



## VMware ESX Server 2

# ESX Server Performance and Resource Management for CPU-Intensive Workloads

VMware® ESX Server® 2 provides a robust, scalable virtualization framework for consolidating multiple systems onto a single hardware platform. By default, machine resources are shared equally among the multiple virtual systems. In addition customers can tailor virtual machine configurations to allocate CPU and system resources based on various environmental, application, and workload factors.

This white paper describes the methods available for allocating and managing resources among ESX Server virtual machines. The paper then examines benchmark results run with the system configurations to illustrate the effects and benefits of using these different scaling and resource allocation methods. The benchmark tests used to measure virtual machine performance are based on a subset of a standard well-known benchmark test suite available from SPEC® (Standard Performance Evaluation Corporation).

This white paper covers the following topics:

- [Introduction](#)
- [Virtual Machine Scalability](#)
- [Virtual Versus Native Scaling](#)
- [Limiting CPU Resources via Maximums](#)
- [Guaranteeing CPU Resources via Minimums](#)
- [Allocating CPU Resources via Shares](#)
- [Setting CPU Affinity](#)
- [Page Sharing](#)
- [Conclusions](#)

This white paper is intended for partners, resellers, and system administrators who are beginning to deploy VMware products and need to understand the implications that various virtual machine scaling and configuration options have on performance and throughput. The benchmark examples and accompanying descriptions and analyses should help both administrators and users understand the impact of virtualization on system and application performance.



## Introduction

VMware ESX Server provides a thin software layer that enables users to create multiple independent virtual machines on the same physical server. By default, physical resources are shared among virtual machines with reasonable efficiency and balance. As the overall system load increases and the hardware resources become overcommitted, ESX Server continues to share and balance resource usage without significantly impacting the total system throughput. The behavior is similar to that of a heavily-loaded system natively running a single operating system instance. In both cases, adding more load does not increase throughput once the system is saturated (that is, when resources are fully utilized).

In many typical customer environments, applications and the underlying physical and virtual machines must meet certain quality-of-service and performance guarantees. ESX Server makes it easier to satisfy these guarantees by providing a comprehensive set of resource-allocation mechanisms, including setting maximum and minimum allowable resource usage, setting up proportional-share resource allocation, and specifying processor affinity for certain virtual machines. ESX Server can also exploit opportunities to reduce overall memory usage by sharing identical physical pages accessed by different virtual machines. Analysis of the experimental benchmark test results included in this paper show the effects of the various resource allocation mechanisms.

## Benchmark Methodology

The virtual resource management concepts and mechanisms explored in this paper pertain primarily to CPU resources and thus require benchmark experiments using CPU-intensive workloads that consume 100 percent of available CPU resources.

The integer component of the SPEC CPU2000 benchmark suite is comprised of a well-known set of CPU-intensive applications. A single application from this group, the GZIP data compression application, was chosen for use in the benchmark tests described in this paper. The SPEC version of the GZIP application benchmark test performs no file I/O other than reading the input. All compression and decompression is performed entirely in memory. The purpose of this study is not to generate results for competitive evaluation or comparison, but to demonstrate the behavior of ESX Server under heavy CPU loads using different CPU resource allocation settings.

**Note:** More information on the SPEC CPU2000 benchmark, and specifically the GZIP application benchmark test, is available on the SPEC web site at [www.spec.org/cpu2000](http://www.spec.org/cpu2000).

No attempt was made to optimize the benchmark test results in any way. Default compilers, tools, and settings were used in all cases. While an experimental benchmark setup based solely upon a single CPU-intensive workload driving a physical system into and past saturation is not a likely customer scenario, it is useful for illustrating ESX Server performance and management.

For each benchmark test run, only the active virtual machines were powered on, since idling virtual machines continue to consume a small amount of resources and can skew results. For each benchmark measurement point, five GZIP application iterations were run within the SPEC benchmark test harness. Each of these measurement points was then repeated five times to ensure repeatability and consistency of results.



Timing for the benchmark tests was keyed to the processor's physical time stamp counter (TSC), rather than the guest's virtual TSC. The TSC is a 64-bit number held in a physical register that is incremented on each clock cycle, so it provides precise timing. Total elapsed time (in seconds) can be determined by taking the difference between the TSC's start and end values, then adjusting the number by the system clock frequency (see Reference 1 for more information on timekeeping in VMware virtual machines).

Overall benchmark performance is measured both by the average run time of the GZIP application and by the overall system throughput. (Run time is the elapsed time it takes to run and complete one instance of the GZIP application execution in an individual benchmark test.) Average run time is presented for each resource-allocation class when resources are allocated unequally. Overall system throughput is computed as the number of GZIP iterations completed per hour.

### **Benchmark System Configuration and Setup**

The system used to run all benchmark tests was an HP ProLiant DL580 server containing four 2.2GHz Intel® Xeon® CPUs with hyper-threading support. Intel hyper-threading technology allows a single processor to execute two independent threads simultaneously. A single processor with hyper-threading does not match the performance of a two-processor system, but can lead to greater throughput for some workload types. In fact, hyper-threading technology should be expected to realize its greatest benefit on CPU-intensive workloads such as the GZIP application. To simplify analysis, all benchmark tests were run with hyper-threading disabled, except those tests which were run to demonstrate the effects of hyper-threading itself. (See Reference 2 for more information about hyper-threading support in VMware ESX Server.) The benchmark test system was configured with 16GB of memory and approximately 140GB of internal SCSI disk storage (two 70GB disks striped using RAID-0). Disks were accessed using the VMFS persistent mode.

All virtual machine tests were run using ESX Server version 2.5.1. One master virtual machine was created with Red Hat® Enterprise Linux® 3 (2.4.21 kernel and gcc 3.2.3) and cloned 16 times using VMware Virtual Center. Each virtual machine was configured with 256MB of memory. For each benchmark test, the appropriate number of virtual machines was powered on. Each virtual machine then ran a single instance of the SPEC CPU2000 GZIP application benchmark test. Resource settings were managed using the Resources window tab inside each virtual machine's Virtual Machine Properties window. Alternatively, resources can be controlled via the ESX Server management interface. All tests were controlled from a Red Hat Enterprise Linux 3 virtual machine on a second system. This controlling system was used only to initiate each benchmark test and then archive the results.

For native operating system benchmark tests, all data was captured using Red Hat Enterprise Linux 3 (2.4.21 kernel and gcc 3.2.3). These tests had access to the full 16GB of system memory. In this case, multiple instances of the GZIP benchmark were run simultaneously using the same experimental benchmark setup as the virtual machine tests. As was the case with the virtual machine benchmarks, the hardware TSC was used to measure elapsed time.



