vmware IMPLEMENTATION METHODOLOGY

Enterprise Java Applications on VMware Best Practices Guide

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

Contents

1. Introduction

1.1 Overview

This *Enterprise Java Applications on VMware Best Practices Guide* provides information about best practices for deploying enterprise Java applications on VMware, including key best practice considerations for architecture, performance, design and sizing, and high availability. This information is intended to help IT Architects successfully deploy and run Java environments on VMware vSphere™.

1.2 Purpose

This guide provides best practice guidelines for deploying enterprise Java applications on VMware vSphere. The recommendations in this guide are not specific to any particular set of hardware or to the size and scope of any particular implementation. The best practices in this document provide guidance only and do not represent strict design requirements because enterprise Java application requirements can vary from one implementation to another. However, the guidelines do form a good foundation on which you can build—many of our customers have used these guidelines to successfully virtualize their enterprise Java application. This document focuses on details that pertain to the deployment of enterprise Java applications on VMware vSphere and it is not necessarily a best practice guide for pure Java. For specific Java best practices refer to the vendor documentation for the JVM you are using.

Virtualizing enterprise Java applications does not require a change in your Java coding paradigm and any performance enhancements that you have done on physical are transferrable as is to the vSphere deployed instance of your application.

1.3 Target Audience

This guide assumes a basic knowledge and understanding of VMware vSphere and enterprise Java applications.

- Architectural staff can use this document to gain an understanding of how the system will work as a whole as they design and implement various components.
- Engineers and administrators can use this document as a catalog of technical capabilities.

1.4 Scope

This guide covers the following topics:

- Enterprise Java Applications on vSphere Architecture – This section provides a high level best practice architecture for running enterprise Java applications on vSphere.
- Enterprise Java Applications on vSphere Best Practices – This section provides best practice guidelines for properly preparing the vSphere platform to run enterprise Java applications on vSphere. Best practices for the Design and Sizing of VMs, guest OS tips, CPU, memory, storage, networking, and useful JVM tuning parameters are presented. Also covered are the various high availability features in vSphere including VMware ESX™ host clusters, resource pools (*horizontal scalability* and *vertical scalability*) along with the VMware Distributed Resource Scheduler (DRS).
- Enterprise Java on vSphere Troubleshooting Primer – There are times when you have to troubleshoot a particular Java application problem when running on vSphere. The vSphere `esxtop` utility is very informative when troubleshooting.
- FAQ: Enterprise Java Application on vSphere – In this section, we answer some frequently asked questions about the deployment of the enterprise Java applications on vSphere.

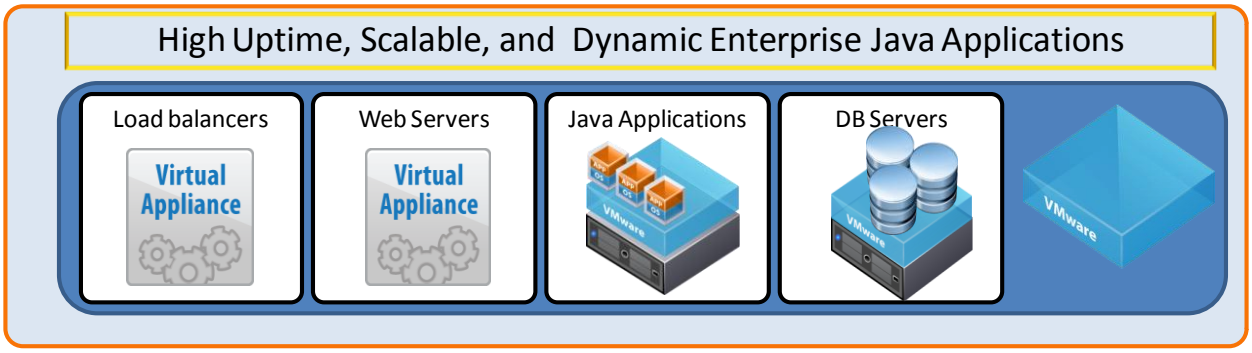
2. Enterprise Java Applications on vSphere Architecture

This section provides details on best practice architecture for enterprise Java applications running on vSphere. Enterprise Java applications are made of four main tiers. These are the:

- Load Balancers tier
- Web Servers tier
- Java Application Server tier
- DB Server tier

A highly scalable and robust Java application has all of these tiers running in VMware vSphere in order to reap the full benefits of scalability features offered by vSphere. Figure 1 shows a multi-tier fully virtualized enterprise Java applications architecture running on VMware vSphere.

Figure 1. Multi-tier Virtualized Enterprise Java Application Architecture



Each one of these tiers is running in a VM that is managed by VMware vSphere, which forms the key building foundation. Best practices are discussed in this guide for vSphere features such as VMware HA, DRS, VMware vMotion™, resource pools, hot plug/hot add, networking and storage.

