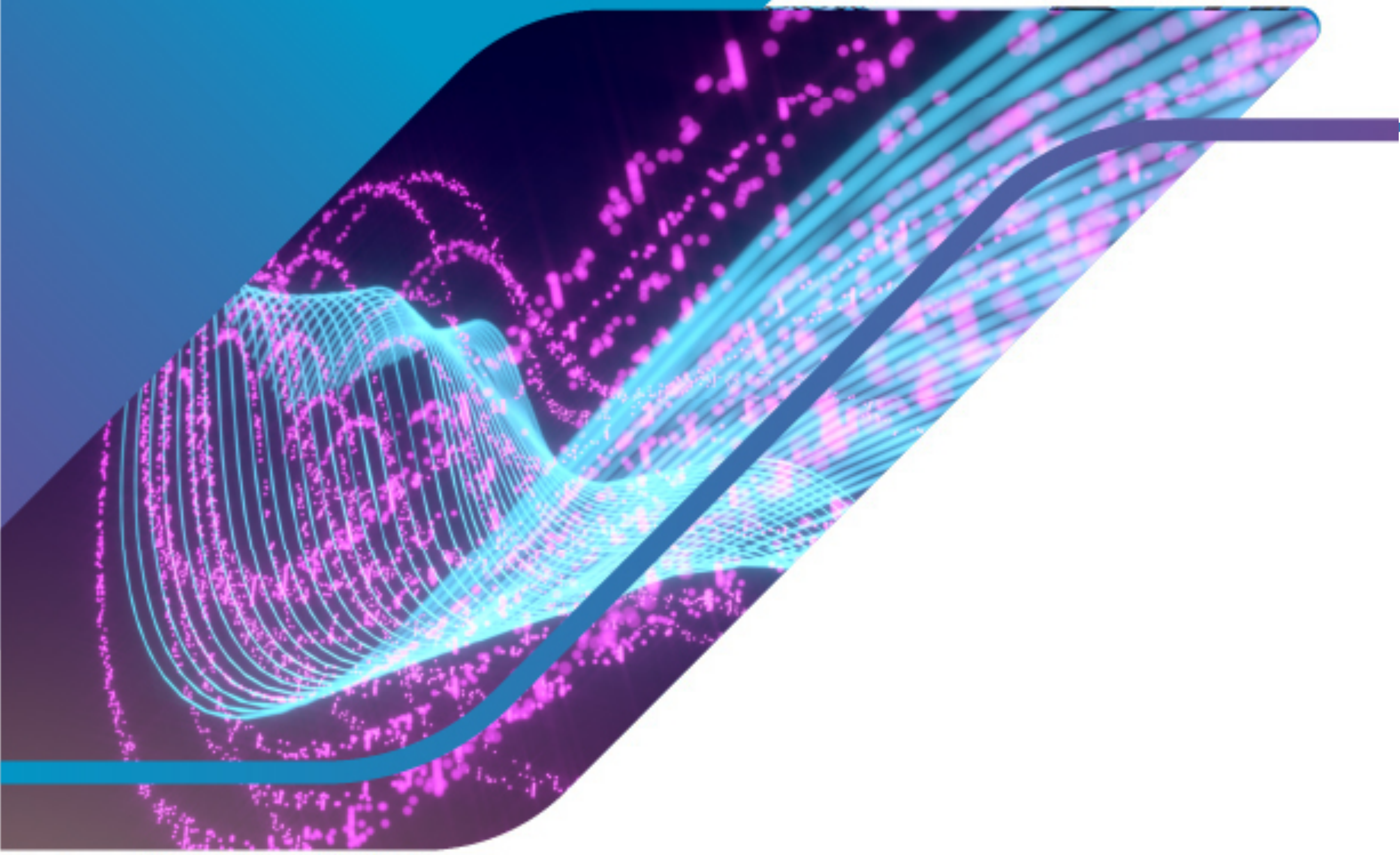
PERFORMANCE TECHNICAL WHITE PAPER
March 2025



VMware vSphere 8.0 Virtual Topology

Performance Study

vmware by Broadcom

Table of contents

Executive summary

This paper describes the performance implications of using virtual topology—introduced in VMware vSphere® 8.0—with several workloads. Virtual topology simplifies a virtual machine's CPU assignments by exposing the appropriate topology at various levels. These topology levels include virtual sockets, virtual non-uniform memory access (vNUMA) nodes, and virtual last-level caches (LLC). Our testing shows that using virtual topology for some typical applications run in a vSphere 8.0 VM will improve the application performance. In some cases, the application performance will remain unchanged.

Terminology

We use the following terminology in this paper:

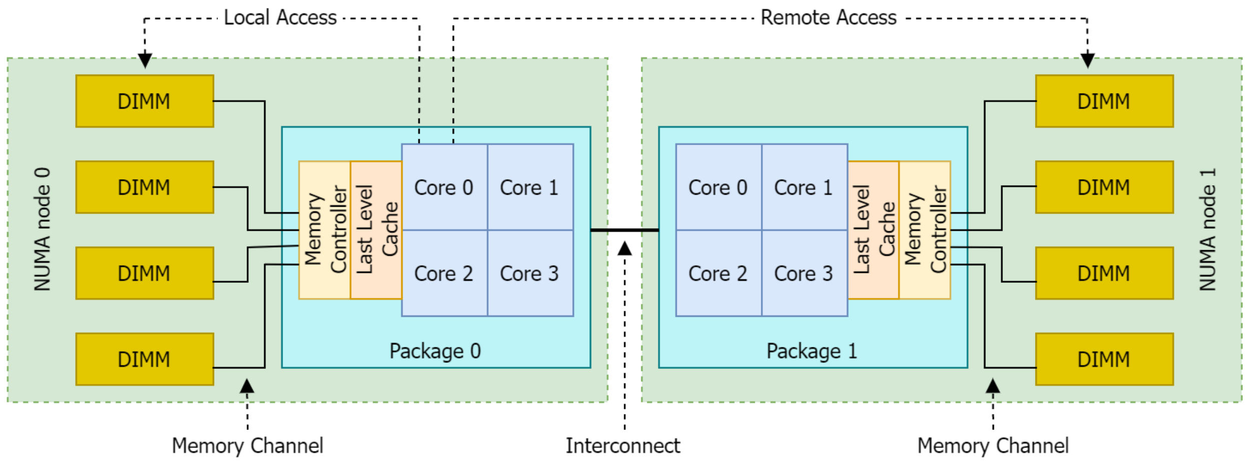
- **CPU:** The CPU, or processor, is the component of a computer system that performs the tasks required for computer applications to run. The CPU is the primary element that performs computer functions. CPUs contain cores.
- **CPU socket:** A CPU socket is a physical connector on a computer motherboard that connects to a single physical CPU. Some motherboards have multiple sockets and can connect multiple multicore processors (CPUs).
- **Core:** A core contains a unit containing an L1 cache and functional units needed to run applications. Cores can independently run applications or threads. One or more cores can exist on a single CPU.
- **Hyper-Threading (HT), simultaneous multithreading (SMT):** Processor technology that allows two logical cores (threads) to run on a single physical core.
- **Virtual NUMA (vNUMA):** vNUMA exposes non-uniform memory access (NUMA) topology to the guest operating systems, allowing NUMA-aware guest operating systems and applications to make the most efficient use of the underlying hardware's NUMA architecture.

Introduction

Emulating physical processor topology for virtual machines (VMs) is critical to performance. ESXi provides options to create virtual CPUs, virtual sockets, virtual NUMA nodes, virtual LLC, and so on. Before vSphere 8.0, the default configuration of a VM was one core per socket. This configuration sometimes created inefficiencies and needed manual tweaks to get optimal performance. Now, ESXi automatically tunes the optimal cores per socket for the VM.

For example, figure 1 illustrates the high-level components of a 2-socket system. Each socket connects to a local memory bank and thus creates a NUMA node. Each socket has 4 cores (each with L1 and L2 caches), and the cores share a common last-level cache (LLC). When a 4-vCPU VM is created, it will be configured with a single socket by default in vSphere 8.0 instead of four sockets, each containing a single vCPU.

Figure 1. A 2-socket system with 2 NUMA nodes



Configuring virtual topology

You can configure the following for vCPUs within the vSphere Client:

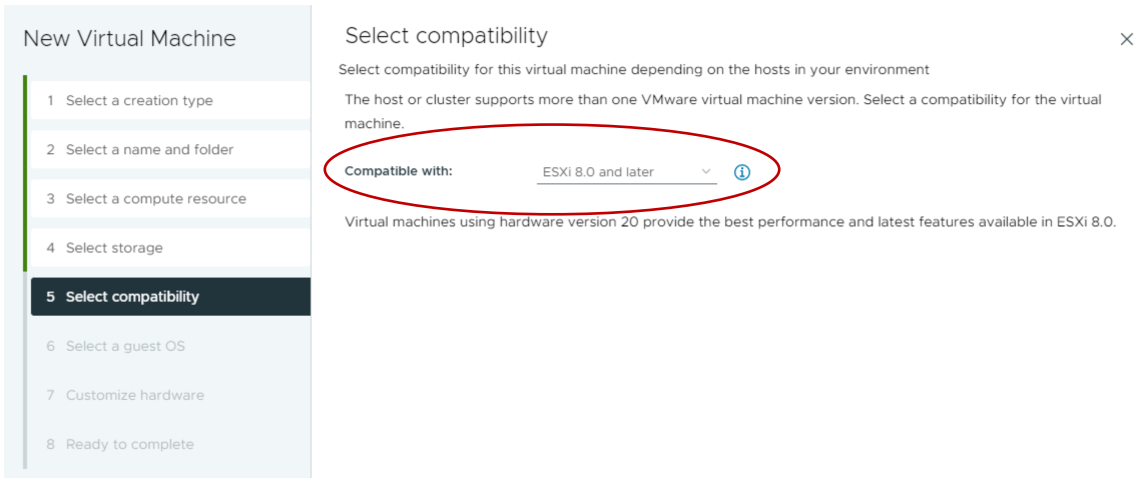
• The total number of vCPUs for the VM

• The number of cores per socket

The following images show the user interface (UI) in the vSphere Client for creating a new VM. When the compatibility setting is **ESXi 8.0 and later**, the virtual topology feature is enabled by default and is shown as **Assigned at power on**. VMs with previous hardware versions retain their existing behavior.