## Benchmark Test Cases

This white paper provides benchmark test results illustrating the behavior of virtual machine performance as you scale the number of virtual machines and also choose various CPU resource allocation methods. Table 1 provides a summary of the benchmark test cases included in this paper.

**Table 1 Summary of Benchmark Test Cases**

Test Cases	Description
<b>Scalability</b>	
Virtual Machine Scalability	System throughput measured as the number of virtual machines is increased; tests were performed with hyper-threading enabled and disabled.
Virtual Versus Native Scaling	System throughput compared for native versus virtual machine execution of workloads to see the effects of virtualization overhead; tests were performed with hyper-threading enabled and disabled.
<b>Virtual Machine Resource Allocation</b>	
Limiting CPU Resources via Maximums	System throughput measured with maximum CPU resource usage limits placed on virtual machines.
Guaranteeing CPU Resources via Minimums	System throughput measured with CPU resource usage guarantees set for virtual machines.
Allocating CPU Resources via Shares	System throughput measured with CPU resource usage for individual virtual machines controlled by proportional share allocation.
Setting CPU Affinity	Benchmark run times measured with virtual machines pinned to specific physical processors versus default resource allocation.
<b>Page Sharing</b>	
Virtual Machine Page Sharing	Benchmark run times and memory usage compared with page sharing enabled and disabled.

The following sections detail each of the benchmark tests, describe the type of system scalability feature being tested, and describe what behavior might be observed based on the workload and type of resource allocation methods used. Analysis of the results following execution of each benchmark test show and explain the behavior of the resource allocation method and its effect on system throughput and performance.



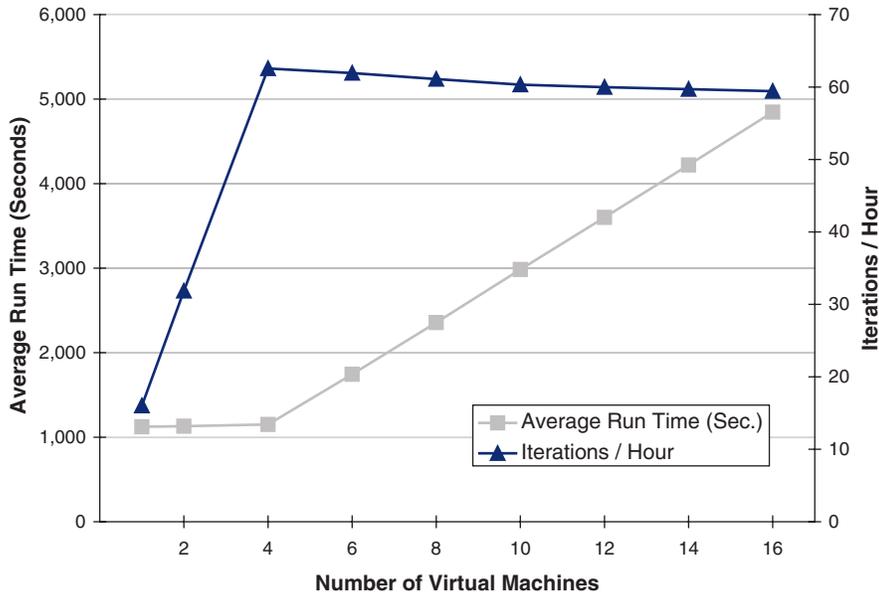
## Virtual Machine Scalability

For many typical CPU-intensive workloads that saturate a single CPU, the overall system throughput should increase as the number of virtual machines increases until the number of virtual machines exceeds the number of physical CPUs (or, when hyper-threading is enabled, exceeds twice the number of hyperthreaded CPU packages). Past this point, the overall system throughput should plateau while the run time of each individual virtual machine increases, due to each virtual machine's diminished share of the fixed physical resources. To demonstrate this behavior, the GZIP benchmark was run with 1, 2, 4, 6, 8, 10, 12, 14, and 16 virtual machines. To demonstrate the potential performance impact of hyper-threading, benchmark tests were run with hyper-threading both disabled and enabled.

### Hyper-Threading Disabled

With hyper-threading disabled, the physical system behaves like a traditional four-CPU server. The results are shown in Figure 1. The average run times remain almost constant for the one, two, and four virtual machine tests, since each additional virtual machine simply consumes another unused CPU. However, there is approximately a two percent increase in run time as the number of virtual machines increases from two to four. This increase can be attributed to ESX Server's service console, which provides services required by the virtual machines and consumes a small amount of resources. (Service console resource usage is proportional to the number of active virtual machines and grows as the number of virtual machines on the system increases.) At this point, the four active virtual machines must share the fully committed resources with the service console. As additional virtual machines are added, the run times increase almost linearly. For example, average run time for four virtual machines is 1151 seconds. For 16 virtual machines, the run time is 4846 seconds, or 4.21 times longer. In other words, adding four times the load to the server incurs five percent additional overhead due to virtualization. This result should be expected, since ESX Server must do additional work for each virtual machine added. However, that cost is relatively low, even for 16 virtual machines.

Table 2 lists the average run times of the GZIP benchmark test as the number of virtual machines increases, as well as the maximum and minimum run times observed for each number of virtual machines tested. The range of run times recorded for individual virtual machines should be narrow if the ESX Server scheduler is dividing the resources equally among all virtual machines. ESX Server exhibits this characteristic narrow range for the GZIP benchmark, which verifies its consistent behavior under both lightly and heavily loaded conditions.



**Figure 1 Performance Vs. Number of Virtual Machines (Hyper-Threading Disabled)**

Figure 1 also shows the equally important gauge of overall throughput as measured in benchmark iterations per hour. As the load is increased from one to four virtual machines, the throughput increases almost linearly from 16 iterations per hour to 62.6 iterations per hour. Once all processors are fully utilized, the overall throughput does not increase, but remains relatively flat, dropping slightly as each additional virtual machine is added. This gauge more clearly illustrates the overhead of managing additional virtual machines.