The following are key architectural attributes of each tier:

- Load Balancer tier – Increasingly feature-rich load balancers are available that provide various load balancing algorithms and API integration with VMware vSphere. This allows the enterprise Java application architecture to scale on demand as traffic bursts occur.
- Web Server tier – The Web Server tier must be appropriately tuned, with the right number of HTTP threads to service your anticipated traffic demands.
- Java Application Server tier – Many of the commonly used application servers have mechanisms to help you tune the Java engine to meet traffic demands. If you have already tuned the available Java Threads, JDBC configurations, and various JVM and GC parameters on physical machines, the tuning information is transferrable as is for the vSphere deployment of enterprise Java applications.
- DB Server tier – Critical to meeting the uptime SLA of enterprise Java applications is having appropriate high availability architecture for the DB server. DB servers can benefit from running on vSphere—see the best practices for your DB server on vSphere. This guide covers best practices to JDBC Connection Pool tuning requirements at the Java application server level that the DB server needs to accommodate.

3. Enterprise Java Applications on vSphere Best Practices

3.1 VM Sizing and Configuration Best Practices Overview

Enterprise Java applications are highly customizable, and consequently, a performance test has to be conducted to establish best sizing.

It is a best practice to establish the size of your VM in terms of vCPU, memory and how many JVMs are needed by conducting a thorough performance test that mimics your production workload profile. The resulting VM from the vertical scalability (scale-up) performance test is referred to as the *building block VM*. The building block VM is a good candidate template on which scaled-out (horizontally scaled) VMs can be based.

