



# Optimizing Oracle VM Server for x86 Performance

ORACLE WHITE PAPER | MAY 2017





## Table of Contents

Introduction	1
Oracle VM Architecture—A Review	1
Oracle VM Architectural Features	1
Performance—What to Measure, What to Tune	2
General Goals for VM Performance	3
Performance Considerations for VM Environments	4
Oracle VM Server for x86 Domain Types	4
CPU Resource Management	5
CPU Resource Allocation	5
Distributed Resource Scheduler	6
Reducing CPU Overhead	6
Memory	7
Optimizing Oracle VM Server for x86 CPU and Memory Performance	8
General Tuning Considerations for Guest VMs	8
Hyper-Threading Technology	11
Page Size and CPU Scheduling	12
Huge Pages	12
CPU Scheduler Control	13
Optimizing Network and Disk I/O Performance	13
Network I/O	13
Latest Enhancements	14



Disk I/O	14
Virtual or Physical Disks	14
Repositories on NFS or Block Storage	15
Sparse or Non-sparse allocation	16
Provide a Robust Storage Back End	16
Optimizations Within the Guest VM	16
Don't Ignore Boot Disks	17
Performance Features in Oracle VM Release 3.4	17
Buffers, multi-queue and I/O Request Sizes on recent kernels	17
Conclusions and Recommendations	18
Learn More	19



## Introduction

This paper addresses performance optimization goals and techniques for Oracle VM Server for x86. Oracle virtualization products are designed to give the best results while requiring as little manual effort as possible, but performance can often be improved by tuning and reconfiguration. This paper gives Oracle VM performance recommendations applicable to private and public cloud environments, and both standard servers and engineered systems like the Oracle Private Cloud Appliance..

Performance must be measured to permit appropriate performance expectations and tuning actions, so this paper discusses how to evaluate performance. This paper also describes performance enhancements introduced in recent versions of Oracle VM.

## Oracle VM Architecture—A Review

The Oracle VM architecture (Figure 1) builds on industry-standard technologies: MySQL database, Oracle WebLogic Server, Oracle Linux, and the open source Xen hypervisor. Installation scripts help to automate the installation and configuration of these components. Oracle provides templates and virtual appliances (also called assemblies) containing pre-built virtual machines (VMs) to make deploying applications easy and fast.

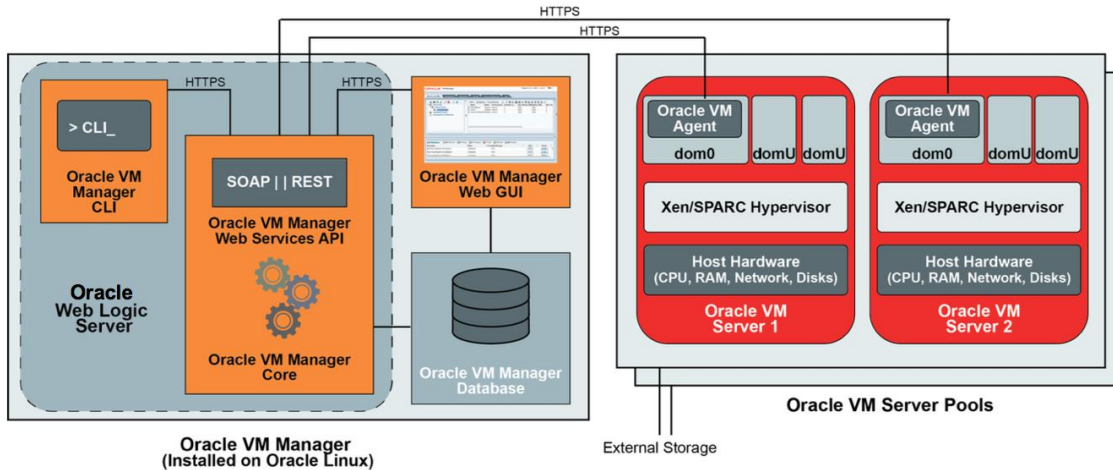


Figure 1: The Oracle VM architecture incorporates state-of-the-art enterprise technologies.

As shown in Figure 1, there are two primary functional areas: pools of Oracle VM Servers (Oracle VM Server for x86 or Oracle VM Server for SPARC) that run the virtual machines, and Oracle VM Manager, which provides a comprehensive management infrastructure through graphical user interface, command line, and Oracle VM web services interfaces (APIs). This paper focuses on virtual machine performance.

### Oracle VM Architectural Features

Oracle VM Server for x86 and Oracle VM Server for SPARC (described here for completeness) have architectural similarities and differences. Both employ a small hypervisor along with a privileged virtual machine, also called a domain, for administration and management of physical and virtual devices (This paper uses the terms “virtual machine” and “domain” interchangeably.)

Oracle VM Server for x86 utilizes the lightweight open-source Xen hypervisor, and domain 0 (**dom0**) is the control domain running a minimized Oracle Linux with Oracle's Unbreakable Enterprise Kernel. Dom0 manages the guest user domains (**domUs**) that host instances of Oracle Linux, Red Hat Enterprise Linux, CentOS Linux, SUSE Linux Enterprise Server, Oracle Solaris, or Microsoft Windows™ operating systems. In contrast, on SPARC, the hypervisor is implemented in server firmware, and uses a Solaris-based control domain.

The similarities and differences regarding how these Oracle VM server products handle systems resources have implications for overall performance optimization.

**TABLE 1. SIMILARITIES AND DIFFERENCES**


	Oracle VM Server for x86	Oracle VM Server for SPARC
CPU	<ul style="list-style-type: none"> <li>» CPUs can be shared, oversubscribed, and time-sliced using a share-based scheduler.</li> <li>» A domain's CPUs can be 'pinned' to specific physical CPUs.</li> <li>» CPUs can be allocated cores (or CPU threads if hyperthreading is enabled).</li> <li>» The number of virtual CPUs in a domain can be changed while the domain is running.</li> </ul>	<ul style="list-style-type: none"> <li>» CPU cores or threads are exclusively dedicated to each domain with static assignment when the domain is "bound."</li> <li>» The number of CPU cores or threads can be changed while the domain is running, either by administrator action or via domain resource management.</li> </ul>
Memory	<ul style="list-style-type: none"> <li>» Memory is dedicated to each domain with no oversubscription.</li> <li>» The hypervisor attempts to assign a VM's memory to a single server node/socket, and uses CPU affinity rules to keep a VM's virtual CPUs near its memory to minimize NUMA latency.</li> </ul>	<ul style="list-style-type: none"> <li>» Memory is dedicated to each domain with no oversubscription.</li> <li>» The hypervisor attempts to assign a VM's memory to a single server node/socket, and uses CPU affinity rules to keep a VM's virtual CPUs near its memory to minimize NUMA latency.</li> </ul>
I/O	<ul style="list-style-type: none"> <li>» Guest VMs are provided virtual network, console, and disk devices managed by dom0.</li> </ul>	<ul style="list-style-type: none"> <li>» Guest VMs are provided a virtual Host Bus Adapter (HBA), network, console, and disk devices managed by the control domain and optional service domains.</li> <li>» VMs can also use physical I/O with direct connection to SR-IOV virtual functions or PCIe buses.</li> </ul>
Domain Types	<ul style="list-style-type: none"> <li>» Guest VMs (domains) may be hardware virtualized (HVM), paravirtualized (PV), or hardware virtualized with PV device drivers (HVMPV).</li> </ul>	<ul style="list-style-type: none"> <li>» Guest VMs (domains) are paravirtualized</li> </ul>

As you can see, the products have different CPU models but a similar memory model. While Oracle VM Server for SPARC uses dedicated CPU cores or threads on servers that have many physical CPUs, Oracle VM Server for x86 uses a more traditional scheduler that time-slices virtual CPUs onto physical CPU threads or cores. Both are aware of NUMA (Non-Uniform Memory Access) effects and try to reduce memory latency. Also, both have virtual networks and virtual disk devices, but Oracle VM Server for SPARC provides additional options for back-end devices and non-virtualized I/O. Oracle VM Server for x86 has more domain types due to the wide range of x86-based operating systems – this has implications for performance, discussed below.

