Performance of Multiple Java Applications in a VMware vSphere™ 4.1 Virtual Machine

March 2011

PERFORMANCE STUDY

vmware®

Table of Contents

Introduction

The consolidation of enterprise Java applications can be performed using a number of different deployment models. In a recent paper, we showed that enterprise Java applications deployed on VMware vFabric tc Server™ have excellent performance when deployed in either scale-up or scale-out configurations on VMware vSphere 4.1. In all of the configurations in that paper, a single application/tc Server/JVM stack was deployed in each virtual machine (VM). In many instances, it is desirable to run multiple enterprise Java applications, each running on a separate JVM, within a single VM. There are many factors that might lead to this model, including software licensing and support costs, and the desire to maintain a deployment model used prior to virtualization.

In this paper, we discuss the results of tests that investigate the performance impact of running applications on multiple JVMs in a single VM. These tests were conducted using the Olio application running in tc Server 2.0 on VMware vSphere 4.1 Update 1. The intent of the investigation is to provide insight into the impact of sizing decisions when selecting the number of JVMs to deploy in a single VM.

The performance impact is examined from two perspectives. We look at the impact on response-time when adding Java applications, and thus JVMs, to a single VM. In order to gain additional insight on the impact of adding JVMs, we also look at the effect on CPU utilization and throughput. We also provide a comparison with the performance impact of running multiple JVMs on a native operating system in equivalent configurations.

Executive Summary

VMware vSphere 4.1 can support multiple Java applications, running on independent JVMs, in a single VM with excellent scalability. The testing discussed in this paper that demonstrates this excellent scalability used Olio, a multi-tier enterprise application which implements a complete social networking Web site. Olio was deployed on VMware vFabric tc Server, running both natively and virtualized on vSphere 4.1. These tests complement previously reported results, which showed that enterprise Java applications have excellent performance when deployed on vSphere 4.1 in scale-up and scale-out configurations with a single JVM per VM. The consistent high performance of Java applications on VMware vSphere 4.1 makes it possible to select a deployment model based on cost or other non-performance-related factors.

A series of tests was performed to investigate the impact on the application response-time of adding JVMs to a VM. The results show that multiple JVMs can be run in a single VM—at up to an overall VM CPU utilization of 70% to 80%—with no impact on application response-times. The tests were run on both 2-vCPU and 1-vCPU VMs, as well as on a native 1-CPU configuration. The impact of running multiple JVMs was similar in all cases.

Tests were also run to quantify the impact of per-JVM overheads on total throughput and CPU utilization. The per-JVM CPU utilization overhead ranged from 1% to 4% depending on the load placed on the application. Understanding these overheads, together with the CPU utilization due to the application load, can be used to guide sizing decisions when combining multiple Java applications in a single VM.

Test Environment

Overview

The workload used for the tests discussed in this paper was Olio, a web application benchmark developed by the Apache Software Foundation. The test environment used was almost identical to that discussed in the recent technical whitepaper on the performance of enterprise Java applications on VMware vSphere 4.1 (http://www.vmware.com/resources/techresources/10158). Refer to that paper for a more detailed discussion of Olio and the test environment.

There were a few significant changes to the test environment from the previous whitepaper:

- The host system for the current tests was a Dell R710 with two Intel Xeon X5680 (Westmere) processors (12 total cores, 24 logical processors) and 144GB of memory. This system was used in order to accommodate a VM with greater than 16GB of memory on a single NUMA node.
- Additional memory was added to the VMs to accommodate the increased Java heap needed for multiple JVMs. The memory available to the VM was always larger than the total heap size required by all JVMs running within the VM. In addition, memory was never overcommitted on the host.
    - The 1-vCPU VM had 24GB, with 21GB allocated as large pages.
    - The 2-vCPU VM had 32GB, with 27GB allocated as large pages.

Test Bed

The test bed consisted of the following components:

- The system under test (SUT). This was the server that hosted the Olio application, whether in VMs or natively. For these tests, the SUT was a Dell R710 with two Intel Xeon X5680 (Westmere) processors (12 total cores, 24 logical processors) and 144GB of memory.
- The primary driver system. This system ran the software to control the tests. It also hosted the Geocoder emulator, a component of the Olio benchmark used to provide geographical information to the application.
- Multiple satellite driver systems.
- The database server. This database hosted the user, event, and other tables for the Olio application.
- The filestore server. This server held the static content used by the Olio application. It exported an NFS mount with the filestore directory, which was mounted by the servers running the Olio application.
- An Ethernet switch. The network infrastructure used in these tests was 1Gbps.
- A Fibre Channel switch for access to the storage arrays.
- A SAN array which hosted the storage for the VMs.
- A SAN array which hosted the storage for the database and filestore.

Additional details about the components and their tuning are given in the Appendix.

