on each package. ESX Server also provides configuration options to control the degree of package sharing for individual virtual machines. See the *Hyperthreading Support in ESX Server 2.1* white paper for more details. Also see the *ESX Server Administration Guide* and the `hyperthreading(8) man` page.

Memory Management

ESX Server gives administrators dynamic control over the physical memory allocation of virtual machines. As many systems underutilize their available memory, ESX Server allows memory to be over-committed so that the total memory size for all running virtual machines exceeds the total amount of physical memory. ESX Server manages the physical memory of the underlying server based on the allocations given to each virtual machine. There are three basic parameters that administrators can use to adjust ESX Server memory allocations: minimum, maximum, and shares. The following section provides a description of these parameters.

Memory Allocation Maximum memory for each virtual machine is specified in the virtual machine's configuration file and sets the amount of memory configured by the guest operating system running in the virtual machine. Administrators can also specify absolute guarantees for minimum memory utilization. This minimum size is a guaranteed lower bound on the amount of memory allocated to a virtual machine, even if the total amount of memory is overcommitted. Unless otherwise specified, the minimum memory size is typically half of the maximum memory setting depending on available unreserved RAM and swap space. It is recommended you set explicit minimum memory sizes above the point of the virtual machines basic working set size, for consistent performance.

Virtual machines are granted their maximum memory size unless memory is overcommitted. When memory is overcommitted, each virtual machine is allocated an amount of memory between its minimum and maximum sizes. The amount of memory granted to a virtual machine above its minimum size may vary with the current memory load. ESX Server automatically determines allocations for each virtual machine based on the number of memory shares the virtual machine has been given and an estimate of its recent working set size.

ESX Server implements proportional-share memory allocation to control the amount of physical memory for each virtual machine. As with CPU shares, memory shares entitle virtual machines to a proportional share of total physical memory. However, virtual machines that are not actively using their currently allocated memory automatically have their effective number of shares reduced, by levying a tax on idle memory. This "memory tax" helps prevent virtual machines from unproductively hoarding idle memory. A virtual machine is charged more for an idle memory page than for a memory page that it is actively using. The memory tax is removed when the virtual machine expands its memory usage.

ESX Server admission control ensures that sufficient unreserved memory and swap space are available before the user powers on a virtual machine. Memory must be reserved for the virtual machine's guaranteed minimum size, plus the overhead memory that is required for virtualization. Note that ESX Server also attempts to keep some memory free at all times to handle dynamic allocation requests efficiently. ESX Server sets this level at approximately six percent of the memory available for running virtual machines. Swap space must be reserved on disk for the remaining virtual machine memory, that is, the difference between the maximum and minimum memory settings. The swap space reservation is needed to ensure that a virtual machine's memory can be saved under any circumstances; in practice, only a small amount of swap space is used. A default swap file size equal to the physical memory size of a server machine is recommended, which allows up to a two-times level of memory over-commitment.

Memory Reclamation ESX Server uses two memory management mechanisms, ballooning and swapping, to expand or contract the amount of memory allocated dynamically for each virtual machine. Ballooning is the preferred method and provides predictable performance, which closely matches the behavior of a native system under similar memory constraints. With ballooning, ESX Server works with a VMware-supplied `vmmemctl` module loaded into the guest operating system to reclaim pages that are considered least valuable by the guest operating system. The `vmmemctl` driver is installed as part of the VMware Tools. Ballooning effectively increases or decreases memory constraints on the guest operating system, and causes the guest to invoke native memory management algorithms. When memory is scarce, the guest operating system identifies which particular pages to reclaim and, if necessary, swaps them to an allocated virtual disk. Note that to use ballooning, the guest operating system must be configured with sufficient swap space.

Swapping is used to forcibly reclaim memory from a virtual machine when no `vmmemctl` driver is available. This may be the case if the `vmmemctl` driver was never installed, has been explicitly disabled, is not running (for example, while the guest operating system is booting), or is temporarily unable to reclaim memory fast enough to satisfy current system demands. Standard demand paging techniques swap pages back in place when the virtual machine needs them.

If the working set (active memory) of the virtual machine resides in physical memory, using the swapping mechanism and having inactive pages swapped out does not affect performance. However, if the working set is so large that active pages are continuously being swapped in and out (that is, the swap I/O rate is high), then performance may degrade significantly. The `vmmemctl` approach (ballooning) is used whenever possible for optimum performance.