*The primary focus of this article is on Oracle VM Server for x86. See [Oracle VM Server for SPARC Best Practices](#) for detailed information about Oracle VM Server for SPARC virtualization.*

## Performance—What to Measure, What to Tune

A good optimization plan starts with understanding what to measure. System administrators should observe how machine utilization and throughput figures relate to application performance and application requirements.



For example, is high or low CPU utilization good or bad? It depends. High CPU utilization (such as 95 percent busy) could show that you're fully utilizing your investment in hardware. It could also be a symptom of a problem, such as program looping or excessive error handling, or having insufficient capacity for the workload. High CPU utilization is a problem when it prevents meeting service level objectives (response time, throughput) due to resource starvation.

On the other hand, low CPU utilization (such as 10 percent busy) may indicate that the workload is idle or perhaps blocked and waiting for I/O. Low utilization could be a sign of excess capacity or a symptom of a bottleneck.

Performance figures must be interpreted carefully to produce valid results. For example, looking at average CPU utilization over time may hide clues for performance problems hidden within peak loads and spikes. Averaging CPU utilization over multiple CPUs in a VM can be misleading: if some CPUs in a VM are 100 percent busy but the majority are idle, average CPU utilization will be low even though the application is CPU constrained on a non-scalable application.

Administrators should avoid the mistake on focusing on metrics that are easy to measure but may not be relevant to the workload. Sometimes referred to as the “[streetlight effect](#)”, there can be a tendency to collect data just because it's easy to collect – such as average CPU utilization for I/O bound applications, or data rates produced by the Linux/Unix ‘`dd`’ command – not because it bears on the performance situation.

Instead, administrators should look at how well applications are meeting their performance objectives and look for factors that limit performance. Objectives are usually expressed in terms of response times (“95 percent of a specified transaction must complete in *X* seconds at a rate of *N* transactions per second”) or throughput rates (“handle *N* payroll records in *H* hours”). This applies equally to virtualized and non-virtualized computer systems.

---

*Performance tuning can be effective only when we are working with a robust infrastructure. Many performance problems are due to inadequate network or storage resources—no amount of tuning or tweaking can overcome such fundamental deficiencies.*

---

## General Goals for VM Performance

The primary goal should be to satisfy service level objectives that meet users' business needs, and not a profile of machine “speeds and feeds”. That said, there are general performance goals for virtual machine environments.

They should be able to:

- » Run workloads as close to bare-metal performance as possible. While there always will be some overhead, Oracle VM technology is engineered to minimize the ‘cost’ of virtualization.
- » Deploy large and small servers efficiently. Large servers are ideal platforms for consolidation by supporting both small and large workloads on the same system. The driving factors for large servers are economy of scale and their ability to run very large VMs, sometimes with hundreds of virtual CPUs and more than a terabyte of memory. The driving factors for smaller servers are granular acquisition costs and lowest price/performance.
- » Support large and small VMs. The goal should be to support a reasonable number of very large virtual machines and a much larger number of small virtual machines with good performance on all of them.
- » Drive high I/O rates to disk and network. I/O has always been a challenge in VM environments due to the overhead of mapping virtual I/O onto real I/O devices through an intermediary software layer. In particular, it can be difficult to achieve high I/O rates for network-intensive applications. High-speed networks generate high interrupt rates, and servicing these interrupts in a VM environment depends on the context switch rate, cache contents displacement, and the NUMA effects of multi-socket servers. This paper describes how Oracle VM technologies approach these challenges.
- » Scale to dozens of servers, hundreds of VMs, and hundreds of LUNs and paths, all under single management control. Current hosting facilities provide services at high scalability under a common administrative interface. For

example, Oracle Managed Cloud Services operates thousands of servers and many times that number of VMs. Oracle VM and Oracle Enterprise Manager also provide integrated and extensive management capabilities.

- » Do all of the above without much manual tuning or intervention. Large VM and cloud environments have too many VMs, tenants, applications, servers, and devices to provide individual attention.

---

*Oracle VM Release 3.4 provides additional enhancements to improve CPU, disk and network I/O, and Oracle VM Manager response time. See Performance Features in Oracle VM Release 3.4 below.*

---

## Performance Considerations for VM Environments

In general, performance management practices for VM environments are the same as for physical servers: *measure user service levels and hunt down the bottlenecks that restrict performance*. Improving performance in some areas does not eliminate all performance issues—they just move to other parts of the technology stack, generally at a higher level of application performance.

However, some practices are different when dealing with VM environments or the cloud, are discussed below.

## Oracle VM Server for x86 Domain Types

One important configuration choice does not exist in the physical server context: choosing the appropriate domain type for a virtual machine. Guest VMs (domU) on Oracle VM Server for x86 use one of the following domain types:

**TABLE 2. DOMAIN TYPES**


Domain type	Characteristics
Hardware virtualized machine (HVM)	HVM guests run unmodified guest operating systems as if they were on bare metal. This mode supports most x86 operating systems and is available on Intel VT or AMDV servers that have enhancements for virtualization, which includes most current server models.  Because the overhead needed to simulate privileged operation exceptions and handle I/O degrades performance, this mode is not recommended for operating systems that can run in the other domain types.
Hardware virtualized with paravirtualized drivers (PVHVM)	PVHVM guests, also called “HVM+PV,” are similar to HVM except they include paravirtualized device drivers written for Oracle VM, and available for Oracle Linux, Oracle Solaris, and Microsoft Windows.  Paravirtualized device drivers use an optimized code path to interact with the Xen hypervisor, and provide performance benefits like those for PVM guests. Starting with Oracle VM 3.4.2, this mode supports dynamic memory configuration for running Linux guests, which was previously available only for PVM.  Together with the hardware virtualization support of recent server models, this is now <i>recommended for all 64-bit guest operating systems</i> .
Paravirtualized (PVM)	PVM is the original Xen domain type used before hardware virtualization support was added to Intel and AMD processors. Guest operating systems supporting PVM (generally Linux) are modified to interact with Xen rather than issue native instructions for context switching, I/O, and memory management.  PVM is available on current hardware and certain operating systems, such as Oracle Linux 6 and earlier. It is not available with Oracle Linux 7. PVM is only recommended for 32-bit operating systems.

---

*Note: These are general rules that do not cover all cases. Users should test and weigh performance gains or losses against operational effectiveness.*

---

Use PVHVM mode for 64-bit operating systems on recent server models. Use either PVHVM or PVM mode on 32-bit operating systems. PVM mode was previously recommended, but PVHVM performs better on 64-bit guests since



it has lower CPU path-length for system calls and kernel/userspace context switching. Dramatic performance improvements have been seen in some cases just by switching to PVHVM. Performance effects depend on workload, and are highest for I/O-intensive applications that benefit the most from the enhanced code path. There may still be exceptions, so it is a good idea to test.

