Best Practices for Performance Tuning of Telco and NFV Workloads in vSphere

TECHNICAL WHITE PAPER

vmware®

Table of Contents

Introduction

Communication Service Providers (CSPs) are looking toward Network Functions Virtualization (NFV) as a means to radically transform their businesses by lowering costs and enabling rapid and agile service innovation. In doing so they aim to reverse the trend of margin erosion driven by the high cost of infrastructure and lack of service differentiation. These trends have long been associated with traditional monolithic and proprietary service platforms.

The European Telecommunications Standards Institute (ETSI) Network Functions Virtualization (NFV) Industry Specification Group (ISG) has published several specifications about the architecture, interfaces and requirements for NFV. Relevant to this white paper in particular are GS NFV-INF 001: NFV Infrastructure Overview, GS NFV-INF-004: NFVI; Hypervisor Domain, and GS NFV-PER 001: NFV Performance & Portability Best Practices.

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, Telco and NFV application workloads are different from the typical Tier I enterprise application workloads, in that they tend to be any combination of latency sensitive, jitter sensitive, or demanding high packet rate throughputs or aggregate bandwidth, and therefore need to be tuned for best performance on vSphere ESXi.

This white paper summarizes our findings and recommends best practices to tune the different layers of an application's environment for Telco and NFV workloads.

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 (live services) environment.

NFV Workload Classification

The ETSI NFV Performance & Portability Best Practices (GS NFV-PER 001) classifies NFV workloads into different classes. At a high level, here are the characteristics distinguishing the workload classes:

- Data plane workloads, which cover all tasks related to packet handling in an end-to-end communication between edge applications. These tasks are expected to be very intensive in I/O operations and memory R/W operations.
- Control plane workloads, which cover any other communication between NFs that is not directly related to the end-to-end data communication between edge applications. This category includes session management, routing, or authentication. When compared to data plane workloads, control plane workloads are expected to be much less intensive in terms of transactions per second, while the complexity of the transactions might be higher.
- Signal processing workloads, which cover all tasks related to digital processing such as the FFT decoding and encoding in a cellular base station. These tasks are expected to be very intensive in CPU processing capacity and highly delay-sensitive.
- Storage workloads, which cover all tasks related to disk storage.

Control plane workloads should be adequately supported using standard "out of the box" configurations on ESXi. Data plane and signal processing workload performance can benefit from further tuning on ESXi as detailed in this paper. Storage workloads are beyond the scope of this document.

Certain telco architectures further sub-classify Data plane workloads into User plane workloads that have higher packet throughputs and carry all user traffic, and Media plane workloads that have moderate packet rates but are more latency and jitter sensitive as they carry media streams. Some NFV applications (for example, virtualized Evolved Packet Core or vEPC) process both sub-classes of traffic, and are therefore both packet throughput intensive and latency-sensitive by nature.

ESXi VM Networking Datapath

In order to understand the tuning suggestions for ESXi in this paper, it is useful to first look at a simplified view of the ESXi VM Networking Datapath architecture. Some tunables modify this default data path in order to deliver better I/O performance for a VM, but can have side effects that impact the hardware resource utilization of the ESXi host or the performance of other VMs running on the same host, that needs to be taken into consideration.

Figure 1 shows a simplified view of the components involved in transmitting and receiving packets over virtual NICs (vNICs) associated with a VM on an ESXi host.