Swapping is a reliable mechanism of last resort that the system uses to reclaim memory only when necessary.

Memory Sharing The VMkernel also implements a page sharing optimization to reduce the amount of memory needed. The process of page sharing maps identical pages in one or more virtual machines to a single page of physical memory on the server. The VMkernel scans pages looking for identical content and then remaps the virtual machines' pages to a single read-only copy of the page. If any virtual machine attempts to write to the shared page, it is allocated a new copy of the page with read-write permissions.

Compared to other methods, page sharing can significantly reduce overall memory requirements. For example, if multiple virtual machines are running the same guest operating system and same applications, their pages can typically be shared. The rate at which the VMkernel scans pages is low, and the scanning itself does not adversely impact performance.

Figure 6 illustrates how page sharing memory virtualization is performed (using the example from Figure 3 in which one page is shared between virtual machines).

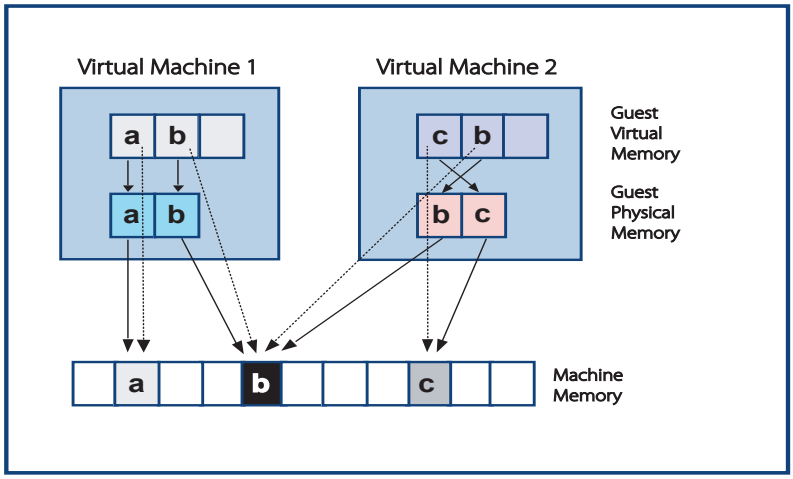

Figure 6: ESX Server Memory Virtualization with Page Sharing.

In this example, the machine page, shown in black in the figure, is transparently shared by both virtual machines. Only a single copy of the page is needed and both virtual machines point to the same machine page. See the *ESX Server 2 Administration Guide* for information on memory management configuration options.

Device Resource Management

ESX Server provides configuration options to control the amount of disk and network bandwidth allocated to each virtual machine, similar to the way options are used to control CPU and memory allocation. For networking, ESX Server uses a traffic-shaping module that limits the amount of outbound traffic for each virtual machine. For disk devices, ESX Server implements a proportional-share algorithm to balance disk bandwidth across virtual machines. See the *ESX Server 2 Administration Guide* for details.

The Hardware Interface

This section describes the hardware interface components and discusses performance implications in choosing various configuration options for hardware device access.

Device Drivers

The VMkernel loads device drivers that access the physical devices on a server machine, such as the network interface card, SCSI adapters, and host bus adapters (HBAs). The ESX Server installation process detects the devices that are assigned to the VMkernel and loads the appropriate driver modules into the VMkernel.

ESX Server uses Linux device drivers that have been ported to run in the VMkernel. The port is facilitated by a Linux compatibility layer that provides an interface between the VMkernel and the Linux drivers.

The VMFS File System

ESX Server implements its own file system called VMFS. VMFS is a distributed file system that allows multiple hosts to concurrently access files on the same VMFS volume. It is designed to provide high performance I/O to virtual machines. VMFS is optimized for storing and accessing large files such as virtual disks and the memory images of suspended virtual machines. Virtual disk files are typically larger than 1GB in size, so VMFS is optimized to work with a smaller number of large files. VMFS uses large file system blocks (1MB to 256MB; the default is 1MB) to keep the amount of metadata (data structures that describe format of files) required for a virtual disk file small. That way, all of the metadata can likely be cached in main memory, and all file system reads and writes can be done without having to access the disk to read or write the metadata. Note that VMFS does not cache data from virtual machine I/O operations. This is to ensure crash consistency and provide stable performance.