If you run Oracle VM Server in a virtual machine, for example under Oracle VM VirtualBox for training or testing, the hardware virtualization capability, if present, is consumed by the first-level hypervisor. That means that virtual machines running in an Oracle VM Server virtual machine must use PVM mode. This is generally only for test or training, and not production workloads due to the performance cost of running "third level" virtual machines.

For the full list of supported operating systems and domain types, see the [Oracle VM Server for x86 Supported Guest Operating Systems](#). Note the different configuration limits for the different domain types, as documented in the [Oracle VM Release Notes for 3.4 - Configuration Limits](#).

---

For more information, see the [Oracle VM Concepts Guide for Release 3.4](#).

---

## CPU Resource Management

CPU contention is not usually a cause of Oracle VM performance problems. Modern servers provide ample CPU capacity, and Oracle VM generally doesn't add substantial CPU overhead. Most performance problems occur elsewhere. Nonetheless, it's important to understand and properly manage CPU resources, in particular for compute-intensive workloads that fully utilize compute resources, and on large highly-scaled platforms.


It is possible to drive Oracle VM environments to high utilization levels while continuing to provide good service. This is done by eliminating unproductive overhead that reduces resources available to applications, and by allocating resources to meet service objectives. Overhead is addressed by properly sizing VMs and assigning CPUs to avoid NUMA effects and cache contention. Resource allocation is addressed by CPU assignment, hypervisor policies, priority and cap settings, and by the Distributed Resource Scheduler (DRS). In many instances, Oracle VM defaults are appropriate and don't require changing.

### CPU Resource Allocation

Oracle VM Server for x86 prevents VMs' CPU consumption from interfering with one another by giving each VM its own physical CPUs when possible. When there are sufficient physical CPUs (more physical CPUs than virtual CPUs), virtual CPUs are assigned different physical CPUs to avoid sharing and resource contention. When virtual CPUs are assigned to different physical CPUs they do not directly compete with one another for CPU resources, though they still compete for cache, memory bandwidth, and I/O access. It is possible to manually pin virtual CPUs to physical CPUs to explicitly isolate them from one another, but this is not typically needed or recommended for performance purposes. Note that CPU pinning assigns physical CPUs to a VM: it does not prevent other VMs from running on the same CPU cores.

If there are more virtual CPUs than physical CPUs (physical CPUs are "oversubscribed"), then Oracle VM assigns multiple virtual CPUs to the same physical CPUs, and VMs compete for CPU cycles. There is generally no problem oversubscribing the number of CPUs: the total number of virtual CPUs on a server can safely exceed the number of physical CPUs. However, no individual VM should ever have more virtual CPUs than the number of physical CPUs on a server. That could expose lock management problems or race conditions within the guest. Ideally, no VM should have more virtual CPUs than the number of physical CPUs on a server socket, as discussed below in the section on NUMA overhead.





When physical CPUs are oversubscribed and CPU utilization is high, Oracle VM Server for x86 uses a “credit scheduler” to proportionally allocate CPU cycles based on VM priorities expressed in the Oracle VM Manager user interface. VMs on the same CPU cores compete with each other for CPU cycles, just as processes in a multitasking OS compete for CPU resources – VMs “steal” CPU access from each other.

Everything else being equal, two VMs on the same physical CPUs with the same priority will each get one-half of the shared CPU capacity. Setting a high priority for important VMs provides preferential access to CPU cycles. This works best when there are low-priority VMs with relaxed service requirements that can tolerate less access to CPU resources, and absorb capacity left over from more important VMs. In situations with strict service requirements for all VMs, the best practice is to provision enough capacity to satisfy all VMs’ peak demands. CPU caps can be set to set an upper bound on the amount of CPU capacity a VM can consume.

CPU stealing is easy to detect in virtual machines running Linux or Oracle Solaris: use `vmstat`, `mpstat`, or `iostat` commands to observe “percent steal” time (`%st` or `%steal`). If steal time is nonzero, the VM was runnable and had work to do, but the hypervisor was unable to give it CPU cycles. In that case, Oracle VM resource management priority and cap controls can be used to administer the CPU resources available to each VM. If steal time is zero, there is no reduced performance due to CPU oversubscription.

### Distributed Resource Scheduler

The Distributed Resource Scheduler (DRS) is a powerful tool for balancing load across servers in an Oracle VM server pool. It works by live migrating VMs from servers under heavy load to less-loaded servers according to a server pool policy. Instead of treating each server as an isolated resource, Oracle VM can use all the servers in the pool as a consolidated resource, fix problems with overcommitted servers, and prevent capacity going unused.

DRS periodically measures CPU utilization on each controlled server. If utilization on a server exceeds a specified threshold, DRS selects a VM and live migrates it to a server in the pool that has lower load. The administrator selects the Policies perspective for a server pool, sets the CPU utilization threshold, how often Oracle VM checks utilization, and which servers in a server pool are controlled by the policy. DRS rules can also be used to balance VMs based on network utilization. This automatic process relieves the administrator of manual effort.


For further information, see the blog <https://blogs.oracle.com/jsavit/oracle-vm-features%3a-distributed-resource-scheduler> and the Oracle VM Concepts Guide [http://docs.oracle.com/cd/E64076\\_01/E64081/html/vmcon-svrpool-drs.html](http://docs.oracle.com/cd/E64076_01/E64081/html/vmcon-svrpool-drs.html)

### Reducing CPU Overhead

The other main point of CPU administration is avoiding overhead, especially using the correct domain type, and avoiding NUMA overhead on multi-socket servers which have a large difference between local and remote memory access times. It is more important in VMs than on “bare metal”, because a non-virtualized OS can observe the server’s CPU and memory topology and co-locate CPU and memory allocations for local memory access and cache affinity. This is not generally possible in VMs because physical server details are hidden from the guest.

Oracle VM Server for x86 avoids NUMA latencies by assigning a VM’s vCPUs and memory to the same server socket whenever possible, based on the size of the VM and the available RAM and CPU resources on each server socket. This yields the best CPU-to-memory latency since all memory access is local to the socket.

This is not possible when a VM’s memory or CPU requirements exceed the unused capacity of a single socket. Assigning a virtual machine more vCPUs than it needs can reduce performance. In addition to multiprocessor scalability issues that affect every environment, this creates NUMA latency when a VM’s virtual CPUs map to physical CPUs on different sockets. The result is higher remote memory access times, and reduced processor



cache effectiveness. Many applications don't scale with more CPUs, so adding more virtual CPUs may be pointless, and can hurt performance. VMs should be allocated only as many virtual CPUs as they can use effectively for their applications.

The control domain, dom0, is a special case since it is not administered through the Oracle VM user interface, and is sized when Oracle VM is installed and booted up when a server starts. Oracle VM Server assigns dom0 CPUs on socket 0, and is limited to 20 CPUs on servers with 20 or more CPUs to prevent dom0 from straddling multiple sockets, as described in the [Oracle VM Installation and Upgrade Guide](#). The number of CPUs can be manually adjusted; one use case would be if hyperthreading is turned off on a larger server reducing total CPU count. Note that guest VMs (domU) can run on the same CPU cores as dom0, which has a very high priority so will run before domU when both need CPU access.

## Memory

Memory considerations often play a bigger role than CPU utilization. As mentioned above, managing CPUs also affects memory performance. Memory management in VM environments differs sharply from that of non-virtualized environments.

On bare-metal systems, all the memory on the server is available to the operating system and its applications. Customers typically choose a server model from a list of standard configurations, and excess memory is used for file buffering or goes unused. Because the cost of over-provisioning memory is low, typical memory requirements can be approximate.