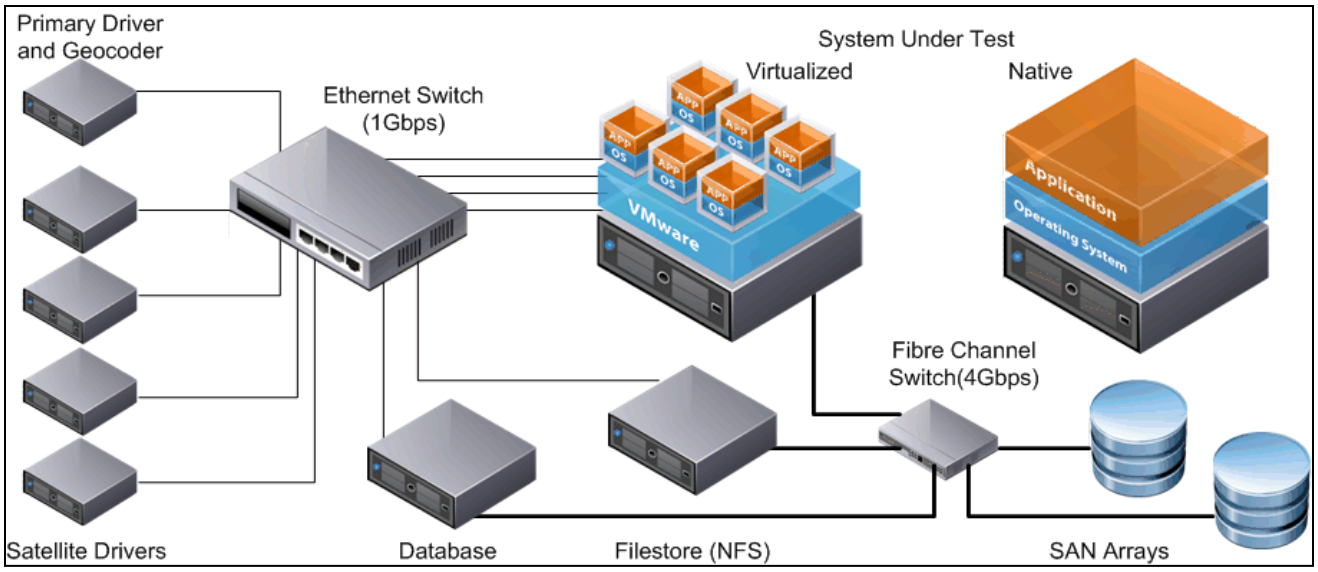

Figure 1. Test Bed

Performance Metrics

The performance achieved on a single run of the Olio workload is measured in terms of response-time and throughput. The Olio workload driver reports the average and 90th percentile response-time for each workload operation. The response-times reported in this paper are the averages for all operations, with each operation's response-time weighted by the frequency of that operation in the workload mix. The throughput of an Olio run is the number of operations per second performed during the steady-state period of the run. The throughput metric is valid only for runs in which all response-time and cycle-time requirements are satisfied. The peak throughput occurs at the largest number of users for which the application passes all requirements.

Multiple JVMs in a 2-vCPU VM

Overview

This section contains the results of the performance tests with multiple JVMs running in a single 2-vCPU VM. This includes an investigation of the impact on the response-time of scaling the load linearly with the number of JVM instances, as well as a look at the per-JVM throughput and CPU utilization overheads.

Response-Time Impact of Multiple JVMs

This section examines the impact on the overall response-time when adding JVMs to a VM. Each JVM in this multiple-JVMs-per-VM scenario represents a lightly loaded application. This test was designed to show how adding more applications to a VM impacts the performance of existing applications running in that VM.

In these tests, a fixed load was used to drive each instance of the Olio/tc Server/JVM stack. The overall load was increased by adding JVMs, each with a fixed number of additional users. The impact on performance was measured using the 90th percentile response-time, averaged over all JVMs.

Figure 2 shows the impact on performance as the number of JVMs running in a 2-vCPU VM is scaled-up. The results of two separate tests are shown, one with 100 Olio users added with each additional JVM, and a second with 200 Olio users added per additional JVM. At 1000 users, this corresponds to configurations with 5 or 10 JVMs running in a single VM. The results show that at lower loads, adding JVMs does not affect the application response-time. As the load increases so that the CPU utilization is greater than 80%, the response-time impact of additional JVMs becomes significant. For example, at 600 users, the difference between running with 3 or 6 JVMs

is minimal. However, at 1000 users, the response-time of the 10 JVM case is significantly higher than with the same load running on 5 JVMs.