Starting with ESX Server 2.1, a VMFS-2 volume can span multiple partitions, across the same or multiple LUNs or physical disks (up to 32). A VMFS-2 volume is a logical grouping of physical extents where each physical extent is a disk partition. Note that adding extents to a VMFS-2 volume may not necessarily improve performance. ESX Server itself does not stripe disk blocks across the extents; rather extents are filled sequentially as the storage is used and the performance benefits achieved are based substantially on the performance of the underlying storage configuration.

Virtual Disk Modes ESX Server supports four disk modes: persistent, nonpersistent, undoable, and append. Using persistent disks typically results in higher disk I/O performance than the other disk modes. When a virtual machine uses a persistent disk (the default), all disk writes go to the VMFS file containing that virtual disk and the virtual disk behaves like a disk drive on a physical machine. When a virtual machine uses the undoable, non-persistent, or append disk modes, disk writes are appended to a redo log. When a virtual machine reads from disk, it first checks the redo log (by looking at a directory of disk blocks contained in the redo log) and, if the redo log is present, reads that information. Otherwise, the read goes to the base disk for the virtual machine.

Raw Disks ESX Server 2.5 introduced support for Raw Device Mapping (RDM), which allows management and access of raw SCSI disks or LUNs as VMFS files. An RDM is a special file on a VMFS-2 volume that acts as a proxy for a raw device. The RDM file contains metadata used to manage and redirect disk accesses to the physical device. Previous ESX Server 2 releases support using raw disks, although the raw disks are not managed as VMFS files. Virtual disks are recommended for most virtual disk storage; raw disks may be needed in some cases. Common applications for raw disks include use as data drives for Microsoft Cluster Services (MSCS) configurations and to run SAN-aware applications inside of virtual machines. For more information on raw device mapping, see the *ESX Server Raw Device Mapping* technical note.

The VMware Service Console

The purpose of the service console is to boot the physical server machines and administer virtual machines. After the machine boots into the service console, the VMkernel is loaded and takes over the machine. The service console supports devices that are not performance critical, such as the mouse, keyboard, screen, floppy drives, CD-ROM, COM ports, and parallel ports. The service console also runs applications that implement support, management and administration functions. The ESX Server resource manager schedules the service console to run, as needed. The service console is special in that it always runs on physical CPU 0. Virtual machines can also run on physical CPU 0, but the service console cannot move to other physical CPUs.

When a remote console is attached to a virtual machine, a VMware application running on the service console and sends the display to the remote console application running on the client. Running remote consoles uses resources on the service console. When connecting many remote consoles, you should monitor the resource utilization (primarily CPU, memory, and networking) of the service console to make sure that no resource becomes a bottleneck. Note that poor remote console performance, for example, if there's insufficient network bandwidth between the remote console and the ESX Server machine, may give a false impression that virtual machine performance is also lagging. If you suspect the service console needs more CPU time, you can increase its CPU minimum (default is 8%). This is more likely to be an issue on larger (8- or 16-way) machines.

An adequate amount of memory, swap, and disk space should also be assigned to the service console. The amount of memory required by the service console is a function of the number of virtual machines that will be run on the associated server machine. Additional memory should be allocated to the service console to accommodate any other applications running directly on the service console, such as system management agents. If the service console has too little memory and begins swapping, its applications slow down considerably. This can adversely affect the performance of all virtual machines running on the same server machine. See the *ESX Server 2 Administration Guide* for guidelines on how to determine the size of memory to allocate to the service console.

Performance issues can arise with the service console when interrupt lines for device controllers are shared between the service console and the VMkernel. (Ideally, each controller has a dedicated interrupt line to the processor that raises an interrupt whenever the processor needs to take action for a device.) When an interrupt line is shared, the processors cannot determine which controller is sending the interrupt, so they have to execute the interrupt handlers for all the controllers that share the same interrupt line. If a VMkernel device and a service console device share an interrupt, calling the controller drivers in sequence results in context switches between the service console and the VMkernel every time there is an interrupt on that line. This can significantly impact performance. Note that devices that are marked as shared by ESX Server at install time are managed by the VMkernel, so performance for those devices may not be affected. Performance issues are more likely to arise when different devices share interrupts, such as a USB controller on the service console sharing an interrupt with a VMkernel NIC, or with a multiport NIC where some of the ports are managed by the VMkernel and other ports are managed by the service console. We recommend disabling any unused devices (such as floppy disk, CD-ROM, and USB controllers) on the system for best performance. See KB article 1290 for information on how to diagnose and work around problems in these situations. We also recommend disconnecting any unused devices within the virtual machines.