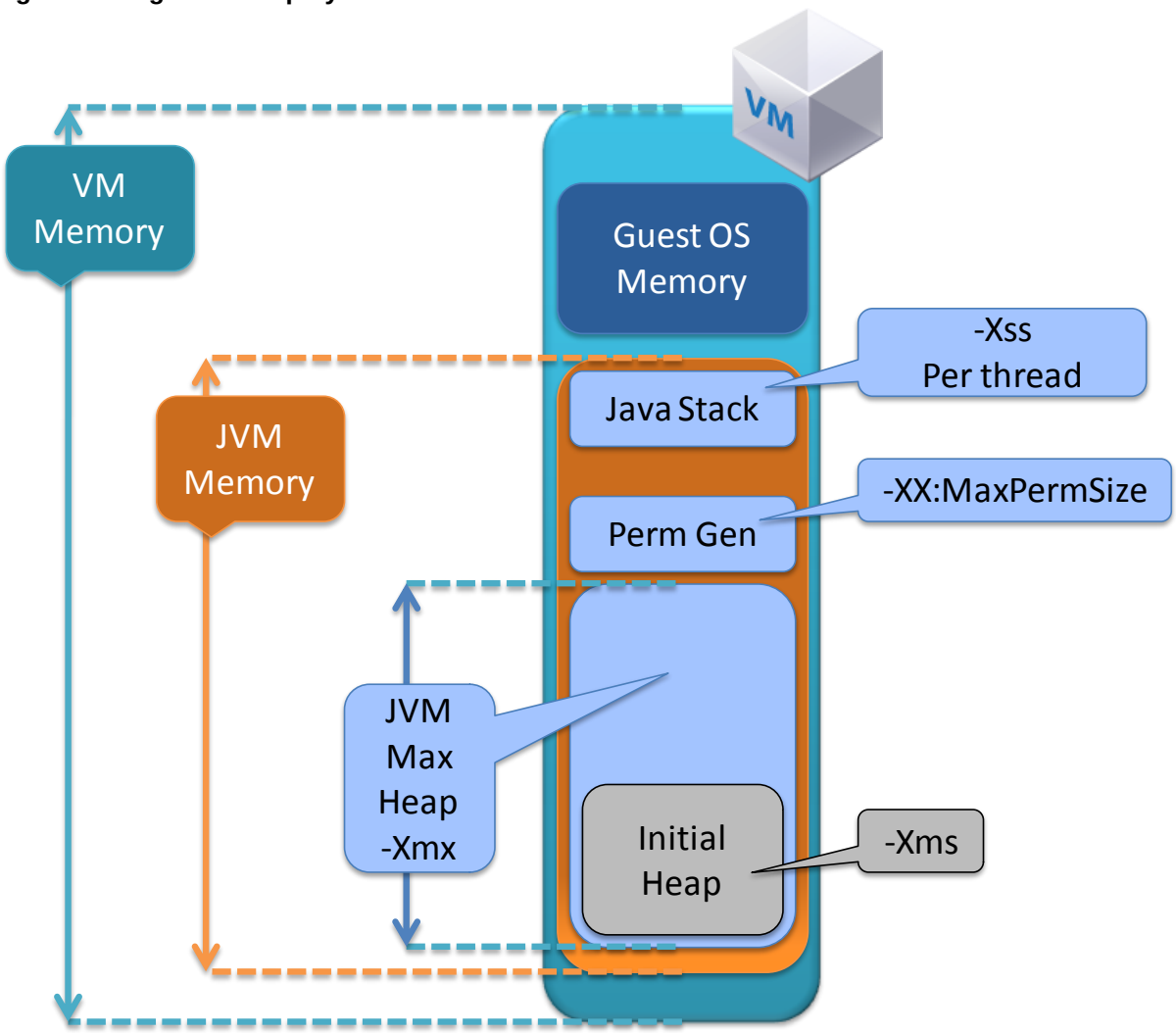
3.2 vCPU for VMs Best Practices

Best Practice	Description
BP1 – VM Sizing and VM-to JVM ratio through a performance load test	Establish a workload profile and conduct a load test to measure how many JVMs you can stack on a particular sized VM. In this test, establish a best case scenario of how many concurrent transactions you can push through a configuration before it can be safely deemed as a good candidate for scaling horizontally in an application cluster.
BP2– VM vCPU CPU Overcommit	For performance-critical enterprise Java applications VMs in production make sure the total number of vCPUs assigned to all of the virtual machines does not cause greater than 80% CPU utilization on the ESX host.
BP3 – VM vCPU Do not over subscribe to CPU cycles that you don't really need	• If your performance load test determines; for example, 2 vCPU is adequate up to 70% CPU utilization, but instead you allocate 4 vCPU to your VM, then potentially you have 2 vCPUs Idle, which is not optimal. • If the exact workload is not known, size the virtual machine with a smaller number of vCPUs initially and increase the number later, if necessary.

3.3 VM Memory Size Best Practice

To understand how to size memory for a VM you must understand the memory requirements of Java and various segments within it. Figure 2 provides an illustration of these separate memory areas.

Figure 2. Single JVM deployed on One VM



VM Memory = Guest OS Memory + JVM Memory

JVM Memory = JVM Max Heap (`-Xmx` value) *+ JVM Perm Gen (`-XX`: MaxPermSize)*
*+ NumberOfConcurrentThreads * (`-Xss`)*

where:

- *Perm Gen* is an area that is in addition to the `-Xmx` (*Max Heap*) value and is not GCed as it holds the class level information about the code. IBM JVMs do not have Perm Gen area.
- The above VM Memory formula is an approximation of the main areas allocated. To more accurately size you need to load test the Java application for additional memory requirements that may be allocated due to NIO buffers: JIT code cache, classloaders, and verifiers. In particular, some Java applications may use NIO buffers, which can have huge additional memory demands.
- The contents of a direct buffer are allocated from the guest operating system memory instead of the Java heap, and non-direct buffers are copied into direct buffers for native I/O operations. Use load testing to appropriately size the effect of these buffers.
- If you have multiple JVMs (N JVMs) on a VM then: *VM memory = guest OS memory + N * JVM memory*.

Best Practice	Description
BP4 – VM memory sizing	• Whether you are using Windows or Linux as your guest OS, refer to the technical specification of the various vendors for memory requirements. It is common to see the guest OS allocated about 1GB in addition to the JVM memory size. However, each installation may have additional processes running on it, for example monitoring agents, and you need to accommodate their memory requirements as well. Figure 2 shows the various segments of JVM and VM memory, and the formula summarizes VM Memory as: *VM Memory (needed) = guest OS memory + JVM Memory,* where *JVM Memory = JVM Max Heap (*`-Xmx` *value) + Perm Gen (*`-XX`*:MaxPermSize) + NumberOfConcurrentThreads * (*`-Xss`*)* • The `-Xmx` value is the value that you found during load testing for your application on physical servers. This value does not need to change when moving to a virtualized environment. Load testing your application when deployed on vSphere will help confirm the best `-Xmx` value. • It is recommended that you do not overcommit memory because the JVM memory is an active space where objects are constantly being created and garbage collected. Such an active memory space requires its memory to be available all the time. If you overcommit memory ballooning or swapping may occur and impede performance. • ESX host employs two distinct techniques for dynamically expanding or contracting the amount of memory allocated to virtual machines. The first method is known as memory *balloon driver* (vmmemctl). This is loaded from the VMware Tools package into the guest operating system running in a virtual machine. The second method involves paging from a virtual machine to a server swap file, without any involvement by the guest operating system. In the page swapping method, when you power on a virtual machine, a corresponding swap file is created and placed in the same location as the virtual machine configuration file (`VMX` file). The virtual machine can power on only when the swap file is available. ESX hosts use swapping to forcibly reclaim memory from a virtual machine when no balloon driver is available. The balloon driver may be unavailable either because VMware Tools is not installed, or because the driver is disabled or not running. For optimum performance, ESX uses the balloon approach whenever possible. However, swapping is used when the driver is temporarily unable to reclaim memory quickly enough to satisfy current system demands. Because the memory is being swapped out to disk, there is a significant performance penalty when the swapping technique is used. Therefore, it is recommended that the balloon driver is always enabled, but monitor to verify that it is not getting invoked as that memory is overcommitted. Both ballooning and swapping should be prevented for Java applications. To prevent ballooning and swapping, refer to BP5 – Set memory reservation for VM needs.