However, in a VM environment, such as a cloud, excess memory allocated to a VM reduces the number of VMs that can be hosted at one time, which ultimately increases costs. Within a VM, excess memory can reduce performance by increasing NUMA latency if memory is spread over multiple sockets.

CPU consumption is easily observed in most hypervisors, but memory is harder to measure. The hypervisor knows how much memory has been allocated to each VM, but doesn't know how much of that memory is being used. Standard tools on Linux, Oracle Solaris, and Microsoft Windows, and in management software like Oracle Enterprise Manager can instrument memory activity within VMs. This generally does not provide instrumentation for NUMA effects, which are not easily measurable.

Operating systems potentially swap or page an application's virtual memory to a swap file when there is not enough physical memory to meet application needs. Some hypervisors do the same with virtual machines' memory contents, producing two levels of virtual memory. This can lead to unpredictable bad performance situations under memory pressure when hypervisors and guests swap memory contents. There can be double paging to disk storage, and thrashing of memory contents to swap files rather than productive work. With nested virtual memory environments, the classic operating system algorithms used to select a page for swapping break down, and the most recently used pages are selected for page-out instead of the least recently used (LRU) pages.

Oracle VM Server intentionally does not over-subscribe memory. This eliminates the double paging problem entirely. It is essential to size memory requirements so sufficient memory is configured. Each VM's memory should be sized to avoid swapping within the guest, just as when running natively. Servers must have enough real memory to support the memory requirements of all the VMs they host. Operating systems may require disk space be allocated for swap files, but the rate of swap operations should be zero.

## Optimizing Oracle VM Server for x86 CPU and Memory Performance

This section describes steps to implement tuning consistent with the discussion above. The first step in managing CPU and memory performance is to configure domains consistent with their application requirements and the hardware resources of the server platform.

Oracle VM uses dom0 as the management control point and to provide virtual I/O to guest domains (domU). Dom0 performance is critical to the operation of all domU – if dom0 isn't serviced, then virtual I/O isn't provided for the guests. As mentioned above, Oracle VM configures dom0 on larger servers so its virtual CPUs are pinned to physical CPUs on the first socket of the server, and allocates its memory based on the server's capacity. This eliminates NUMA memory latency on dom0 by having its CPUs and memory on the same socket. Sometimes it is useful to change dom0 memory or CPU settings, as described in the Oracle VM documentation, but the default settings are generally good.

### General Tuning Considerations for Guest VMs

Guest VMs, domU, are explicitly defined by system administrators. Here are guidelines to consider:

- » *Avoid NUMA latency: the number of virtual CPUs should not exceed the number of physical CPUs per socket, and the amount of memory should not exceed the memory per socket.* There is NUMA overhead if CPUs are on more than one socket or if memory is on different sockets from CPUs. That happens if there are too many CPUs or too much RAM for a socket to provide, or very commonly if those resources are partially consumed when a VM is started. It is sometimes necessary to give a VM more CPUs or RAM than are on a single socket because the workload requires more than a socket's capacity. This achieves scale at the cost of NUMA efficiency.

Oracle VM will generally allocate resources optimally, but if necessary, a server's physical configuration of sockets and cores can be determined by logging into dom0 and using the commands shown below. This is normally of consequence only for large VMs or fragmented memory, and isn't usually needed.

Use the `xenpm get-cpu-topology` or `xl info -n` command to display the relationship between CPU number, cores, and sockets. In this example on a two-socket server with hyperthreading, socket 0 has CPU threads 0 to 15 labeled CPU0 to CPU15, and socket 1 has threads 16 to 31. Cores on each socket are numbered from 0 to 7: On this server, a VM with more than 16 virtual CPUs would span two sockets and have NUMA latency overhead. You can see the assignment of physical CPUs by issuing the command `xm vcpu-list`.

```
root # xenpm get-cpu-topology
CPU      core    socket  node
CPU0     0       0       0
CPU1     0       0       0
CPU2     1       0       0
CPU3     1       0       0
CPU4     2       0       0
CPU5     2       0       0
CPU6     3       0       0
CPU7     3       0       0
CPU8     4       0       0
CPU9     4       0       0
CPU10    5       0       0
CPU11    5       0       0
CPU12    6       0       0
CPU13    6       0       0
CPU14    7       0       0
```

```

CPU15    7      0      0
CPU16    0      1      1
CPU17    0      1      1
CPU18    1      1      1
CPU19    1      1      1
CPU20    2      1      1
CPU21    2      1      1
CPU22    3      1      1
CPU23    3      1      1
CPU24    4      1      1
CPU25    4      1      1
CPU26    5      1      1
CPU27    5      1      1
CPU28    6      1      1
CPU29    6      1      1
CPU30    7      1      1
CPU31    7      1      1

```

```
root # xm vcpu-list
```

```

Name                                     ID  VCPU  CPU State  Time(s) CPU Affinity
0004fb00000600000f6355f82c1d1620      10   0    26  -b-    5035.9 16-31
0004fb00000600000f6355f82c1d1620      10   1    31  -b-   17563.9 16-31
...snip...
0004fb00000600000bcc9f9e08d870c8a      9    0    27  -b-    4746.4 16-31
...snip...

```

The following commands show how memory is assigned to sockets with different “NUMA nodes”. We see two NUMA nodes, one per socket, their starting memory addresses and sizes, and which CPUs are local to each. We can also see that memory for domains 37 and 38 are contained on a single socket, but domain 40 has memory on both sockets. If there is a performance problem with that domain it might be useful to reduce its size, or migrate it to a server with sufficient available memory on a single socket. For other examples, see Wim Coekaerts' blog at <https://blogs.oracle.com/wim/understanding-memory-allocation-in-oracle-vm-xen>

```
root # xm debug-key u ; xm dmesg
```

```

...snip...
(XEN) 'u' pressed -> dumping numa info (now=0x3FE1E0:2A072134)
(XEN) idx0 -> NODE0 start->0 size->34078720
(XEN) phys_to_nid(00000000000001000) -> 0 should be 0
(XEN) idx1 -> NODE1 start->34078720 size->33554432
(XEN) phys_to_nid(0000002080001000) -> 1 should be 1
(XEN) CPU0 -> NODE0
(XEN) CPU1 -> NODE0
(XEN) CPU2 -> NODE0
(XEN) CPU3 -> NODE0
(XEN) CPU4 -> NODE0
(XEN) CPU5 -> NODE0
(XEN) CPU6 -> NODE0
(XEN) CPU7 -> NODE0

```

```
(XEN) CPU8 -> NODE0
(XEN) CPU9 -> NODE0
(XEN) CPU10 -> NODE0
(XEN) CPU11 -> NODE0
(XEN) CPU12 -> NODE0
(XEN) CPU13 -> NODE0
(XEN) CPU14 -> NODE0
(XEN) CPU15 -> NODE0
(XEN) CPU16 -> NODE1
(XEN) CPU17 -> NODE1
(XEN) CPU18 -> NODE1
(XEN) CPU19 -> NODE1
(XEN) CPU20 -> NODE1
(XEN) CPU21 -> NODE1
(XEN) CPU22 -> NODE1
(XEN) CPU23 -> NODE1
(XEN) CPU24 -> NODE1
(XEN) CPU25 -> NODE1
(XEN) CPU26 -> NODE1
(XEN) CPU27 -> NODE1
(XEN) CPU28 -> NODE1
(XEN) CPU29 -> NODE1
(XEN) CPU30 -> NODE1
(XEN) CPU31 -> NODE1
(XEN) Memory location of each domain:
(XEN) Domain 0 (total: 1457660):
(XEN)   Node 0: 694143
(XEN)   Node 1: 763517
(XEN) Domain 37 (total: 2098164):
(XEN)   Node 0: 2098164
(XEN)   Node 1: 0
(XEN) Domain 38 (total: 2098164):
(XEN)   Node 0: 2098164
(XEN)   Node 1: 0
(XEN) Domain 40 (total: 7865332):
(XEN)   Node 0: 3932666
(XEN)   Node 1: 3932666
```