**Table 2 Benchmark Run Times With Increasing Number of Virtual Machines (Hyper-Threading Disabled)**

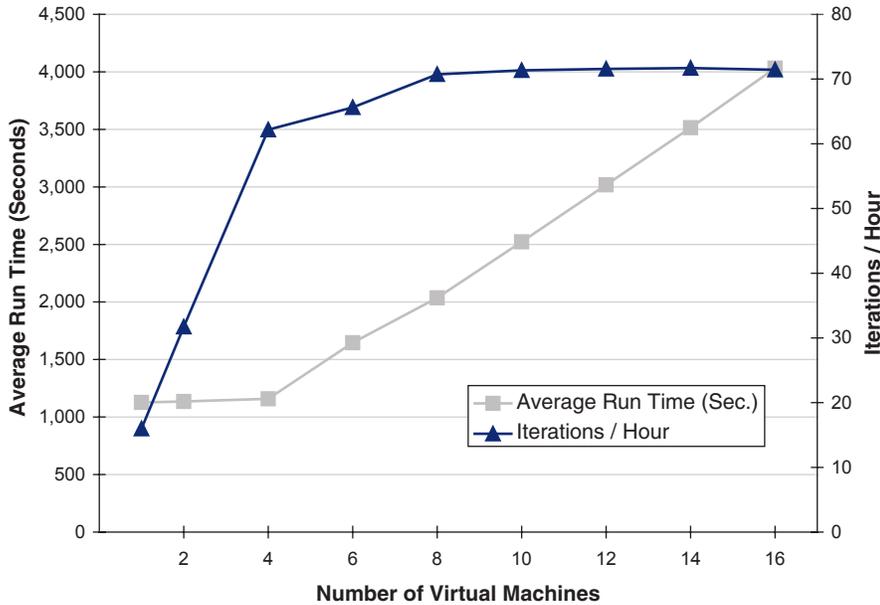
Number of Virtual Machines	Average Run Time (in Seconds)	Maximum Run Time (in Seconds)	Minimum Run Time (in Seconds)	Percent (%) Difference Between Max and Min
1	1124	1127	1119	0.7
2	1130	1138	1126	1.1
4	1151	1156	1144	1.0
8	2357	2367	2342	1.1
16	4846	4877	4785	1.9

### Hyper-Threading Enabled

The same virtual machine scaling tests were run with hyper-threading enabled. Figure 2 plots the GZIP benchmark test results. While the number of virtual machines does not exceed the number of physical processors (four), hyper-threading provides no additional benefit. As expected, the results with one to four virtual machines are similar to those without hyper-threading. Hyper-threading begins to demonstrate benefits when the number of virtual machines increases past the number of physical processors. The throughput does not begin to



peak until eight virtual machines are used, providing two threads of execution for each physical processor. As the number of virtual machines continues to increase, the throughput increases marginally until reaching a peak at 14 virtual machines, and then beginning to drop when 16 virtual machines are deployed.



**Figure 2 Performance Vs. Number of Virtual Machines (Hyper-Threading Enabled)**

Table 3 lists the average, maximum, and minimum run times for benchmark tests as the number of virtual machines increases (with hyper-threading enabled). As in the hyper-threading-disabled case, ESX Server allocates resources equally among the virtual machines, as shown by the narrow range between the maximum and minimum run times.

**Table 3 Benchmark Run Times With Increasing Number of Virtual Machines (Hyper-Threading Enabled)**

Number of Virtual Machines	Average Run Time (in Seconds)	Maximum Run Time (in Seconds)	Minimum Run Time (in Seconds)	Percent (%) Difference Between MAX and MIN
1	1127	1129	1126	0.3
2	1135	1139	1129	0.9
4	1159	1166	1149	1.5
8	2036	2048	2020	1.4
16	4032	4099	3984	2.8

Figure 3 shows the relative performance benefit of enabling hyper-threading for CPU-intensive workloads. As expected, no benefit is gained until the number of virtual machines exceeds the number of physical processors. When the system is saturated, hyper-threading increases throughput by about 20 percent over the non-hyper-threading case by utilizing each processor more effectively. However, the benefit of hyper-threading technology varies significantly with



workload and cannot be easily generalized to other workloads. In particular, applications with more networking, disk I/O, or operating system calls will likely achieve less dramatic results.

