Best Practices for Performance Tuning of Latency-Sensitive Workloads in vSphere VMs

TECHNICAL WHITE PAPER

vmware

Table of Contents

Introduction 3
BIOS Settings 3
NUMA 4
Choice of Guest OS 5
Physical NIC Settings 5
Virtual NIC Settings 6
VM Settings 7
Polling Versus Interrupts 8
Guest OS Tips and Tricks 9
Appendix 10

Introduction

The vSphere ESXi hypervisor provides a high-performance and competitive platform that effectively runs many Tier 1 application workloads in virtual machines. By default, ESXi has been heavily tuned for driving high I/O throughput efficiently by utilizing fewer CPU cycles and conserving power, as required by a wide range of workloads. However, many applications require I/O latency to be minimized, even at the expense of higher CPU utilization and greater power consumption.

We recently investigated the performance of an in-memory, distributed data management platform, measured in the number of puts/second of 1KB data objects from one VM on one ESXi host to another VM on a different ESXi host across physical Ethernet networks. The performance benchmarks for this platform are very latency sensitive, in that any added latency in delivering the packets containing the data objects from one VM to another directly impacts the number of puts/sec observed.

Following the recommendations described in this paper helped us improve from 4400 puts/sec when we started, to 7200 puts/sec for a specific single-threaded, single vCPU application performance benchmark, which is 78% of the 9200 puts/sec achieved on bare metal on the same hardware. This specific benchmark is serial in nature, with only one outstanding put at a time.

We also investigated the performance of two latency micro-benchmarks, one for InfiniBand devices and another for networking devices, in VM DirectPath I/O (passthrough) mode, which bypasses most of the virtualization stack except the path for delivering interrupts from the underlying physical devices to the guest OS. Applying the recommendations reduced latency and therefore increased the score of these latency micro-benchmarks in a virtualized environment, bringing them closer to bare metal performance.

This white paper summarizes our findings and recommends best practices to tune the different layers of an application's environment for similar latency-sensitive workloads. By latency-sensitive, we mean workloads that require optimizing for a few microseconds to a few tens of microseconds end-to-end latencies; we don't mean workloads in the hundreds of microseconds to tens of milliseconds end-to-end-latencies. In fact, many of the recommendations in this paper that can help with the microsecond level latency can actually end up hurting the performance of applications that are tolerant of higher latency.

Please note that the exact benefits and effects of each of these configuration choices will be highly dependent upon the specific applications and workloads, so we strongly recommend experimenting with the different configuration options with your workload before deploying them in a production environment.

BIOS Settings

Most servers with new Intel and AMD processors provide power savings features that use several techniques to dynamically detect the load on a system and put various components of the server, including the CPU, chipsets, and peripheral devices into low power states when the system is mostly idle.

There are two parts to power management on ESXi platforms:

1. The BIOS settings for power management, which influence what the BIOS advertises to the OS/hypervisor about whether it should be managing power states of the host or not.
2. The OS/hypervisor settings for power management, which influence the policies of what to do when it detects that the system is idle.

For latency-sensitive applications, any form of power management adds latency to the path where an idle system (in one of several power savings modes) responds to an external event. So our recommendation is to set the BIOS setting for power management to "static high," that is, no OS-controlled power management, effectively disabling any form of active power management. Note that achieving the lowest possible latency and saving power on the hosts and running the hosts cooler are fundamentally at odds with each other, so we recommend carefully

evaluating the trade-offs of disabling any form of power management in order to achieve the lowest possible latencies for your application's needs.

Servers with Intel Nehalem class and newer (Intel Xeon 55xx and newer) CPUs also offer two other power management options: C-states and Intel Turbo Boost. Leaving C-states enabled can increase memory latency and is therefore not recommended for low-latency workloads. Even the enhanced C-state known as C1E introduces longer latencies to wake up the CPUs from halt (idle) states to full-power, so disabling C1E in the BIOS can further lower latencies. Intel Turbo Boost, on the other hand, will step up the internal frequency of the processor should the workload demand more power, and should be left enabled for low-latency, high-performance workloads. However, since Turbo Boost can over-clock portions of the CPU, it should be left disabled if the applications require stable, predictable performance and low latency with minimal jitter.