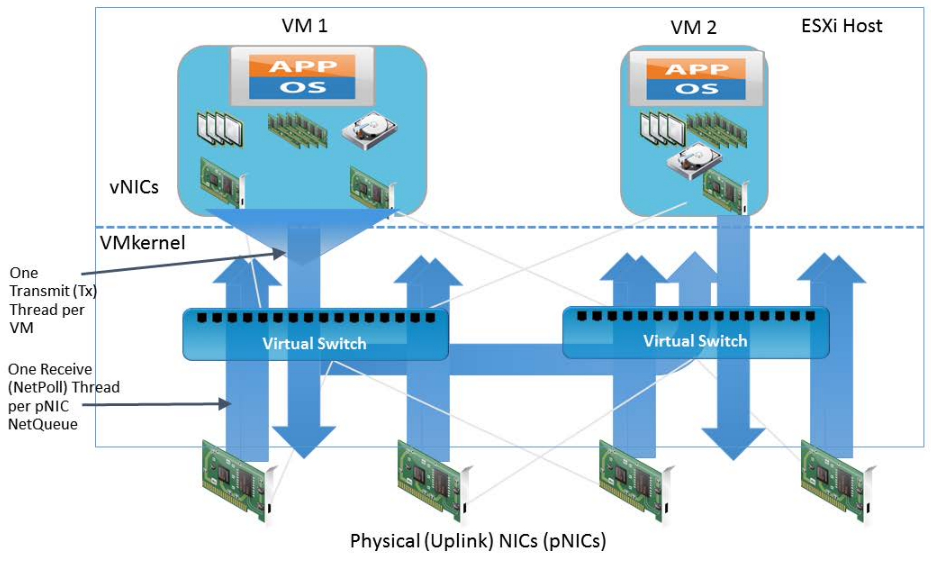
VM 1
VM 2
ESXi Host
APP
OS
APP
OS
vNICs
VMkernel
One Transmit (Tx) Thread per VM
One Receive (NetPoll) Thread per pNIC NetQueue
Virtual Switch
Virtual Switch
Physical (Uplink) NICs (pNICs)

Figure 1. Simplified ESXi VM Networking Datapath

A VM on ESXi can have between 1 and 10 vNICs connected to one or more virtual switch (vSwitch) port groups. By default, there is one transmit thread for each VM, regardless of how many vNICs it has. The transmit thread executes all parts of the virtual networking stack including the vNIC emulation (for example, VMXNET3 or E1000), the vSwitch, any network virtualization overlays such as VXLAN, the uplink layer and packet scheduling, to the physical NIC (pNIC) driver. If the vSwitch determines that a packet is destined for another VM on the same host, the transmit thread also handles the tasks for receiving the packet on the destination VM's vNIC port and emulating that destination vNIC's receive operation.

Each pNIC on the ESXi host has a receive thread (NetPoll) dedicated to it. If the pNIC supports the NetQueue feature, where it has multiple receive queues programmed with the unicast MAC addresses associated with multiple vNICs, then there is one NetPoll thread for each NetQueue. This helps in scaling receive performance on an ESXi host by executing NetPoll threads on different physical CPUs, thereby achieving parallelism. Some pNICs from certain NIC vendors also support multiple transmit queues, and often a transmit queue is paired up with a receive queue in a queue pair.

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]:

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 performance," 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. One approach to still achieving power savings for less performance demanding workloads in a vSphere cluster is to use the Distributed Power Management (DPM) feature of vSphere Distributed Resource Scheduler (DRS), and creating a smaller cluster of ESXi hosts with the "static high performance" BIOS power profile where DRS can place the most performance demanding workloads.

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.

Hyper-threading technology allows a single physical processor core to behave like two logical processors, essentially allowing two independent threads to run simultaneously. Unlike having twice as many processor cores—that can roughly double performance—hyper-threading can provide anywhere from a slight to a significant increase in system performance by keeping the processor pipeline busier.

If the hardware and BIOS support hyper-threading, ESXi automatically makes use of it. For the best performance we recommend that you enable hyper-threading [2] on ESXi hosts (it is enabled by default).

For extremely latency-sensitive or extremely demanding packet rate workloads (millions of packets/second) it may be necessary to rely on DirectPath I/O or SR-IOV [3] direct assignment of physical NICs. In order to use these features on an ESXi host, enable the BIOS setting for Intel VT-d, and for enabling SR-IOV on the host.

Consult your server documentation for further details on how to change these BIOS settings. Also see "DirectPath I/O" and "Single Root I/O Virtualization (SR-IOV)" in "Managing Network Resources [4]."

Physical NIC Settings

Most 1GbE or 10GbE network interface cards (NICs) 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.

The ESXi uplink pNIC layer also maintains a software Tx queue of packets queued for transmission, which by default holds 500 packets. If the workload is I/O intensive with large bursts of transmit packets, this queue may overflow leading to packets being dropped in the uplink layer.