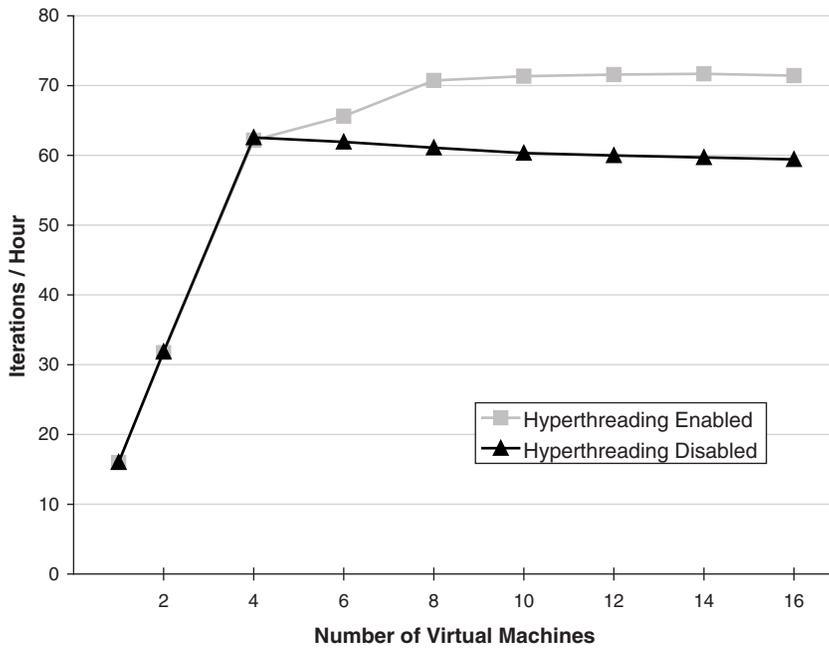


Figure 3 Comparison of Run Time Performance With and Without Hyper-Threading

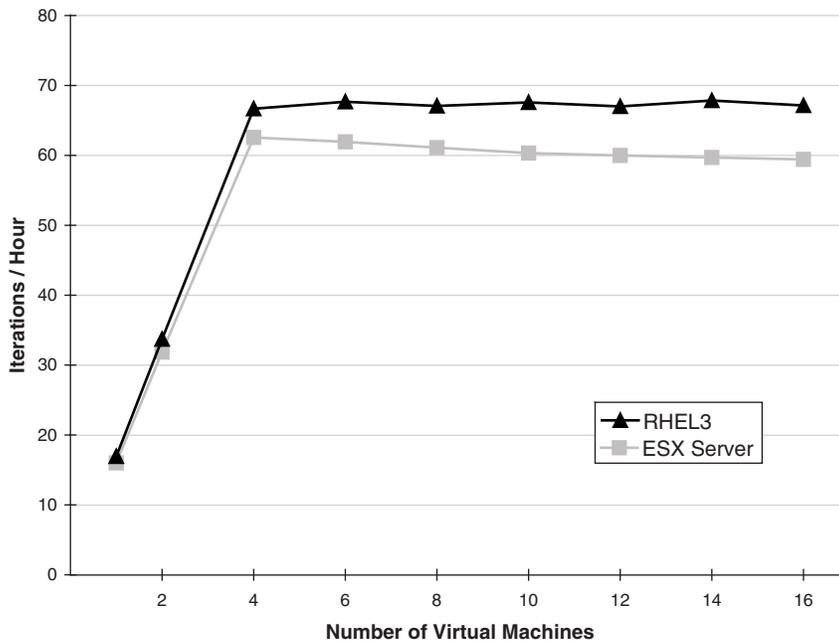


## Virtual Versus Native Scaling

A throughput comparison between applications run in a virtualized environment and applications run natively on the same hardware should show that the virtual environment scales in the same fashion as the native environment and should also quantify the relatively small virtualization overhead of CPU-intensive applications such as GZIP. A set of benchmark tests using a single natively-run operating system was performed using the same physical system. As before, individual tests were run both with and without hyper-threading enabled.

### Hyper-Threading Disabled

Figure 4 plots throughput results comparing native and virtual environment benchmark test execution with hyper-threading disabled. For one or two instances of the GZIP benchmark, ESX Server achieves 95 percent of the native environment's throughput, which indicates a measure of the virtualization overhead when there is little or no contention for system resources. With four instances of the benchmark running, ESX Server drops to 94 percent of the native environment's throughput as a result of the demands of the service console in the now fully utilized system. Once the native environment becomes saturated, throughput remains fairly constant as more benchmark instances are added. Throughput in the virtual environment decreases slightly as more benchmark instances are added. This behavior can be expected since each benchmark instance not only incurs virtualization and management overhead, but also includes the overhead of running an additional operating system instance for each benchmark instance, while the native benchmark tests use only a single operating system instance. As a result, the virtual environment achieves 89 percent of the performance of the native environment when running 16 benchmark instances. Note that it is unlikely that servers would run with such heavy workloads in real-world scenarios.



**Figure 4 Comparison of Run Time Performance Between Virtual and Native Environments (Hyper-Threading Disabled)**



### Hyper-Threading Enabled

The comparison of virtual and native environments with hyper-threading enabled is shown in Figure 5. For a single benchmark instance, the virtual environment again sustains 95 percent of the native environment's throughput. Surprisingly, the virtual environment achieves 5 percent and 15 percent higher throughput than the native environment running two and four benchmark instances, respectively. This is primarily due to the behavior of the Red Hat Enterprise Linux 3 scheduler. When hyper-threading is enabled optimal performance is obtained when one thread is scheduled on each processor prior to a second thread being scheduled on any processor. The Red Hat Enterprise Linux 3 scheduler does not follow this guideline and can schedule two threads on a single processor while other processors remain idle. (This behavior can be easily verified by watching the CPU field in the output of the `top` command while the benchmark is running.) On the other hand, ESX Server schedules threads across all processors to achieve more consistent and predictable results. (Note that the behavior of the Red Hat Linux 3 scheduler should not be generalized to other Linux releases.) As the number of benchmark threads increases to 16 and the system becomes fully utilized, the results look more like those you might expect. With 16 benchmark threads, the virtual environment has 13 percent lower throughput than the native environment due to the virtualization overhead.

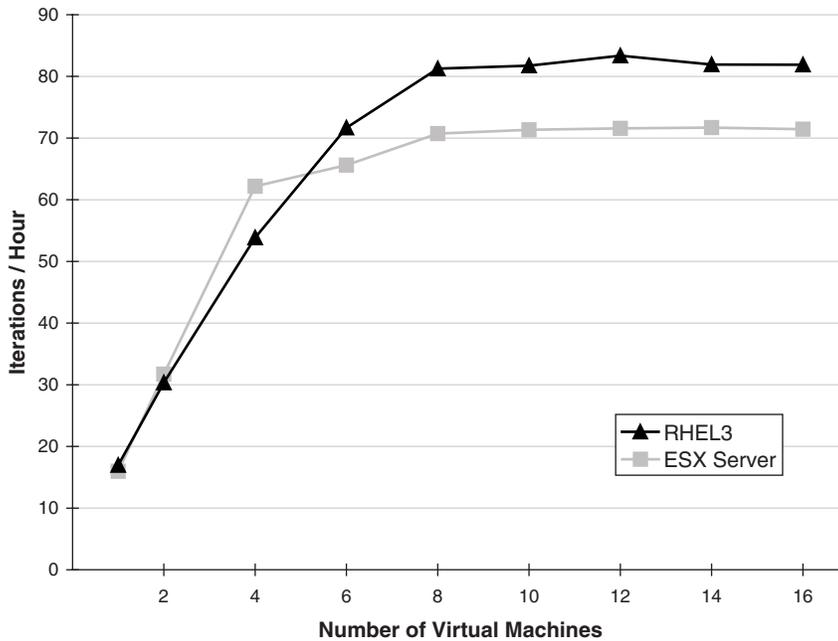


Figure 5 Comparison of Run Time Performance Between Virtual and Native Environments (Hyper-Threading Enabled)



## Limiting CPU Resources via Maximums

One method of providing more consistent and predictable behavior is to limit the total resources a virtual machine may consume. ESX Server can limit the maximum amount of CPU resources a virtual machine may consume. This approach makes it easier to keep quality-of-service guarantees (even as additional virtual machines are powered on) by maintaining more predictable and consistent CPU usage levels. For example, on a four-CPU system, an administrator might set the maximum CPU usage for a CPU-intensive workload to 25 percent or 50 percent, to provide predictable response times and throughputs for up to 16 or 8 virtual machines, respectively.

Limiting each virtual machine to 25 percent or 50 percent of a CPU's resources also provides insight into ESX Server's behavior when running lightly loaded virtual machines, as is more typical in customer environments. The effects of these two CPU usage settings (25 percent and 50 percent) are shown in Figure 6. Limiting each virtual machine to 25 percent of a CPU provides a nearly linear throughput-scaling curve until the system is fully loaded with 16 virtual machines. In essence, each virtual machine is allocated the share of the system it would receive with sixteen active virtual machines, regardless of how many virtual machines are actually active. The maximum throughput is not reached as quickly due to idling resources, but the behavior of individual virtual machines is much more predictable and each virtual machine's throughput remains constant as the overall load increases. When each virtual machine is allocated 50 percent of a CPU, the maximum throughput is achieved with eight virtual machines and then matches the default scaling behavior up to 16 virtual machines.