Monitoring Resource Utilization in ESX Server

As with any system, it is important to monitor the resource utilization of your virtual machines as well as the physical machines that run ESX Server. You should make sure that there are sufficient resources for running all your virtual machines, since they will be sharing resources, and if any one resource becomes a bottleneck, the performance of the virtual machines will suffer. The following section provides a description of tools that are available to monitor resource utilization. It also provides information on keeping accurate time in virtual machine environments to ensure accurate benchmarking and performance measurements.

VMware Performance Tools

VMware products include a number of tools to monitor the resource utilization of both virtual machines and the underlying physical server machines when running with ESX Server. Many customers also run system management agents such as Dell OpenManage, HP Insight Manager, or IBM Director, to monitor their virtual machines. This section provides a brief overview of each of the VMware performance tools and includes references to additional documentation.

VMware VirtualCenter allows administrators to monitor and control groups of VMware servers and virtual machines from a single management console. It shows resource utilization summaries and also provides historical graphs for CPU, memory, networking, and disk resources. The majority of customers running ESX Server today use VirtualCenter to manage their virtual infrastructure. For more information on VirtualCenter, refer to the *VirtualCenter User's Manual*.

The ESX Server Management User Interface (MUI) displays simple summaries of physical and virtual machine performance statistics, but does not provide historical graphs. More information on the ESX Server MUI can be found in the *ESX Server Administration Guide*.

The command-line tools `esxtop` and `vmkusage` come bundled with ESX Server. The `esxtop` tool provides a real-time view of CPU and memory utilization (updated every five seconds by default) for each virtual machine as well as the service console and certain VMkernel system services. It also shows CPU utilization per physical processor, memory utilization, and disk and networking bandwidth for each physical disk and network device available to virtual machines. The `vmkusage` tool shows historical graphs of resource utilization for a single physical host running ESX Server and its associated virtual machines. For more information on `esxtop`, see the *Using esxtop to Troubleshoot Performance Problems* technical note. For `vmkusage`, see the *Using vmkusage to Isolate Performance Problems* technical note.

CPU and memory are typically the most important resources to monitor for an ESX Server machine (or system). For CPU utilization, many VMware customers consider average CPU utilization in the range of 60% to 80% for the ESX Server to be a reasonable level, though different organizations may have their own standards as to how close to maximum capacity they want to run their servers. (Sufficient CPU resources must also be made available to handle peaks in workload demand.)

ESX Server allows memory to be over-committed, as described in Memory Management on page 14. In many cases, actual physical memory utilization is less than the maximum memory requested by all virtual machines. However, when ESX Server becomes strained for memory, it reclaims memory from virtual machines using ballooning or swapping, which can cause performance to degrade.

Keeping Time in Virtual Machines

Within a virtual machine, it is important that the guest operating system keep time accurately, for obvious reasons, but also because many performance monitoring applications and

benchmarks are timed from within the virtual machine. If time measured in a virtual machine were to pass more slowly than real time, then applications within the virtual machine would appear to run more quickly. That is, if the virtual machine runs for ten minutes in real time, but only reports that five minutes have elapsed, then an application would appear to have completed work in only five minutes, instead of the ten minutes it actually took. Since many systems and applications have to perform to service level agreements (SLA), such as executing a certain number of transactions per second, it is important, to provide accurate time-keeping within virtual machines to determine if SLAs are being met.

Most operating systems keep track of time by counting timer interrupts from timer devices. Server machines typically have several different hardware timers, so the operating system sets up one or more of these timers to generate periodic interrupts at a fixed rate. Common timer interrupt rates are 100 or 1000 interrupts per second. The operating system then keeps a running count of the number of interrupts it has actually handled to calculate the time.

In a virtual machine, the timer devices operate in the same way in that they generate interrupts that are processed by the guest operating system. However, since virtual machines share the underlying processors, a particular virtual machine may not actually be running at any specific time. If a virtual machine is not running when a timer interrupt comes in from a timer device, ESX Server attempts to schedule the virtual machine so that it can accept the interrupt. However, by the time the virtual machine runs again and checks for interrupts, it may have fallen behind real time or even missed several timer interrupts. To keep time accurate in such cases, ESX Server delivers virtual timer interrupts at a higher rate so that the virtual machine's time can catch up with real time. If the backlog of interrupts is over 60 seconds, then ESX Server drops the undelivered interrupts to avoid running for too long at the higher timer interrupt rate. The VMware Tools also synchronizes the virtual machine's time to the underlying server's time once every minute.