- » *Never give an individual virtual machine more virtual CPUs than the number of physical CPUs on the server.* Giving more CPUs than CPUs per socket may be necessary, but giving a VM more CPUs than are on the server always harms performance and can lead to errors. On the server above, no virtual machine should have more than 32 virtual CPUs, and ideally has no more than 16 so it can fit on a single socket.
- » *The total number of virtual CPUs over all VMs can be higher than physical CPUs.* CPU oversubscription (in terms of number of virtual CPUs rather than their utilization) is not a cause for concern. There is some overhead for running the CPU scheduler, so it's best to not allocate more CPUs than are needed, but this is a mild effect.
- » *Avoid excessive oversubscription of active virtual CPUs (those with high percentage of CPU load).* It's not the number of virtual CPUs that's the problem—it's distributing a fixed resource (a server's total CPU capacity) over

more requesters that can degrade performance. Time-slicing CPU cycles for many active consumers just means that they get smaller slices. This is classic queuing theory: service times (elapsed time to get service) get longer as capacity approaches saturation.

To avoid this, monitor CPU utilization at the system level (Oracle Enterprise Manager provides observability, as does Oracle VM Manager's **Health** tab) to see if servers are approaching full utilization. Be cautious about averages: a smoothed average over time of, say, "75 percent CPU busy" may hide peak intervals that were 100 percent busy. In the guest, use `vmstat` and `mpstat` to observe CPU consumption. Guest VM CPU activity does not appear in `dom0`, so commands to observe CPU utilization should be run in the guest VMs. For systematic data collection, use the OSWatcher tool, which can run in `dom0` and in guest VMs. For further details, see the section [Diagnostic Tools for Oracle VM Server](#) in the [Oracle VM Administrator's Guide](#).

Reaching CPU saturation is not necessarily a problem—it means you're utilizing full capacity. CPU saturation with unsatisfied resource demand, as shown by non-zero `%steal` values in `vmstat` and `mpstat`, may be a problem. However, even a high steal rate is not a problem for low priority "cycle soaker" VMs with compute-intensive workloads and lax service levels that can run when there are cycles not needed by other VMs.

- » *Use the appropriate domain type* – PVHVM is recommended in most cases, as described earlier.

## Hyper-Threading Technology

---

*Intel® Hyper-Threading Technology (HTT) is available on the Intel® Core™ processor family, the Intel® Core™ M processor family, and the Intel® Xeon® processor family.*

---

HTT improves performance in some cases, but degrades it in others. With HTT enabled, each CPU core has two CPU threads instead of one. A server with 8 cores would appear to have 16 CPU threads. Each thread is treated as a dispatchable CPU, time-sliced (in hardware) onto the core's resources.

If one thread is idle or stalls on a cache miss, the other thread can continue to execute. This potentially improves throughput, especially for workloads whose memory accesses have frequent cache misses that stall the instruction stream. In those cases, HTT lets the CPU continue processing by running the other thread rather than stall and do nothing productive. On the other hand, both threads compete for the core's resources, and each thread may run slower than if it owned the entire core. Not only are they competing for the core's logic resources, but each thread potentially displaces the other thread's level 1 cache (L1\$) contents, causing both threads to run slower. This is especially visible for compute intensive workloads that might normally fit in cache without HTT.

Each thread is assignable to a VM virtual CPU by the hypervisor, and is dispatched as a CPU. The performance effect is workload dependent, so it must be tested on different workload types.

- » Linux and other guest operating systems try to spread work over cores, but guests don't know if virtual CPUs are on the same core or socket. Real topology data isn't available, so the guest dispatcher may make poor scheduling choices that increase overhead.
- » HTT can help scalable applications when the number of vCPUs is greater than the number of cores per socket but less than the number of threads per socket. This can keep CPU allocation on the same socket.
- » HTT can keep a core busy when a CPU thread stalls on a cache miss, which can increase overall throughput. In general, HTT is best for multithreaded applications that can drive multiple vCPUs, but it can reduce per-vCPU performance and affect single-thread-dominated applications. It needs to be evaluated in the context of the application.

To determine if HTT is available, log into `dom0` on Oracle VM Server and issue the commands shown below:

```

root# dmidecode | grep HTT
           HTT (Hyper-Threading Technology)
           HTT (Hyper-Threading Technology)

root# egrep 'siblings|cpu cores' /proc/cpuinfo | head -2
siblings      : 2
cpu cores     : 1

root# grep '^flags\b' /proc/cpuinfo | tail -1
flags          : fpu de tsc msr pae mce cx8 apic sep mca cmov pat clflush acpi mmx
fxsr sse sse2 ss ht syscall nx lm constant_tsc rep_good nopl nonstop_tsc pni
pclmulqdq est ssse3 cx16 sse4_1 sse4_2 popcnt aes fl6c rdrand hypervisor lahf_lm ida
arat epb pln pts dts fsgsbase erms
root# dmidecode | grep Count          # tests that HTT is available *and* enabled
Core Count: 1
Thread Count: 2
Core Count: 1
Thread Count: 2

```

## Page Size and CPU Scheduling

Virtual memory page size and CPU scheduling policies can also be tuned for better performance.

### Huge Pages

All modern CPUs divide real and virtual memory into uniformly sized “pages” of memory. Each virtual memory page is individually mapped to a real memory page, requiring its own page table entry (PTE) in a page translation table.

Intel (and compatible) servers have a page directory and a page table, such that each entry in the page directory points to a different page table of 1,024 PTEs, with each PTE mapping one page and occupying 4 bytes. With the default page size of 4 KB, a process with an address space of 1 GB requires 262,144 four-byte entries—a megabyte of memory for each gigabyte of address space. This becomes very expensive in terms of memory footprint and CPU time to search and update the tables. The overhead can be reduced by enabling “huge pages” supporting a 2 MB page size, yielding a 512:1 reduction in page table size and CPU overhead. This makes a big difference for large memory applications, such as Oracle Database with a large system global area (SGA).

Huge pages are recommended for PVHVM and HVM guests, especially for large-memory 64-bit virtual machines that benefit from page table size reduction and work best in those domain modes. No Oracle VM administrative effort is needed to enable huge pages for HVM and PVHVM guests. Instead, this is configured within the guest operating system. Oracle Linux administrators enable the “HugePages” feature as described in the [Oracle Linux Administrator’s Guide](#). Also see MOS notes “HugePages on Linux: What It Is... and What It Is Not... (Doc ID 361323.1)” and “HugePages on Oracle Linux 64-bit (Doc ID 361468.1)”. For Oracle Solaris guests, the available page sizes are displayed with the command `pagesize -a`.

Huge pages are deprecated for PVM virtual machines. Huge page and default page size VMs should not be mixed on the same server, because allocation for small page guests can cause memory fragmentation that could prevent a huge page guest from starting.