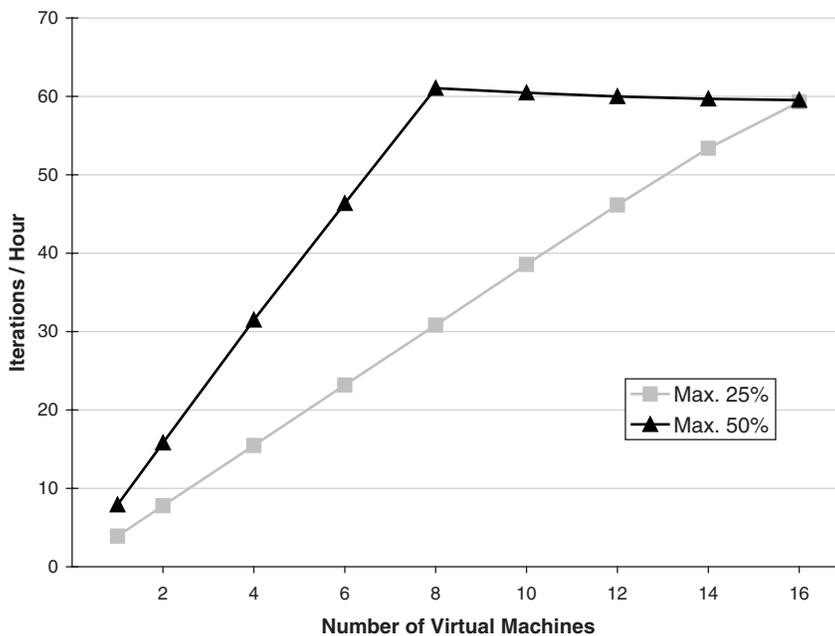


Figure 6 Effect of Setting CPU Usage Maximums on Virtual Machine Performance



## Guaranteeing CPU Resources via Minimums

Various types of workloads require guaranteed amounts of resources to meet response time or throughput requirements. ESX Server can be configured to assure that virtual machines running those applications receive sufficient CPU resources. When these virtual machines do not consume their entire resource allocations, other virtual machines may be allowed to use those resources. If ESX Server cannot reserve sufficient resources required by a virtual machine due to existing commitments, the admission-control mechanism prevents the virtual machine from powering on. This limits the number of high-priority virtual machines that can be run by a server when using ESX Server's minimum resource settings.

Figure 7 shows a scheduling example with one of eight virtual machines elevated to high priority for three consecutive time slices. The high-priority virtual machine (VM1) is scheduled for every time slice, while the remaining seven virtual machines share the remaining resources among themselves, and occasionally with the service console (labelled COS in Figure 7).

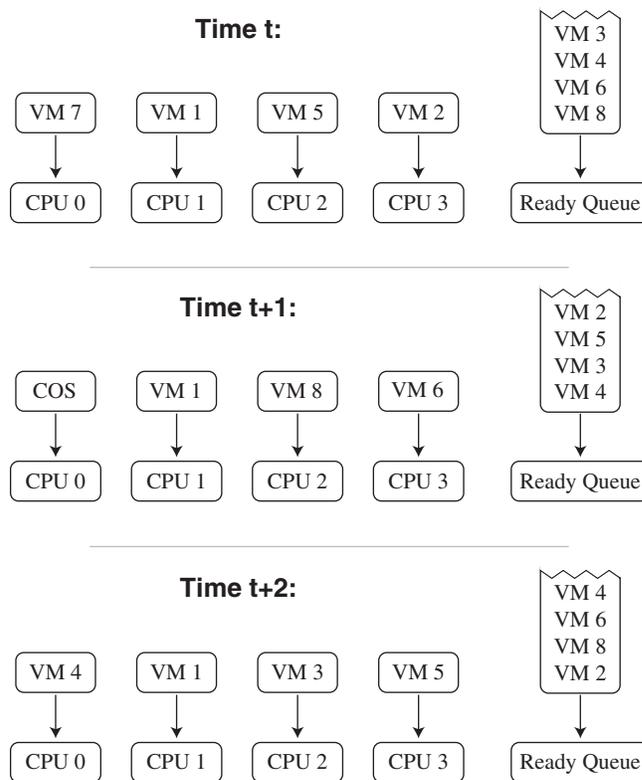


Figure 7 Virtual Machine Priority Assignments in Three Consecutive Time Slices



To test the effects of the ESX Server minimum resource setting, some virtual machines running the GZIP benchmark were elevated to high priority and guaranteed 100 percent of a CPU's resources. The GZIP benchmark tests were run with 8 and 16 active virtual machines, respectively, and one, two, and three virtual machines elevated to high priority. (On a four-CPU test system, no more than three virtual machines can be set to operate in this high-priority 100 percent class, due to other resource commitments.)

Table 4 shows the run times of the two resource allocation classes for each of the benchmark tests. With a single virtual machine allocated 100 percent CPU out of eight total active virtual machines, the high-priority virtual machine completes the GZIP benchmark in 1141 seconds, roughly the same time as the 1124 seconds taken by a single active virtual machine. The 2 percent increase in run time is due to the extra overhead created by the additional seven virtual machines. The seven remaining virtual machines complete the benchmark in 2360 seconds, which is comparable to the 2357-second run time when all eight virtual machines run with equal priority. Thus, despite the difference in individual run times between the two virtual machine classes, the total throughput is unchanged from the default virtual machine resource allocation case. When the high-priority virtual machine completes the benchmark and goes idle, ESX Server then allocates the extra resources to the remaining active virtual machines, allowing them to complete their remaining operations more quickly. However, if the high-priority virtual machines were to run continuously, the low-priority virtual machines would continue to run at their slower initial rate, lowering their throughput. The results for eight virtual machines do not change significantly when two or three virtual machines are given higher resource allocations. Likewise, the benchmark tests run with 16 virtual machines behave similarly, but with slightly higher virtualization overhead, as expected.