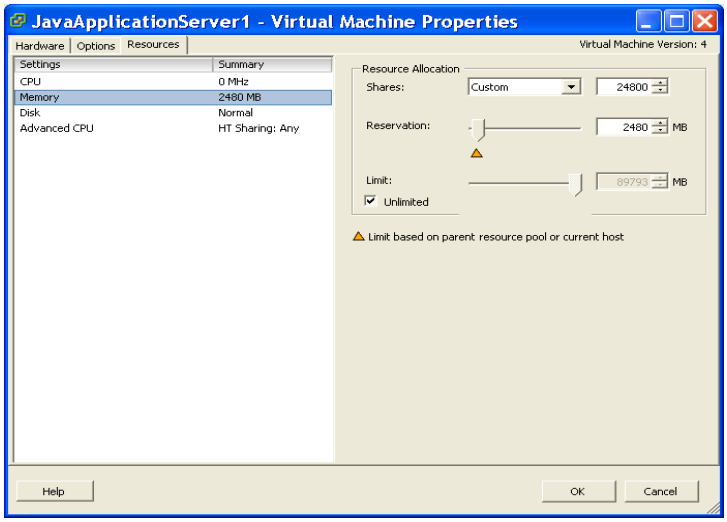
BP5 – Set memory reservation for VM memory needs	• JVMs running on VMs have an active heap space requirement that must always be present in physical memory. Use the VMware vSphere Client to set the reservation equal to the needed VM memory. *Reservation Memory = VM Memory = guest OS Memory + JVM Memory* • You may set this reservation to the active memory being used by the VM for a more efficient use of the amount of memory available. Or, a simpler approach is to set the reservation equal to the total configured memory of the VM.
BP6 – Use Large Memory Pages	• Large memory pages help performance by optimizing the use of the Translation Look-aside Buffer (TLB), where virtual to physical address translations are performed. Use large memory pages as supported by your JVM and your guest operating system. The operating system and the JVM must be informed that you want to use large memory pages, as is the case when using large pages in physical systems. • Set the `-XX:+UseLargePages` at the JVM level for Sun HotSpot. • On the IBM JVM it is `-Xlp`, and `JRockit -XXlargePages`. • You also need to enable this at the guest OS level. For information, see *Large Page Performance: ESX Server 3.5 and ESX Server 3i v3.5*.

3.3.1 BP5-How to Set Memory Reservation

Set the memory reservation value in the vSphere Client to the size of memory for the virtual machine. In Figure 3, the memory reservation is set to 2480MB. This virtual machine is always allocated this amount of memory on any ESX host on which it runs.

To set the memory reservation select the VM, right-click and select **Edit Settings** > **Resources** tab. Figure 3 shows how to set the memory reservation for a virtual machine in the vSphere Client.

Figure 3. Setting the Memory Reservation



3.4 VM Timekeeping Best Practices

Best Practice	Description
BP7 – Timekeeping Use NTP Source	• Timekeeping can be different in virtual machines than on physical machines for a variety of reasons (see *Timekeeping in VMware Virtual Machines: VMware ESX 4.0/ESXi 4.0, VMware Workstation 7.0*. • Timekeeping can have an effect on Java programs if they are sensitive to accurate measurements over periods of time, or if they need a timestamp that is within an exact tolerance (such as a timestamp on a shared document or data item). VMware Tools contains features that are installable into the guest operating system to enable time synchronization and the use of those tools is recommended. The effects of timer interrupts are also discussed as the frequency of those interrupts can have an effect on the performance of your Java application.

3.4.1 Configuration for Timekeeping Best Practices

- Synchronize the time on the ESX host with an NTP source. See *Timekeeping in VMware Virtual Machines: VMware ESX 4.0/ESXi 4.0, VMware Workstation 7.0*.
- Synchronize the time in the virtual machine's guest operating system:
  - For Linux guest operating systems using an external NTP source, see *Timekeeping in VMware Virtual Machines: VMware ESX 4.0/ESXi 4.0, VMware Workstation 7.0*.
  - For Windows guest operating systems use W32Time. Refer to your Windows administration guide for detailed information.