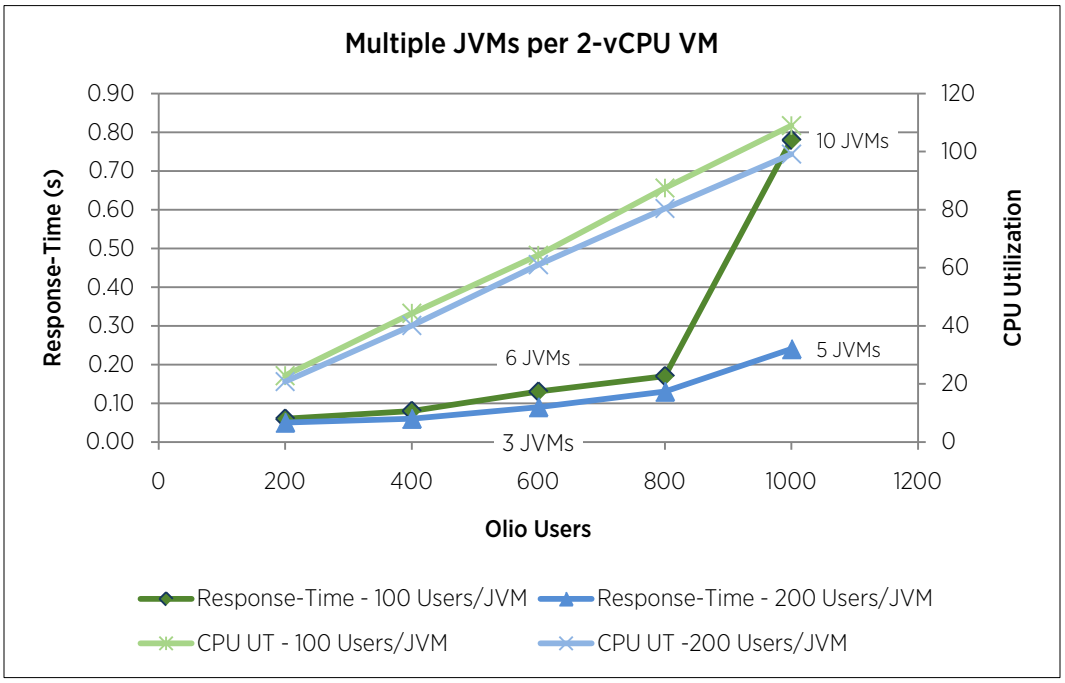

Figure 2. Impact of Adding JVMs with Fixed Load per JVM

Throughput and CPU Utilization Impact of Multiple JVMs

The approach of scaling up the load along with the number of JVMs represents a realistic deployment scenario. However, the increase in the overall demands (for example, the CPU utilization) on the VM as the number of JVMs is increased can complicate the understanding of the per-JVM overheads. In order to help separate the impact of running multiple JVMs from the impact of increased load, we ran two additional sets of tests:

- Tests in which the CPU utilization of the VM was kept constant as the number of JVMs was increased. This allowed us to examine the impact of additional JVMs on the overall throughput of the VM.
- Tests in which the load on the VM was kept constant as the number of JVMs was increased. This allowed us to examine the impact of additional JVMs on the CPU utilization of the VM.

When the CPU utilization of a VM is held constant, the overhead of adding JVM instances results in a drop in the overall throughput of the applications running in the VM. Figure 3 shows the per-JVM drop in throughput, expressed as a percentage of the throughput achieved by a single JVM, for the Olio application when the VM is held at a fixed 40% and 70% CPU utilization. For each JVM that is added to the VM, the throughput that can be maintained at the given CPU utilization drops by between 2% and 3%. The per-JVM impact on throughput is similar at 40% and 70% CPU utilization.