To create a new VM in vSphere 8.0 using the vSphere Client and configure it with the virtual topology feature:

1. When you create a new VM, at **Select compatibility**, make sure **Compatible with** shows **ESXi 8.0 and later**.



Select compatibility

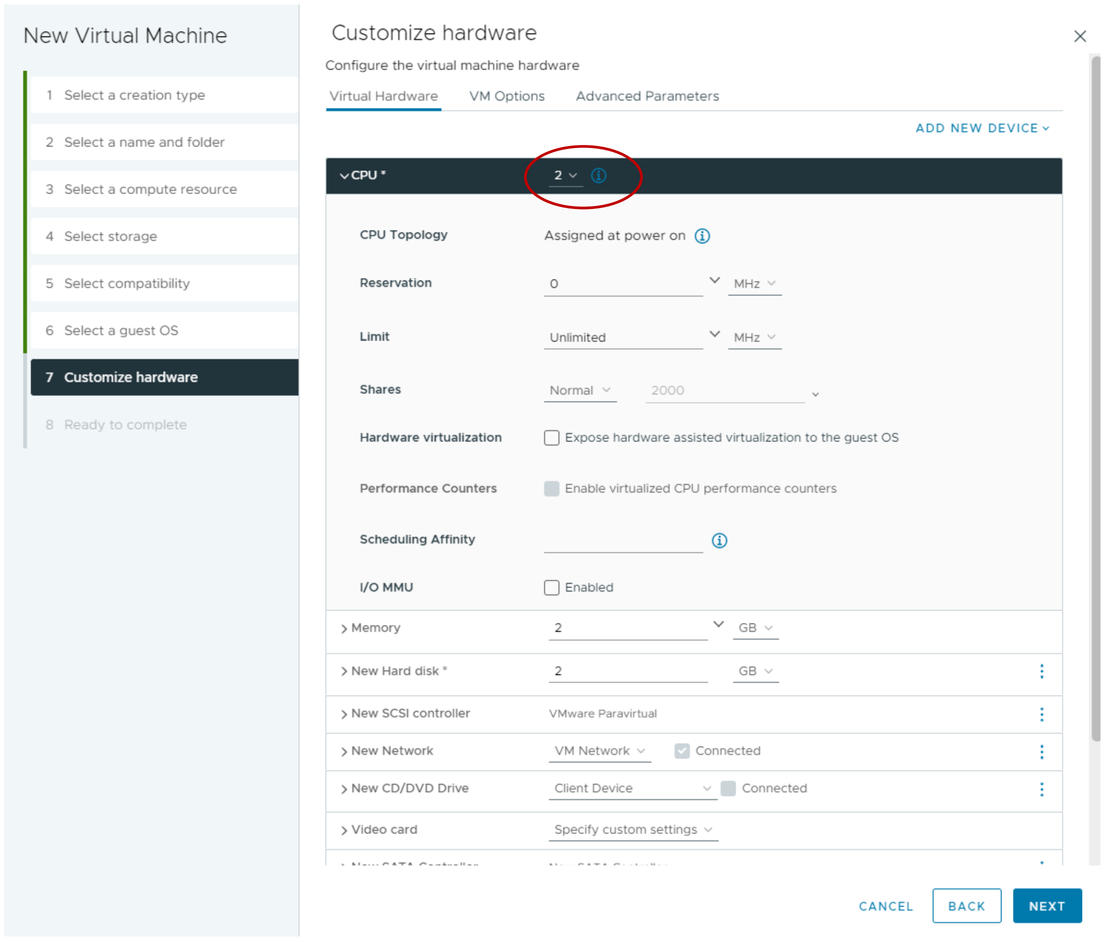
Select compatibility for this virtual machine depending on the hosts in your environment

The host or cluster supports more than one VMware virtual machine version. Select a compatibility for the virtual machine.

Compatible with: ESXi 8.0 and later

Virtual machines using hardware version 20 provide the best performance and latest features available in ESXi 8.0.

2. In the **Customize hardware** section, select the number of virtual CPUs under the **Virtual Hardware** tab.



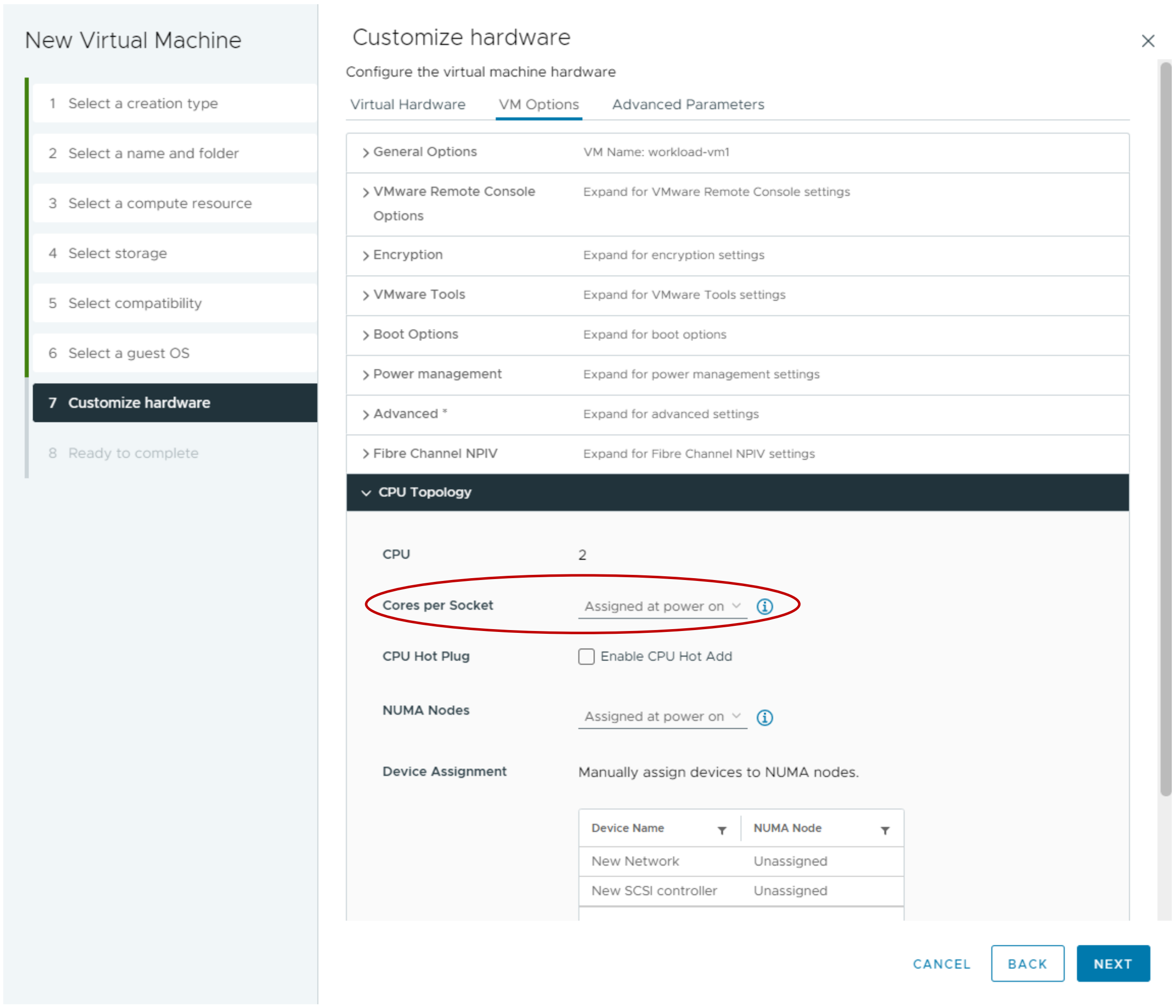
Configure the virtual machine hardware

CPU * 2

CPU Topology: Assigned at power on
Reservation: 0 MHz
Limit: Unlimited MHz
Shares: Normal 2000
Hardware virtualization: Expose hardware assisted virtualization to the guest OS
Performance Counters: Enable virtualized CPU performance counters
Scheduling Affinity:
I/O MMU: Enabled

Memory: 2 GB
New Hard disk *: 2 GB
New SCSI controller: VMware Paravirtual
New Network: VM Network, Connected
New CD/DVD Drive: Client Device, Connected
Video card: Specify custom settings

3. In the **VM Options** tab, there is a new **CPU Topology** section. Ensure **Cores per Socket** is set to **Assigned on power on** (the default).



> General Options	VM Name: workload-vm1
> VMware Remote Console Options	Expand for VMware Remote Console settings
> Encryption	Expand for encryption settings
> VMware Tools	Expand for VMware Tools settings
> Boot Options	Expand for boot options
> Power management	Expand for power management settings
> Advanced *	Expand for advanced settings
> Fibre Channel NPIV	Expand for Fibre Channel NPIV settings
⌄ CPU Topology	
CPU	2
Cores per Socket	Assigned at power on
CPU Hot Plug	☐ Enable CPU Hot Add
NUMA Nodes	Assigned at power on
Device Assignment	Manually assign devices to NUMA nodes.

Device Name	NUMA Node
New Network	Unassigned
New SCSI controller	Unassigned

Note: Equivalent options are also available in the VM configuration (`.vmx`) file.