---

*For more details, see the discussion of huge pages in the [Oracle VM Concepts Guide for Release 3.4](#).*

---

## CPU Scheduler Control

Oracle VM Manager provides settings you can use to manage the CPU resources a VM can consume. These settings can be made when a VM is created or by editing an existing VM.

- » **Maximum Processors / Processors:** Sets the number of CPU processors, up to 128, that can be allocated to the VM. The **Maximum Processors** value can be changed only when the VM is not running. The **Processors** value can be changed while the VM is running, up to the **Maximum Processors** value.

If HTT is enabled, “processor” is a CPU thread; otherwise, it’s an entire core. Regardless of priority or cap values, a VM can consume no more CPU resources than the number of virtual CPUs it has. VMs running single-threaded non-scalable applications will not benefit from adding more virtual CPUs. Even reasonably scalable applications may demonstrate diminishing returns from additional CPUs.

- » **Priority:** Represents a share or weight setting that expresses the VM’s relative importance for CPU access compared to other VMs. The higher the priority, the more CPU cycles it will obtain relative to other VMs when there is contention for CPU cycles. The default value is 50; higher or lower priorities assign relative importance. When there is excess CPU capacity, this value has no effect.
- » **Processor Cap %:** Sets the maximum CPU consumption a virtual CPU can have. The default is 100 percent, indicating that the VM’s virtual CPUs can be given all the CPU capacity they can consume, subject to availability and their priority. Set this to a lower number if you want to throttle the VM’s CPU consumption. If you want the VM to consume as many cycles as it needs when no other VM needs CPU cycles, leave the default Cap value but set a low priority.

As mentioned earlier, run monitoring tools such as `vmstat`, `mpstat`, and `iostat` on the VM to get the best indication of CPU resource needs. Those tools report “steal” time when the VM wanted to run but was prevented because CPU cycles weren’t available. Nonzero steal values indicate that a VM could benefit from higher priority. Use `mpstat` to see if more CPUs would help performance (`mpstat` would show most CPUs busy—if `mpstat` shows idle virtual CPUs, there could be no benefit to adding CPUs). A very helpful tool is OSWatcher, which can be used to systematically collect performance data (see MOS note OSWatcher Analyzer User Guide (Doc ID 461053.1))

---

*For more details, see the [Oracle VM Manager User's Guide for Release 3.4](#).*

---


## Optimizing Network and Disk I/O Performance

### Network I/O

Network performance is an essential component of overall system performance, especially in a virtual machine environment. Networks are used in separate roles: for live migration, for cluster heartbeats, for access to virtual disks on NFS or iSCSI devices, for system management, and for the guest virtual machines that own network traffic.

These roles, and other aspects of Oracle VM networking, are explained in the article [Looking "Under the Hood" at Networking in Oracle VM Server for x86](#) by Greg King and Suzanne Zorn. Similar helpful Oracle VM white papers can be found at the end of this article.





dom0 has physical access to the network adapter cards, so it has native I/O performance. The tips to use at the hypervisor level are the same as in a physical network environment, such as using bonding for resiliency and isolation, using large MTU sizes (as long as all network devices in the transmission path are enabled for large MTU sizes), and using separate networks for latency and isolation. In particular, protect the Oracle VM cluster heartbeat from being delayed by heavy network I/O for storage or VM access. The same applies to heartbeats for guests using Oracle Real Application Clusters (Oracle RAC) and other cluster technologies that are sensitive to packet delay.

Guest VM network traffic rides on virtual networks with the “VM” channel defined. Many of the same rules apply here as with physical networks, especially the rule for using separate networks for different types of traffic.

For more network tuning advice, see the white paper [Oracle VM 3: 10GbE Network Performance Tuning](#). Driving a 1 GbE network at line speed is no challenge for today’s servers, but can be more difficult with 10 GbE and higher network speeds. Surprisingly, the most important tuning actions on large servers relate to controlling NUMA latency. Many of the outlined tuning steps are now Oracle VM defaults, reducing the amount of tuning required.

### Latest Enhancements

Oracle VM Release 3.4 added network performance improvements:

- » Network performance shows double-digit improvement for both PVM and PVHVM guests, with the larger increases for PVM guests.
- » Xen hypervisor *netback/netfront multi-queue* further improves network performance 80 percent and better than physical line speed on 20 Gb/sec bonds.
- » PVM guests outperform PVHVM in some cases, so PVM should be considered when maximal network performance is a top priority. PVHVM mode still remains the general recommendation for 64-bit guests, and it generally outperforms PVM mode. PVM mode is available only for Oracle Linux 6 and earlier.

### Disk I/O

Disk I/O performance is critical for most applications, especially commercial workloads that are typically database- or file-intensive. Providing good disk I/O performance is the most important performance need for many workloads.

---

*A good resource on disk I/O performance is Brendan Gregg's book [Systems Performance: Enterprise and the Cloud](#). Oracle VM's use of storage is described in the [Oracle VM Concepts Guide for Release 3.4](#).*


---

There are disk I/O requirements for Oracle VM Manager and for the Oracle VM Server on each host, but those are lightweight in nature and not in the direct I/O path for workloads. Requirements are illustrated in the [Oracle VM Installation and Upgrade Guide for Release 3.4](#). Most attention should be on virtual machine disk performance.

Virtual machine disk images typically reside on shared storage, as described in the white paper [Oracle VM 3: Architecture and Technical Overview](#). Oracle VM Servers are grouped into server pools where every server has access to shared storage, which can be NFS, Fibre Channel, or iSCSI. This allows VMs in the pool to run on any physical server in the pool.

Local storage (a server’s builtin disks) can also be configured, but is often not appropriate for production, because it can be a single point of failure. However, Solid State Drives (SSD) and Non-volatile Memory Express (NVMe) devices have high performance and make local server storage more attractive.

### Virtual or Physical Disks



The first important deployment choice when configuring a virtual machine is whether to use virtual disks in a repository or physical disks, as described in [Understanding Repositories](#) and [How are Virtual Disks Managed?](#) in the *Oracle VM Concepts Guide for Release 3.4*. The main considerations are:

- » **Virtual disks** are disk image files in a repository. Virtual disks provide good performance, permit scalability, and help resource sharing: a single LUN or NFS share can host hundreds of virtual disks. They are easy to administer, and simplify cloning VMs and using templates. Repositories are always used for metadata about each VM, and optionally may contain virtual disk images, ISO images, assemblies, and templates. Keep in mind that storage bandwidth and IOPS are shared by all the VMs in a repository, so it is important to not excessively load too many active virtual disks in the same repository. Use `iostat` and `nfsiostat` (if NFS is used) in `dom0` to see if the disk resource for a repository is busy and experiencing higher service times, and spread work over multiple repositories for higher performance and to isolate VMs from one another.
- » **Physical disks** are LUNs accessed directly by virtual machines: each LUN becomes a disk visible to the VM. LUNs are presented to the Oracle VM server, which makes them available to its VMs. Physical disks generally provide much better performance than virtual disks but require additional administrative effort to create LUNs and present them to each physical server in a server pool. Use them when performance is important. They are strongly recommended for Oracle RAC.

In practice, most VM disk images are virtual disks in repositories. Use physical disks when needed for higher performance. Bear in mind that there are limits on the number of physical disks and paths that can be presented to a server, as described in the [Oracle VM Release Notes](#), so they should be used selectively for performance..

### Repositories on NFS or Block Storage

Repositories are always needed, whether or not they are used for virtual disks. Repositories should be hosted on a storage back end that provides sufficient performance and capacity, or they will be a bottleneck for the entire environment.