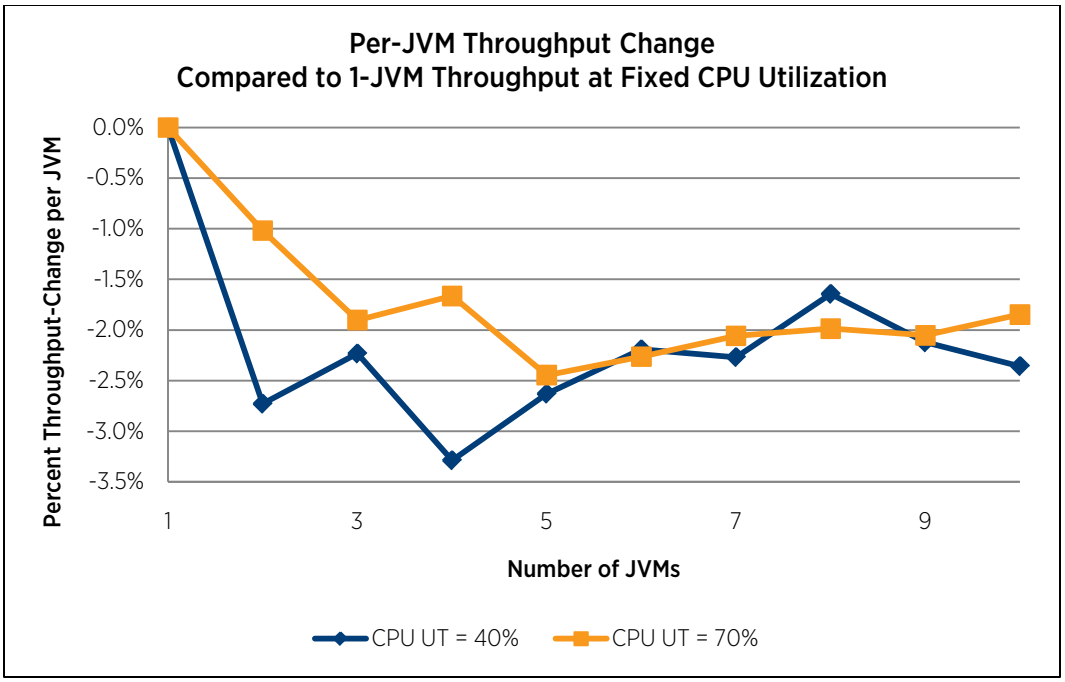

Figure 3. Per-JVM Throughput Decrease at Fixed CPU Utilization

Figure 4 shows the per-JVM increase in CPU utilization for fixed loads from 200 to 1000 users. These load levels correspond roughly to total CPU utilizations of 20% to 90+%. There are two important trends that can be observed in this data. First, the per-JVM CPU utilization overhead increases as the load increases. For example, at 200 users the CPU utilization increases by only 1% when moving from 1 to 2 JVMs, while at 1000 users the increase is about 4% Second, the per-JVM CPU utilization overhead levels off after about 4 JVMs. As a result, if we were sizing a multi-JVM deployment of this application and workload, we could use a worst-case estimate of about 4% CPU utilization overhead per JVM. This would be in addition to the CPU utilization added by the load on the application running in the additional JVM. Using the response-time and CPU utilization curves from the previous section (Response-Time Impact of Multiple JVMs on page 5) to select a targeted maximum CPU utilization allows us to determine the maximum number of application/JVM stacks that could be supported in a single VM. It is important to note that these results are for a specific application and workload, and the overheads will likely differ for different applications. In addition, this sizing approach assumes that CPU utilization is the primary bottleneck. It is important to consider other demands, such as storage or networking, which the combined applications will place on the VM.

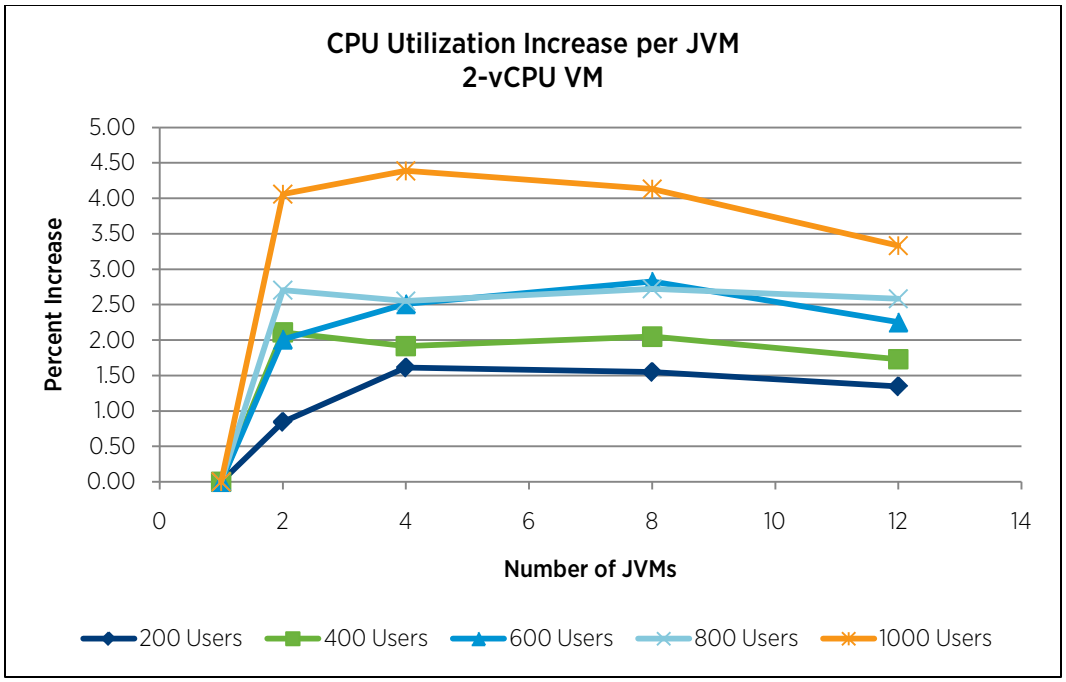

Figure 4. Per-JVM CPU Utilization Increase at Fixed Loads

Multiple JVMs in a 1-vCPU VM

Overview

This section contains the results of the performance tests with multiple JVMs running in a single 1-vCPU VM. The results for a 1-vCPU VM are quite similar to those presented in the previous section, Multiple JVMs in a 2-vCPU VM on page 5. As a result, we present only a subset of the response-time and CPU utilization overhead data shown for the 2-vCPU VM. In addition, we present the results of equivalent tests performed with the guest operating system running natively on a 1-CPU configuration.