Lower the clock interrupt rate on the virtual CPUs in your virtual machines by using a guest operating system that allows lower timer interrupts. Examples of such operating systems are RHEL 4.7 and later, RHEL 5.2 and later and the SuSE Linux Enterprise Server 10 SP2. See *Timekeeping best practices for Linux guests* for more information on timekeeping best practices for Linux.

- Use the Java features for lower resolution timing that are supplied by your JVM, such as the option for the Sun JVM on Windows guest operating systems:

```
-XX:+ForceTimeHighResolution
```

- You can also set the `_JAVA_OPTIONS` variable to this value on Windows operating systems using the technique given (useful in cases where you cannot easily change the Java command line).
- The following is an example of how to set the Sun JVM option. To set the `_JAVA_OPTIONS` environment variable:
  1. Click **Start** > **Settings** > **Control Panel** > **System** > **Advanced** > **Environment Variables**.
  2. Click **New** under **System Variables**. The variable name is `_JAVA_OPTIONS`. The variable value is `-XX:+ForceTimeHighResolution`.
  3. Reboot the guest operating system to properly propagate the variable.
- Avoid using the `/usepmtimer` option in the `boot.ini` system configuration for Windows guest operating systems that use an SMP HAL.

3.5 Vertical Scalability Best Practices

If an enterprise Java application deployed on vSphere experiences heavy CPU utilization and you have determined that an increase in the vCPU count will help resolve the saturation, you can use vSphere hot add to add additional vCPU.

Best Practice	Description
BP8 – Hot Add CPU/Memory	• VMs with a guest OS that supports hot add CPU and hot add memory can take advantage of the ability to change the VM configuration at runtime without any interruption to VM operations. This is particularly useful when you are trying to increase the ability of the VM to handle more traffic. • Plan ahead and enable this feature. The VM must be turned off to have the hot plug feature enabled, but when enabled you can hot add CPU and hot add memory at runtime without VM shutdown (if the guest OS supports it). • Refer to VM memory size best practices in Section 3.3. • When you need to increase Java heap space, along with this is usually an increase in vCPU count to get the best GC cycle performance. Keep in mind that there are many other aspects to GC tuning and you should refer to your JVM documentation.

3.6 Horizontal Scalability, Clusters, and Pools Best Practices

Enterprise Java applications deployed on VMware vSphere can benefit from using vSphere features for horizontal scalability using ESX host clusters, resource pools, host affinity and DRS.

Best Practice	Description

BP9 – Use ESX host cluster	• To enable better scalability use ESX host clusters.
	• When creating clusters enable VMware HA and VMware DRS:
	  ○ VMware HA – Detects failures and provides rapid recovery for the VM running in a cluster. Core functionality includes host monitoring and VM monitoring to minimize downtime.
	  ○ VMware DRS – Enables vCenter Server to manage hosts as an aggregate pool of resources. Cluster resources can be divided into smaller pools for users, groups, and VMs. It enables vCenter to manage the assignment of VMs to hosts automatically, suggesting placement when VMs are powered on, and migrating running VMs to balance load and enforce allocation policies.
	• Enable EVC (for either Intel or AMD). EVC is Enhanced vMotion Compatibility; it configures a cluster and its hosts to maximize vMotion compatibility. When EVC is enabled, only hosts that are compatible with those in the cluster may be added to the cluster.

BP10 – Use resource pools	Multiple resource pools can be used within a cluster to manage compute resource consumption by either reserving the needed memory for the VMs within a resource pool or by limiting/restricting it to a certain level. This feature also helps you meet quality of service and requirements. For example, you can create a Tier-2 resource pool for the less critical applications and a Tier-1 resource pool for business critical applications.
BP11 – Affinity Rules	In addition to exiting anti-affinity rules, the VM-Host affinity rule was introduced in vSphere 4.1. The VM-Host affinity rule provides the ability to place VMs on a subset of hosts in a cluster. This is very useful in honoring ISV Licensing requirements. Rules can be created so that VMs run on ESX hosts in different blades for higher availability. Conversely, limit the ESX host to one blade in case network traffic between the VMs needs to be optimized by keeping them in one chassis location.
BP12 – Use vSphere-aware load balancers	vSphere makes it easy to add resources such as host and VMs at runtime. It is possible to provision these ahead of time. However, it is simpler if you use a load balancer that is able to integrate with vSphere APIs to detect the newly added VMs and add them to its application load balancing pools without downtime.

3.7 Inter-tier Configuration Best Practices

As discussed in Section 2, there are four critical technology tiers that sit on top of vSphere. These tiers are the Load Balancer tier, the Web Server tier, the Java Application Server tier, and the DB Server tier. The configurations for compute resources at each tier must translate to an equitable configuration at the next tier. For example, if the Web Server tier is configured to handle 100 HTTP requests per second, then of those requests you must determine how many Java application server threads are needed, and in turn how many DB connections are needed in the JDBC Pool configuration.