How power management–related settings are changed depends on the OEM make and model of the server.

For example, for HP ProLiant servers:
1. Set the Power Regulator Mode to Static High Mode.
2. Disable Processor C-State Support.
3. Disable Processor C1E Support.
4. Disable QPI Power Management.
5. Enable Intel Turbo Boost.

For Dell PowerEdge servers:
1. Set the Power Management Mode to Maximum Performance.
2. Set the CPU Power and Performance Management Mode to Maximum Performance.
3. Processor Settings: set Turbo Mode to enabled.
4. Processor Settings: set C States to disabled.

Consult your server documentation for further details.

NUMA

The high latency of accessing remote memory in NUMA (Non-Uniform Memory Access) architecture servers can add a non-trivial amount of latency to application performance. ESXi uses a sophisticated, NUMA-aware scheduler to dynamically balance processor load and memory locality.

For best performance of latency-sensitive applications in guest OSes, all vCPUs should be scheduled on the same NUMA node and all VM memory should fit and be allocated out of the local physical memory attached to that NUMA node.

Processor affinity for vCPUs to be scheduled on specific NUMA nodes, as well as memory affinity for all VM memory to be allocated from those NUMA nodes, can be set using the vSphere Client under **VM Settings → Options tab → Advanced General → Configuration Parameters** and adding entries for "numa.nodeAffinity=0, 1, ...," where 0, 1, etc. are the processor socket numbers.

Note that when you constrain NUMA node affinities, you might interfere with the ability of the NUMA scheduler to rebalance virtual machines across NUMA nodes for fairness. Specify NUMA node affinity only after you consider the rebalancing issues. Note also that when a VM is migrated (for example, using vMotion) to another host with a different NUMA topology, these advanced settings may not be optimal on the new host and could lead to sub-optimal performance of your application on the new host. You will need to re-tune these advanced settings for the NUMA topology for the new host.

ESXi 5.0 and newer also support vNUMA where the underlying physical host's NUMA architecture can be exposed to the guest OS by providing certain ACPI BIOS tables for the guest OS to consume. Exposing the physical host's NUMA topology to the VM helps the guest OS kernel make better scheduling and placement decisions for applications to minimize memory access latencies.

vNUMA is automatically enabled for VMs configured with more than 8 vCPUs that are wider than the number of cores per physical NUMA node. For certain latency-sensitive workloads running on physical hosts with fewer than 8 cores per physical NUMA node, enabling vNUMA may be beneficial. This is achieved by adding an entry for "numa.vcpu.min = N", where N is less than the number of vCPUs in the VM, in the vSphere Client under **VM Settings → Options tab → Advanced General → Configuration Parameters**.

To learn more about this topic, please refer to the NUMA sections in the "vSphere Resource Management Guide" and the white paper explaining the vSphere CPU Scheduler:
http://www.vmware.com/files/pdf/techpaper/VMware-vSphere-CPU-Sched-Perf.pdf

Choice of Guest OS

Certain older guest OSes like RHEL5 incur higher virtualization overhead for various reasons, such as frequent accesses to virtual PCI devices for interrupt handling, frequent accesses to the virtual APIC (Advanced Programmable Interrupt Controller) for interrupt handling, high virtualization overhead when reading the current time, inefficient mechanisms to idle, and so on.

Moving to a more modern guest OS (like SLES11 SP1 or RHEL6 based on 2.6.32 Linux kernels, or Windows Server 2008 or newer) minimizes these virtualization overheads significantly. For example, RHEL6 is based on a "tickless" kernel, which means that it doesn't rely on high-frequency timer interrupts at all. For a mostly idle VM, this saves the power consumed when the guest wakes up for periodic timer interrupts, finds out there is no real work to do, and goes back to an idle state.

Note however, that tickless kernels like RHEL6 can incur higher overheads in certain latency-sensitive workloads because the kernel programs one-shot timers every time it wakes up from idle to handle an interrupt, while the legacy periodic timers are pre-programmed and don't have to be programmed every time the guest OS wakes up from idle. To override tickless mode and fall back to the legacy periodic timer mode for such modern versions of Linux, pass the `nohz=off` kernel boot-time parameter to the guest OS.