Sockets and NUMA

Since vSphere 6.5, changing the **Cores per Socket** value no longer influences vNUMA or the configuration of the vNUMA topology [1]. The configuration of **Cores per Socket** only affects the presentation of the virtual processors to the guest operating system. In the past, some software licensing schemes had limitations on socket counts, but not on core counts. For this reason, configuring the socket count to `1` resulted in better performance for some application workloads. Many of these licenses now use core count as the primary method for licensing and not socket count. Therefore, providing VMs with optimal topologies for their hardware simplifies tracking and offers greater flexibility for running these VMs in data centers with different configurations.

Some applications behave differently with cores per socket variations. For example, a single instance of SQL Server Standard edition [2] running on an 8-vCPU VM with 1 core per socket would be able to utilize only 4 vCPUs. But if the same VM were configured with 1 socket (that is, 8 cores per socket), then all 8 vCPUs would be leveraged. In the past, we provided guidelines to appropriately choose the CPU and cores per socket values. But, with the automatic **Cores per Socket** setting, you no longer need to set this manually. An optimal CPU topology is applied by default at the initial power-on of the VM; the configuration also persists when the VM is migrated with vMotion.

Checking sockets in the guest operating system

Let's look at a 4-vCPU VM. With 1 core per socket, the VM is presented to the guest operating system as a 4-socket system (each socket contains 1 CPU). With virtual topology, the VM is presented to the guest operating system as a 1-socket system (with that socket containing 4 CPUs). Table 1 shows a section of CoreInfo [3] in a VM running Windows Server 2022 on an Intel Xeon Gold 6148 Processor. In both cases, all the cores belong to a single NUMA node.

Table 1. CoreInfo output difference with virtual topology

Without virtual topology	With virtual topology
`Processor signature: 00050654`	`Processor signature: 00050654`
`Logical to Physical Processor Map:`	`Logical to Physical Processor Map:`
`*--- Physical Processor 0`	`*--- Physical Processor 0`
`-*-- Physical Processor 1`	`-*-- Physical Processor 1`
`--*- Physical Processor 2`	`--*- Physical Processor 2`
`---* Physical Processor 3`	`---* Physical Processor 3`
`Logical Processor to Socket Map:`	`Logical Processor to Socket Map:`
`*--- Socket 0`	`**** Socket 0`
`-*-- Socket 1`	`Logical Processor to NUMA Node Map:`
`--*- Socket 2`	`**** NUMA Node 0`
`---* Socket 3`	
`Logical Processor to NUMA Node Map:`	
`**** NUMA Node 0`	

In Linux and an 8-vCPU VM, we can use the `lscpu` or `numactl` command to see the CPU and socket settings. Table 2 shows the `lscpu` output differences inside a VM running Red Hat Enterprise Linux 8.1 on an AMD EPYC 7763 processor.

Table 2. Linux lscpu output difference with virtual topology

Without virtual topology		With virtual topology	
Architecture:	x86_64	Architecture:	x86_64
CPU op-mode(s):	32-bit, 64-bit	CPU op-mode(s):	32-bit, 64-bit
Byte Order:	Little Endian	Byte Order:	Little Endian
CPU(s):	8	CPU(s):	8
On-line CPU(s) list:	0-7	On-line CPU(s) list:	0-7
Thread(s) per core:	1	Thread(s) per core:	1
Core(s) per socket:	1	Core(s) per socket:	8
Socket(s):	8	Socket(s):	1
NUMA node(s):	1	NUMA node(s):	1
Vendor ID:	AuthenticAMD	Vendor ID:	AuthenticAMD
CPU family:	25	CPU family:	25
Model:	1	Model:	1
Model name:	AMD EPYC 7763 64-Core Processor	Model name:	AMD EPYC 7763 64-Core Processor
Stepping:	1	Stepping:	1
CPU MHz:	2445.405	CPU MHz:	2445.405
BogoMIPS:	4890.81	BogoMIPS:	4890.81
Hypervisor vendor:	VMware	Hypervisor vendor:	VMware
Virtualization type:	full	Virtualization type:	full
L1d cache:	32K	L1d cache:	32K
L1i cache:	32K	L1i cache:	32K
L2 cache:	512K	L2 cache:	512K
L3 cache:	262144K	L3 cache:	32768K
NUMA node0 CPU(s):	0-7	NUMA node0 CPU(s):	0-7

Overview of workloads

Guest operating system schedulers and applications optimize workloads based on the available hardware topology. Some optimizations may place worker threads running within the same socket and may not migrate threads between sockets to avoid overhead from increased memory latencies. For a larger VM (for example, one with 32 vCPUs), the number of sockets with 1 core per socket by default might not be ideal and practical regarding available system implementations.

The latest processors provide various configurations with BIOS controls. Tuning an optimal setting of cores, sockets, and NUMA values for various workloads has become more complicated. Examples include sub-NUMA clustering (SNC) in Intel Xeon Scalable Processors and NUMA nodes per socket (NPS) in AMD EPYC processors. Virtual topology in vSphere 8.0 reduces this complexity while providing optimal performance for most workloads.

To analyze the performance impact of the new automatic cores per socket setting in vSphere 8.0, we looked at workloads varying from large-scale databases and applications to micro-benchmarks. We ran each workload with the automatic cores per socket feature enabled and then disabled.

- Database workloads:
  - Oracle Database in Linux
  - SQL Server in Windows Server 2019
- Virtual desktop infrastructure (VDI) workload
- VMmark benchmark
- Storage benchmark tests:
  - Iometer tests in Windows
  - Fio benchmark in Linux
- Network benchmark tests:
  - Netperf tests in Windows
  - Netperf tests in Linux

We saw good performance in large database workloads (up to a 14% increase in Oracle Database and a 17% increase in Microsoft SQL Server). Some workload performance (for example, the VMmark benchmark) remained unchanged. A few workloads exhibited performance regression, such as the storage benchmarks. We collected additional performance data to analyze and root cause the performance changes.

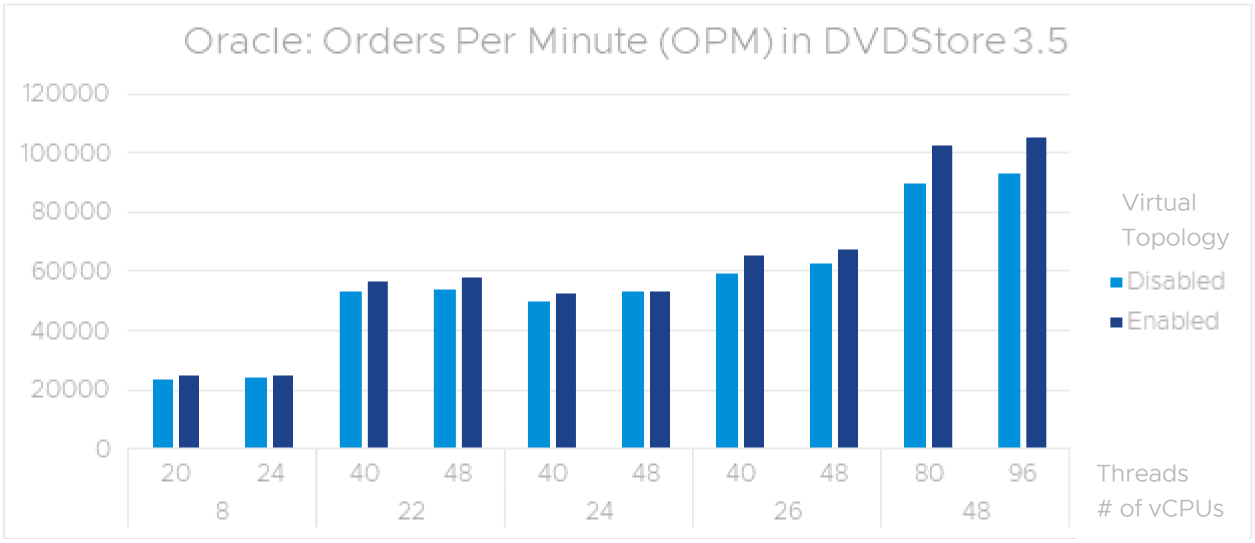
Database workloads

Oracle Database in Linux

We used the open-source database benchmark application DVD Store 3.5 [4]. It simulates an online store with users logging on, searching for DVDs, reading and rating reviews, purchasing DVDs, and so on. The benchmark includes a client program that generates load against the database and reports throughput in orders per minute (OPM). We selected different numbers of threads in different VM configurations.

We configured the VMs with different numbers of virtual CPUs (vCPUs) ranging from 8 to 51 and ran the benchmark application with and then without virtual topology. We based the number of vCPUs used on the 24 cores per socket of the host, and we tested just above and below this number. We saw an average gain of 8.9% in OPM across the workloads with a maximum of up to 14%. The improvement correlates with an increase in CPU utilization and I/O operations per second (IOPS).