Best Practice	Description
BP13 – Establish appropriate thread ratios that prevents bottlenecks (HTTP threads:Java threads:DB connections)	• This is the ratio of HTTP threads to Java threads to DB connections. • Establish initial setup by assuming that each layer requires a 1:1:1 ratio of HTTP threads:Java Threads:DB-connections, and then based on the response time and throughput numbers, adjust each of these properties accordingly until you have satisfied your SLA objectives. • For example, if you have 100 HTTP requests submitted to the Web Server initially, assume that all of these will have an interaction with Java threads, and in turn, DB connections. Of course, in reality, during your benchmark you will find that not all HTTP threads are submitted to the Java application server, and in turn, not all Java application server threads each require a DB connection. That is, you may find your ratio for 100 requests translates to 100 HTTP threads:25 Java threads:10 Db connections, and this depends on the nature of your enterprise Java application behavior. Benchmarking helps you establish this ratio.
BP14 – Load balancer algorithm choice and VM symmetry	• Take into account the available algorithms of your load balancer. Make sure that when using the scale-out approach all of your VMs are receiving an equal share of the traffic. Some industry standard algorithms are Round Robin, Weighted Round Robin, Least Connections, and Least Response Time. You may want to initially default to Least Connections and then adjust as you see fit in your load test iterations. • Keep your VMs symmetrical in terms of the size of compute resource. For example, if you decide to use 2 vCPU VMs as a repeatable, horizontally scalable building block, this helps with your load balancing algorithm, working more effectively as opposed to if there were a pool of non-symmetrical VMs for one particular application. That is, mixing 2 vCPU VMs with 4 vCPU VMs in one load balancer-facing pool is non-symmetrical and the load balancer has no notion of weighing this unless you configure for it at the Load Balancer level, which is time consuming.

3.8 High Level vSphere Best Practices

It is important to follow the best practices. See *Performance Best Practices for VMware vSphere 4.0*. The following is a summary of some of the key networking, storage and hardware-related best practices that are commonly used.

3.8.1 vSphere Networking Best Practices

Best Practice	Description
BP15 – vSphere Networking	Follow vSphere networking best practices. In addition to *Performance Best Practices for VMware vSphere 4.0*, refer to VMware *Virtual Networking Concepts and Best Practices*.

3.8.2 vSphere Storage Best Practices

Best Practice	Description
BP16 – vSphere Storage	• VMware recommends a minimum of four paths from an ESX host to a storage array, which means the host requires at least two HBA ports. • Follow vSphere storage best practices. For detailed description of VMware storage best practices refer to the VMware *SAN System Deployment and Design Guide*.

3.8.3 vSphere Server Hardware Configuration Best Practices

Best Practice	Description
BP17 – ESX Host Hardware	• For hardware configuration best practices, refer to VMware *vCenter Server Performance and Best Practices Guide*. Also see *The CPU Scheduler in VMware ESX4.1 Guide*. • Disable any other power-saving mode in the BIOS. • NUMA Considerations – IBM (X-Architecture), AMD (Opteron-based), and Intel (Nehalem) non-uniform memory access (NUMA) systems are supported by ESX. On AMD Opteron-based systems, such as the HP ProLiant DL585 Server, BIOS settings for node interleaving determine whether the system behaves like a NUMA system or like a uniform memory accessing (UMA) system. • By default, ESX NUMA scheduling and related optimizations are enabled only on systems with a total of at least four CPU cores and with at least two CPU cores per NUMA node. • Virtual machines with a number of vCPUs equal to or less than the number of cores in each NUMA node are managed by the NUMA scheduler and have the best performance. • Hardware Bios – Verify the BIOS is set to enable all populated sockets, and enable all cores in each socket. • Enable Turbo Mode if your processors support it. • Make sure hyper-threading is enabled in the BIOS.

4. Troubleshooting Primer

Troubleshooting problems with enterprise Java applications involves the investigation of each of the tiers, the Load balancer tier, The Web Server tier, the Java Application Server tier, the DB Server tier, and vSphere. VMware vSphere, in turn, has a dependency on networking and storage. The next few sections provide information about how to begin troubleshooting, and effective utilities you can use.

4.1 Open a Support Request Ticket

If you suspect the VMware vSphere is not configured optimally and is cause of the bottleneck, file a support request (http://www.vmware.com/support/contacts/file-sr.html). In addition, you may want to:

- Follow the troubleshooting steps outlined in the *Performance troubleshooting guide for ESX4.0*.
- Verify that you have applied all of the best practices discussed in this *Best Practices Guide*.
- Run the `vm-support` utility. Execute the following command at the service console:

  ```
  vm-support -s
  ```

  This collects necessary information so that VMware can help diagnose the problem. It is best to run this command at the time when the symptoms occur.