ESX Server's virtual time implementation is able to keep time accurately overall, but there may be minor time inaccuracies within small time periods. The amount of the inaccuracy depends on the overall CPU load on the server machine. If the server is lightly loaded, the virtual machines are able to process their timer interrupts without delay; if the server is heavily loaded, virtual machines may fall behind real time briefly, until the guest time is synchronized again with VMware Tools. To keep time as accurate as possible, we recommend enabling VMware Tools time synchronization. For more information, see the Timekeeping in VMware Virtual Machines white paper also available on VMware's web site.

Guest Operating System Performance Tools

It is functionally possible to run performance tools from inside a guest operating system running in a virtual machine. For example, Windows tools such as `perfmon` and Task Manager are well known and widely used on physical machine platforms. However, the CPU utilization reported by such tools for applications running inside the virtual machine may not accurately reflect the actual physical CPU utilization for that application, especially when the virtual machine is sharing the underlying processor. Such tools assume that the guest operating system is the only operating system running and does not take in account situations in which the underlying physical CPU is shared.

Operating systems typically compute per-process CPU utilization in the following way. When the operating system fields a timer interrupt, it increments a counter for the currently running process. It keeps track of other attributes as well, such as whether the process is running user code or privileged code. The estimated CPU utilization for each process is then calculated as the interrupt count for the process times the interrupt interval.

To understand why inaccuracies arise, consider the case of two CPU-intensive uniprocessor virtual machines running on a two-way server machine. If the two virtual machines, A and B, are tied to physical CPU 1, each of the virtual machines receive only 50% of physical CPU 1's time. If a timer interrupt comes in for virtual machine A while virtual machine B is running, virtual machine A is re-scheduled to take the interrupt. The process in virtual machine A that is currently running is charged for the entire interval (for example, 10 milliseconds for a rate of 100 interrupts per second). In this example, the guest operating system tools show the application running inside virtual machine A taking 100% of the CPU, when it actually used only 50% of a single processor, or 25% of the underlying two-way server machine's total capacity.

If the virtual machines have some amount of idle time, as is typically the case, then guest-reported CPU utilization can either be over-reported or under-reported. The guest CPU utilization is attributed either to a process or to system idle time, depending on whether a process or the idle loop happens to be running when the virtual machine is scheduled. As described in Keeping Time in Virtual Machines on page 19, if virtual machine A's time falls behind, then ESX Server increases the timer interrupt rate, or in some cases, drops the backlog of timer interrupts. This is necessary to keep accurate time, although it can result in further inaccurate measurements in guest-reported CPU utilization.

As a result, platform-based performance tools are the most accurate way to assess CPU utilization and collect other timer-based metrics. Other metrics in guest tools, such as network packets and disk I/O statistics, do accurately reflect the guest operating system's counts of these events, although they may not necessarily match the number of I/O operations performed on the underlying physical device.

Conclusion

ESX Server 2 is designed to help you deploy and manage scalable, high-performance, virtual machine platforms for mission-critical environments. It can be used to effectively run many virtual machines on a single physical server. Performance of individual virtual machines varies depending on the applications it is running and the guest operating systems deployed. There are a number of configuration options provided by ESX Server that can be used to tune various components of the system, including CPU, memory, networking, and storage. Consult the *ESX Server 2 Administration Guide* for specific configuration option and component tuning details.

Although much of the discussion in this paper centers on resource sharing and the associated overhead, in practice, most workloads perform at desired performance levels without saturating server resources. For robust large scale deployments, it is necessary to understand the performance metrics of the specific applications being consolidated (such as the number of transactions per second or the number of concurrent users the system must be able to handle). Then, after you create the virtualized system environment, performance of the applications running on virtual machines in the consolidated environment needs to be measured to make sure it meets production requirements.

VMware ESX Server is widely used to optimize IT infrastructure by partitioning industry-standard servers into virtual machines for use by Windows and Linux applications that require isolation. ESX Server allocates shares of the physical server resources to each virtual machine, and some of the physical server resources have to be reserved for the ESX Server software. When consolidated with ESX Server, workloads that don't fully utilize the resources of their respective physical systems to achieve target application throughput show great improvement in the server environment's total cost of ownership and can then take advantage of the other strategic benefits of deploying a virtual infrastructure.

vmware®

Glossary

Active Memory: Memory recently used by a virtual machine, an estimate of its working set size.

