Performance Optimizations in VMware vSphere 7.0 U2 CPU Scheduler for AMD EPYC Processors

Performance Study - March 24, 2021

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

Table of Contents

Executive Summary

VMware vSphere® 7.0 U2 includes a CPU scheduler that is architecturally optimized for AMD EPYC™. This scheduler is designed to take advantage of the multiple last-level caches (LLCs) per CPU socket offered by the AMD EPYC processors. This paper shows that vSphere 7.0 U2 can achieve near optimal performance with this new CPU scheduler using out-of-the-box options and configurations. An extensive performance evaluation using both enterprise benchmarks and microbenchmarks shows that the CPU scheduler in vSphere 7.0 U2 achieves up to 50% better performance than 7.0 U1.

Background and Introduction

AMD EPYC processors are built using the Multi-Chip Module (MCM) architecture where a set of Core Complex Dies (CCDs) are connected to an I/O Die via Infinity Fabric.

In total, there are up to eight CCDs in the EPYC 7002 Series Processors, codenamed "Rome," as shown in the simplified logical diagram in figure 1.

Two Core Complexes (CCXs) make up each CCD. A CCX is up to four cores sharing an L3 cache of 16MB in size, as shown in figure 2 for a Rome CCD. As seen at the software layer, inter-CCX traffic still travels through the Infinity Fabric. In AMD EPYC 7003 Series Processors, codenamed "Milan," there is just one CCX/LLC per CCD and up to eight cores per LLC.

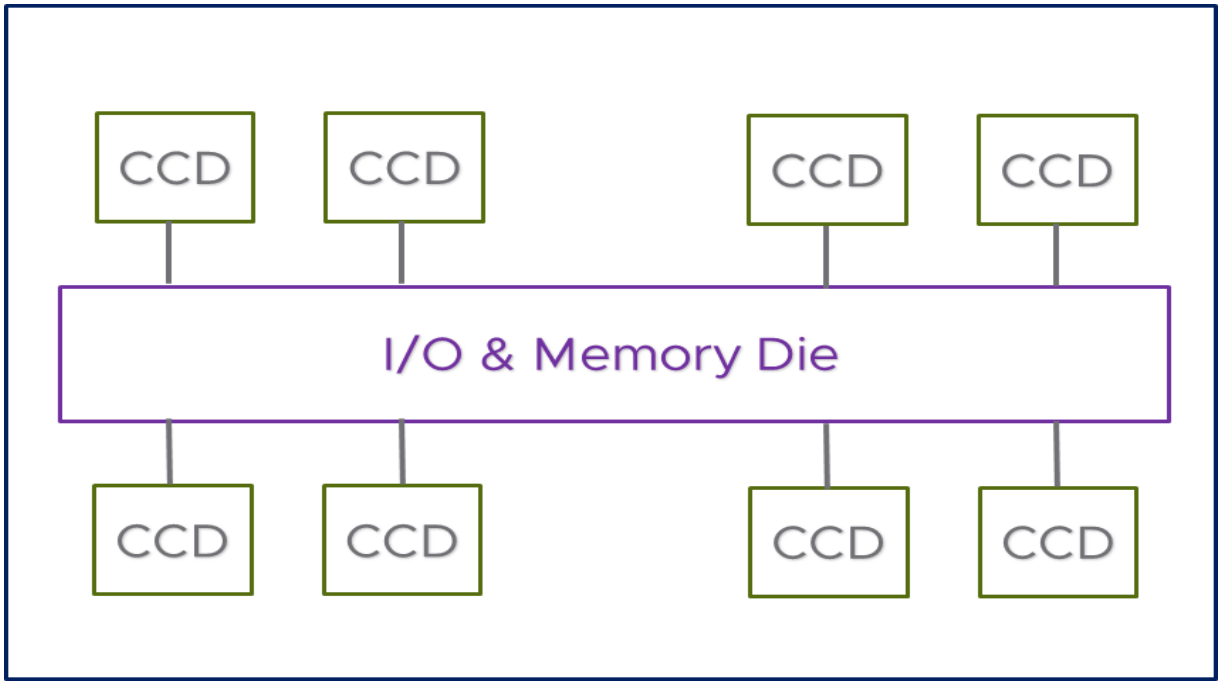

Figure 1. Simplied view of a single socket AMD EPYC 7702 (Rome) processor architecture

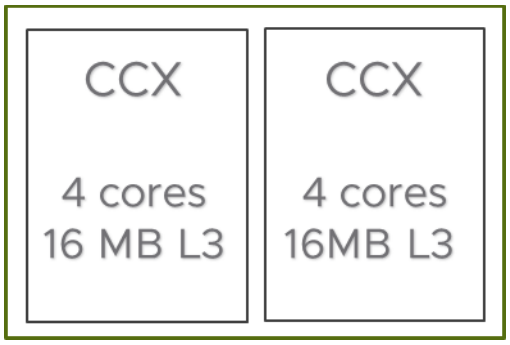

Figure 2. Compute/Core Complex Die (CCD) for AMD EPYC 7702 (Rome)

AMD EPYC 7702/7742 processors each contain 64 cores and 4 cores per LLC, with a total of 16 LLCs. AMD EPYC sockets have a similar CCX-based design across all generations: Gen1 (Naples), Gen2 (Rome), and Gen3 (Milan). On AMD EPYC, the cost of cache-to-cache sharing may also vary greatly depending on the relative location of the caching/home agent and the target cache line, as well as processor generations. Such sharing cost tends to remain constant within a socket on other x86 processors. For example, as seen in software, the cache access latency can increase up to 3x for inter-CCX cache access compared to intra-CCX cache access latency. In short, one Rome socket has up to 256MB (16x16MB) system-wide L3 cache, but this large cache size comes at the cost of non-uniform cache-access (NUCA) latencies. Software vendors must use these architectural features to optimize their software stacks.

AMD also presents a firmware setting called NUMA node per socket (NPS), which changes the memory interleaving policy to present either 1, 2, or 4 NUMA nodes per single socket. Depending on the NPS settings, each NUMA node can have multiple LLCs, up to 16 LLCs per NUMA node under the default NPS-1 configuration. There is another BIOS option called CCX-as-NUMA, which presents each LLC/CCX as a NUMA node to operating systems, but it does not change the memory channel interleave policy that is still dictated by the NPS setting. The CCX-as-NUMA option was invented to aid operating system schedulers (hypervisor and guest) when they are not optimized or aware of AMD EPYC topology. However, this option will skew certain OS assumptions about NUMA architecture and can degrade performance for some applications, for example, VMware View Planner™ and VMware VMmark® (we discuss these later in this technical paper).

CPU Scheduler Behavior on EPYC before vSphere 7.0 U2

This section covers what changes were made to the CPU scheduler for AMD EPYC processors before the vSphere 7.0 U2 release and the typical behavior of that scheduler. We also discuss how BIOS options were used as workarounds to optimize performance before vSphere 7.0 U2.

CPU Scheduler changes and behavior before vSphere 7.0 U2

One of the keys to achieving good performance on AMD EPYC processors is optimizing for their unique CPU topology where a NUMA node or a CPU socket contains multiple last-level caches (LLC). Traditionally, the vSphere CPU scheduler has been designed and optimized based on the assumption that a NUMA node contains only a single LLC.

VMs running on AMD EPYC with earlier vSphere versions may suffer from significant CPU contention, in some cases (see KB 67996). More specifically, the vSphere CPU scheduler before 6.5 U3 and 6.7 U2 is built with the assumption that a NUMA node boundary is aligned with the LLC boundary. When a VM boots, it is divided into logical groups, and each logical group is placed on a single NUMA node and LLC. Such logical groups are sized to maximize locality (cache and memory) and minimize contention (CPU and memory). However, this same policy is not optimal on AMD EPYC. When a logical group is put on an AMD EPYC NUMA node, all vCPUs might be packed into one of the LLCs in that NUMA node because the scheduler was not aware of the multiple LLCs per NUMA node. vCPUs may experience significant contention if the number of vCPUs of the logical group exceeds the capacity of the LLC, and performance may decrease.

In order to provide some initial performance improvements for AMD EPYC, workarounds in the vSphere CPU scheduler have been implemented and backported to vSphere 6.5 U3, 6.7 U2, and 7.0. These workarounds try to spread the vCPUs of a VM across the cores within a NUMA node based on the CPU utilization. It leverages full compute capacity in the NUMA node for a more balanced placement to avoid CPU hot spots and to improve the performance of some workloads by up to 40%. In some corner cases, CPU contention may still occur pre-vSphere 7.0 U2.

A limitation of the workaround is that the vSphere CPU scheduler may spread vCPUs from the same VM across multiple LLCs within a single NUMA node. To explain the workarounds and limitations, consider the following scenario: two 4-vCPU VMs (VM1 and VM2) and one 8-vCPU VM (VM3) as shown in figure 3 (hyper-threading is enabled but not shown for brevity) are running on an AMD EPYC system.