These newer guest OSes also have better support for MSI-X (Message Signaled Interrupts) which are more efficient than legacy INT-x style APIC -based interrupts for interrupt delivery and acknowledgement from the guest OSes.

Since there is a certain overhead when reading the current time, due to overhead in virtualizing various timer mechanisms, we recommend minimizing the frequency of reading the current time (using `gettimeofday()` or `currentTimeMillis()` calls) in your guest OS, either via the latency-sensitive application doing so directly, or via some other software component in the guest OS doing this. The overhead in reading the current time was especially worse in Linux versions older than RHEL 5.4, due to the underlying timer device they relied on as their time source and the overhead in virtualizing them. Versions of Linux after RHEL5.4 incur significantly lower overhead when reading the current time.

To learn more about best practices for time keeping in Linux guests, please see the VMware KB 1006427: http://kb.vmware.com/kb/1006427. To learn more about how timekeeping works in VMware VMs, please read http://www.vmware.com/files/pdf/Timekeeping-In-VirtualMachines.pdf.

Physical NIC Settings

Most 1GbE or 10GbE NICs (Network Interface Cards) support a feature called interrupt moderation or interrupt throttling, which coalesces interrupts from the NIC to the host so that the host doesn't get overwhelmed and spend all its CPU cycles processing interrupts.

However, for latency-sensitive workloads, the time the NIC is delaying the delivery of an interrupt for a received packet or a packet that has successfully been sent on the wire is the time that increases the latency of the workload.

Most NICs also provide a mechanism, usually via the `ethtool` command and/or module parameters, to disable interrupt moderation. Our recommendation is to disable physical NIC interrupt moderation on the ESXi host as follows:

```
# esxcli system module parameters set -m ixgbe -p "InterruptThrottleRate=0"
```

This example applies to the Intel 10GbE driver called ixgbe. You can find the appropriate module parameter for your NIC by first finding the driver using the ESXi command:

```
# esxcli network nic list
```

Then find the list of module parameters for the driver used:

```
# esxcli system module parameters list -m <driver>
```

Note that while disabling interrupt moderation on physical NICs is extremely helpful in reducing latency for latency-sensitive VMs, it can lead to some performance penalties for other VMs on the ESXi host, as well as higher CPU utilization to handle the higher rate of interrupts from the physical NIC.

Disabling physical NIC interrupt moderation can also defeat the benefits of Large Receive Offloads (LRO), since some physical NICs (like Intel 10GbE NICs) that support LRO in hardware automatically disable it when interrupt moderation is disabled, and ESXi's implementation of software LRO has fewer packets to coalesce into larger packets on every interrupt. LRO is an important offload for driving high throughput for large-message transfers at reduced CPU cost, so this trade-off should be considered carefully.

Virtual NIC Settings

ESXi VMs can be configured to have one of the following types of virtual NICs (http://kb.vmware.com/kb/1001805): Vlance, VMXNET, Flexible, E1000, VMXNET 2 (Enhanced), or VMXNET 3.

We recommend you choose VMXNET 3 virtual NICs for your latency-sensitive or otherwise performance-critical VMs. VMXNET 3 is the latest generation of our paravirtualized NICs designed from the ground up for performance, and is not related to VMXNET or VMXNET 2 in any way. It offers several advanced features including multi-queue support: Receive Side Scaling, IPv4/IPv6 offloads, and MSI/MSI-X interrupt delivery. Modern enterprise Linux distributions based on 2.6.32 or newer kernels, like RHEL6 and SLES11 SP1, ship with out-of-the-box support for VMXNET 3 NICs.

VMXNET 3 by default also supports an adaptive interrupt coalescing algorithm, for the same reasons that physical NICs implement interrupt moderation. This virtual interrupt coalescing helps drive high throughputs to VMs with multiple vCPUs with parallelized workloads (for example, multiple threads), while at the same time striving to minimize the latency of virtual interrupt delivery.

However, if your workload is extremely sensitive to latency, then we recommend you disable virtual interrupt coalescing for VMXNET 3 virtual NICs as follows.

To do so through the vSphere Client, go to **VM Settings → Options tab → Advanced General → Configuration Parameters** and add an entry for `ethernetX.coalescingScheme` with the value of `disabled`.