Response-Time and CPU Utilization Impact of Multiple JVMs

The response-time impact of adding JVMs to a 1-vCPU VM is similar to that shown for a 2-vCPU VM in Response-Time Impact of Multiple JVMs on page 5. Figure 5 shows the response-times for tests in which the 25, 50, or 100 Olio users were added with each additional JVM. At low loads the response-times are essentially identical regardless of the number of JVMs. As the load increases, larger numbers of JVMs lead to higher response-times. This is particularly true where the CPU utilization overhead of the additional JVMs pushes the VM's CPU utilization beyond about 70%.

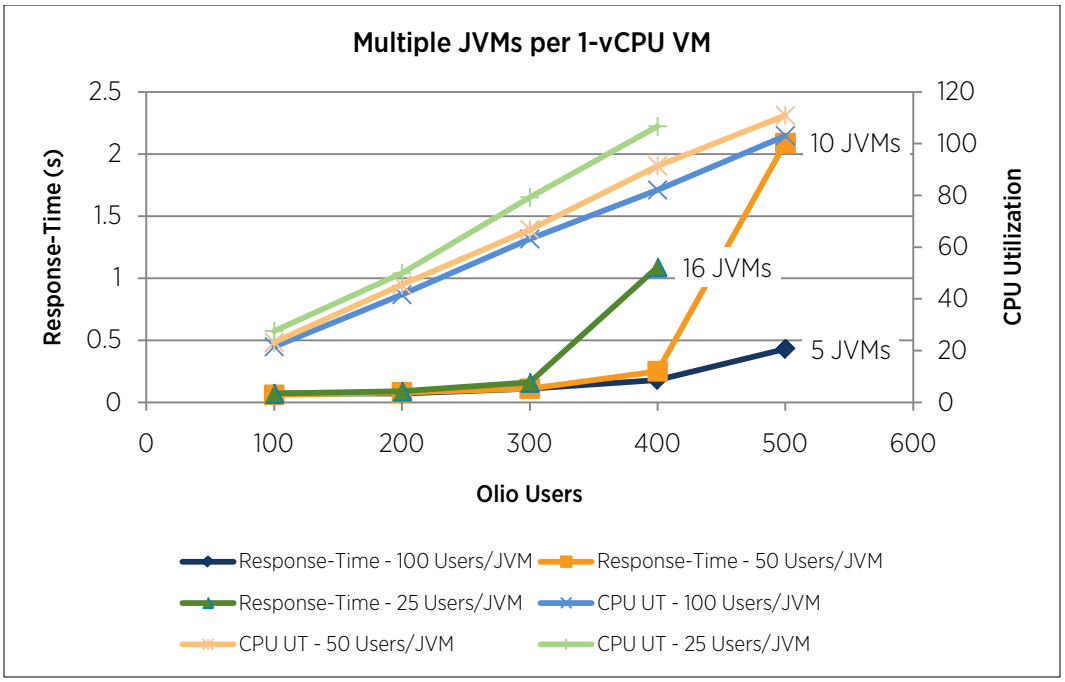

Figure 5. Performance Impact of Multiple JVMs at Increasing Load for a 1-vCPU VM

Figure 6 shows the per-JVM increase in CPU utilization for fixed loads from 200 to 1000 users. These load levels correspond roughly to total CPU utilizations of 20% to 90+%. This data shows the same trends as the 1-vCPU VM results presented in Throughput and CPU Utilization Impact of Multiple JVMs on page 6. The per-JVM CPU utilization overhead increases as the load increases. In addition, the per-JVM CPU utilization overhead levels off after about 4 JVMs. The per-JVM CPU utilization overheads for the 2-vCPU VM are slightly higher than for the 1-vCPU VM, particularly at the higher load. However, the maximum per-JVM CPU-utilization overhead is only around 4%, and is below 3% for loads which are not as close to CPU saturation.