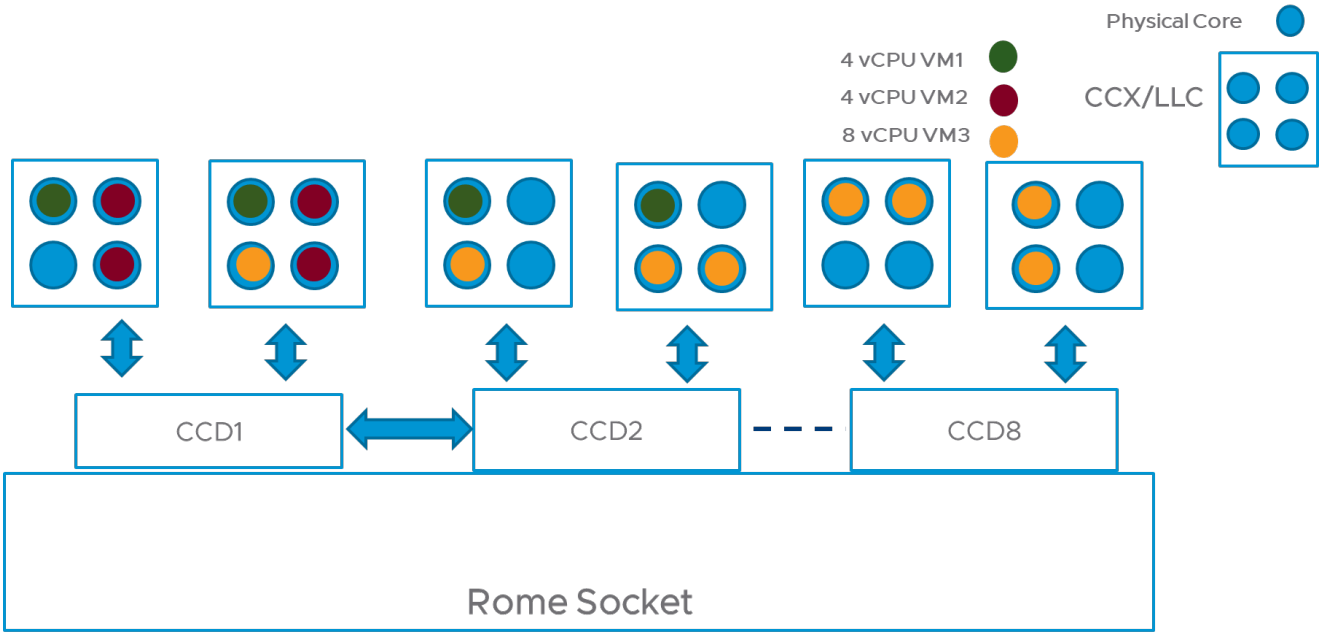

Figure 3. Example of the CPU scheduler behavior on AMD EPYC before vSphere 7.0 U2

NOTE: The example shows just one possible scheduling of these three VMs and depending on the characteristics of your workload and communication pattern, you may experience a different scheduling decision.

VM1's vCPUs are scheduled such that it has one vCPU per LLC. VM2's vCPUs are scheduled such that two vCPUs sit on an LLC. Similarly, the 8-vCPU VM3 is also scheduled such that the vCPUs are spanning five LLCs. Even though the VMs are not experiencing any CPU contention, the VMs won't be able to take advantage of LLC locality because the vCPUs are scattered across LLCs. On top of that, vCPUs from all the VMs may suffer from a "noisy neighbor" effect, which can result in cache contention/pollution because the vCPUs are from different VMs. This example illustrates how the CPU scheduler can lead to non-optimal VM (and its vCPUs) placement decisions if it is not fully aware of the architectural characteristics.

The placement of vCPUs across LLCs plays a crucial role for performance, and the unique topology design of AMD EPYC processors has necessitated changes in the vSphere CPU Scheduler in order to optimize performance. Even though some applications may benefit from a larger total LLC footprint, aligning a single VM to the fewest number of LLCs possible without overpacking should give better performance for the majority of workloads. Such a performance boost may come directly from reduced communication cost and more efficient cache sharing if vCPUs are placed on the same LLC.

Workarounds by changing BIOS settings before vSphere 7.0 U2

Prior to vSphere 7.0 U2, the CPU scheduler could not do the placement automatically to take advantage of LLC locality on AMD EPYC processors, so VMware published a list of performance blogs to educate customers about some possible complex tunings (for example, various NPS modes and CCX-as-NUMA) based on workload types and VM configurations to regain some of the LLC locality benefits. See the list of performance blogs in the References section at the end of this paper.

For example, NPS-4 mode can be used to rely on the NUMA rebalancing policy to maximize LLC locality, although NPS modes are designed for workloads that benefit from memory interleaving and not to influence the software in a manner that is used today on AMD EPYC processors.

By using the CCX-as-NUMA setting, the system presents each LLC as a NUMA node. The hypervisor scheduler then considers LLCs as NUMA nodes and the NUMA rebalancing policy becomes responsible to manage VMs across LLCs. This CCX-as-NUMA workaround theoretically makes sense and can be beneficial for some custom situations and scenarios, but it has many practical limitations. A dual-socket AMD EPYC processor has 32 NUMA nodes when configured with CCX-as-NUMA. There are not many machines that have 32 NUMA nodes, and the NUMA rebalancing policy is not designed to work optimally with such a large number of NUMA nodes.

The NUMA rebalancing also does not distinguish between which NUMA nodes belong to a particular socket, and it treats all NUMA nodes equally when doing load balancing. On top of that, the vSphere memory scheduler also depends on certain assumptions; for example, remote memory access is costly across NUMA nodes and it would copy memory pages when VMs are migrated to different NUMA nodes. In CCX-as-NUMA, where memory access cost is almost uniform within a socket, such page migrations would be unnecessary. The memory scheduler also does page sharing optimizations within NUMA node boundaries, and with so many NUMA nodes, page sharing would be reduced—that can result in memory ballooning causing performance problems. We documented such a case in VDI environments in one of our blogs. Overall, it is extremely complex to optimize AMD EPYC performance today.

Given the above issues, designing a CPU scheduler that is organically aware of multiple LLCs per NUMA node is the key to achieve out-of-the box optimal performance on AMD EPYC. vSphere 7.0 U2 presents new optimizations to the vSphere CPU scheduler, which is designed from the ground up to explore and make full use of the special CPU topology on AMD EPYC processors. We show that the CPU Scheduler in vSphere 7.0 U2 can achieve **up to 50% better performance** on AMD EPYC processors with out-of-the-box BIOS options and vSphere settings.

Design of AMD-specific vSphere CPU Scheduler Changes

We augmented the vSphere scheduler such that it is aware of the topology where each NUMA node may have multiple LLCs with distinct cache cost or non-uniform cache access latency (NUCA). The goal is to automatically explore the locality benefits offered by the hardware architecture by placing related contexts, such as vCPUs of the same VM, or multiple communicating contexts, onto as few LLCs as possible. Intuitively, a VM needs to be divided into logical groups that can be individually placed on LLCs to benefit from LLC locality without over-packing.

To explain this new approach and how the VMs are split up into these logical groups, consider a scenario where you have four VMs running on vSphere on EPYC Rome processor as shown in figure 4.