The primary decision is whether to base repositories on a LUN block device or on an NFS share:


- » **A LUN block device** is LUNs from a storage array, presented to each server in the pool that will use the repository, and formatted with the OCFS2 file system (Oracle Cluster File System).
- » **An NFS share** is from a file system exported to servers in the pool.

There may be no clear-cut advantage of one over the other. Performance seems to be equivalent except for cloning (explained below). Some performance reports have shown a few percentage points for one over the other. There are high-speed transports (Fibre Channel LUNs, 10 GbE for iSCSI LUNs or NFS) in all cases, and the limiting constraint may well be the robustness of the storage underlying any of them rather than the protocol.

The main advantage of LUN block-device repositories is that they permit *thin cloning*, where clones are created using reference links to common blocks. Cloning a VM or template is almost instantaneous and consumes only as much disk space as the differences between the cloned images and their source. The disadvantage is administration: it's harder to grow a LUN and its OCFS2 file system, for example. An OCFS2 file system on a single LUN potentially represents a single point of failure if the file system is corrupted.

The main advantage of an NFS-based repository is that it is easier to set up and administer. Backing it up is easy, increasing its size is easy, too, by just increasing the quota on the file system it resides on. Performance over 10 GbE networks with a large MTU size gives good results. However, cloning a template or VM requires copying complete disk images rather than aliasing them, and consumes both space and time.

Also, NFS permits per-virtual disk analytics on the storage subsystem. With block-storage repositories, all of the virtual disks in a repository are on a single LUN and the storage server can't distinguish one from the other. With NFS-based storage, each virtual disk is a different file, and file servers such as Oracle ZFS Storage Appliance can



show individual per-vdisk I/O statistics. That can be incredibly valuable for answering questions such as "which VM is hammering storage?" This can and should be done with individual physical disks too, as mentioned earlier.

### **Sparse or Non-sparse allocation**

When a virtual disk is created for a VM, the administrator chooses between sparse and non-sparse allocation. With sparse allocation, the space consumed by the disk is initially small, and grows as new blocks on the disk are written. With non-sparse disks, the entire disk blocks are allocated when the disk is created. Creating a sparse allocation disk is faster than non-sparse, but there can be a performance penalty when the guest VM writes to previously unused blocks, causing the virtual disk to grow. Eventually, if the disk is populated with data, most blocks will have been used at some point so all the underlying storage will have been allocated, and there will be no further performance impact.

Note: sparse allocation is useful for over-subscribing underlying storage, as you can allocate more apparent disk space than exists if the space is not expected to be used. If this is done, the administrator should monitor disk space consumption in the repository and storage array to make sure that physical space isn't exhausted.

### **Provide a Robust Storage Back End**

Regardless of protocol or interconnect, virtual machine disk performance is determined by the underlying physical storage performance, just as with non-virtualized environments. A slow storage back end or a low-speed storage network will result in bad performance regardless of other decisions. There must be enough disk drives (if rotating media is being used), RAM, cache, and solid-state disk (SSD) and a fast enough interconnect to meet application requirements.

Many storage arrays provide cache and SSD, which improves performance by eliminating rotational and seek latency. This can increase performance until cache or SSD becomes full, so these resources should be sized and monitored to ensure they are adequate for the workload.

Write-optimized SSD and non-volatile disk cache are valuable for write-intensive workloads, just as in non-virtual machine contexts. In ZFS terms, that means "logzilla" SSDs for the ZFS Intent Log (ZIL) to quickly store to SSD blocks that will later be written to the regular part of the ZFS pool. Read-intensive workloads on ZFS benefit from memory used as a large Adaptive Replacement Cache (ARC) and read-optimized SSDs as an extension of memory cache (L2ARC). The better the hit ratio from memory or SSD, the fewer I/O operations need to be satisfied from slower rotating media. Some storage arrays are SSD-only, and have no issue overflowing from SSD to rotating disk.


Otherwise, I/O is limited to the 150 to 300 IOPS that a rotating media device can deliver, multiplied by the number of disks and amount of parallelism that can be achieved. When rotating media is used, spread the workload over sufficient spindles to avoid excessive device utilization. Disks with high utilization deliver high service times. As with non-virtualized environments, use tools like `iostat` to watch for high device %busy and high service times.

When ZFS is used for the storage backend, a ZFS mirror is preferred over RAIDZ1 or RAIDZ2 because it permits higher IOPS while retaining redundancy to protect against media errors.

### **Optimizations Within the Guest VM**

Within the guest VM, the main tuning action is to make sure hardware virtualized guests use the paravirtualized device drivers. This will outperform native device drivers by a large margin. Guests configured as PVM or HVM with PV drivers behave the same in this regard.

Applications that try to do seek optimization will see little or no benefit to guest VMs: what guests see as contiguous disk blocks may be scattered over multiple physical disks. Trying to schedule seeks will only add unnecessary



overhead. If the guest OS provides control over seek optimization, it might be worth disabling it, as discussed in My Oracle Support Note “I/O Scheduler Selection to Optimize Oracle VM Performance” (Doc ID 2069125.1).

Another consideration is how virtual disks are partitioned. Oracle VM repositories use an underlying block size of 4096 bytes, and a virtual disk partition that is not aligned on a 4K boundary may have suboptimal performance because VM writes and reads will have to be split over multiple repository blocks. Partitions should be allocated on a 4K boundary or larger, meaning that starting sector numbers should be divisible by 8 (512 byte sectors times 8).

For further details, see in My Oracle Support Note “Optimise Oracle VM 3.x Oracle Linux guest write I/O performance through guest/host partition boundary alignment” (Doc ID 1632576.1), and the white paper [Aligning Partitions to Maximize Storage Performance](#).

### Don't Ignore Boot Disks

Besides application I/O, there's I/O for the guest operating systems. You can get “boot storms” when many systems are restarted after an outage. Every VM starting up has to load its OS images and fire up processes from its boot drives. This sometimes catches people by surprise; they think they can avoid planning for OS and boot disk performance because “those have only light I/O load”—but that's not always true. Treat boot disks like other disks to avoid unpleasant surprises.

One performance benefit is available automatically when you use thin-cloned VMs on OCFS2-based LUN repositories. VM boot disks cloned from the same template or VM source have many disk blocks in common, at least until the guest operating systems are individually upgraded. The first OS to read a disk block warms the disk read cache (if there is one) and other guests reading the same block get quick access directly from the cache.

## Performance Features in Oracle VM Release 3.4

Oracle VM Release 3.4 incorporates many new features that substantially improve VM performance without additional administrative overhead. This release also delivers better response times in the Oracle VM Manager user interface and increases the limits on the number of VMs and servers allowed and the resources they use. Additionally, Oracle Linux customers can tune I/O performance by changing certain default settings.

Actual results will depend on the infrastructure and workload. A storage environment that is already fully saturated won't see much improvement just by tuning these parameters. However, these changes will help Oracle VM environments fully exploit the capabilities of the underlying hardware and approach bare-metal performance:

- » Increased Scale: Three times as many VMs per host allowed; vCPUs per host raised from 900 to 4096; maximum vdisk size on NFS raised from 2 TB to 10 TB and on OCFS2 from 10 TB to 64 TB.
- » Support for Fibre Channel over Ethernet (FCoE) and Unified Extensible Firmware Interface (UEFI).
- » Disk I/O improvements mentioned below.
- » Network performance improvements for paravirtualization and netfront/netback multi-queue drivers, mentioned in the section Latest Enhancements *above*.