**Table 4 Effect of Setting CPU Usage Minimums on Virtual Machine Performance**

Number of Virtual Machines	Virtual Machines Set to 100% CPU Usage	Average High Priority Run Time (in Seconds)	Average Low Priority Run Time (in Seconds)
8	1	1141	2360
	2	1144	2352
	3	1152	2371
16	1	1192	4879
	2	1185	4923
	3	1189	4973



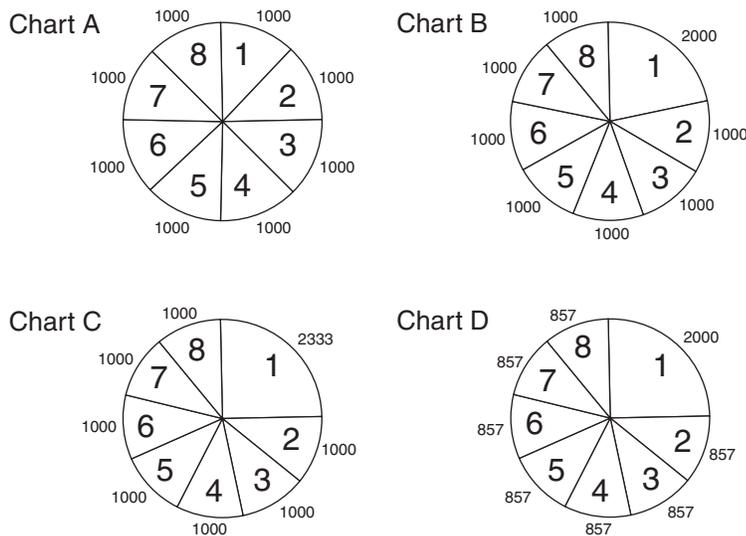
## Allocating CPU Resources via Shares

ESX Server also supports a proportional-share resource allocation mechanism that provides greater flexibility than the straightforward, yet fairly rigid, maximum and minimum resource allocation methods that are most commonly used. Using proportional-share resource allocation, each virtual machine is guaranteed a resource allocation equal to its fractional share of the total shares of CPU resources available. That is, shares represent relative resource rights that depend on the total number of shares of CPU resources available.

**Note:** Virtual machines proportionally benefit from extra resources when some allocations are underutilized. Also, resource allocations degrade gracefully in overload situations.

Adding virtual machines to a system or modifying share allocations within a system changes the total shares and each individual virtual machine's fractional allocation. Shares are used to allocate the resources available after the maximum and minimum allocations have been allotted. While proportional shares provide additional flexibility for resource allocation, the underlying subtlety of this feature can lead to non-intuitive behavior and poorly configured systems.

Attempting to recreate the behavior of the minimum CPU benchmark tests described in the previous section illustrates the complexity of proportional-share allocation. By default, all single-processor virtual machines receive an allocation of 1000 shares and have equal access to resources. With eight virtual machines running the GZIP benchmark, a total of 8000 shares (1000 shares per virtual machine) are automatically allocated and in active use. The relative share allocation for this case is shown in Figure 8 (Chart A). Note that the service console and other ESX Server applications hold some shares that are largely underutilized, so these are not represented in the share allocation shown in the figure.



**Figure 8** Effect of Assigning Proportional Shares on Virtual Machine Performance

To provision one virtual machine with 100 percent of a CPU, an inexperienced administrator might simply double its allocation to 2000 shares. However, in that case, the total number of active shares increases to 9000, reducing each virtual machine's fractional share of the total shares. As shown in Figure 8 (Chart B), the high-priority virtual machine now holds 22 percent of



the total shares (2000 of 9000), rather than 25 percent, and is therefore given only 88 percent of a CPU.

Table 5 shows that the run time of the high priority virtual machine is now 1296 seconds compared to 1141 seconds shown in Table 4 using the minimum 100 percent CPU usage setting. This is the result we would expect for a virtual machine with 88 percent of a CPU:

$$1141/0.88 = 1296$$

To correctly provision resources, we must account for the increase in the total number of shares. Either increasing the shares given to the high-priority virtual machine or reducing the shares of all low-priority virtual machines can satisfy the resource constraints. For the first approach, shown in Figure 8 (Chart C), the 7000 shares of the seven low-priority virtual machines should be limited to 75 percent of the system:

$$0.75 * (\text{total shares}) = 7000$$

$$\text{total shares} = 9333$$

Therefore, to achieve the desired balance of resource sharing, we allocate the remaining 2333 shares to the high-priority virtual machine while leaving the low-priority virtual machines with the default value. A second approach, shown in Figure 8 (Chart D), is to hold the total number of shares constant at 8000 by reducing the shares for the low-priority virtual machines. The seven low-priority virtual machines divide 6000 shares (75 percent of the total):

$$6000/7 = 857$$

Each low-priority virtual machine would therefore be allocated 857 shares.

This approach has the same effect as increasing the shares for the high-priority virtual machine. The results for these two share allocations, shown in Table 5, closely resemble the results using the 100 percent minimum CPU usage setting. The 1 percent increase in the high-priority run times is due to the service console utilizing a small fraction of its share allocation. Since this usage is small, it was neglected in the computations.

Table 5 also contains the benchmark test data for alternately correctly and incorrectly provisioning two high-priority virtual machines. Simply doubling the allocated shares for the high-priority virtual machines and neglecting the increase in total shares to 10,000 leads to greater share dilution and less accurate resource allocation. Determining the correct share total to allocate 50 percent of the total shares to the six low-priority virtual machines leads to correct results by allocating each high-priority virtual machine 3000 shares:

$$0.5 * (\text{total shares}) = 6000$$

$$\text{total shares} = 12,000$$

$$6000 \text{ shares} / 2 \text{ high-priority virtual machines} = 3000 \text{ shares per high-priority machine}$$



Finally, Table 5 shows the correct share allocations for three high-priority virtual machines.

**Table 5 Effect of Increasing Share Allocation on Benchmark Run Times**

Number of Preferred Virtual Machines	Number of Shares per Preferred Virtual Machine	Number of Shares per Regular Virtual Machine	High Priority Time (in Seconds)	Low Priority Time (in Seconds)
1	2000	1000	1296	2352
1	2333	1000	1157	2357
1	2000	857	1153	2350
2	2000	1000	1470	2363
2	3000	1000	1159	2359
3	5000	1000	1159	2360

Table 6 shows the correct share settings to mimic the CPU-minimum guarantees for one, two, and three high-priority virtual machines and 16 total virtual machines.

**Table 6 Adjusting Share Allocation to Mimic CPU Minimum Settings**

Number of Preferred Virtual Machines	Number of Shares per Preferred Virtual Machine	Number of Shares per Regular Virtual Machine	Fastest Time (in Seconds)	Slowest Time (in Seconds)
1	5000	1000	1214	4927
2	7000	1000	1226	4970
3	13,000	1000	1198	4964

For the examples in Table 6, the high-priority virtual machines had their share allocation increased while all other virtual machines were left at the default settings. The computations are left as an exercise to the reader. Notice that powering on eight additional virtual machines increases the total number of default shares from 8000 to 16,000.