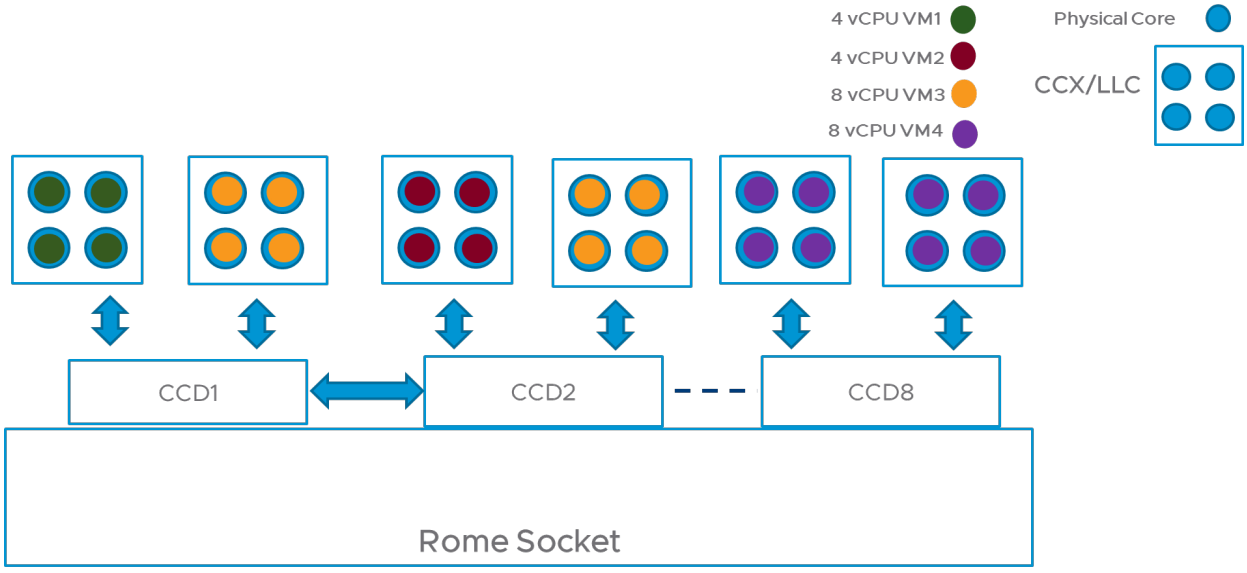

Figure 4. Example of vSphere 7.0 U2 behavior of CPU scheduler on AMD EPYC

VM1 and VM2 are 4-vCPU VMs. VM3 and VM4 are 8-vCPU VMs. As shown in figure 4, both 4-vCPU VMs are nicely aligned on a single LLC (the blue box with 4 physical cores each—blue dots) to maximize locality. VM3 has 8 vCPUs (larger than the LLC size of 4 cores per LLC). It is broken into two logical groups, and each group is 4 vCPUs, which matches the number of cores per LLC. Each group is placed on a single LLC. VM4's behavior is the same. However, note that in this example, VM3's logical groups (two logical groups with 4 vCPUs each) are placed on different CCDs, whereas they are placed on the same CCD for VM4. This is due to the lack of topology information exposed by the hardware (to operating systems) that describes the difference between CCX and CCD. The vSphere CPU scheduler will not try to place them on the same CCD, even though peak performance may be achieved if they are placed on the LLCs that are on the same CCD. The latest

Gen 3 AMD EPYC processor "Milan" has a single CCX/LLC per CCD and such a placement is not needed anymore.

Performance Testbed and Methodology

To quantify the performance impact, we conducted a wide variety of experiments using enterprise-level benchmarks and microbenchmarks that stress all the components of a system, including CPU, memory, storage, and network. We based our evaluation on a single and dual-socket AMD EPYC 7702 (Rome) setup, ProLiant DL385 Gen10 Plus, with each socket containing 64 cores (128 hyper-threads), 4 cores per LLC. The system ran at 2.0 GHz base frequency with up to 2TB of total DRAM. We also used Gen 3 AMD EPYC processor (Milan) for some tests as well. These systems were dual-socket AMD EPYC 7713 64-core processors, 8 cores per LLC, with a base frequency of 2.0GHz with 1TB of total DRAM. All the experiments were performed using out-of-the-box BIOS options, namely NPS-1 and default ESXi options, unless otherwise specified in certain scenarios to make the results clearer.

We first present the enterprise benchmark results, which reflect the most common customer scenarios.

> NOTE: We internally tested the same 7.0 U2 build and turned off the scheduler-related optimizations to isolate the impact from scheduler-related enhancements. This is to make sure that most of the performance gains we claim in this paper between 7.0 U2 and 7.0 U1 are due to scheduler optimizations and not some other software component.

Enterprise Benchmark Performance

SQL Server using HammerDB

To begin quantifying the LLC locality benefits, we first ran a controlled experiment using the HammerDB version 3.1 benchmark with a SQL Server database and a TPC-C OLTP workload profile.[1]

[1] This is a non-compliant, fair-use implementation of the TPC-C workload for testing the benefits of a feature; our results should not be compared with official results.

Benchmark specifications and configuration:

- We used local Samsung PM1725a NVMEs for this test, which ran on SQL Server 2016 on a Windows Server 2016 virtual machine using the HammerDB TCP-C workload profile of 70% reads and 30% writes.
- We configured the workload profile to have 80 users per 8-vCPU VM using 32GB RAM and 1,000 warehouses.
- We included a 100GB database.
- We ran the test for 15 min with 5 min ramp up.
- We loaded the HammerDB load client on each of the SQL Server VMs locally.
- We looked at the metric of transactions per minute and show normalized throughput with respect to vSphere 7.0 U1 in figure 5.

System under test (SUT) specifications:

- The system under test was a dual-socket AMD EPYC 7702 with 2TB DRAM, with 64 cores in total, 4 cores/LLC, and 2 LLCs per CCD.

We evaluated a couple of scenarios where we manually placed the vCPUs by pinning them to a particular LLC across just one CCD and across eight CCDs. We observed that when vCPUs are spread across all eight CCDs on one socket, performance was worse (50%) compared to when all vCPUs were on a single CCD. We believe the reduced throughput is a result of reduced locality due to more shared database meta-data access misses, remote data access misses served from remote CCXs, and local data accesses that become remote when the guest OS migrates the threads accessing the data. This simple experiment really shows the value of maximizing LLC locality for database workloads.

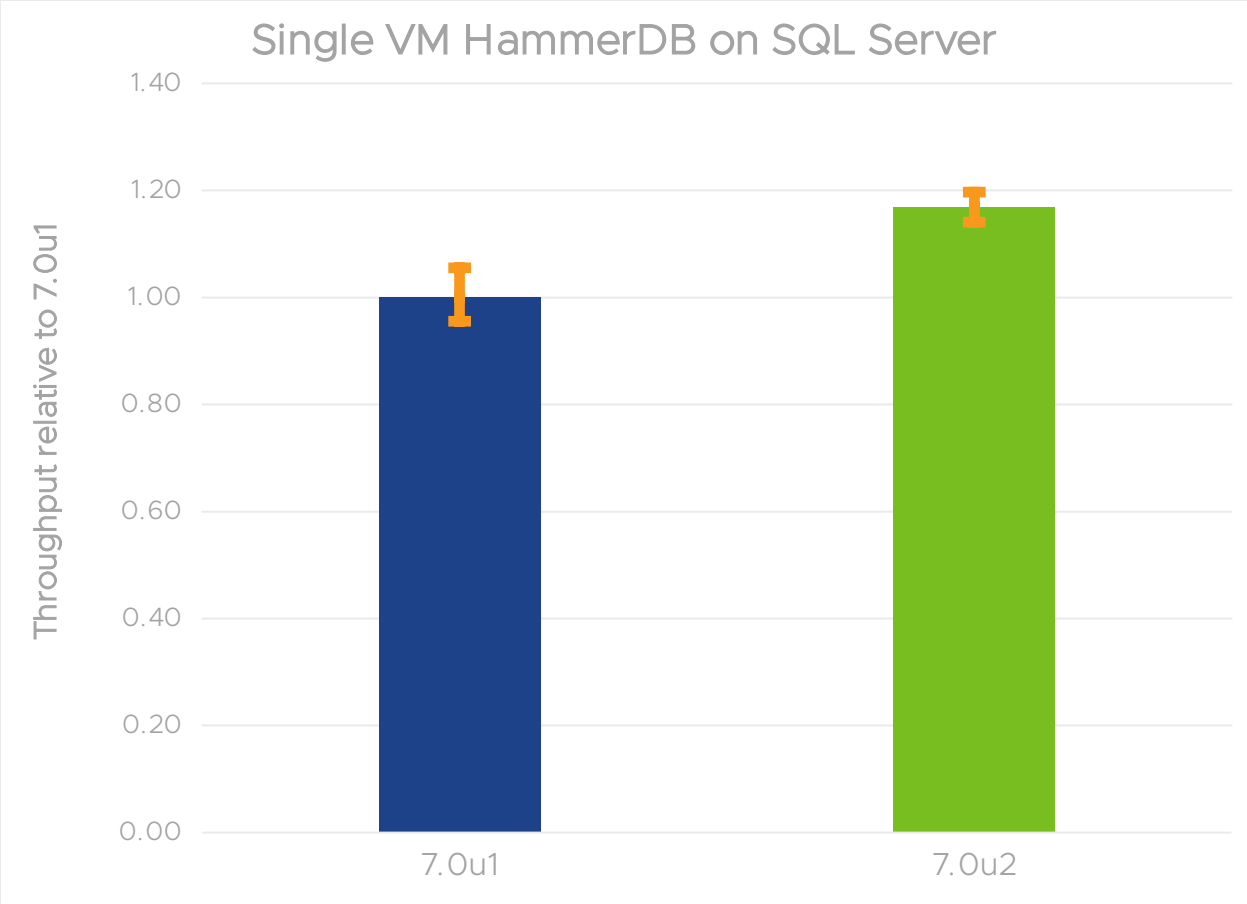

Figure 5. Multiple runs of the single VM HammerDB benchmark show vSphere 7.0 U2 CPU scheduler is much better than 7.0 U1

We then conducted out of the box, single VM tests using HammerDB. Figure 5 above shows the results of the average of eight single VM runs conducted on 7.0 U1 and 7.0 U2 and compares their performance. We performed several runs to show that there is a lot of variation in 7.0 U1. The reason for this is that the scheduler does not have any particular policy to schedule vCPUs on an architecture with multiple LLCs. It will just try to spread the vCPUs to minimize contention. The lowest score corresponds to the scheduling where vCPUs were spread across more CCDs and vCPUs were scheduled on fewer CCDs with the highest score. In 7.0 U2, the performance is not only increased up to 24% better (17% better on average), but it has minimal variance. Overall, there is 50% reduction in standard deviation (4x reduction in variance) in 7.0 U2. This is because in vSphere 7.0 U2 the CPU scheduler will always try to place the 8 vCPUs on just two LLCs.[2]

[2] These 8 vCPUs won't be placed on LLCs located on a single CCD by policy design, but it can happen by chance. In these runs, vCPUs were usually placed on LLCs that were on separate CCDs. Due to this phenomenon (vSphere not knowing the location of LLCs on CCDs), we cannot achieve peak performance (which was 10-12% higher if we manually placed all vCPUs on a single CCD).