This Tx queue size can be increased up to 10,000 packets by using the following ESXi command:

```
# esxcli system settings advanced set -i 10000 -o /Net/MaxNetifTxQueueLen
```

Depending on the physical NIC and the specific version of the ESXi driver being used on the ESXi host, sometimes packets can be dropped in the pNIC driver because the transmit ring on the pNIC is too small and is filled up. Most pNIC drivers allow you to increase the size of the transmit ring using the command:

```
# ethtool -G vmnic0 tx 4096
```

This command increases the Tx ring size to 4096 entries. The maximum size you can set for a specific pNIC driver, as well as the current Tx ring size in effect, can be determined using the following command:

```
# ethtool -g vmnic0
Ring parameters for vmnic0:
Pre-set maximums:
RX:            4096
RX Mini:       0
RX Jumbo:      0
TX:            4096
```

```
Current hardware settings:
RX:             512
RX Mini:        0
RX Jumbo:       0
TX:             4096
```

Some pNIC drivers such as Intel's ixgbe and Broadcom's bnx2x also support a feature called "queue pairing" which indicates to the ESXi uplink layer that the receive thread (NetPoll) will also process completion of transmitted packets on a paired transmit queue. For certain transmit-heavy workloads, this can cause delays in processing transmit completions and therefore the transmit ring for the vNIC to run out of room for transmitting additional packets, forcing the vNIC driver in the guest OS to drop packets.

This feature can be disabled on an ESXi host for all pNICs, thereby creating a separate thread for processing transmit completions for the pNICs so that they are processed on a timely manner to make room in the vNIC's transmit ring for additional packets. The ESXi command to disable queue pairing is:

```
# esxcli system settings advanced set -o /Net/NetNetqRxQueueFeatPairEnable -i 0
```

For this to take effect, the ESXi host needs to be rebooted.

Like some of the other configuration options, this too should be carefully considered as it increases the CPU usage with the additional thread to handle transmit completions, impacting the amount of CPU resources available for running other workloads on the same host.

Virtual NIC Settings

ESXi VMs can be configured to have one of the following types of virtual NICs [5]: Vlance, VMXNET, Flexible, E1000, VMXNET2 (Enhanced), or VMXNET3.

We highly recommend you choose **VMXNET3** virtual NICs for your latency-sensitive or otherwise performance-critical VMs. VMXNET3 is the latest generation of our paravirtualized NICs designed from the ground up for performance, and is not related to VMXNET or VMXNET2 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 VMXNET3 NICs.

VMXNET3 by default also supports an 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 VMXNET3 virtual NICs as follows.

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

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 "Advanced Networking Performance Options" for details [6].

Another feature of VMXNET3 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
```

As Figure 1 showed, by default ESXi uses one transmit thread regardless of the number of vNICs associated with a VM. In order to achieve significant parallelism for high I/O workloads where the transmit thread becomes CPU bottlenecked, you can configure a VM to use one transmit thread per vNIC, by adding the following setting to the VM's configuration:

```
ethernetX.ctxPerDev = "1"
```

Replace “X” with the right number for each vNIC in the VM's configuration of type VMXNET3.

While this configuration will help drive higher transmit throughput from the VM, it should be considered carefully as it impacts the CPU resources used on the host, and therefore available for running other VMs on the same host.

VM Settings

The virtual machine hardware version determines the virtual hardware available to the virtual machine, which corresponds to the physical hardware available on the host. Virtual hardware includes BIOS and EFI, available virtual PCI slots, maximum number of CPUs, maximum memory configuration, and other characteristics. In order to benefit from the newest virtual machine features in an ESXi release, upgrading the virtual hardware version to the latest supported in the ESXi release can help. For example, ESXi 5.5 supports hardware version 10, while ESXi 6.0 supports hardware version 11.

When configuring virtual hardware associated with a VM, choose paravirtual devices for best performance. For a SCSI disk controller, choose “VMware Paravirtual SCSI” (PVSCSI) [7].

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 physical CPUs (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 contend with other scheduling contexts, like vCPUs from other VMs or ESXi helper worlds. The best practice is to allocate only enough vCPUs to the VM as the number of active, CPU consuming processes or threads in your VM.

When a VM is network I/O intensive, the ESXi I/O threads shown in Figure 1 also consume pCPU resources, so when sizing the VM on a host, take into account the number of vCPUs allocated to the VM as well as the number of transmit and receive threads in ESXi handling the I/O needs of the VM.

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 Web 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.

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. In order to ensure the VM gets dedicated pCPUs, you need to specify the maximum CPU reservations of the latency-sensitive VM for best performance.
- Specifying 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.
- Disabling interrupt coalescing and LRO automatically for VMXNET3 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, "Deploying Extremely Latency-Sensitive Applications in VMware vSphere 5.5 [8]." (This vSphere 5.5 paper is still relevant to vSphere 6.0.)

NUMA Considerations

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 operating systems, 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 Web 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 of the new host. Specifying maximum CPU and memory reservations for the VM is a way to ensure optimal performance on the new host after vMotion.

ESXi also supports vNUMA, which 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 Web 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 [9] and "The CPU Scheduler" technical white paper [10]. ("The CPU Scheduler" is written for vSphere 5.1, but it is still relevant to vSphere 5.5 and 6.0.)

Another NUMA consideration is related to I/O from physical PCIe devices such as pNICs, and the non-uniform memory access times they experience depending on whether the memory they are accessing is attached to the local CPU socket or a remote CPU socket. When specifying NUMA node affinities for a VM, picking a NUMA node where the pNIC handling most of the I/O for the VM can improve I/O performance, as long as the pCPUs on that local NUMA node are not overcommitted to vCPUs of other VMs or other scheduling contexts.

Alternatively, if the I/O traffic on a given host flows through different PCIe pNICs for different VMs, then it may be beneficial to physically plug in the pNIC handling a particular VM's traffic into a PCIe slot attached to the CPU socket where the VM is running.

Guest OS Recommendations

Specifying the closest Guest OS type for the VM based on the original OS distribution will enable ESXi to automatically enable the optimal virtual hardware configurations for the Guest OS. For example, if the Guest OS in the VM is a derivative of RHEL6, then specify the Guest OS as "Linux" and the Guest OS Version as "Red Hat Enterprise Linux 6" of the right type (32-bit or 64-bit).

Moving to a more modern guest OS (like SLES11 SP1 or RHEL6 based on 2.6.32 Linux kernels or newer) minimizes 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.

You can use performance tuning tools in the guest operating system for software profiling. This capability is useful for software developers who optimize or debug software that runs in the virtual machine. You can enable the virtual performance monitoring counters (vPMCs) feature to allow software running inside virtual machines to access CPU Performance Monitoring Counters (PMCs), just as it would when running on a physical machine.

To learn more about best practices for time keeping in Linux guests, please see VMware KB 1006427 [11]. Also see "Timekeeping in VMware Virtual Machines [12]."

Poll Mode Drivers

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), 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 sends the data up the protocol stack immediately to the latency-sensitive application waiting for the data.