4.2 Troubleshooting Techniques for vSphere with esxtop

4.2.1 Performance Guides and References

- *Performance Troubleshooting for VMware vSphere 4 and ESX 4.0* (http://communities.vmware.com/docs/DOC-10352)
- *Interpreting esxtop 4.1 Statistics* (http://communities.vmware.com/docs/DOC-11812)

4.2.2 esxtop Primer

See *Interpreting esxtop Statistics* for detailed information about metrics. The following table summarizes some of the metrics you may use when troubleshooting.

Display	Metric	Threshold	Description
CPU	%RDY	10	Over-provisioning of vCPU, excessive usage of vSMP or a limit (check %MLMTD) has been set. This %RDY value is the sum of all vCPUs %RDY for a VM. For example, if the max value of %RDY of 1vCPU is 100% and 4vCPU is 400%. If %RDY is 20 for 1 vCPU then this is problematic, as it means 1 vCPU is waiting 20% of the time for VMkernel to schedule it.
CPU	%CSTP	3	Excessive usage of vSMP. Decrease amount of vCPUs for this particular VM.
CPU	%MLMTD	0	If larger than 0 the worlds are being throttled. Possible cause is a limit on CPU.
CPU	%SWPWT	5	VM waiting on swapped pages to be read from disk. You may have overcommitted memory.
MEM	MCTLSZ	1	If larger than 0, host is forcing VM to inflate balloon driver to reclaim memory as the host is overcommitted.
MEM	SWCUR	1	If larger than 0 host has swapped memory pages in the past. You may have overcommitted.
MEM	SWR/s	1	If larger than 0 host is actively reading from swap. This is caused by excessive memory overcommitment.
MEM	SWW/s	1	If larger than 0 host is actively writing to swap. This is caused by excessive memory overcommitment.
MEM	N%L	80	If less than 80, VM experiences poor NUMA locality. If a VM has memory size greater than the amount of memory local to each processor, the ESX scheduler does not attempt to use NUMA optimizations for that VM.
NETWORK	%DRPTX	1	Dropped packages transmitted, hardware is overworked due to high network utilization.
NETWORK	%DRPRX	1	Dropped packages received, hardware is overworked due to high network utilization.
DISK	GAVG	25	Look at DAVG and KAVG as GAVG = DAVG + KAVG.
DISK	DAVG	25	At this level you have disk latency that is likely to be caused by storage array.
DISK	KAVG	2	Disk latency caused by the VMkernel. High KAVG usually means queuing. Check QUED.
DISK	QUED	1	Queue has maxed out. Possibly queue depth is set too low. Check with array vendor for optimal queue value.
DISK	ABRTS/s	1	Aborts issued by VM because storage is not responding. For Windows VMs this happens after 60-second default. Can be caused by path failure, or storage array Is not accepting IO.
DISK	RESET/s	1	The number of commands resets per second.

4.3 Java Troubleshooting Primer

Refer to your JVM documentation for troubleshooting guides.

The following sections provide information about how to begin troubleshooting. Information is given about some severe Java application problems which are GC/memory leakage-related, and some that are thread contention-based. For JDBC-based errors, refer to the JDBC driver provided to you by the database vendor. Of particular importance for performance are errors leading to OutofMemory, Stackoverflow, and Thread Deadlock.

4.3.1 Java Memory Problem Troubleshooting

Consider an example where you observe load increases and decreases over a period of time. If memory continues to build without reclamation (in the worst case), or a GC reclamation occurs but not everything is reclaimed, you may have a memory leak. This is very likely if these symptoms persist to a point where the application suffers from an OutofMemory error. In this case you need to investigate the GC frequency and setting.

- To turn on GC verbose mode:
  - `verbose:gc` — prints basic information about GC to the standard output.
  - `-XX:+PrintGCTimeStamps` – Prints the times that GC executes.
  - `-XX:+PrintGCDetails` – Prints statistics about different regions of memory in the JVM.
  - `-Xloggc:<`*`file`*`>` – Logs the results of GC in the specified file.
- Re-inspect the `-Xmx`, `-Xms`, `-Xss` settings.
- If you're using JDK 6, you can use tool called `jmap` on any platform. Running `jmap` may add additional load on your environment so plan for the best time to run it.
- If you're using JDK 5:
  - If you're running Linux with JDK 5 you can use jmap.
  - If you're using JDK 5 update 14 or later, you can use the `-XX:+HeapDumpOnCtrlBreak` option when starting JVM, then use the Ctrl+Break key combination on Windows to dump the heap.