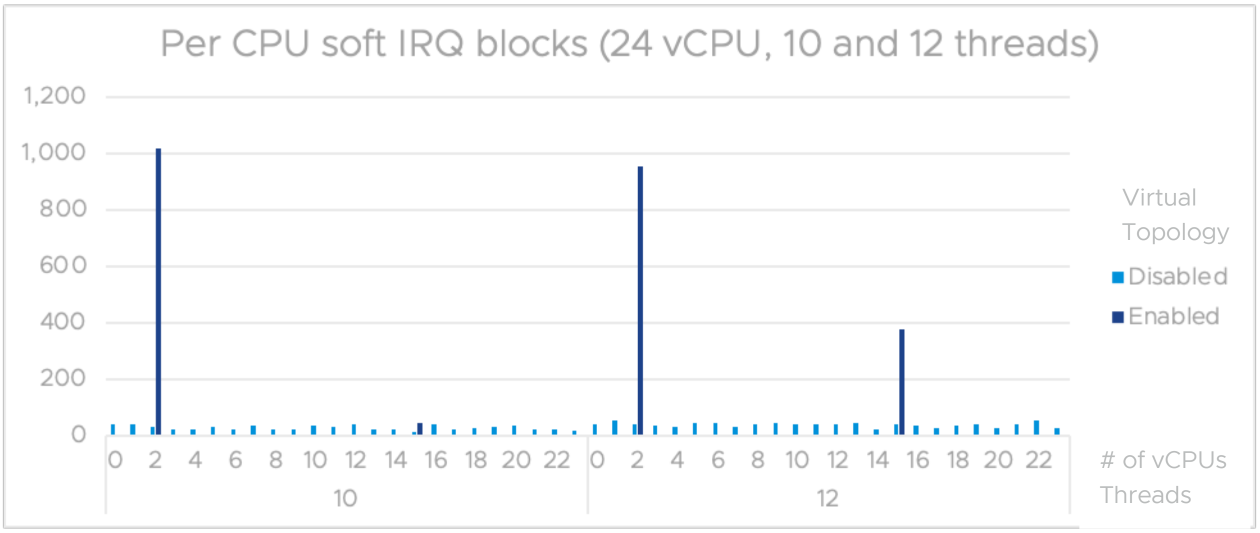
Figure 2. Orders per minute in DVD Store 3.5 tested with an Oracle DB workload running in a Linux VM, with and without virtual topology



We collected CPU interrupts, memory, and I/O operations using various Linux tools during the workload run for a subset of the cases. We observed consistently higher CPU utilization for the user mode and kernel mode components with virtual topology. Disk I/O throughput was also higher for both reads and writes.

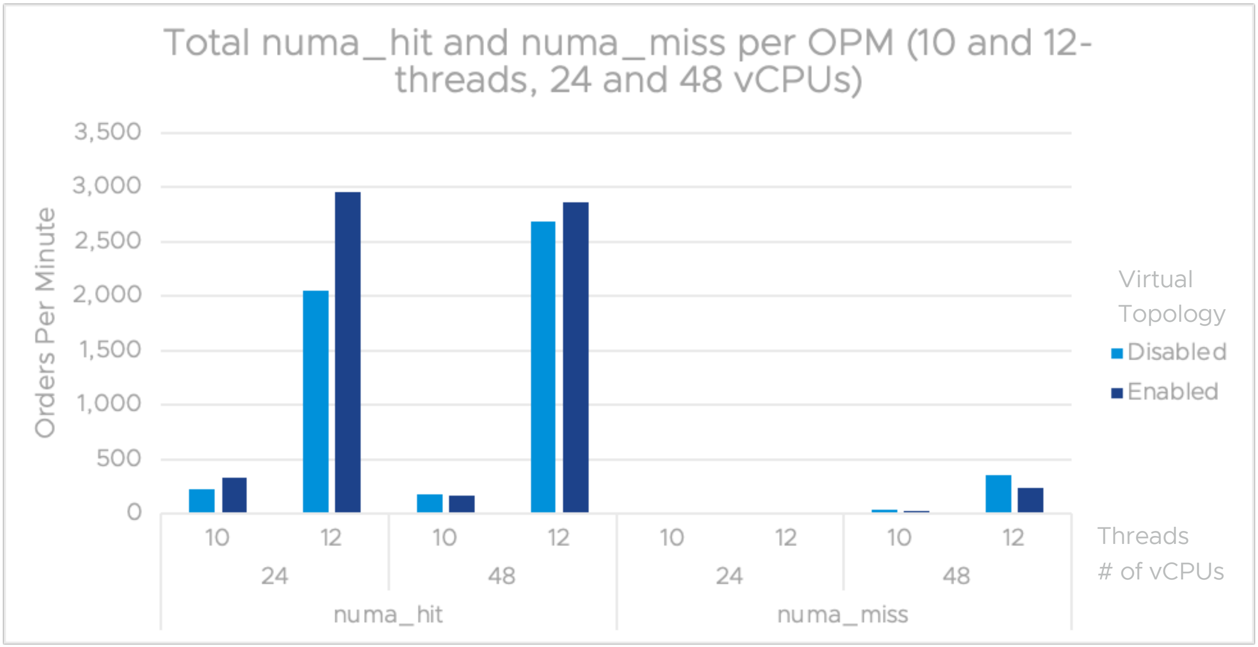
We next collected performance data for Linux guests with various tools to analyze the root cause. Using the `mpstat` utility, we observed that the soft IRQ of the block devices was no longer distributed across all cores when virtual topology was enabled. This means that only a few CPUs (instead of all of them) were handling these soft IRQs.

Figure 3. Per CPU soft IRQ blocks with and without virtual topology



When we enabled virtual topology, we observed more NUMA hits and fewer NUMA misses from the NUMA statistics collected in the Linux guest. The following charts show the data from `numa_stats` normalized to OPM values.

Figure 4. numa_hit and numa_miss per orders per minute with and without virtual topology



When virtual topology is enabled, Oracle DB performs better in the Linux VM.

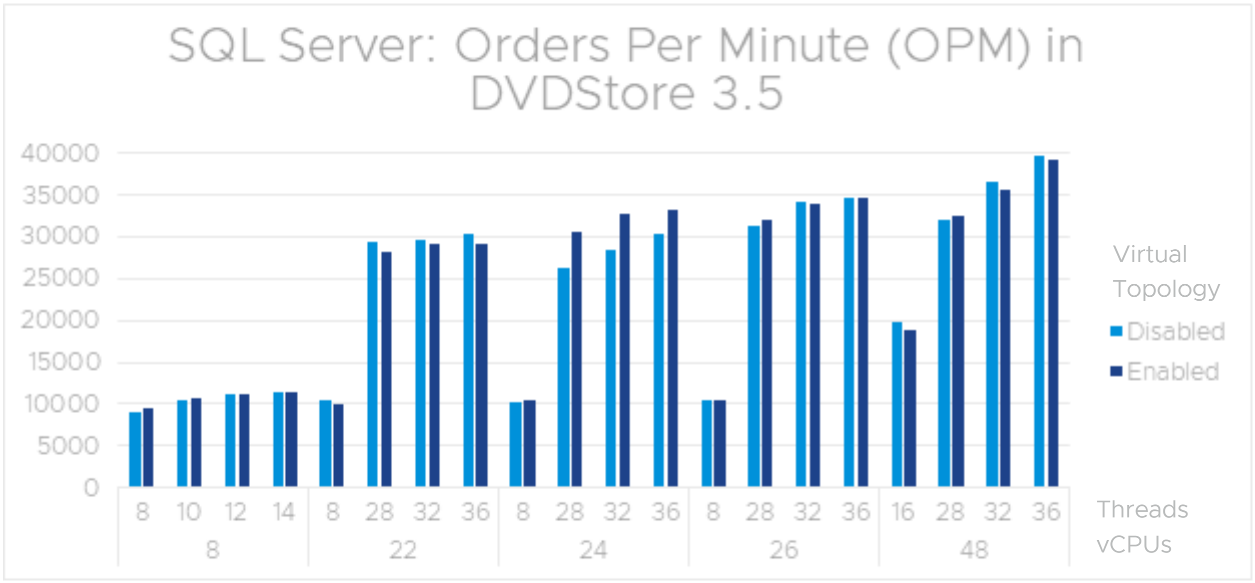
The following components comprised the testbed:

- A 4-socket Broadwell server with 24 cores per socket and 2TB of RAM
- Oracle 12c on CentOS 7 with 64GB of RAM and a system global area (SGA) of 25GB
- DVD Store 3.5 with 5 stores of 3GB each; the workload ran 45GB of data and generated 17GB of logs

SQL Server in Windows Server 2019

We ran DVD Store 3.5 against a SQL Server VM and set a specific number of threads for each vCPU configuration. The average performance of the application remains unchanged with the feature across different configurations. We observed up to a 17% performance gain in the OPM metric. In some cases, the performance was neutral, and in a few cases, we noticed a slight drop. Since the server had 24 cores per socket, the settings with 24 vCPUs matched the physical configuration. Virtual topology provides good improvements in this test case.

Figure 5. Orders per minute (OPM) in DVD Store 3.5 on SQL Server with and without virtual topology



We did not manually configure soft NUMA [5]; instead, we let the SQL Server database engine detect the CPU cores and create soft NUMA nodes automatically. Using virtual topology showed a significant performance improvement in several configurations.

The testbed configuration included:

- A 4-socket Broadwell server with 24 cores per socket and 2TB of RAM
- DVD Store 3.5 with 2 SQL instances, each with 5 stores of 3GB each and SQL memory set to 50GB
- SQL Server 2019 on Windows Server 2019 with 512GB
- SQL Server 2019 developer edition

VDI workload

For a virtual desktop infrastructure (VDI) workload, we used Virtual Session Indexer (Login VSI) [6].

Login VSI is an industry-standard benchmarking tool for measuring the performance and scalability of centralized desktop environments. It gradually increases the number of simulated users until the CPU and other resources (such as memory and disk) are saturated. When the system is saturated, the response times of the applications increase significantly. The latency indicates that the system is almost overloaded. After the test is run, the response times can be analyzed to calculate the maximum active sessions per desktop capacity. This metric is called VSImax.

The simulated desktop workload is scripted in a 70-minute loop when a simulated Login VSI user is logged on and performs generic office “knowledge” worker activities (examples: Microsoft Office, Internet Explorer, and Adobe Reader). We loaded the system under test with the number of VMs to achieve VSImax.

We ran the workload with virtual topology enabled and then disabled. We did not observe any significant changes in latency and throughput or CPU utilization for this workload. The VMs created by Login VSI are small, and virtual topology does not affect their performance.