The Data Plane Development Kit framework [13], which is commonly used to build NFV dataplane applications, is an example of poll-mode drivers. DPDK uses several techniques to achieve both higher packet throughput and lower latency than the traditional guest OS kernel networking stack. For more details on usage models of DPDK in vSphere ESXi VMs, read the paper, "Intel Data Plane Development Kit with VMware vSphere [14]."

For NFV applications based on DPDK, we recommend using DPDK-1.8 or newer, as they include VMXNET3 Poll-Mode Drivers (PMD) with features and offloads for achieving higher packet throughputs, such as using a second data ring consisting of buffers pre-mapped in the VMkernel for packets smaller than 128-bytes for transmit, and VLAN transmit and IPv4 receive checksum offloads.

For applications sensitive to packet drops, we recommend increasing transmit and receive ring sizes for the VMXNET3 PMD using the DPDK APIs. VMXNET3 supports transmit and receive ring sizes up to 4096 entries, while the default transmit and receive ring sizes in DPDK for VMXNET3 are 512 and 128 entries respectively.

Appendix

Here is a tabulated summary of all the performance tuning options at various levels of the virtualization stack described in this document.

For ESXi host settings that do not persist across reboots, the settings can be added to `/etc/rc.local.d/local.sh` that is executed during the ESXi boot process.

HOST SETTINGS		
BIOS		
Power Management[1, 2]	Static High/Max Performance	
	Disable Processor C-states including C1E	
	Disable chipset power management	
Hyper-threading[1, 2]	Enabled	
ESXi HOST SETTINGS		**Persistent Across Reboot**
Physical NIC		
Disable interrupt moderation[1]	`esxcli system module parameters set -m <driver> -p "param=val"`	Yes
Increase Tx Queue Size[2]	`esxcli system settings advanced set -i 10000 -o /Net/MaxNetifTxQueueLen`	Yes
Increase pNIC Tx Ring Size[2]	`ethtool -G vmnic0 tx 4096`	No
Disable pNIC queue pairing[2]	`esxcli system settings advanced set -o /Net/NetNetqRxQueueFeatPairEnable -i 0`	Yes

VM SETTINGS	
NUMA	
vCPU and memory affinities[1, 2]	VM Settings → Options tab → Advanced General → Configuration Parameters `numa.nodeAffinity = "0, 1, …"`
Reduce idle-wakeup latencies[1]	VM Settings → Options tab → Latency Sensitivity → High
VIRTUAL NIC	
Disable interrupt coalescing[1]	VM Settings → Options tab → Advanced General → Configuration Parameters ethernetX.coalescingScheme = "disabled"
Configure separate ESXi transmit thread per vNIC[2]	VM Settings → Options tab → Advanced General → Configuration Parameters `ethernetX.ctxPerDev = "1"`
GUEST OS SETTINGS	
Disable LRO[1]	`modprobe -r vmxnet3; modprobe vmxnet3 disable_lro=1`
Turn off iptables[1, 2]	`/etc/init.d/iptables stop`
Disable SELinux[1, 2]	`SELINUX=disabled in /etc/selinux/config`

Notes:

1. Recommended for latency-sensitive workloads
2. Recommended for high packet throughput workloads

Resources

[1] Qasim Ali. (2013, August) Host Power Management in VMware vSphere 5.5.
http://www.vmware.com/files/pdf/techpaper/hpm-perf-vsphere55.pdf

[2] VMware, Inc. (2014) Enable Hyperthreading. https://pubs.vmware.com/vsphere-55/topic/com.vmware.vsphere.hostclient.doc/GUID-BD79D611-58D3-4E96-A2CE-11CD0C90D8F1.html

[3] VMware, Inc. (2014) DirectPath I/O vs SR-IOV. https://pubs.vmware.com/vsphere-55/topic/com.vmware.vsphere.networking.doc/GUID-D416429C-1A35-4A6C-8F3E-192B9B0ED7A6.html

[4] VMware, Inc. (2014) Managing Network Resources. https://pubs.vmware.com/vsphere-55/topic/com.vmware.vsphere.networking.doc/GUID-D1597704-154D-421F-A618-512B3EA2FFA2.html

[5] VMware, Inc. (2015, March) Choosing a network adapter for your virtual machine (1001805).
http://kb.vmware.com/kb/1001805

[6] Scott Drummonds. (2009, November) Advanced Networking Performance Options.
http://communities.vmware.com/docs/DOC-10892

[7] VMware, Inc. (2014) Add a Paravirtual SCSI Controller in the vSphere Web Client.
https://pubs.vmware.com/vsphere-55/topic/com.vmware.vsphere.vm_admin.doc/GUID-5D976292-4D8B-409B-9460-34490E72989E.html

[8] Jin Heo and Lenin Singaravelu. (2013) Deploying Extremely Latency-Sensitive Applications in vSphere 5.5.
http://www.vmware.com/files/pdf/techpaper/latency-sensitive-perf-vsphere55.pdf

[9] VMware, Inc. (2014) vSphere Resource Management. https://pubs.vmware.com/vsphere-55/topic/com.vmware.ICbase/PDF/vsphere-esxi-vcenter-server-55-resource-management-guide.pdf

[10] VMware, Inc. (2013, Feb) The CPU Scheduler in VMware vSphere 5.1.
http://www.vmware.com/files/pdf/techpaper/VMware-vSphere-CPU-Sched-Perf.pdf

[11] VMware, Inc. (2015, March) Timekeeping Best Practices for Linux Guests (1006427).
http://kb.vmware.com/kb/1006427

[12] VMware, Inc. (2011, November) Timekeeping in VMware Virtual Machines.
http://www.vmware.com/files/pdf/Timekeeping-In-VirtualMachines.pdf

[13] dpdk.org. DPDK: Data Plane Development Kit. http://www.dpdk.org

[14] VMware, Inc. (2014, August) Intel Data Plane Development Kit with VMware vSphere.
http://www.vmware.com/files/pdf/techpaper/intel-dpdk-vsphere-solution-brief.pdf