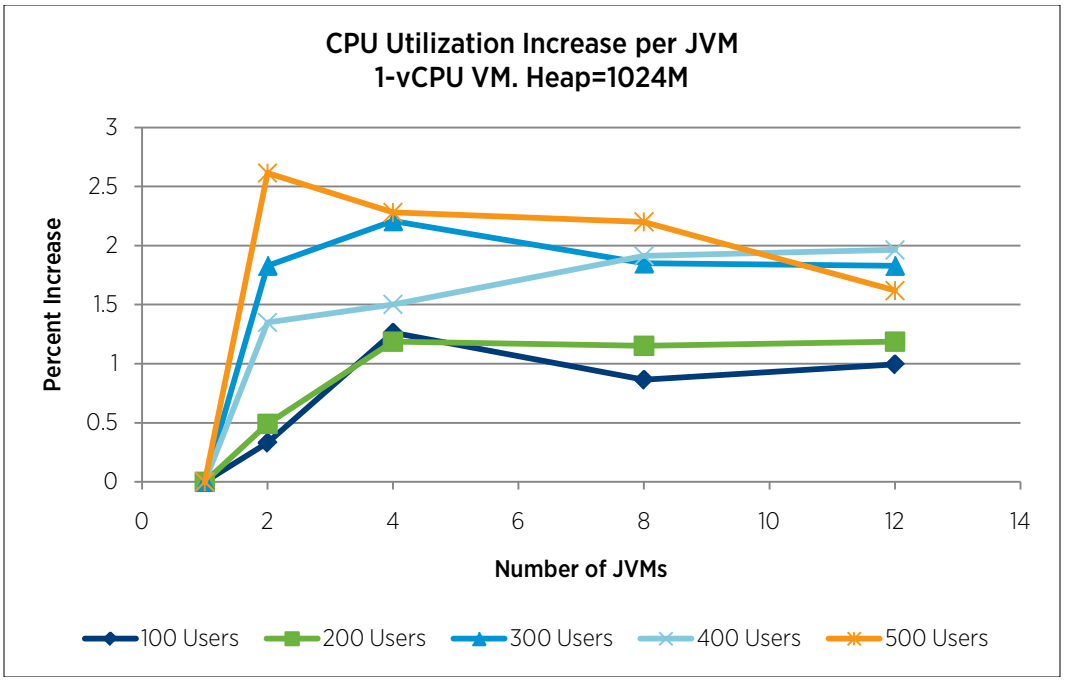

Figure 6. Per-JVM CPU Utilization Increase at Fixed Loads for a 2-vCPU VM

Comparison with Multiple JVMs Running on a Native Operating System

The previous sections discussed the response-time impacts and other overheads of running multiple JVMs in a single VM on VMware vSphere 4.1. It is interesting to consider how those impacts differ from running multiple JVMs in an equivalent configuration with the operating system running natively on the hardware.

Figure 7 shows a comparison of the response-time and CPU utilization of the VM and native cases for the 50 users per JVM scenario. The response-times for the two cases are almost identical, although there is a small increase in CPU utilization when running in the VM.

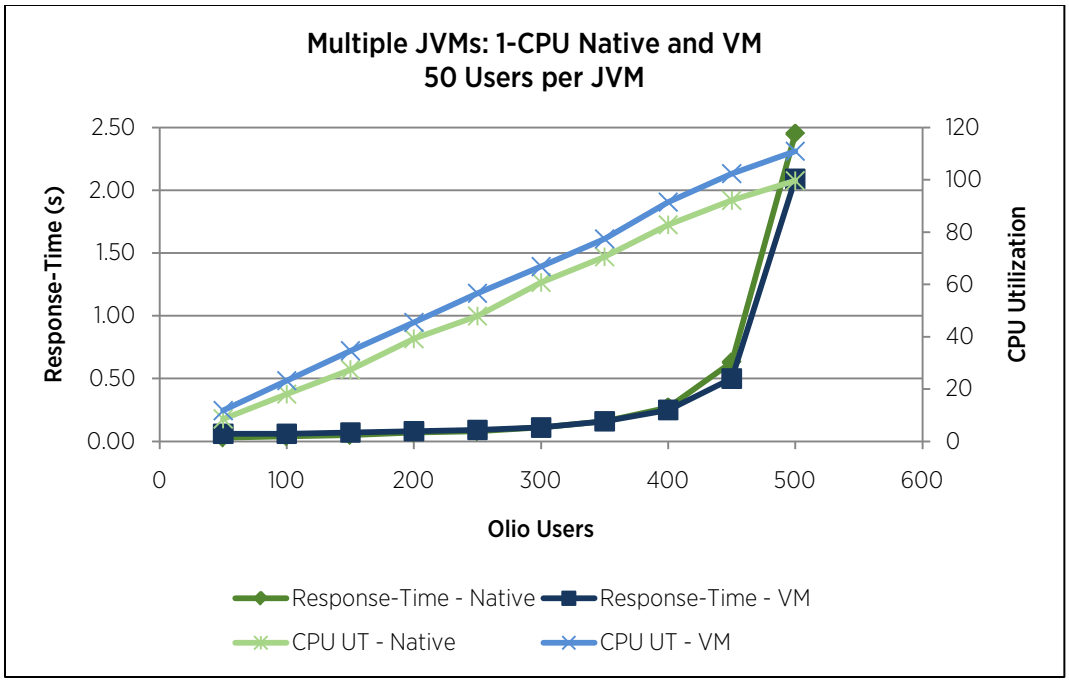

Figure 7. Comparison of VM and Native at 50 Users per JVM

Figure 8 shows the per-JVM drop in throughput for the native and VM configurations at a fixed 70% CPU utilization. Figure 9 shows the per-JVM CPU utilization overheads at fixed loads for the native 1-CPU configuration. The overheads are almost identical in the native and VM configurations. Based on these results, we can conclude that the types of overheads that we observed when running multiple JVMs in a VM are also present when running natively.