The testbed used here included:

- Dual socket, 40 cores, Intel Xeon Platinum 8380 Processor @ 2.30GHz
- Memory: 1TB
- VM: 2-vCPU, 2GB memory with knowledge worker profile
- Guest: Windows 10 Enterprise version 21H2
- Baseline: 1 core per socket
- Virtual topology: 2 cores per socket

VMmark benchmark

VMmark [7] is a free tool used to measure the performance and scalability of virtualization platforms. We used version 3 of the tool with multiple configurations. We were able to run the maximum number of tiles with no performance change in the workload.

The testbed included:

- Dell PowerEdge R640
- 2 Xeon Platinum 8260C servers @ 2.40GHz
- 768GB RAM
- 2 Intel X710 10Gb NICs
- QLE2692 HBA
- Dell EMC XtremIO X2 storage array with a Violin V6000 flash memory array

Storage benchmark tests

Iometer in Windows

Iometer [8] is an I/O subsystem measurement and characterization tool for single and clustered systems. The tool consists of two executables: Iometer and Dynamo.

Iometer creates a worker thread for each disk and assigns it to a processor. It does not explicitly provide any CPU affinity setting for the threads. The worker thread waits for I/O completions before issuing the next I/O. The workload is not compute-intensive. When a thread has a specific affinity to a CPU core, the operating system scheduler will re-use the same core (unless it is not available due to a higher-priority thread running there).

We looked at the storage drivers for the VMware paravirtual SCSI (PVSCSI) controller and the non-volatile memory express (NVMe) controller and workloads with varying sizes from 4, 64, and 256KB and a combination of read and write operations with sequential and random I/O patterns. Results are shown for four tests:

1. 4096_100_0: 4K, 100% read, all sequential
2. 4096_100_100: 4K, 100% read, all random
3. 65536_0_0: 64K, 100% write, all sequential
4. 262144_70_100: 256K, 70% read, and 30% write, 100% random

We measured performance using CPU cycles per I/O (CPIO). This metric is influenced by both I/O throughput and CPU utilization. We calculated CPU cycles with an internal tool that relies on a CPU performance counter (`CPU_CLK_UNHALTED`). Throughput, measured in I/O operations per second (IOPS), increased up to 2.27% with virtual topology. We also saw an increase in CPU utilization in the CPIO metric (up to 21%), as shown in figures 6 and 7.

Figure 6. IOPS in Iometer test with and without virtual topology

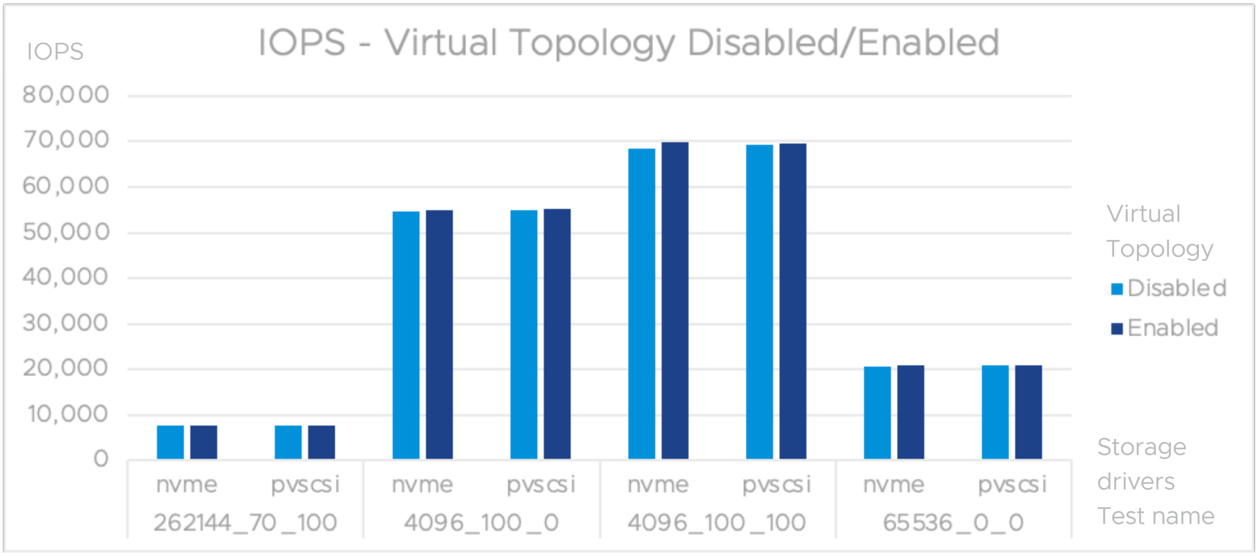
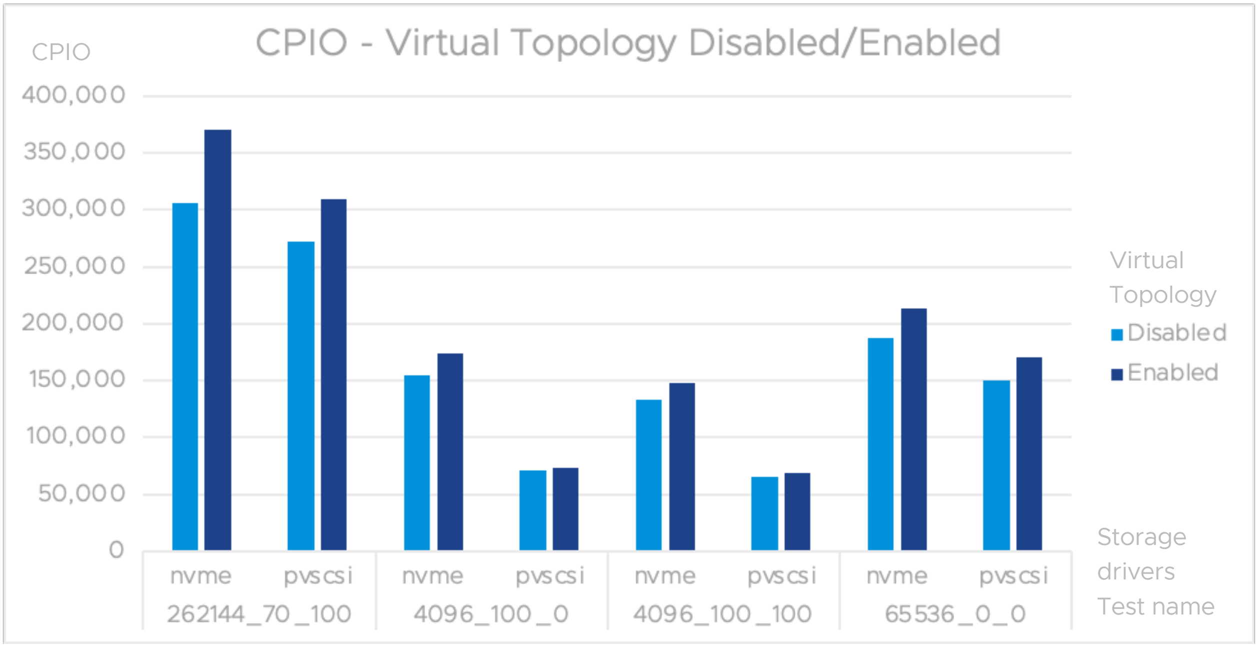


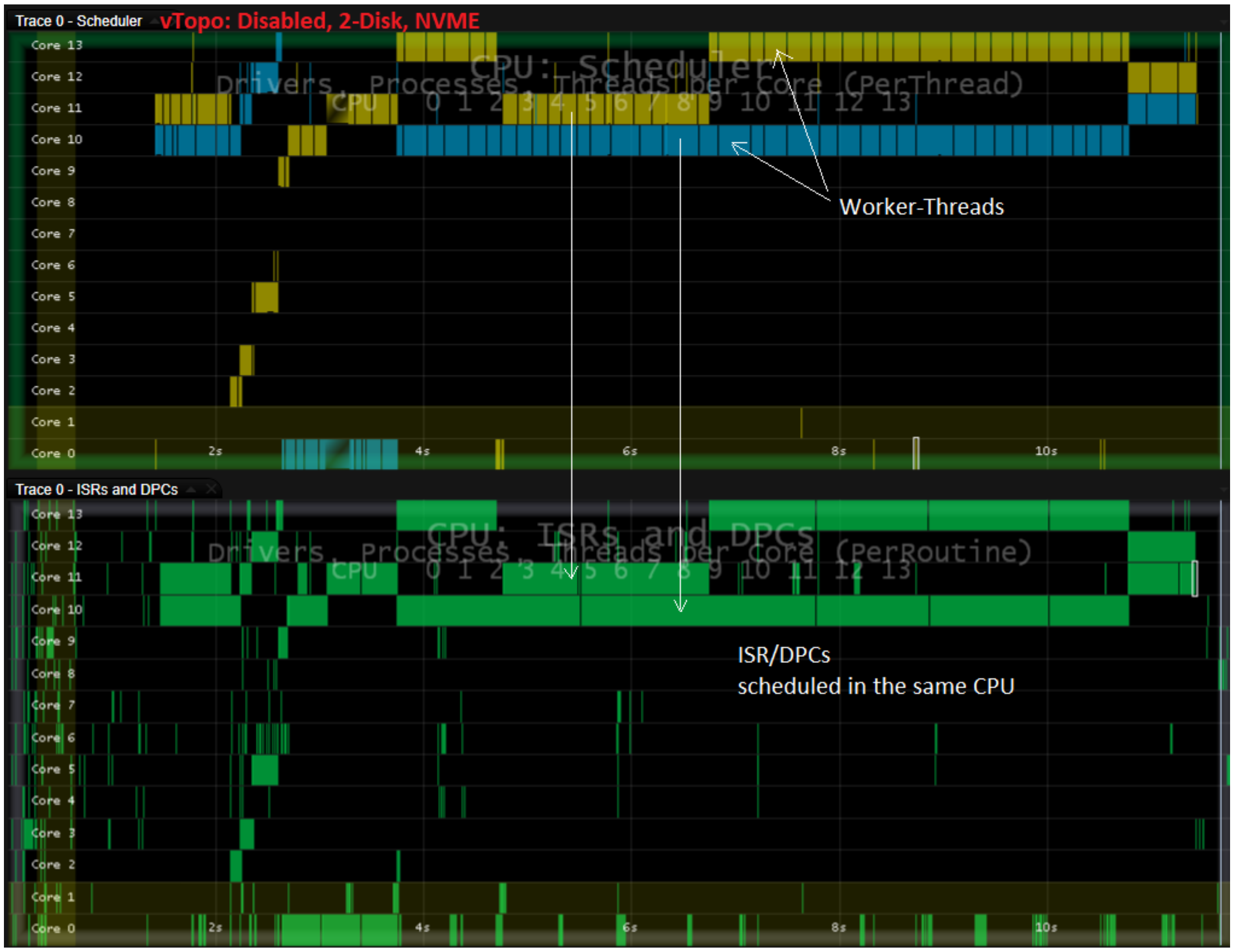
Figure 7. CPIO in Iometer test with and without virtual topology