## Setting CPU Affinity

ESX Server also supports processor affinity for pinning virtual machines to a subset of the physical processors. This can be useful for creating a fixed partitioning of the physical processors available. However, invoking CPU affinity can lead to performance problems by preventing the ESX Server scheduler from effectively load-balancing the overall system. Prior to ESX Server 2.0, affinities were particularly beneficial on NUMA systems, where keeping a virtual machine on a single NUMA node can improve performance by reducing memory latency. However, a heavily-loaded NUMA node with lower memory latency is likely outperformed by a lightly-loaded NUMA node with higher memory latency, so even using processor affinity in this case should be closely monitored, since load imbalances may occur. Users should also be aware that affinity settings are not portable across physical hosts. The scheduler in ESX Server 2.0 and subsequent releases, however, is NUMA-aware and properly schedules virtual machines across NUMA nodes. In fact, setting affinity for a virtual machine excludes it from NUMA optimizations, such as page migration. Affinity is still useful in certain limited situations, such as the segregation of two distinct organizations sharing the same physical hardware for administrative purposes. In many cases, mechanisms such as setting CPU minimums can be used to achieve the same management goals without constraining the ESX Server scheduler. For example, rather than using affinity, setting the CPU minimum to 100 percent effectively dedicates a processor to a virtual machine. In general, the decision to use affinity should be carefully considered and the resulting configuration carefully monitored.

Two different affinity configurations, all-on-one and round-robin, were tested using the GZIP benchmark and compared to the default resource allocation behavior. The first benchmark test pins all virtual machines to a single CPU, CPU 1. (CPU 0 was not chosen due to the presence of the service console.) While this configuration may not be entirely realistic, similar configurations can arise in dynamic environments where virtual machines become active or inactive at irregular intervals. As expected, the average run times shown in Table 7 are roughly four times the non-affinity run times. Clearly, leaving three out of four processors idle is not a wise use of system resources and is not recommended. When using affinity you must carefully balance and monitor workloads in order to avoid overcommitting some processors while other processors remain underutilized. In general, you can most often achieve better utilization by allowing the ESX Server scheduler the freedom to migrate virtual machines between processors as workloads change.

**Table 7 Effect of Processor Affinity Configuration on Benchmark Run Times**

Number of Virtual Machines	Default Average Time (in Seconds)	All-on-One Average Time (in Seconds)	Round-Robin Average Time (in Seconds)
4	1151	4654	1163
8	2357	9463	2356
16	4846	19457	4838



A more realistic example would be to spread the virtual machines equally among the processors using a round-robin scheme. While a more efficient use of processor resources such as the cache might be expected using this approach, Table 7 shows the average run time roughly equivalent to default resource allocation.

**Table 8 Maximum and Minimum Run Times Using Default and Round-Robin Affinity Resource Allocation**

Number of Virtual Machines	Default Maximum Time (in Seconds)	Default Minimum Time (in Seconds)	Round-Robin Maximum Time (in Seconds)	Round-Robin Minimum Time (in Seconds)
4	1156	1144	1174	1136
8	2367	2342	2437	2317
16	4877	4785	5217	4618

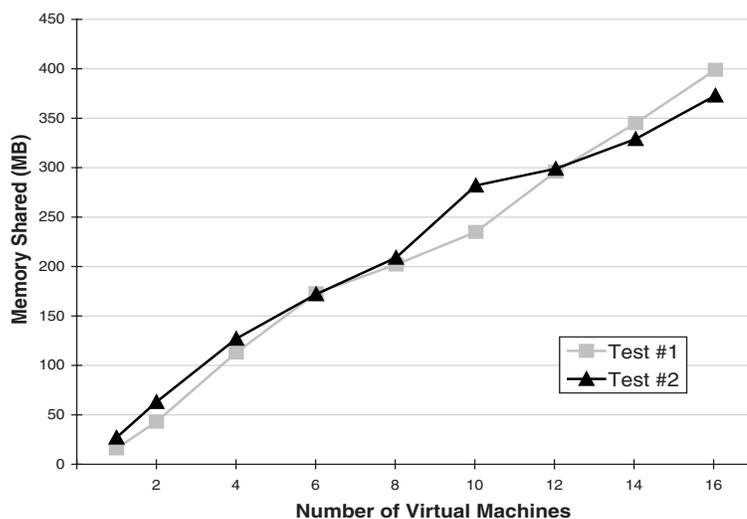
Further analysis shows that the individual run times are far less consistent when using affinity. Table 8 lists the maximum and minimum individual run times observed for both the default configuration and the round-robin affinity benchmark tests. The default benchmark tests show more consistent and predictable behavior; the maximum and minimum run times vary by only 1 percent for four and eight virtual machines, and 2 percent for sixteen virtual machines. In contrast, the round-robin affinity run times differ by 3 percent, 5 percent, and 12 percent for four, eight, and sixteen virtual machines, respectively. This variance is due to the interaction between the affinity settings and the scheduler logic defined in the underlying architecture of ESX Server, which pins the service console to CPU 0. Since the virtual machines pinned to CPU 0 must share resources with the service console, and the ESX Server scheduler cannot migrate these virtual machines to equalize the resource allocation among all virtual machines, these virtual machines run more slowly. As the total number of virtual machines increases, the overhead of the service console increases, widening the performance disparity among the remaining virtual machines. Allowing the scheduler the freedom to migrate virtual machines and evenly distribute resources maintains more equitable sharing in varying circumstances and workload environments.



## Page Sharing

A primary application of ESX Server is consolidating multiple servers onto a single physical server using virtual machines. This consolidation also creates numerous opportunities for sharing memory between virtual machines. For example, virtual machines may be running instances of the same guest operating system and applications that contain common data. ESX Server by default exploits these sharing opportunities, so that server workloads running in virtual machines on a single physical machine can often consume less memory than they would if they were running separately. ESX Server scans guest pages at random to search for potential matches and then consolidates memory by eliminating redundant copies. A copy-on-write mechanism is used when pages can no longer be shared due to page updates by a virtual machine. An in-depth discussion of memory resource management in ESX Server can be found in Reference 3.

Data on page sharing was collected during the benchmark tests performed for previous sections of this white paper. Figure 9 shows the amount of memory shared for two sets of benchmark runs. It might be unexpected, but some sharing is achieved even with only a single virtual machine. This is due to shared copies of redundant structures such as the zero page. As the number of virtual machines increases, the random selection of page candidates for sharing creates irregularities in the graph's curves and makes exact reproduction of results impossible.