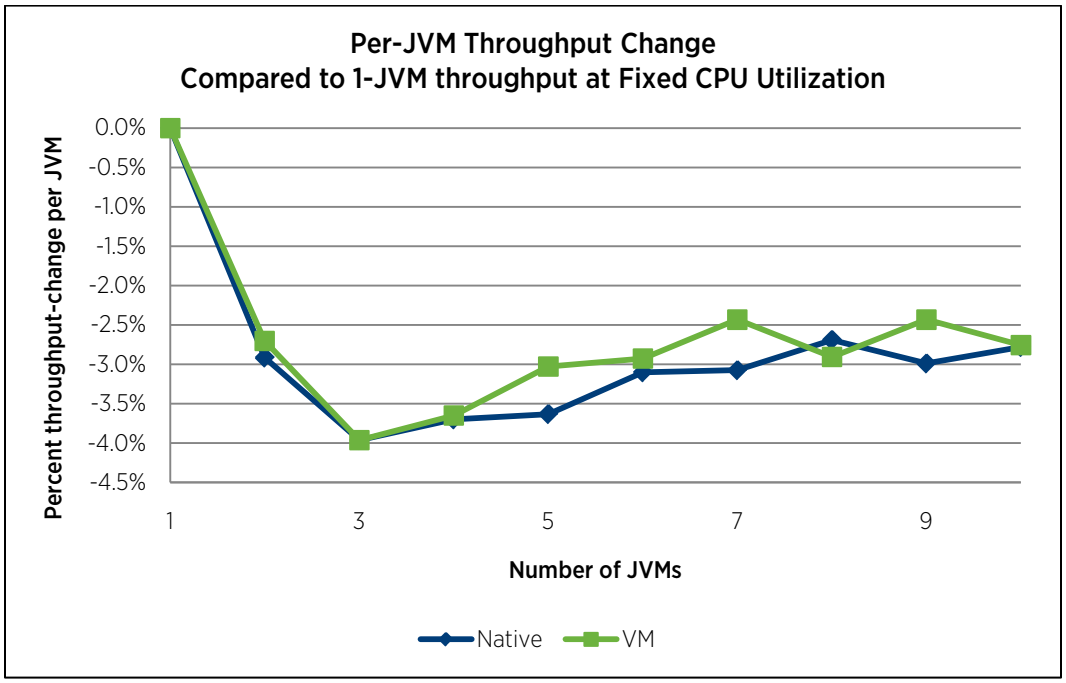

Figure 8. Native/VM Throughput Decrease for Increasing Numbers of JVMs

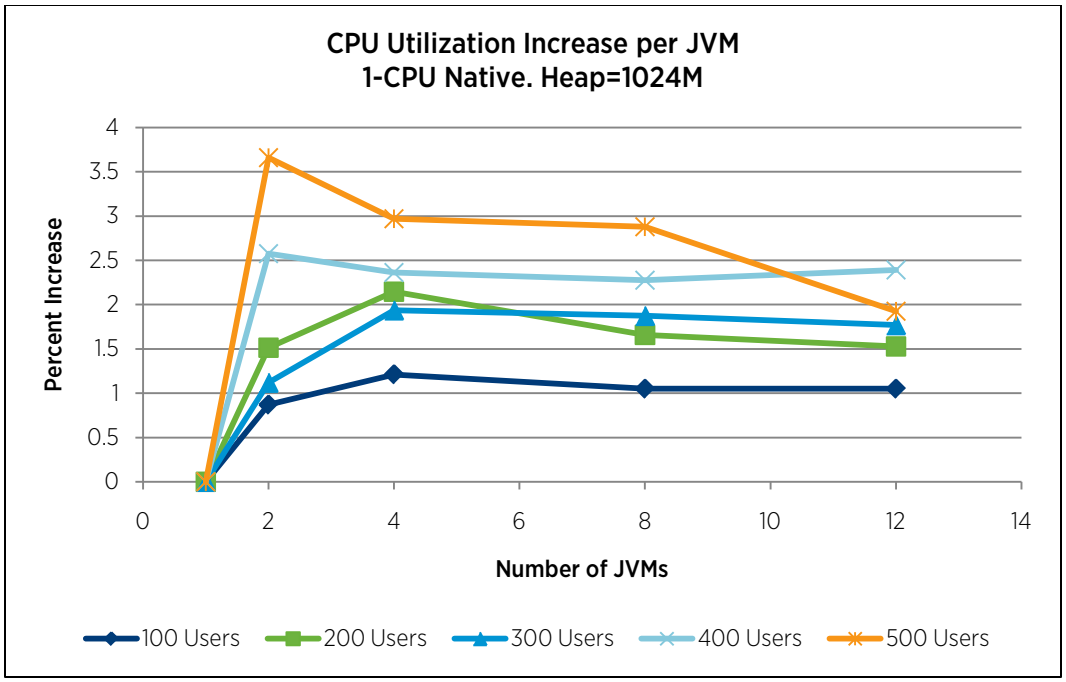

Figure 9. Per-JVM CPU Utilization Increase at Fixed Loads for Native 1-CPU

Conclusion

The combination of VMware vSphere 4.1 and VMware vFabric tc Server provides a powerful and flexible platform for virtualizing enterprise Java applications. The results of the tests discussed in this paper show that, with proper sizing, multiple Java applications can be virtualized in a single VM while maintaining the end-user experience.