To find the cause of this CPU utilization, we profiled the application and looked at the scheduling behavior. Overall, the CPU utilization depends on the number of IOPS, which is higher when the request size is small (that is, 4K). We collected event tracing for Windows (ETW) in the guest and analyzed the traces using Media eXperience Analyzer (MXA).

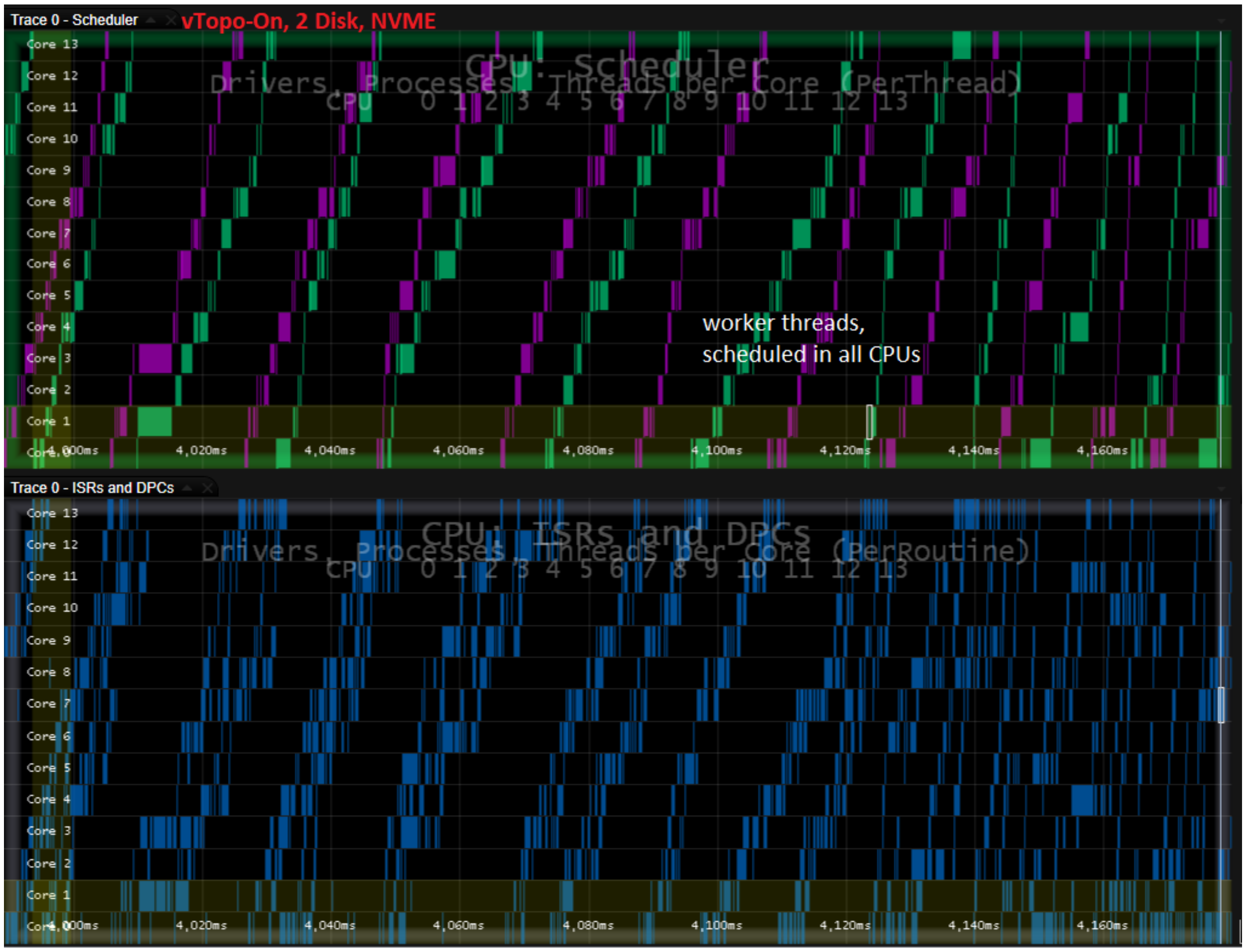
Figure 8 shows the scheduling patterns of the NVMe controller with 2 disks in a VM and 14 CPUs when virtual topology is disabled (for example, 1 core per socket). Here, the x-axis shows the timeline in seconds. The y-axis shows per-core thread scheduling of the worker threads in `Dynamo.exe` at the top, and interrupt service routines (ISR) and deferred procedure calls (DPC) filtered for the storage controller at the bottom. The two worker threads are mostly scheduled in the same CPU cores: core-10 for the blue-colored thread and core-11 or core-12 for the yellow-colored thread. ISR/DPC from the NVMe storage controller are scheduled in the corresponding CPU cores. The CPU caches benefit the thread execution because it stays in the same CPU.

Figure 8. CPU scheduler and ISRs and DPCs view in Media eXperience Analyzer without virtual topology



When virtual topology is enabled, all 14 cores belong to the same socket. As figure 9 shows, the threads are being scheduled in all cores in a round-robin fashion. The NVMe controller interrupts, and DPCs follow the same cores. As a thread moves to a new CPU within a short interval, it generates many L1 and L2 cache misses and takes more CPU cycles to complete the run.

Figure 9. CPU scheduler and ISRs and DPCs view in Media eXperience Analyzer with virtual topology



When we started the application with a specific CPU affinity (for example, `start /affinity 0x4 Iometer.exe` ...), we noticed better performance compared to the case without affinity. This configuration assigned the process to the specified set of processors, while still allowing the threads from the process to migrate freely among the permitted CPU cores. We had to modify the application code to set thread affinity for a fair comparison.

Iometer with thread affinity to fixed CPU cores

We modified Iometer so that it created worker threads to have affinity to specific CPU cores. We named this Iometer2. We collected data with virtual topology enabled and disabled. With Iometer2, we produced a new baseline. We no longer observed consistent differences in the throughput (IOPS) and CPU usage (CPIO) metrics. Figures 10 and 11 show the data for the NVMe controller.

Figure 10. IOPS in Iometer2 (with thread-affinity) test with and without virtual topology

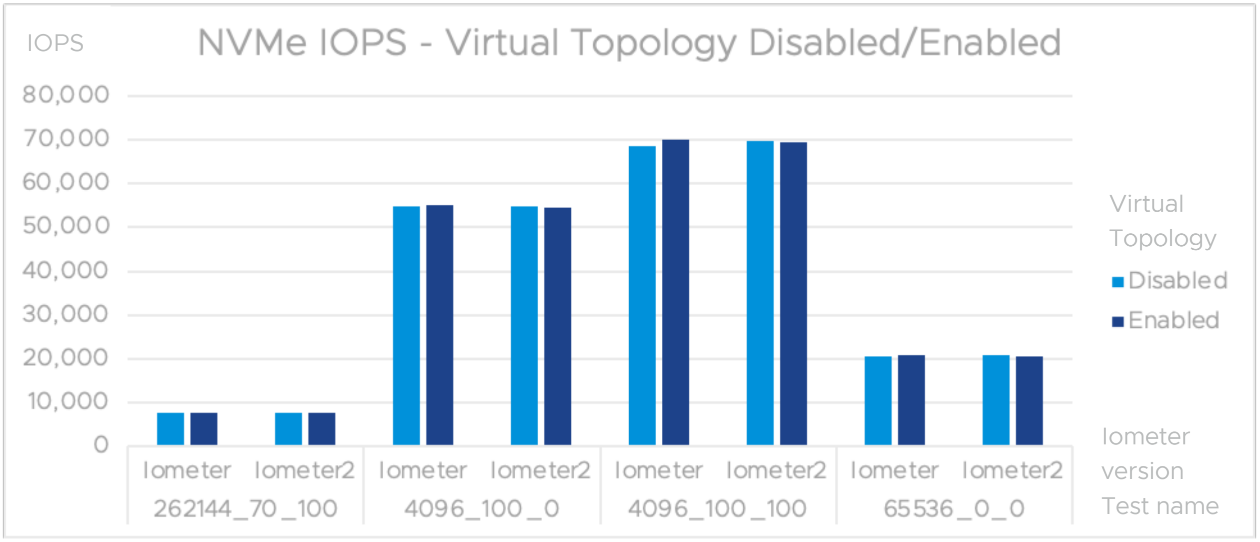
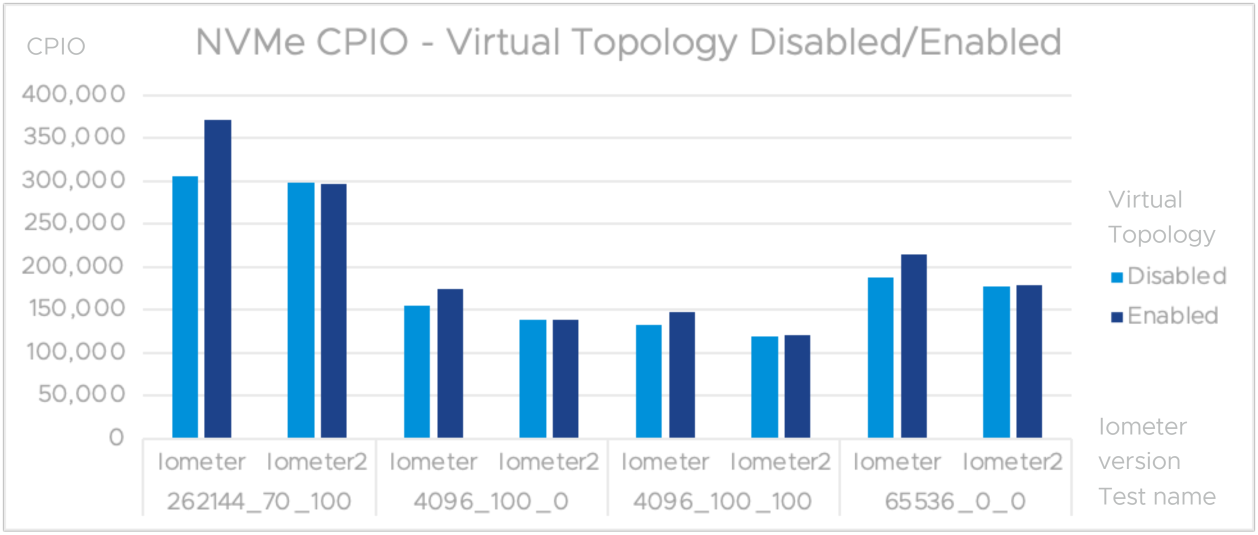