We also scaled out this test by increasing the number of VMs until we reached the peak performance at 30 VMs (16 million transactions per minute). The performance gain observed in these cases is up to 16%. In the multiple VM scenario, the gains are not just due to maximizing LLC locality—they are also because, in 7.0 U2, the multiple VMs were confined across their own set of LLCs and were not interfering with other VM's cache usage (noisy neighbor effect).

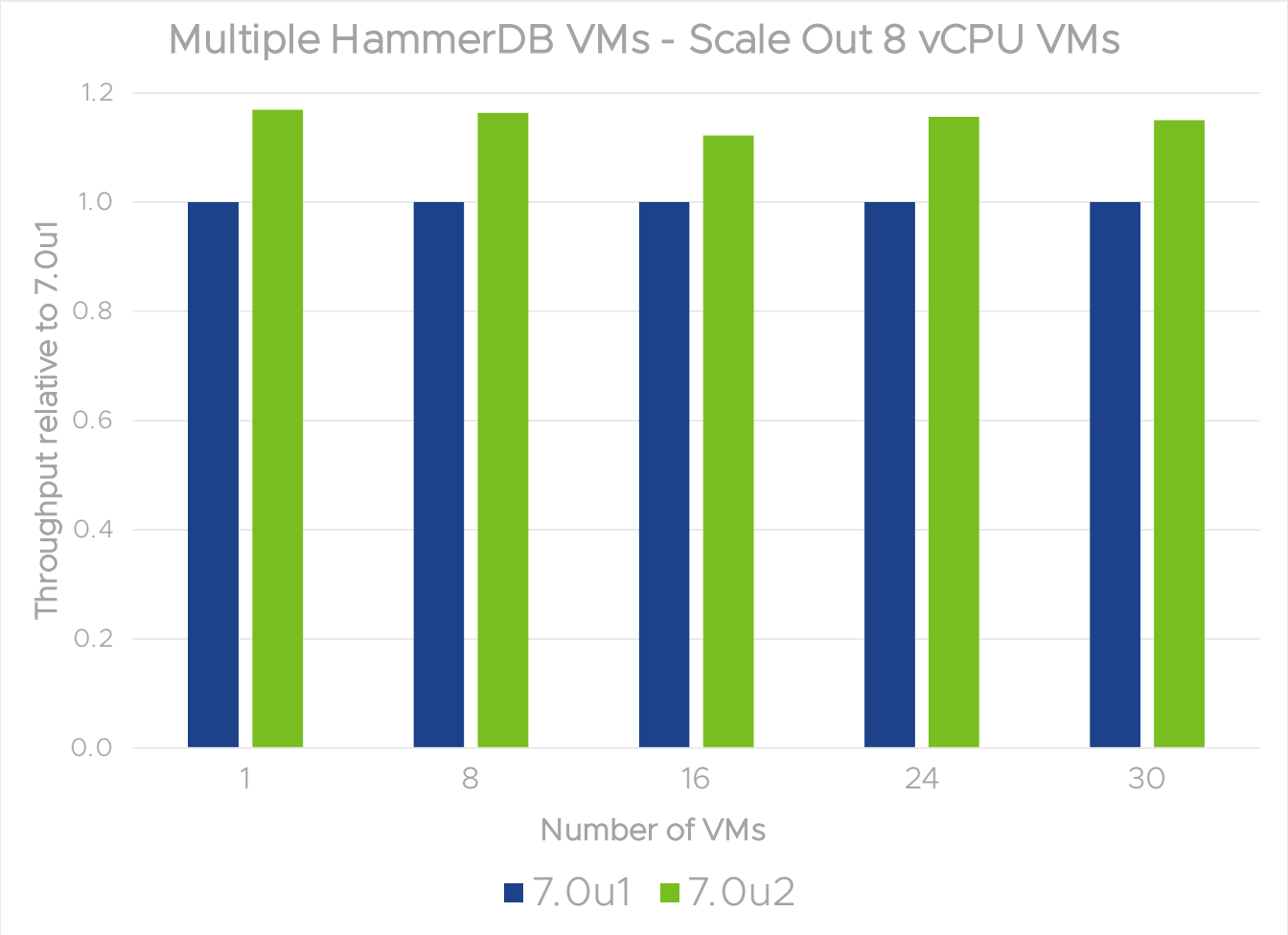

Figure 6. When scaling out multiple HammerDB VMs, the vSphere 7.0 U2 CPU scheduler performs better than the 7.0 U1 scheduler.

Virtualization infrastructure and mixed load – VMmark3

VMware VMmark® version 3 is VMware's virtualization benchmark that combines several workloads to simulate the mix of applications in customer datacenters. These workloads include a scalable web simulation, an e-commerce simulation, and a standby server. The scalable web simulation is Weathervane (WV), which is an application-level benchmark for virtual infrastructure and cloud performance tests. The e-commerce simulation is DVD Store version 3 (DS3), which is a complete online e-commerce test application with a back-end database component, a web application layer, and driver programs. Both workloads, DS3 and WV, also generate cyclical load on the system to mimic a bursty profile. For more details about the benchmark, see the blog Introducing VMmark3.

These commonly virtualized applications are bundled into predefined units called "tiles" in VMmark. Each tile contains 19 VMs needed for the workloads that make up VMmark. Load is incremented by adding more tiles of VMs. Each workload in all tiles must meet the defined quality of service (QoS) metrics for a run to be valid. At the end of the run, a score is generated based on the number of tiles and how well the workloads performed within each tile. The most important VMmark score we look at is the DS3 workload score.

VMmark performance on AMD EPYC processor "Rome"

For these tests, we ran VMmark in single host mode—all infrastructure operations were disabled and as many tiles as possible were run on a single host[3]. We used a Dell EMC Unity 650F storage array in these experiments.

We determined the maximum number of tiles that could be supported (determined by the workload QoS metrics) for 7.0 U1 vs 7.0 U2, as well as the VMmark application scores for these runs. Figure 7 shows the normalized VMmark application scores relative to 7.0 U1.

[3] This was done within the academic rules of the benchmark and external publication/competitive comparisons are not allowed to be configured in this way. Please review the VMmark Run and Report Rules Guide for more information.