4.3.2 Java Thread Contention Problem

If you suspect that your enterprise Java application is suffering from long pauses, or just has general response time issues to the point where the JVM needs to be restarted to resolve the issue, you may need to also inspect the Java thread dump. You can obtain a Java thread dump by pressing Ctrl+Break for a Windows OS or in Linux by issuing `Kill -3` on the Java process ID. It is important to take the thread dump right at the point where problematic symptoms appear. This is especially true if you are conducting a benchmark load test—take the thread dump at max peak load, and inspect the behavior of the various application threads.

There are many widely used thread analysis tools that interpret the thread dump and highlight in red the hot threads or threads waiting for a lock. You can begin your code investigation from that point and follow the call stack.

5. FAQ: Enterprise Java Applications on vSphere

With UNIX-based hardware I have very large machines running all of my Java applications. What should be my migration sizing strategy and what are the VMware vSphere maximums that I need to know about?

- One of the most important steps is to conduct a load test to help you determine the ideal individual VM size and how many JVMs you can stack up (vertical scalability). Based on this repeatable building block VM you can scale-out to determine what is best for your application traffic profile.
- Know the VMware vSphere maximums. See *Configuration Maximums: VMware vSphere 4.1*.

The following table provides a summary of maximums for per VM, per Host, and per vCenter.

vSphere Configuration	Maximum
Per VM	• 8 vCPUs • 255GB • 2TB of storage minus 512 bytes
Per Host	• 512 vCPUs • 320 VMs • 25 vCPUs per core
Per vCenter	• 1000 hosts • 10000 Powered on VMs • 15000 registered VMs • 10 Linked vCenter Servers • 3000 hosts in linked vCenter servers • 30000 powered on VMs in linked vCenter Servers • 50000 registered VMs in linked vCenter Servers • 100 concurrent vSphere Clients • 400 hosts per datacenter

What decisions must be made due to virtualization?

You have to determine the size of the repeatable building block VM. This is established by benchmarking, along with total scale-out factor. Determine how many concurrent users each single vCPU configuration of your application can handle, and extrapolate that to your production traffic to determine the overall compute resource requirement. Having a symmetrical building block; for example, every VM having the same number of vCPUs, helps keep load distribution from your load balancer even. Essentially, your benchmarking test helps you determine how large a single VM should be (vertical scalability) and how many of these VMs you will need (horizontal scalability).

You need to pay special attention to scale-out factor, and see up to what point it is linear within your application running on top of VMware. Enterprise Java applications are multi-tier and bottlenecks can appear at any point along the scale out performance line and quickly cause non-linear results. The assumption of linear scalability may not always be true, and it is essential to load test a pre-production replica (production to be) of your environment to accurately size for you traffic.

I have conducted extensive GC sizing and tuning for our current enterprise Java application running on physical. Do I have to adjust anything related to sizing when moving this Java application to a virtualized environment?

- No. All tuning that you would perform for your Java application on physical is transferrable to your virtual environment. However, because virtualization projects are typically about driving a high consolidation ratio, it is advisable that you conduct adequate load testing to establish your ideal compute resource configuration for individual VMs, number of JVMs within a VM and overall number of VMs on the ESX host.
- Additionally, because this type of migration involves an OS/platform change as well as a JVM vendor change, it is advisable to read through Section 2 of this document along with your vendors tuning advice for both OS and JVM.

How many and what size of virtual machines will I need?

This depends on the nature of your application. We most often see 2 vCPU VMs as a common building block for Java applications. One of the guidelines is to tune your system for more scale-out as opposed to scale up. This rule is not absolute as it depends on your organization's architectural best practices. Smaller more scaled-out VMs may provide better overall architecture, but you will incur additional guest Os licensing costs. If this is a constraint then you can tune towards larger 4 vCPU VMs and stack more JVMs on them.

What is the correct number of JVMs per virtual machine?

- There is no one definite answer. This largely depends on the nature of your application. The benchmarking you conduct can determine the limit of the number of JVMs can be stacked up on a single VM.
- The more JVMs you put on a single VM the more JVM overhead/cost of initializing a JVM is incurred. Alternately, instead of stacking up multiple JVMs within a VM, you can instead increase the JVM size vertically by adding more threads and heap size. This can be achieved if your JVM is within an application server such as Tomcat. Then, instead of increasing the number of JVMs you can increase the number of concurrent threads available and resources that a single Tomcat JVM services for your n-number of applications deployed and their concurrent requests per second. The limitation of how many applications you can stack up within a single application server instance/JVM is bounded by how large you can afford your JVM heap size to be and performance. The trade-off of very large JVM heap size beyond 4GB needs to be tested for performance and GC cycle impact. This concern is not specific to virtualization as it equally applies to physical server setup.