Figure 11. CPIO in Iometer2 (with thread-affinity) test with and without virtual topology



We concluded that virtual topology is not the root cause behind the regression in the tests. The guest scheduler contributed to the additional CPU cost, and the appropriate application changes mitigated this cost. This benchmark, however, does not reflect a real-life workload, so the regression due to the application's nature is not concerning.

The following components comprised the testbed:

- Server: Dell PowerEdge R930
- CPU: 4 Intel Xeon Processors E7-8880 v4 @2.2GHz with 22 physical cores
- Hyper-Threading (HT): Yes
- Memory: 1024GB
- Physical disks: 3 non-RAID SSDs
- Guest operating system: Windows Server 2022

Fio in Linux

Flexible IO Tester (Fio) [9] is a benchmarking and workload simulation tool for Linux/Unix. It is highly tunable and widely used for storage performance benchmarking. We used the tool with various configurations to simulate different workloads. For example, we used the following command to perform a sequential write test with a 64K block size on 8 virtual disks for a duration of 10 minutes:

```
fio -name=fio-test -ioengine=libaio -iodepth=4 -rw=write -bs=65536 -thread -direct=1 -numjobs=8 -
group_reporting=1 -time_based -runtime=600 -
filename=/dev/sdb:/dev/sdc:/dev/sdd:/dev/sde:/dev/sdf:/dev/sdg:/dev/sdh:/dev/sdi -significant_figures=10
```

While I/O throughput remained the same, we saw a 5.86% increase in CPU utlization.

Table 3. IOPS and CPIO with and without virtual topology

Metric	Virtual Topology		%Diff
	Disabled	Enabled	
IOPS	12530	12553	0.18
CPIO	151278	160143	5.86

We collected traces in the host using `esxtop` and Linux guest statistics using `vmstat`, `pidstat`, `mpstat`, and so on to understand the changes. `esxtop` showed higher CPU and memory utlization of the VM when virtual topology was enabled.

In the `numastat` output, we observed that the VM was allocating more memory (that is, higher `numa_hit`) on node-0. `vmstat` also showed a higher memory cache usage. This should have had a positive impact. However, we saw higher context switching and more activity between vCPUs. In `/proc/interrupts`, we saw a much higher rate of rescheduling interrupts, which indicated that the driver was trying to distribute the load among the processors. In the `mpstat` output, we saw the `BLOCK` interrupt (soft IRQ of block devices) was scheduled evenly among the CPUs when virtual topology was disabled. But when virtual topology was enabled, all the soft IRQ of block devices were sent to one CPU, and this eventually resulted in a higher rescheduling of interrupts. This guest behavior resulted in a higher CPU usage with virtual topology turned on.

The testbed included the following:

- Server: Dell PowerEdge R740xd
- CPU: Intel Xeon Gold 5120 Processor @ 2.20GHz, 2 sockets, 28 cores total
- Memory: 128GB
- VM configuration: CentOS 7, 4 vCPUs, 4GB memory

Network benchmark tests

Netperf in Windows

Netperf [10] is a benchmarking tool that can be used to measure the performance of different types of networking. It can be used to measure the throughput and latency of TCP and UDP packets.

Netperf TCP_RR (request-response) tests report round-trip latency and throughput between two systems. We used the following command line that uses small packets without any delay.

```
Netperf -l 60 -H DestinationIP -p port -t TCP_RR -v 2 -- -s 8K -S 8K -r 1,1 -m 1 -b 4 -D
```

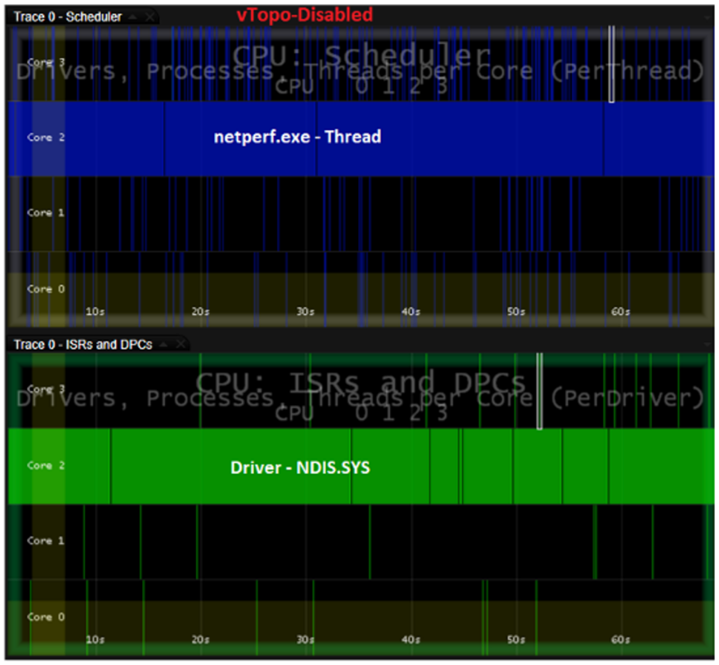
With virtual topology, we saw the networking throughput degrade and latency increase, which brings it more in line with a multi-core, bare-metal performance of a similar configuration.

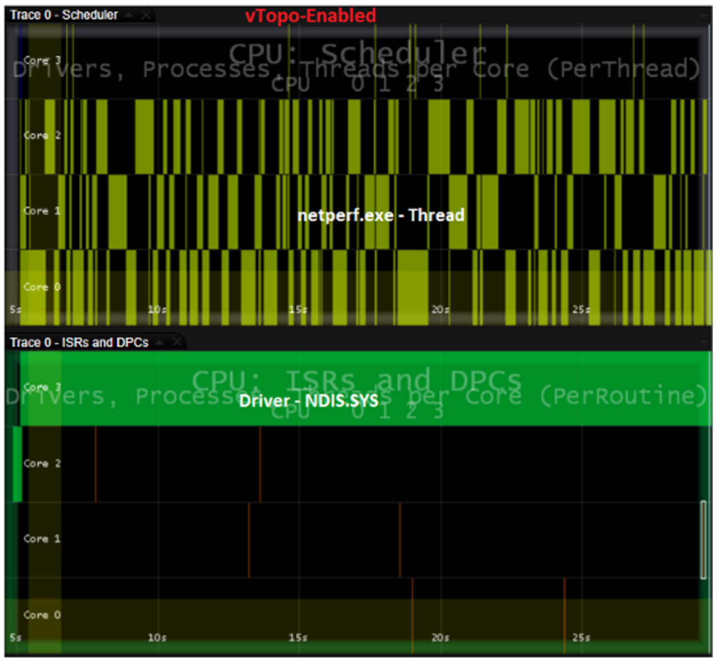
Table 4. Roundtrip latency and transfer rate of Netperf tests in Windows

Setting	Roundtrip Latency µsec/Transfer	Transfer Rate per Second
4 vCPU VM (1 core per socket)	122.32	40926.38
4 vCPU VM (4 cores per socket)	147	34115.58
Bare-metal (4-CPU), HT-off	144.63	34571.06

With a setting of 1 core per socket, the worker thread and networking driver (`ndis.sys`) run in the same core (that is, core-2 in figure 12). When virtual topology is enabled, the worker thread mostly runs in three cores (cores 0, 1, 2) and skips the core where the networking driver's interrupt/DPCs are scheduled (core-3 in figure 13).