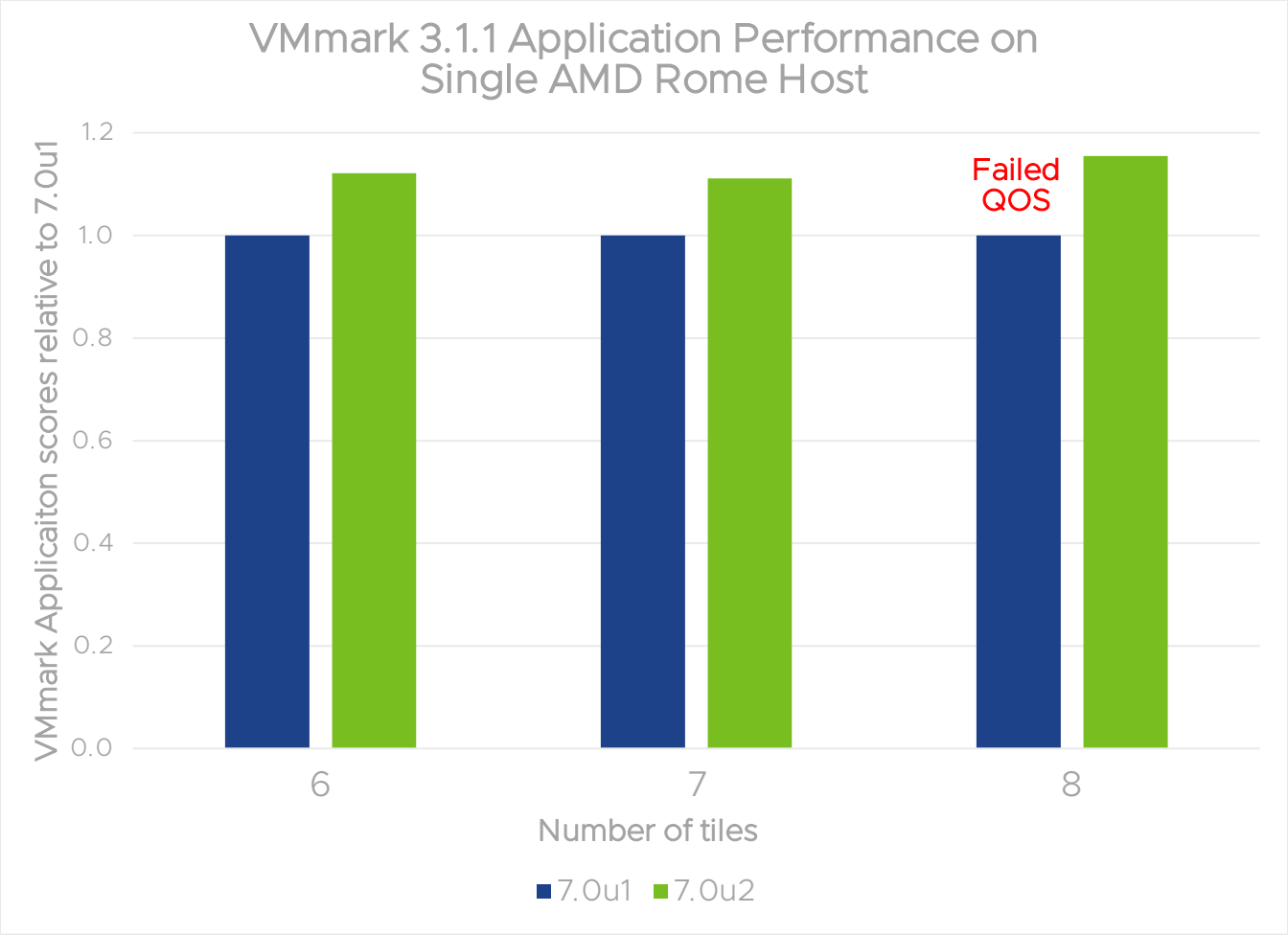

Figure 7. VMmark performance results show the CPU Scheduler in vSphere 7.0 U2 is consistently higher than vSphere 7.0 U1.

We started with six tiles and continued to increase the number of tiles until VMmark failed its application QoS metrics. In 7.0 U1, only seven tiles passed QoS. In 7.0 U2, we were able to run eight tiles with no QoS issues. Even though 7.0 U1 passed the QoS metric at seven tiles, it was very close to failing. We see up to 16% improvement in VMmark application scores in 7.0 U2. We used both sockets of the AMD EPYC "Rome" server that had a total of 256 logical threads (128 cores). Seven tiles translate to 301 vCPUs, and eight tiles translate to 344 vCPUs. This means that with eight tiles, we were 344/256 = 1.34x over-committed in terms of vCPUs. This level of over-commitment tests the new CPU scheduler's load-balancing algorithm in addition to maximizing locality. In 7.0 U2, being able to run eight tiles without failing QoS vs seven tiles in 7.0 U1 translates to 14% higher VM density in 7.0 U2.

In-depth performance analysis

We did some fine-grained analysis and looked at latencies of the two workloads. Figures 8 and 9 below show the raw latencies reported by three DS3 web servers driving load against the DS3

database for each tile, in the eight-tile experiment. The X-axis is time, and the Y-axis is the latency of operations in milliseconds. For eight tiles, the total number of web servers are 8x3= 24, which corresponds to the number of colors on the scatter plots at a particular time. You can see that latencies are approaching 4,500 milliseconds in 7.0 U1 vs latencies being well below the 2,600-millisecond mark in 7.0 U2 (lower is better).

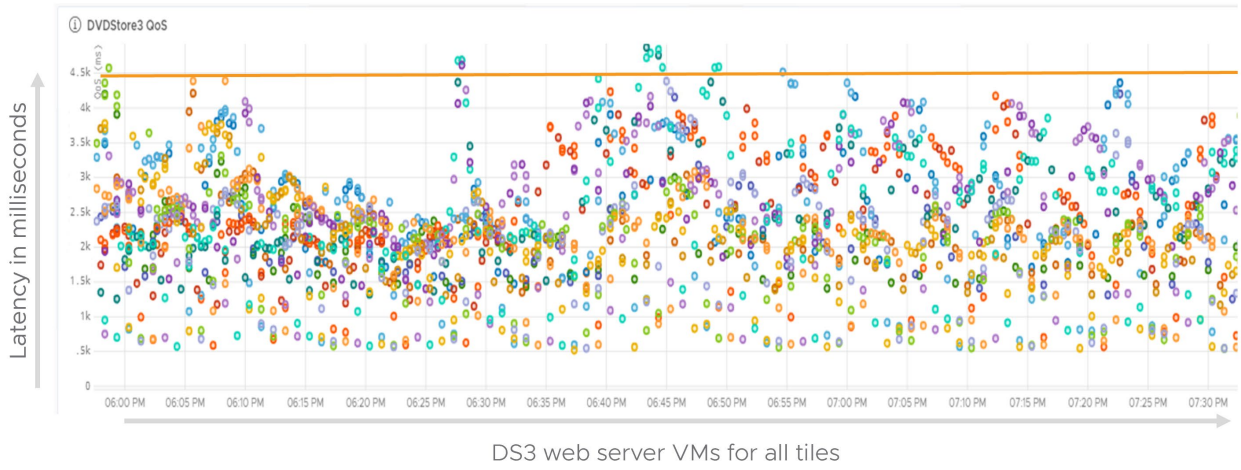

Figure 8. DS3 web server latencies over time in 7.0 U1 for 8 tiles

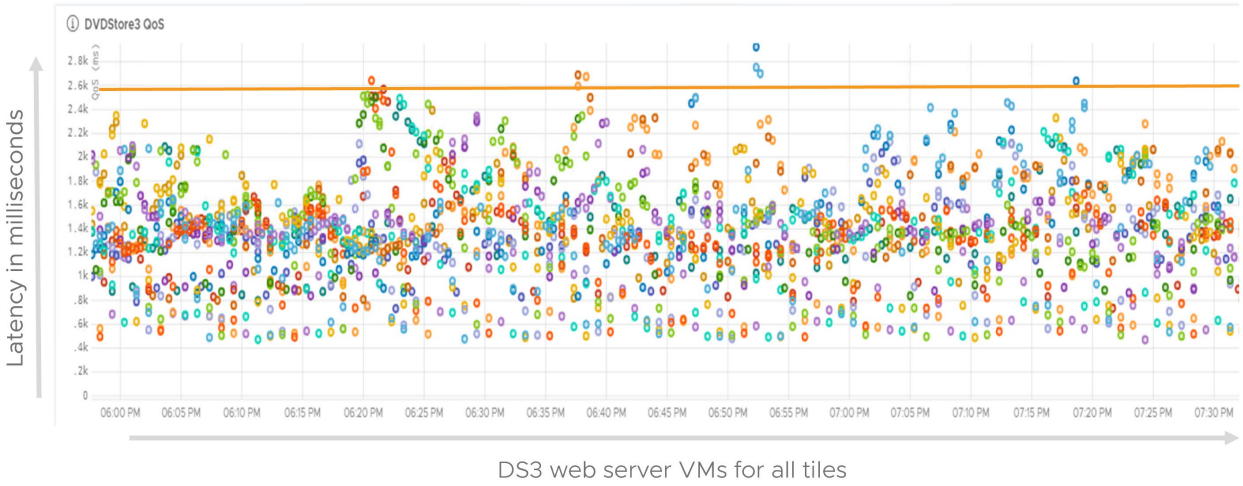

Figure 9. DS3 web server latencies over time in 7.0 U2 for 8 tiles

We also calculated the 99th percentile latencies for both workloads of DS3 and WV. This calculation was based on the average latencies reported periodically by the benchmark for both workloads. For seven tiles, the latencies in 7.0 U1 for DS3 and WV were 1.41x and 1.18x higher respectively when compared to 7.0 U2, which means that the system was much more efficient overall in 7.0 U2, not just in terms of raw scores but in terms of latencies/response times as well. At eight tiles, the DS3 and WV latencies in 7.0 U1 were 1.72x and 1.18x higher compared to 7.0 U2.

The reason for this performance gain is two-fold. First, gains are due to VMs achieving better LLC/CCX locality. Second, gains are due to better load balancing, resulting in better CPU latency and less CPU contention (ready times and hyper-threading contention).

One of the other fine-grained metrics we used internally was inter-LLC vCPU migrations. Figure 10 shows the inter-LLC migrations for the DS3 database VM vCPUs. The X-axis has the 8 vCPUs of these VMs for seven tiles (total of 56 vCPUs). These 56 vCPUs were monitored for 2.5 hours for inter-LLC migrations and plotted. 7.0 U1 has 8x more inter-LLC migrations compared to those in 7.0 U2. There were 2.3 inter-LLC migrations per second in 7.0 U1 compared to just 0.3 migrations per second in 7.0U2. Similar inter-LLC migration reductions were seen for the Weathervane workload VMs as well (not shown here). Due to the improvement in inter-LLC migrations, there were fewer cache misses and overall CPU cycles were up to 10% lower in 7.0 U2, which resulted in up to 20% better CPU efficiency (CPU cycles divided by VMmark application scores).