Please note that this new configuration option is only available in ESXi 5.0 and later. An alternative way to disable virtual interrupt coalescing for all virtual NICs on the host which affects all VMs, not just the latency-sensitive ones, is by setting the advanced networking performance option (**Configuration → Advanced Settings → Net**) `CoalesceDefaultOn` to `0` (disabled). See http://communities.vmware.com/docs/DOC-10892 for details.

Another feature of VMXNET 3 that helps deliver high throughput with lower CPU utilization is Large Receive Offload (LRO), which aggregates multiple received TCP segments into a larger TCP segment before delivering it up to the guest TCP stack. However, for latency-sensitive applications that rely on TCP, the time spent aggregating smaller TCP segments into a larger one adds latency. It can also affect TCP algorithms like delayed ACK, which now cause the TCP stack to delay an ACK until the two larger TCP segments are received, also adding to end-to-end latency of the application.

Therefore, you should also consider disabling LRO if your latency-sensitive application relies on TCP. To do so for Linux guests, you need to reload the vmxnet3 driver in the guest:

```
# modprobe -r vmxnet3
```

Add the following line in `/etc/modprobe.conf` (Linux version dependent):

```
options vmxnet3 disable_lro=1
```

Then reload the driver using:

```
# modprobe vmxnet3
```

VM Settings

If your application is multi-threaded or consists of multiple processes that could benefit from using multiple CPUs, you can add more virtual CPUs (vCPUs) to your VM. However, for latency-sensitive applications, you should not overcommit vCPUs as compared to the number of pCPUs (processors) on your ESXi host. For example, if your host has 8 CPU cores, limit your number of vCPUs for your VM to 7. This will ensure that the ESXi vmkernel scheduler has a better chance of placing your vCPUs on pCPUs which won't be contended by other scheduling contexts, like vCPUs from other VMs or ESXi helper worlds.

If your application needs a large amount of physical memory when running unvirtualized, consider configuring your VM with a lot of memory as well, but again, try to refrain from overcommitting the amount of physical memory in the system. You can look at the memory statistics in the vSphere Client under the host's **Resource Allocation** tab under **Memory → Available Capacity** to see how much memory you can configure for the VM after all the virtualization overheads are accounted for.

If you want to ensure that the VMkernel does not deschedule your VM when the vCPU is idle (most systems generally have brief periods of idle time, unless you're running an application which has a tight loop executing CPU instructions without taking a break or yielding the CPU), you can add the following configuration option. Go to **VM Settings → Options tab → Advanced General → Configuration Parameters** and add `monitor_control.halt_desched` with the value of `false`.

Note that this option should be considered carefully, because this option will effectively force the vCPU to consume all of its allocated pCPU time, such that when that vCPU in the VM idles, the VM Monitor will spin on the CPU without yielding the CPU to the VMkernel scheduler, until the vCPU needs to run in the VM again. However, for extremely latency-sensitive VMs which cannot tolerate the latency of being descheduled and scheduled, this option has been seen to help.

A slightly more power conserving approach which still results in lower latencies when the guest needs to be woken up soon after it idles, are the following advanced configuration parameters (see also http://kb.vmware.com/kb/1018276):

- For > 1 vCPU VMs, set `monitor.idleLoopSpinBeforeHalt` to `true`
- For 1 vCPU VMs, set `monitor.idleLoopSpinBeforeHaltUP` to `true`

This option will cause the VM Monitor to spin for a small period of time (by default 100 us, configurable through `monitor.idleLoopMinSpinUS`) before yielding the CPU to the VMkernel scheduler, which may then idle the CPU if there is no other work to do.

New in vSphere 5.5 is a VM option called **Latency Sensitivity**, which defaults to **Normal**. Setting this to **High** can yield significantly lower latencies and jitter, as a result of the following mechanisms that take effect in ESXi:

- Exclusive access to physical resources, including pCPUs dedicated to vCPUs with no contending threads for executing on these pCPUs.
- Full memory reservation eliminates ballooning or hypervisor swapping leading to more predictable performance with no latency overheads due to such mechanisms.
- Halting in the VM Monitor when the vCPU is idle, leading to faster vCPU wake-up from halt, and bypassing the VMkernel scheduler for yielding the pCPU. This also conserves power as halting makes the pCPU enter a low power mode, compared to spinning in the VM Monitor with the `monitor_control.halt_desched=FALSE` option.
- Disabling interrupt coalescing and LRO automatically for VMXNET 3 virtual NICs.
- Optimized interrupt delivery path for VM DirectPath I/O and SR-IOV passthrough devices, using heuristics to derive hints from the guest OS about optimal placement of physical interrupt vectors on physical CPUs.