**Figure 9** Shared Aggregate Memory Increases with Number of Virtual Machines

Neglecting the slight random variation, Figure 9 shows that the amount of aggregate memory shared increases by approximately 25MB per virtual machine in a roughly linear fashion. This translates into a savings of nearly 10 percent of each virtual machine's 256MB memory. (Each GZIP benchmark test's data set consumes about 200MB and is unlikely to be shared since it is actively being updated.)

**Note:** Results vary significantly with the mix of guest operating systems and applications.

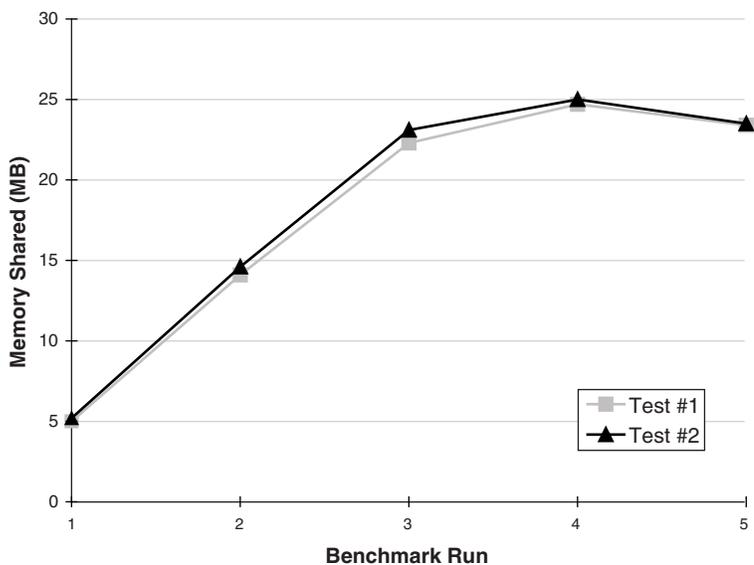
The memory savings provided by page sharing for the GZIP benchmark workload is achieved with no discernible performance consequence. Table 9 shows the average run times for a set of benchmark tests performed with page sharing disabled. The results are essentially identical to those captured with page sharing enabled.



**Table 9 Average Run Times for Benchmark Tests (With and Without Page Sharing)**

Number of Virtual Machines	Average Run Time without Page Sharing (Seconds)	Average Run Time with Page Sharing (Seconds)	Performance Impact
1	1121	1124	0.26%
2	1128	1130	0.17%
4	1153	1151	0.17%
6	1749	1744	0.29%
8	2355	2357	0.08%
10	2971	2984	0.44%
12	3600	3601	0.03%
14	4222	4221	-0.02%
16	4844	4846	0.04%

After a virtual machine is powered on, the page sharing mechanism begins to scan pages looking for sharing opportunities. Over time, the page sharing in a virtual machine should plateau. Figure 10 shows the page sharing ramp-up for two recently powered-on virtual machines after each of five sequential benchmark runs. (Each benchmark run lasts approximately 30 minutes.)



**Figure 10 Page Sharing Ramp-Up After Sequential Benchmark Test Runs**

The time required to reach a plateau depends upon the resource utilization of the system and the rate of shared-page invalidations due to writes, compared to the rate of discovery of shared pages. As Figure 10 shows, page sharing peaks during the fourth benchmark run, but has largely reached a plateau by the third run. Once a plateau has been reached, slight variations over time are to be expected due to some shared pages being invalidated by writes while new shared pages are found. The slight drop during the fifth run is a consequence of this effect.



## Conclusions

ESX Server provides a robust and powerful virtualization framework for consolidating multiple systems onto a single hardware platform. Benchmark tests run with CPU-intensive workloads show fair resource allocation, good scaling, and fairly consistent behavior, even when overcommitted, and also while achieving performance comparable with native throughput. ESX Server supports several mechanisms for fulfilling the quality-of-service guarantees frequently required for server consolidation. Maximum and minimum CPU allocations are shown to provide consistent and predictable behavior. The proportional-share approach provides more flexibility, but is not as straightforward to use, and requires careful configuration. Benchmark testing demonstrates and verifies the correct method for allocating shares. The pitfalls and idiosyncrasies of using processor affinities are also shown. Finally, the benefits of page sharing as demonstrated through virtual machine scaling benchmark test results are shown. This white paper demonstrates that ESX Server provides a stable platform for virtualization, with a powerful set of resource management tools to help users make the appropriate resource-allocation decisions as they deploy and scale virtual machines.



## References

1. *Timekeeping in VMware Virtual Machines*, a VMware white paper, [http://www.vmware.com/pdf/vmware\\_timekeeping.pdf](http://www.vmware.com/pdf/vmware_timekeeping.pdf)
2. *HyperThreading Support in VMware ESX Server 2*, a VMware white paper, [http://www.vmware.com/pdf/esx21\\_hyperthreading.pdf](http://www.vmware.com/pdf/esx21_hyperthreading.pdf)
3. Carl A. Waldspurger, *Memory Resource Management in VMware ESX Server*, Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI '02), Boston, Massachusetts, December 2002.



## Glossary

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

**Hyper-Threading:** A processor architecture feature that allows a single processor to execute multiple independent threads simultaneously. Hyper-threading was added to Intel's Xeon and Pentium® 4 processors. Intel uses the term "package" to refer to the entire chip, and "logical processor" to refer to each hardware thread.

**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, but 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.

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

**Virtual CPU:** A processor within a virtual machine. ESX Server 2 currently supports up to two virtual CPUs 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 additional resource utilization or decreased performance.

---

VMware, Inc., 3145 Porter Drive, Palo Alto, CA 94304 USA

Tel 650-475-5000 Fax 650-475-5001 www.vmware.com

Copyright (c) 1998-2005 VMware, Inc. All rights reserved. Protected by one or more of U.S. Patent Nos. 6,397,242, 6,496,847, 6,704,925, 6,711,672, 6,725,289, 6,735,601, 6,785,886, 6,789,156, 6,795,966, 6,880,022 6,961,941, 6,961,806 and 6,944,699; patents pending. VMware, the VMware "boxes" logo and design, Virtual SMP and VMotion are trademarks or registered trademarks of VMware, Inc. in the United States and/or other jurisdictions. All other marks and names mentioned herein may be trademarks of their respective companies.

Revision: 20051202 Item: ESX-ENG-Q405-133

---