We also tested a CCX-as-NUMA configuration at seven tiles, but this test failed QoS with similar scores compared to the 7.0 U2 NPS-1. For eight tiles, we observed 12% lower performance (also failed QoS) compared to the 7.0 U2 NPS-1 configuration.

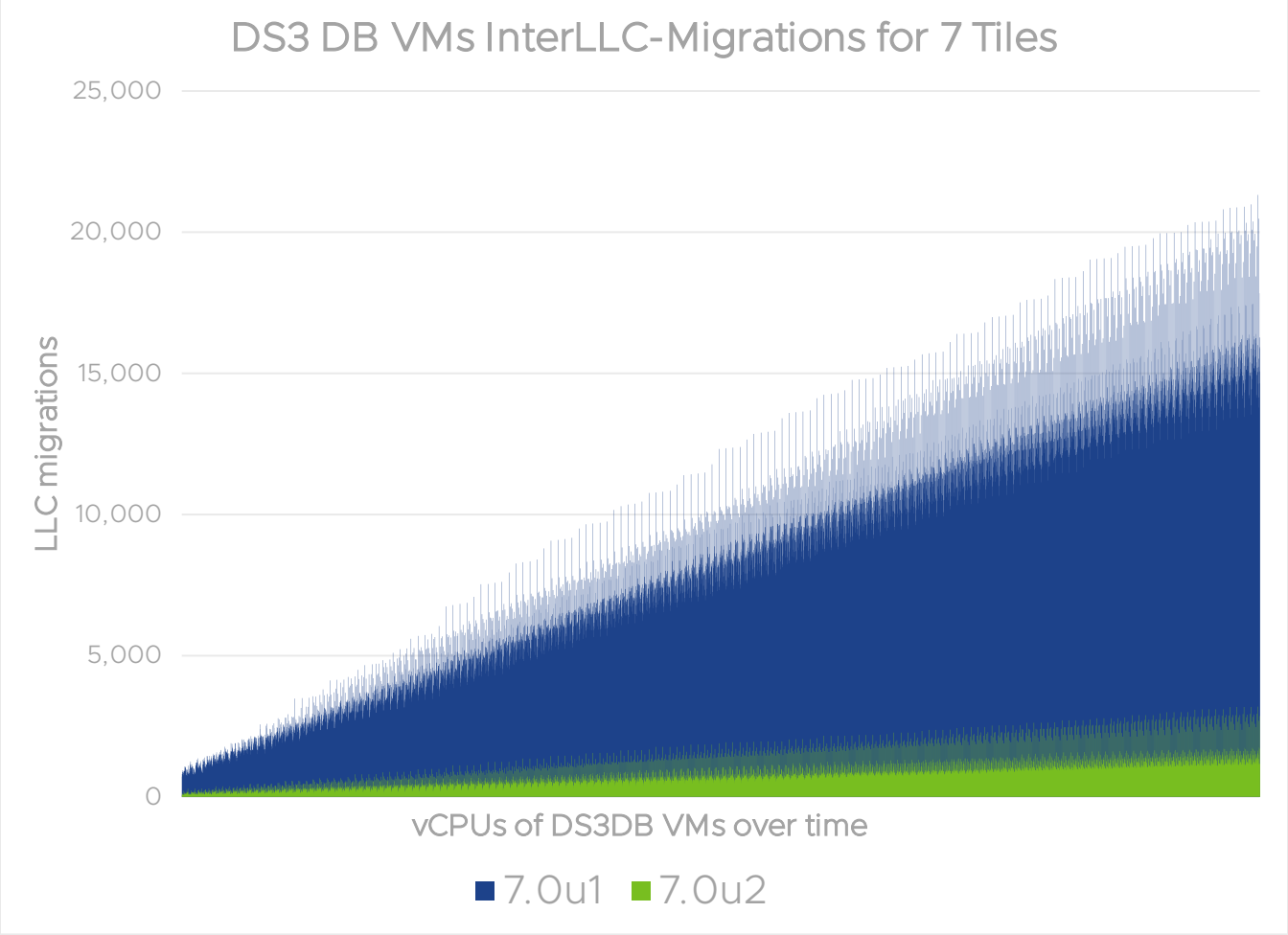

Figure 10. Inter-CCX/LLC migrations for DVD Store 3 (DS3) database VM in VMmark; lower is better

VMmark performance on AMD EPYC "Milan"

We observed similar improvements on a 2-host Milan setup. In this case, since we were using two hosts, the infrastructure operations were included in the full VMmark scores. The hosts used 2-socket AMD EPYC 7713 processors with 1TB of RAM.

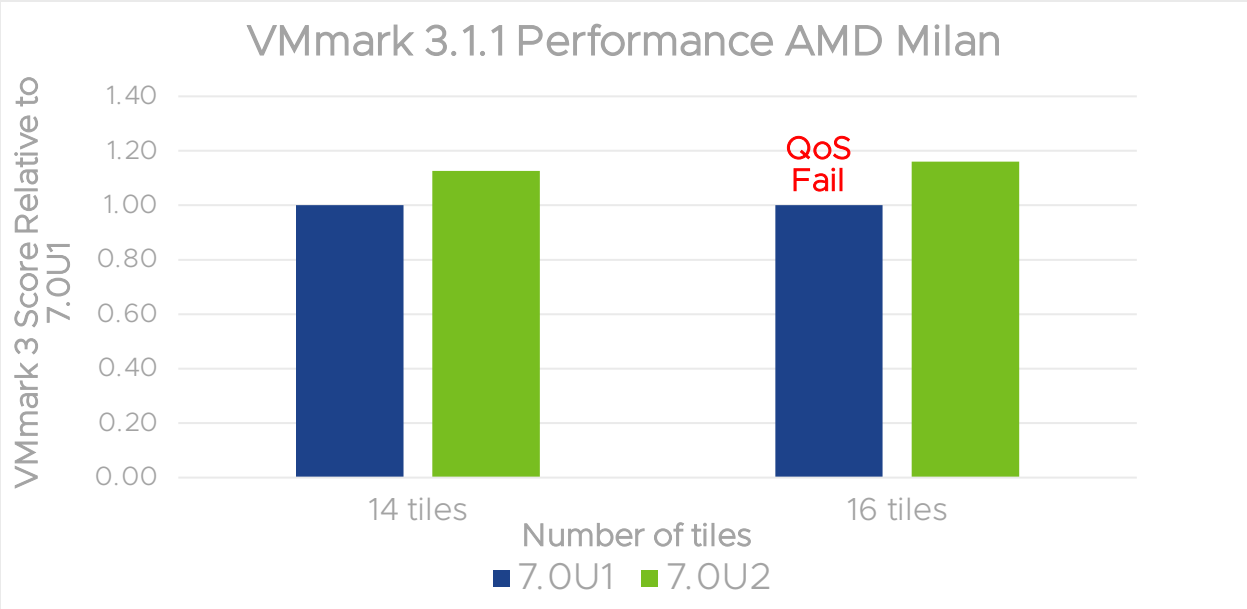

Figure 11. The vSphere 7.0 U2 CPU scheduler scored higher relative to 7.0 U1 for both 14 tiles and 16 tiles. At 16 tiles, 7.0 U1 could not achieve its quality-of-service goals.

The maximum performance on these hosts using 7.0 U1 was found at 14 tiles. Multiple QoS failures occurred when we tried to increase the tiles to 16. 7.0 U2 increases performance without needing any tunings and allows the benchmark to pass at 16 tiles.

We also tested a CCX-as-NUMA configuration at sixteen tiles. We observed 5% lower performance compared to the 7.0 U2 NPS-1 configuration.

CockroachDB using YCSB

We created a CockroachDB cluster of 6 nodes using VMs installed with CentOS 8, and configured with 16 vCPUs and 32GB of RAM. The host featured AMD EPYC Milan processors with 2TB of RAM and 3TB of local NVMe storage. The AMD EPYC host had two sockets with 64 cores per socket for a total of 128 cores and 256 threads. We installed CockroachDB 20.2.4 and initialized the database cluster with 25% of VM memory for cache. We used a seventh VM as a driver for the tests; it was also running on the same host.

We used the Yahoo Cloud Service Benchmark (YCSB) that CockroachDB includes as its workload command. This is a throughput-based benchmark that simulates a cloud service. YCSB was initially released as open source in 2010. This benchmark has been implemented here for CockroachDB, but it is also available for many other databases and has been widely used in recent years.

We conducted these tests specifically to see if the performance of a clustered value-store CockroachDB database would perform better with 7.0 U2 on the AMD EPYC platform compared to 7.0 U1.