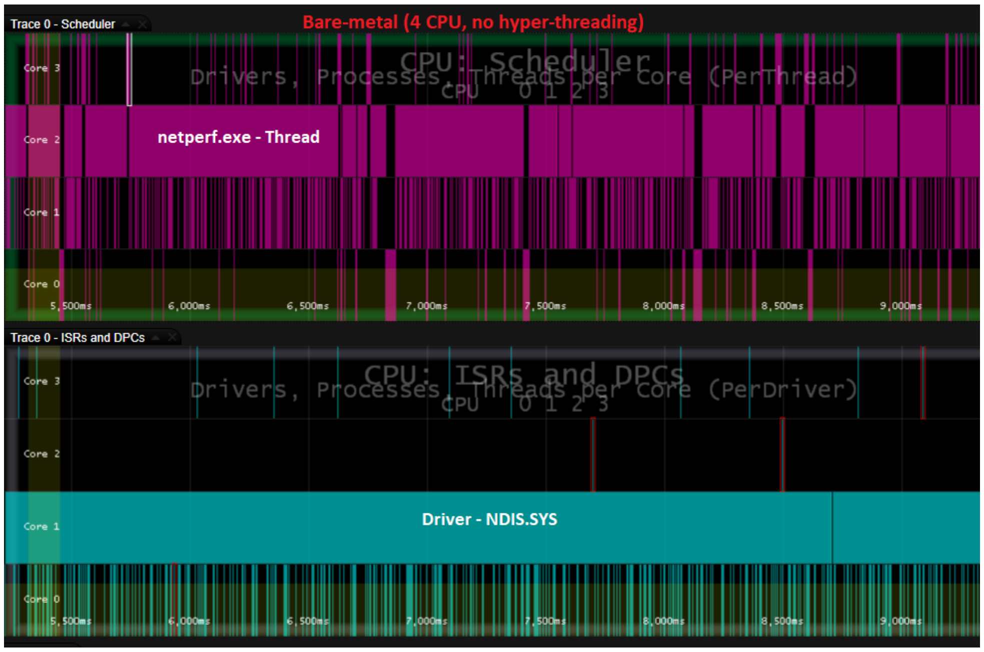
Figure 12. (left) CPU scheduler and ISRs and DPCs view in Media eXperience Analyzer without virtual topology during Netperf test. Figure 13. (right) CPU scheduler and ISRs and DPCs view in Media eXperience Analyzer with virtual topology





The scheduling pattern and driver behavior with virtual topology on is similar to a bare-metal system running Windows Server 2022, as shown in figure 14. The number of CPUs is constrained to 4 (as a boot parameter), and Hyper-Threading is turned off in the BIOS. We see that the network driver's interrupt and DPC routines are scheduled in one core (core-1). The worker thread of the application is mostly scheduled in another core (core-2) with occasional migration to other cores.

Figure 14. CPU scheduler and ISRs and DPCs view in Media eXperience Analyzer during Netperf test in Windows running on a bare-metal host



`netperf.exe` uses a single worker thread to process network traffic. When the VM has a setting of 1 core per socket, the Windows scheduler keeps the thread within the same socket, and we do not see thread migrations between cores. The thread's execution benefits from the code locality and instruction cache of the processor. Moreover, the network driver's interrupt and DPC routines are scheduled within the same socket (that is, in the same core). When the driver copies the data packets to the process memory, they also benefit from data caching. Therefore, a VM with a 1 core per socket setting lowers latency and utilizes less CPU than either running on a bare metal (4-CPU server) or on a VM with a 4 cores per socket setting. We recommend continuing to manually configure the same setting to retain optimal performance under this benchmark condition.

The testbed included:

- Server: Dell PowerEdge R930
- CPU: 4 Intel Xeon Processors E7-8880 v4 @2.2GHz with 22 cores
- Hyper-Threading (HT): Yes
- Memory: 1024GB
- Disks: 3 SSD non-RAID physical disks
- Operating system: Windows Server 2022

Netperf in Linux

Among different tests using Netperf, we analyzed one that sends UDP traffic from a bare-metal client to a Linux VM. The send and receive socket buffer size was 256K in both the client (which ran the netperf binary) and the server (which ran the netserver binary). 256 bytes of the buffer were passed in during send/receive calls.

```
netperf -l 60 -H DestinationIP -p port -t UDP_STREAM -- -s 256K -S 256K -m 256 -M 256 -n 1
```

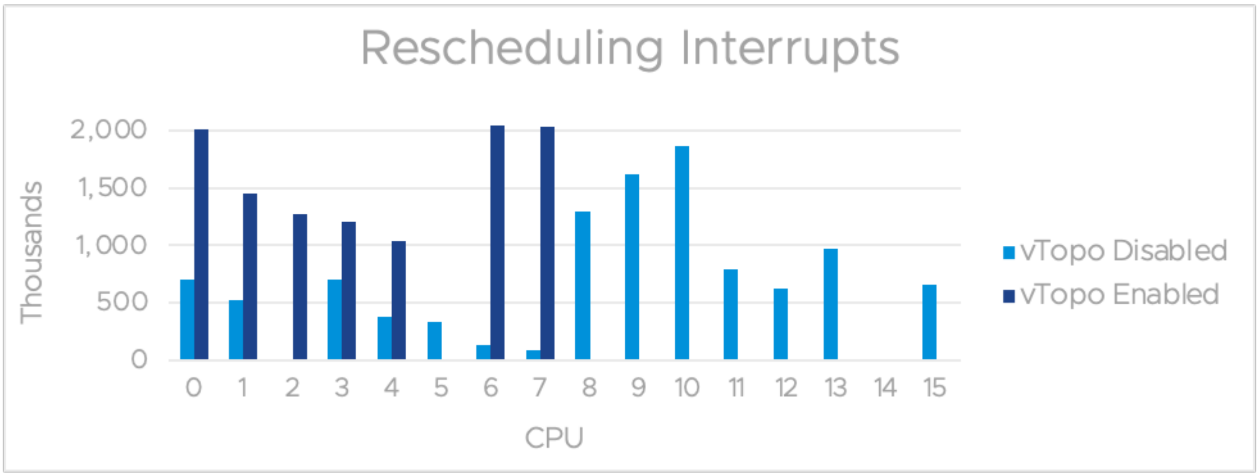
We see a decrease in throughput as received messages per second drops about 26%.

Table 5. Comparison of virtual topology disabled and enabled using Netperf

Metric	Virtual Topology		%Diff
	Disabled	Enabled	
Recv Msg per Second	352,777.5	259,852.25	26.34

We collected stats in the Linux VM, which had 16 vCPUs. When we enabled virtual topology, `vmstat` revealed a slightly higher interrupt rate. Both user-level and kernel-level CPU utilization were higher, according to `pidstat`. There was no significant difference in rescheduling interrupts, but their distribution was different. With virtual topology disabled, we saw an even distribution among all the cores. However, upon enabling virtual topology, rescheduling interrupts activated only a subset of all the cores. The CPU utlization of the cores followed similar patterns. These scheduling differences in the guest operating system contribute to the regression of the scenario.

Figure 15. Rescheduling interrupts Iometer test results with virtual topology disabled and enabled



The following components comprised the testbed:

- Server: Dell PowerEdge R740xd
- CPU: 2 Intel Xeon Gold 5120 Processors @ 2.20GHz with 14 cores
- Hyper-Threading (HT): Yes
- Memory: 192GB
- Adaptor: Mellanox Technologies MT27700 Family (ConnectX-4)
- VM configuration: 16 vCPU, 32GB memory

Conclusion

Virtual topology simplifies VM settings and provides the optimal configuration to match the physical hardware. We saw performance improvements in several workloads. We also noticed some micro-benchmark workloads show degradation due to the behavior of the guest operating system or application design choices. We noticed we could mitigate these micro-benchmark degradations by either staying in a previous hardware version or manually setting cores per socket to 1. We expect that performance will be unchanged or improved using virtual topology for most typical customer workloads.

References

[1] M. Achtemichuk, "Virtual Machine vCPU and vNUMA Rightsizing – Guidelines," 18 Jul 2019. https://blogs.vmware.com/performance/2017/03/virtual-machine-vcpu-and-vnuma-rightsizing-rules-of-thumb.html.

[2] Microsoft, "Compute capacity limits by edition of SQL Server," 1 Feb 2021. https://learn.microsoft.com/en-us/sql/sql-server/compute-capacity-limits-by-edition-of-sql-server?view=sql-server-ver16.

[3] M. Russinovich, "Coreinfo v3.6," 9 Sep 2022. https://docs.microsoft.com/en-us/sysinternals/downloads/coreinfo.

[4] T. Muirhead, "DVD Store 3.5," 13 Jan 2022. https://github.com/dvdstore/ds35.

[5] Microsoft, "Soft-NUMA (SQL Server)," 14 Sep 2022. https://learn.microsoft.com/en-us/sql/database-engine/configure-windows/soft-numa-sql-server?view=sql-server-ver16.

[6] "Login VSI," https://www.loginvsi.com/.

[7] VMware, "VMmark," 7 Apr 2020. https://www.vmware.com/products/vmmark.html.

[8] "Iometer," http://www.iometer.org/.

[9] "Flexible I/O Tester," 30 Aug 2022. https://github.com/axboe/fio.

[10] Hewlett Packard, "Netperf," 20 Jul 2015. https://github.com/HewlettPackard/netperf.

About the authors

Mohammad Hossain is a software engineer in Broadcom's VMware Cloud Foundation (VCF) Performance Engineering team. His interests include hypervisor performance, resource management, and power efficiency.

Tanmay Nag is a software engineer in the VCF Performance Engineering team. His work focuses on investigating and tracking different aspects of vCenter and ESXi performance. Part of his work also involves automating tools and benchmarks for performance debugging and measurement.

Todd Muirhead is a software engineer in the VCF Performance Engineering team where he works on database, storage, and processor performance. He is also one of the co-creators and maintainers of the DVD Store open-source benchmark.

Qasim Ali is a distinguished engineer in the VCF Performance Engineering team. His interests include hypervisor performance and optimizing resource management—for example, power, memory, and CPU. Qasim graduated with a PhD in Computer Engineering from Purdue University in 2009.

Acknowledgments

We thank **Mark Achtemichuk** and **Valentin Bondzio** for reviewing and providing input for the paper.