There are a number of important lessons from these results that can be applied to deploying multiple Java applications in a single VM on VMware vSphere 4.1:

- The overall CPU utilization of the VM should be kept below the range of 70% to 80% CPU utilization. The exact target will depend on the characteristics of the applications. If it is not possible to perform pre-production performance testing with the actual applications, then a lower CPU utilization target should be used to ensure good performance. It is also necessary to ensure that other components, such as storage or network bandwidth, do not become bottlenecks when multiple applications are deployed in a single VM.
- When deploying multiple Java applications in a single VM, it is important to consider the CPU overhead of the JVMs as well as the CPU utilization due to the application load. While the per-JVM overheads are low, they can become significant when consolidating many JVMs in a VM, or when the application load is high.

About the Author

Hal Rosenberg is a performance engineer at VMware. His focus areas are middleware, Java, and performance troubleshooting. Prior to coming to VMware, he had over 10 years of experience working on performance engineering and analysis for hardware and software projects at IBM and Sun Microsystems. Hal will be blogging with additional information about this paper and other performance topics at http://communities.vmware.com/blogs/haroldr.

Appendix

Test Bed Details

Table 1 and Table 2 show the hardware and software configurations for the test bed.

Hardware Configuration

COMPONENT	DETAILS
Drivers	
System Model	Dell PowerEdge 2950
Processors	Two Intel Xeon 5160 @ 3GHz, 4 total cores
Total Memory	8GB
System Under Test (SUT)	
System Model	Dell R710
Processors	Two Intel Xeon X5680 @ 3.33GHz, 12 total cores. Hyper-Threading Enabled
Total Memory	144GB
Network Controller	Intel NC364T quad-port NIC, all four ports connected to the network switch
Storage Controller	QLogic ISP2432 4Gbps Fibre Channel HBA
Storage Configuration	- Native OS and ESX 4.1 U1 booted from an EMC CX500 Fibre Channel storage array - VMs stored in a VMFS3 datastore on an EMC CX500 Fibre Channel storage array
SUT/VM Configuration	
Number of vCPUs	1,2, 4
VM Memory Size	- 5GB with 1 or 2 vCPUs - 6GB with 4 vCPUs
Virtual NIC	vmxnet3

Database	
System Model	HP ProLiant DL380 G5
Processors	Two Intel Xeon X5460 @ 3.16Ghz, 16 total cores
Total Memory	32GB
Storage Controller	QLogic ISP2432 4Gbps Fibre Channel HBA
Storage Configuration	Data stored in a 10 disk, RAID10, LUN on an EMC CX3-10 Fibre Channel storage array
Filestore	
System Model	HP DL580 G5
Processors	Intel Xeon X7350 @ 2.93GHz, 16 total cores
Total Memory	128GB
Storage Controller	QLogic ISP2432 4Gbps Fibre Channel HBA
Storage Configuration	NFS mounted on a 10 disk, RAID0, LUN on an EMC CX3-10 Fibre Channel storage array

Table 1. Hardware Configuration

Software Configuration

COMPONENT	DETAILS
Drivers	
Workload Driver	Faban 1.0
tc Server for Geocoder	tc Server 2.0
JVM Version	Sun JDK 1.6.0_19 64-bit
Operating System	Red Hat Enterprise Linux 5 Update 4 x86_64
Additional Notes	The primary driver ran the DNS server
System Under Test (SUT)	
Operating System	Red Hat Enterprise Linux 5 Update 4 x86_64 When running natively, booted with kernel options: "mem=6144M numCPU=1"
tc Server Version	tc Server 2.0
JVM Version	Sun JDK 1.6.0_19 64-bit
JVM Parameters	1 CPU: -Xmx1536 –Xms1536 –Xss192K –XX:+UseLargePages 2 CPU: -Xmx2048 –Xms2048 –Xss192K –XX:+UseLargePages
VMware ESX Version	ESX 4.1 Update 1 (build 320357) The Olio application has very high network demands. As a result, we used the following network-level tuning: /adv/Net/vmxnetThroughputWeight=24 (default 0)

Database	
Database	Percona Server 5.1.47 (MySQL 5.1.47 with the Percona XtraDB storage engine)
Operating System	Red Hat Enterprise Linux 5 Update 4 x86_64
Filestore	
Operating System	Red Hat Enterprise Linux 5 Update 4 x86_64
Additional Notes	Mount exported using NFS v3

Table 2. Software Configuration

VMware, Inc. 3401 Hillview Avenue Palo Alto CA 94304 USA Tel 877-486-9273 Fax 650-427-5001 www.vmware.com