Figure 12 below shows the results of the tests with 7.0 U2 performing approximately 50% better than 7.0 U1. During testing with 7.0 U1, we observed that all 6 VMs had been packed onto one NUMA node. This was due to sub-optimal NUMA rebalancing policy placement decisions. In 7.0 U2, some changes were made around the NUMA rebalancing policy to avoid such non-optimal decisions; these policy changes were agnostic to the CPU vendor. In order to quantify the gains due to the CPU scheduler changes made for the EPYC processors, we disabled an advanced NUMA config option, `LocalityWeightActionAffinity`, because NUMA rebalancing policy optimizations were around this option in 7.0 U2.

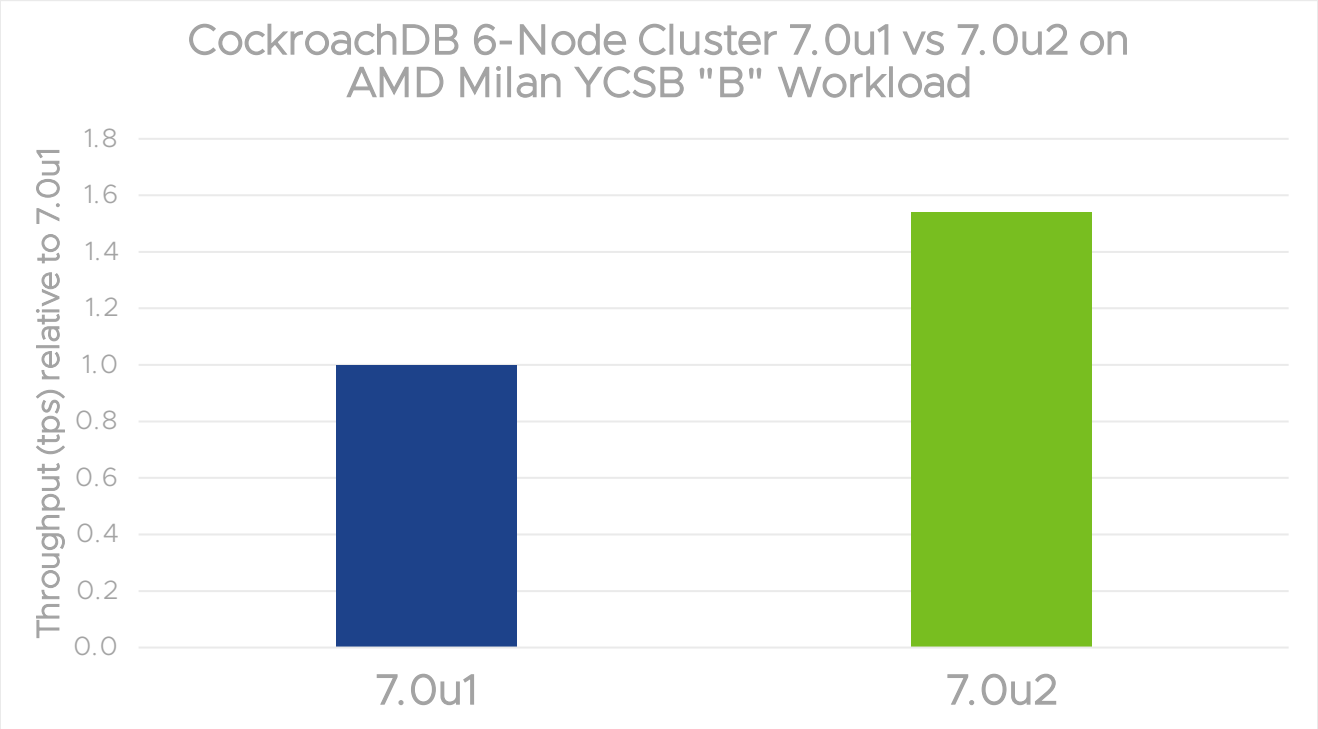

Figure 12. The vSphere 7.0 U2 CPU scheduler performed approximately 50% better than 7.0 U1 with a simulated cloud-service workload (YCSB "B") when run against a clustered value-store database (CockroachDB).

Figure 13 shows that with this option disabled, we see up to 7% performance gains. The rest of the gains (up to 43%) observed in 7.0 U2 are due to the NUMA rebalancing policy changes not specifically tied to the EPYC architecture. Disabling the `LocalityWeightActionAffinity` option had no impact on 7.0 U2 because NUMA placement decisions were already optimal in 7.0 U2.

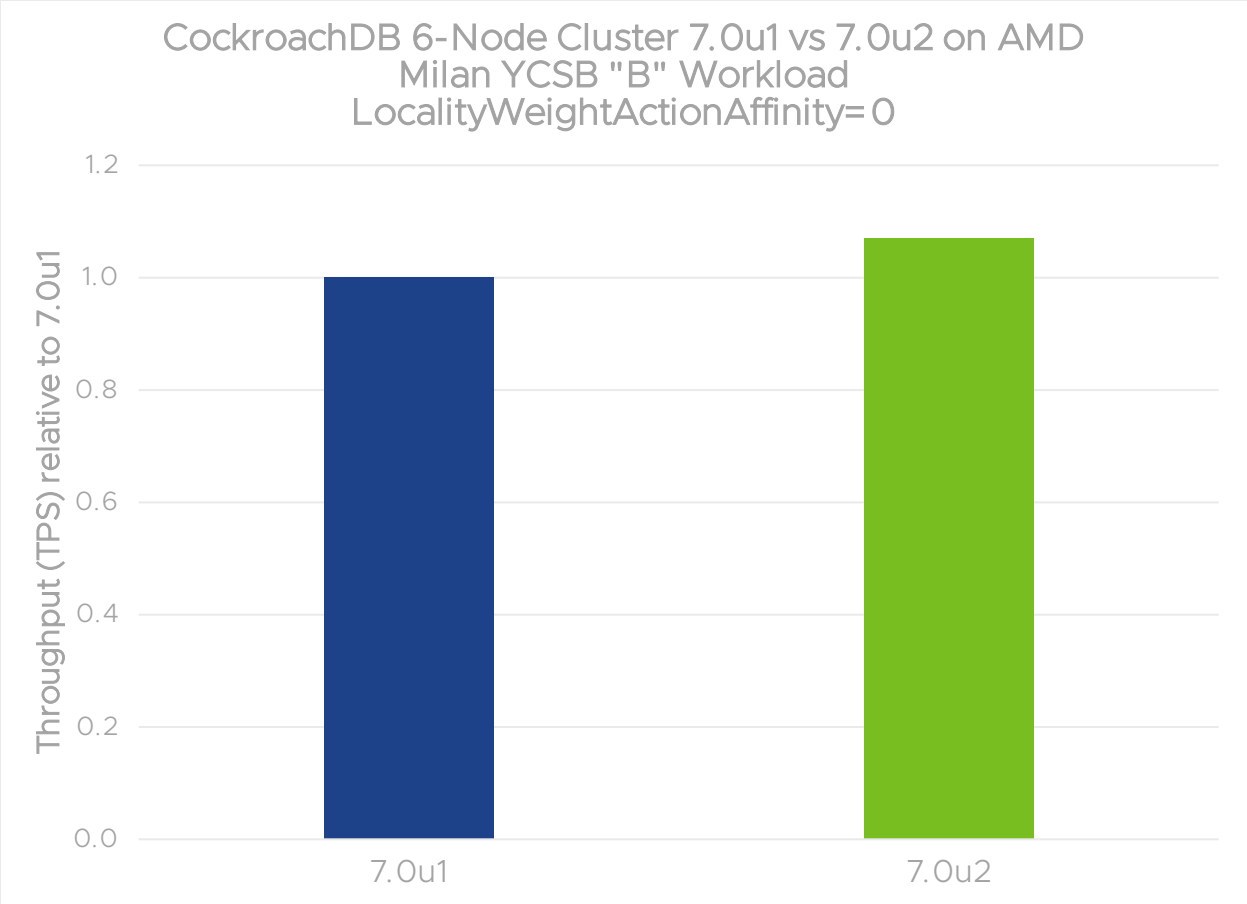

Figure 13. The vSphere 7.0 U2 CPU scheduler showed only a 7% improvement over the 7.0 U1 CPU scheduler when changing the `LocalityWeightActionAffinity` option to `0`.

Oracle Database and SQL Server using DVD Store

We used a dual-socket AMD EPYC Milan host with 64 cores per socket for a total of 128 cores and 256 logical threads for testing. The host had 2TB of RAM and 3TB of high speed local NVMe flash storage. For the Oracle tests, we created a VM with 8 vCPUs and 128GB of RAM and installed it with CentOS 8 and Oracle 12c; the VM's storage was on the local NVMe. We created a DVD Store 3 benchmark database of approximately 100GB for testing. DVD Store is an open-source benchmark that simulates an online store. We then cloned the VM 16 times to create the testbed for the multi-VM tests conducted.

In all test configurations, we drove up the load higher by steadily increasing the number of test users that were running on a separate load-driver VM located on a separate host. The benchmark reports the maximum throughput for each configuration. Normalized throughput with respect to 7.0 U1 is reported in figure 14.