Direct Execution: A mode in which the virtual machine monitor can run the virtual machine's code directly on the underlying processor. Code that runs in direct execution has negligible virtualization overhead.

Granted Memory: Memory allocated to a virtual machine.

Guest Operating System, or Guest: The operating system running within a virtual machine.

Guest Device Driver: A device driver loaded into the guest operating system which controls the virtual devices of the virtual machine.

Hyperthreading: Processor architecture feature that allows a single processor to execute multiple independent threads simultaneously. Hyperthreading was added to Intel's Xeon and Pentium IV processors. Intel uses the term package to refer to the entire chip, and logical processor to refer to each hardware thread.

Logical CPU: A hardware thread on a hyperthreaded processor.

Machine Pages: The physical memory pages on the physical machine running ESX Server.

Native Execution: Execution of an application directly on a physical server, as contrasted with running the application in a virtual machine.

Non-Uniform Memory Access (NUMA) Architecture: A server architecture that consists of multiple nodes, where each node contains a small number of processors connected to main memory on the same system bus. Nodes are then connected to other nodes via a high-speed interconnect. Processors can access memory on any node, however accesses to memory on the same node (local memory) are much faster than accesses to memory on other nodes (remote memory). Thus the memory access times are non-uniform, depending on the location of the memory.

Physical CPU: A processor on a physical server. A single physical CPU with hyperthreading can contain multiple logical CPUs.

Raw Device Mapping (RDM): An RDM is a special file on a VMFS-2 volume that acts as a proxy for a raw device. It allows management and access of raw SCSI disks or LUNs as VMFS files.

Reserved Memory: Memory committed for guaranteed allocations to existing virtual machines.

Service Console: The service console boots the systems and runs support, management, and administration applications.

Shadow Page Table: A map between the guest operating system's virtual memory pages and the underlying physical machine pages. The shadow page tables are maintained by ESX Server and are used to efficiently virtualize memory accesses.

SMP (Symmetric Multi-Processing) virtual machine: An SMP virtual machine is a virtual machine with more than one virtual CPU.

Unused Memory: Memory that has not yet been allocated to a virtual machine.

Virtual CPU: A processor within a virtual machine. ESX Server 2 currently supports up to two virtual processors per virtual machine.

Virtualization Overhead: The cost of running an application within a virtual machine as compared to running the same application natively. Since running in a virtual machine requires

an extra layer of software, there is by necessity an associated cost. This cost may be in terms of additional resource utilization or performance.

VMkernel: The part of ESX Server that controls and manages the physical resources of the server. The VMkernel implements the virtualization, resource management, and hardware interface components of ESX Server.

VMkernel Device Driver: A device driver loaded into the VMkernel which controls the physical devices on the server.

Virtual Machine Monitor (VMM): The part of ESX Server that implements the x86 CPU virtualization.

Virtual SMP: The version of ESX Server that supports multiprocessor virtual machines. Currently ESX Server supports up to two-way virtual machines.

Additional Resources

VMware ESX Server 2.5 Installation Guide. VMware, Inc. (2004)

VMware ESX Server 2.5 Administration Guide. VMware, Inc. (2004)

VMware VirtualCenter 1.2 User's Manual (2004)

ESX Server 2: NUMA Support: http://www.vmware.com/pdf/esx21_numa.pdf

Hyperthreading Support in ESX Server 2.1: http://www.vmware.com/pdf/esx21_hyperthreading.pdf

ESX Server Raw Device Mapping: http://www.vmware.com/pdf/esx/esx25_rawdevicemapping.pdf

Networking Throughput in a Virtual Infrastructure: http://www.vmware.com/pdf/esx_network_planning.pdf

Using esxtop to Troubleshoot Performance Problems: http://www.vmware.com/pdf/esx2_using_esxtop.pdf

Using vmkusage to Troubleshoot Performance Problems: http://www.vmware.com/pdf/esx2_using_vmkusage.pdf

Comparing the MUI, VirtualCenter, and vmkusage: http://www.vmware.com/pdf/mui_vmkusage2.pdf

Cpu(8), Mem(8), Numa(8), and Hyperthreading(8) man page

VMware, Inc. 3145 Porter Drive Palo Alto, CA 94304 www.vmware.com