### Buffers, multi-queue and I/O Request Sizes on recent kernels

Features introduced with Oracle VM Release 3.4 can improve guest disk I/O performance for Oracle Linux guests by increasing the number of in-flight I/O requests, the amount of data transferred in each request, and the number of hardware queues to drive simultaneously. These enhancements apply to Oracle Linux guests, and require administrative steps in the guest domains' boot parameters. Since these changes require efforts in each guest, they should be considered if general performance measures, such as using physical disk have been performed and there

is still a performance shortfall. They are not a substitute for providing adequate physical resources and proper domain configuration, but can drive well configured systems to higher I/O rates. There is no “best number” – tuning these settings requires testing different values under load.

Guests with paravirtualized drivers (PVM or PVHVM) process disk I/O through a ring buffer shared between domU (guest VMs) and dom0. By default, the ring buffer has 32 entries of 128 KB bytes, each representing an in-flight I/O request. I/O requests larger than the buffer size are split into multiple requests, which can decrease performance.

With recent Oracle Linux kernels, the maximum size of an I/O request can be controlled by a GRUB kernel parameter, `xen-blkfront.max_indirect_segments`, measured in 4 KB units. The maximum value for this setting is 256 (`xen-blkfront.max_indirect_segments=256`), which enables I/O requests up to one 1 MB (1024 KB) without breaking them into multiple requests. Higher values can improve streaming sequential I/O. This parameter can be set on Oracle Linux 6 and Oracle Linux 7 when running the Unbreakable Enterprise Kernel (UEK). Some older UEK kernels use the variable name `blkfront.max`. You can determine which name to use, or whether the capability is available at all, by logging into the VM and issuing the command `modinfo xen-blkfront` to display the available parameters.

If you know that the application block size is bigger than the default (`iostat` in the guest VM could tell, as would knowledge of the application), increasing the buffer size could improve performance.

If the storage devices aren't saturated and you suspect the system can handle more concurrent I/O (especially for physical disk rather than vdisk from a repository), there could also be a benefit to increasing the number of concurrent I/O operations (buffer pages). Another setting, `xen-blkfront.max_ring_page_order`, controls the number of pages used for the buffers. Increasing the number of pages permits more concurrent in-flight disk I/O operations. Pages for shared ring buffers must be allocated in units of powers of two, and can be adjusted by setting the exponent in  $2^N$ . `xen-blkfront.max_ring_page_order=4` would permit  $2^4 = 16$  ring buffers pages, and would support as many in-flight operations as fit in that number of pages based on the buffer size. This feature is available in Oracle Linux 7.2 and later, and requires Unbreakable Enterprise Kernel 3.8.13-90 or later.

A further setting, `xen-blkfront.max_queues`, controls the number of hardware queues/rings used per virtual disk. This applies to devices that support multiple I/O queues, which currently is exclusive to NVMe devices. The number of queues is the larger of the number of CPUs in dom0 and in the guest: if dom0 has 20 vCPUs and the guest has 4 CPUs, then it can meaningfully be set to 4; if the guest has 32 vCPUs it can go up to 20. A tuned guest with a recent Oracle Linux kernel and NVMe disk can achieve I/O rates close to the NVMe rated speed.

The variables can be set in the `/etc/modprobe.conf` domU configuration file, for example

```
options xen-blkfront.max_indirect_segments =256
options xen-blkfront.max_ring_page_order=4
```

Or they can be set on the guest kernel line in GRUB kernel line like this (split over multiple lines for illustration):

```
kernel /vmlinuz-3.8.13-90.1.4.el6uek.x86_64 ro root=LABEL=/ \
SYSFONT=latacyrheb-sun16 \
LANG=en_US.UTF-8 KEYTABLE=us \
xen-blkfront.max_indirect_segments=64 xen-blkfront.max_ring_page_order=4
```

---

See the [Oracle VM Release Notes for 3.4.2](#) for details.

---

## Conclusions and Recommendations

Oracle VM products are designed for out-of-the-box performance whenever possible. However, optimization tuning may still be needed to achieve service level requirements. Here are some golden rules discussed in this paper for meeting performance goals in VM and cloud environments:

- » Apply best practices (which are often the defaults).
- » Keep current with Oracle VM releases to reap the benefits from the latest enhancements. Release 3.4, in particular, added a number of significant performance improvements.
- » Measure and tune to target workloads, not to abstract average or benchmark statistics.
  - » Use the appropriate tools to measure and monitor baseline performance before performance problems appear.
  - » Use baseline figures so you can discover trends that might lead to a performance problem.
  - » Use metrics and tests that are meaningful to application workloads and service level requirements, such as throughput and response times.
- » Avoid excessive oversubscription of active virtual CPUs.
- » Provide a robust infrastructure from the start, including a network and back-end storage capacity sufficient to handle expected workloads, that can scale effectively.

## Learn More

For more information about Oracle virtualization products, visit <http://www.oracle.com/us/technologies/virtualization>, call +1.800.ORACLE1 to speak to an Oracle representative, or visit the web resources below.

**TABLE 3. RESOURCES**

Oracle virtualization overview	<a href="http://www.oracle.com/us/technologies/virtualization/overview/index.html">http://www.oracle.com/us/technologies/virtualization/overview/index.html</a>
"Oracle VM 3: Architecture and Technical Overview" white paper	<a href="http://www.oracle.com/us/technologies/virtualization/ovm3-arch-tech-overview-459307.pdf">http://www.oracle.com/us/technologies/virtualization/ovm3-arch-tech-overview-459307.pdf</a>
Oracle virtualization documentation	<a href="http://docs.oracle.com/en/virtualization">http://docs.oracle.com/en/virtualization</a>
Oracle VM Release 3.4 documentation	<a href="http://docs.oracle.com/cd/E64076_01/index.html">http://docs.oracle.com/cd/E64076_01/index.html</a>
Additional Oracle VM white papers	<a href="http://www.oracle.com/technetwork/server-storage/vm/overview/index.html">http://www.oracle.com/technetwork/server-storage/vm/overview/index.html</a>
Xen hypervisor project	<a href="http://www.xenproject.org/users/virtualization.html">http://www.xenproject.org/users/virtualization.html</a>
"Oracle Server for SPARC Best Practices" white paper	<a href="http://www.oracle.com/technetwork/server-storage/vm/ovmsparc-best-practices-2334546.pdf">http://www.oracle.com/technetwork/server-storage/vm/ovmsparc-best-practices-2334546.pdf</a>
"Oracle VM 3: 10GbE Network Performance Tuning" white paper	<a href="http://www.oracle.com/technetwork/server-storage/vm/ovm3-10gbe-perf-1900032.pdf">http://www.oracle.com/technetwork/server-storage/vm/ovm3-10gbe-perf-1900032.pdf</a>







**Oracle Corporation, World Headquarters**

500 Oracle Parkway  
Redwood Shores, CA 94065, USA

**Worldwide Inquiries**

Phone: +1.650.506.7000  
Fax: +1.650.506.7200

CONNECT WITH US

-  [blogs.oracle.com/oracle](https://blogs.oracle.com/oracle)
-  [facebook.com/oracle](https://facebook.com/oracle)
-  [twitter.com/oracle](https://twitter.com/oracle)
-  [oracle.com](https://oracle.com)

**Integrated Cloud Applications & Platform Services**

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. This document is provided for information purposes only, and the contents hereof are subject to change without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document, and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group. 0615

Optimizing Oracle VM Server for x86 Performance  
May 2017  
Author: Jeff Savit