To learn more about this topic, please refer to the technical whitepaper:
http://www.vmware.com/files/pdf/techpaper/latency-sensitive-perf-vsphere55.pdf

Polling Versus Interrupts

For applications or workloads that are allowed to use more CPU resources in order to achieve the lowest possible latency, polling in the guest for I/O to be complete instead of relying on the device delivering an interrupt to the guest OS could help. Traditional interrupt-based I/O processing incurs additional overheads at various levels, including interrupt handlers in the guest OS, accesses to the interrupt subsystem (APIC, devices) that incurs emulation overhead, and deferred interrupt processing in guest OSes (Linux bottom halves/NAPI poll, Windows DPC), which hurts latency to the applications.

With polling, the driver and/or the application in the guest OS will spin waiting for I/O to be available and can immediately indicate the completed I/O up to the application waiting for it, thereby delivering lower latencies. However, this approach consumes more CPU resources, and therefore more power, and hence should be considered carefully.

Note that this approach is different from what the `idle=poll` kernel parameter for Linux guests achieves. This approach requires writing a poll-mode device driver for the I/O device involved in your low latency application, which constantly polls the device (for example, looking at the receive ring for data to have been posted by the device) and indicates the data up the protocol stack immediately to the latency-sensitive application waiting for the data.

Guest OS Tips and Tricks

If your application uses Java, then one of the most important optimizations we recommend is to configure both the guest OS and Java to use large pages. Add the following command-line option when launching Java:

```
-XX:+UseLargePages
```

For other important guidelines when tuning Java applications running in VMware VMs, please refer to http://www.vmware.com/resources/techresources/1087.

Another source of latency for networking I/O can be guest firewall rules like Linux iptables. If your security policy for your VM can allow for it, consider stopping the guest firewall.

Similarly, security infrastructure like SELinux can also add to application latency, since it intercepts every system call to do additional security checks. Consider disabling SELinux if your security policy can allow for that.

Appendix

Here is a tabulated summary of all the performance tuning options at various levels of the virtualization stack described in this document, along with their availability in specific releases of vSphere. This serves more as a checklist for evaluating which options might be appropriate and yield the best performance for your latency-sensitive workloads.

BIOS		
Power Management	Static High/Max Performance	All versions of vSphere
	Disable Processor C-states including C1E	
	Disable chipset power management	
NUMA		
vCPU and memory affinities	VM Settings → Options tab → Advanced General → Configuration Parameters numa.nodeAffinity = "0, 1, ..."	vSphere 5.0 and newer
PHYSICAL NIC		
Disable interrupt moderation	`esxcli system module parameters set -m <driver> -p "param=val"`	vSphere 5.0 and newer
VIRTUAL NIC		
Disable interrupt coalescing	VM Settings → Options tab → Advanced General → Configuration Parameters `ethernetX.coalescingScheme = "disabled"`	vSphere 5.0 and newer
	Configuration → Advanced Settings → Net `CoalesceDefaultOn = 0`	vSphere 4.0/4.1
Disable LRO	`modprobe -r vmxnet3; modprobe vmxnet3 disable_lro=1`	All versions of vSphere
VM		
Reduce idle-wakeup latencies	VM Settings → Options tab → Advanced General → Configuration Parameters `monitor_control.halt_desched = "false"` OR For > 1 vCPU VMs: `monitor.idleLoopSpinBeforeHalt = "true"` For 1 vCPU VMs: `monitor.idleLoopSpinBeforeHaltUP = "true"`	vSphere 5.1 and older
	VM Settings → Options tab → Latency Sensitivity → High	vSphere 5.5 and newer
GUEST OS		
Java: use large pages	`-XX:+UseLargePages`	All versions of vSphere
Turn off iptables	`/etc/init.d/iptables stop`	
Disable SELinux	`SELINUX=disabled in /etc/selinux/config`	

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