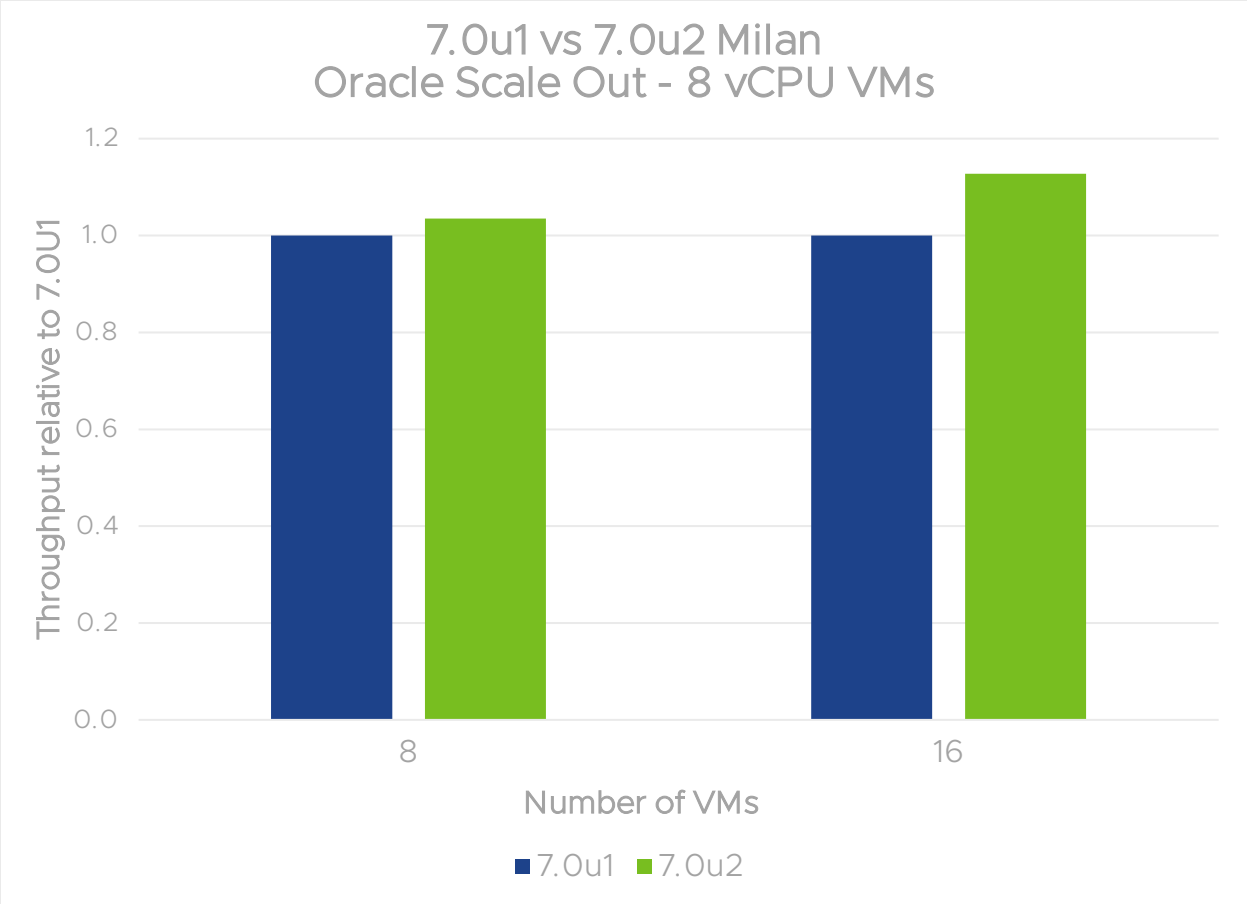

Figure 14. The vSphere 7.0 U2 CPU scheduler performed 13% higher than 7.0 U1 with 16 VMs running the DVD Store workload against Oracle Database. With 8 VMs, the performance was on par.

The performance with 8 VMs is 3.5% better, but at 16 VMs we see that the gain for 7.0 U2 reaches 13%. The 8 VMs gain is lower because in 7.0 U1, up to four vCPUs of certain VMs were confined to a single LLC, and many vCPUs were also not overlapping with other VM's vCPUs. This was just a coincidence and not by design in 7.0 U1.

We ran similar tests with a set of SQL Server VMs on the same host. We installed a VM with Windows Server 2019 and SQL Server 2019 with 8 vCPUs and 64GB of RAM. We created a DVD Store 3 benchmark database of approximately 50GB for testing. The VM was then cloned 16 times to create the testbed. Results from testing showed performance gains for 7.0 U2 with 8 VMs and 16 VMs of 21% and 13.4%, respectively. See figure 15.

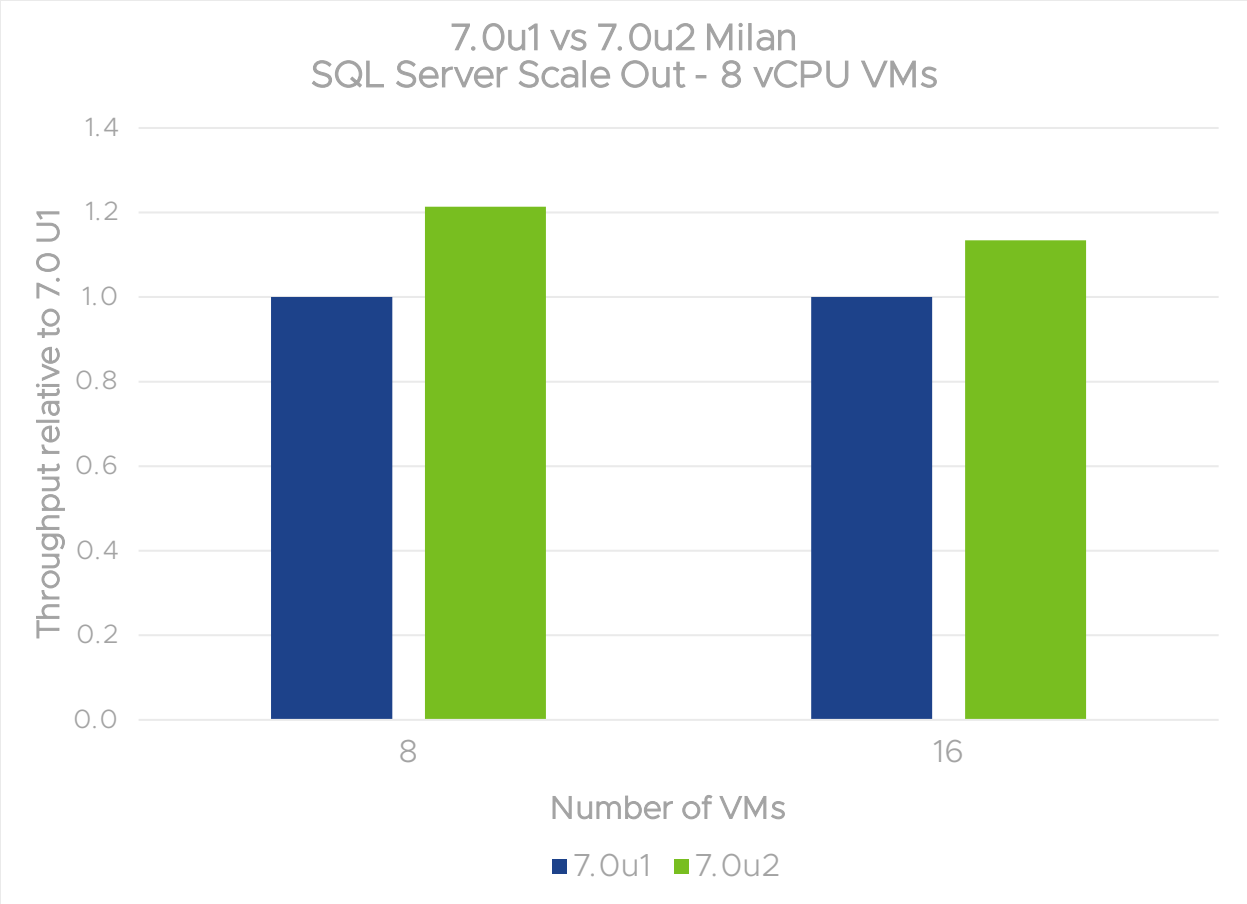

Figure 15. When we ran the DVD Store workload in 8 and then 16 VMs against a SQL Server database, the performance of the vSphere 7.0 U2 CPU scheduler increased in both cases over the CPU Scheduler in vSphere 7.0 U1.

Virtual desktop infrastructure with View Planner

VMware View Planner™ is a benchmark for virtual desktops. View Planner simulates users running many popular applications doing common user tasks. The tests measure how many virtual desktops can be supported while still maintaining the response times better than the defined maximum allowed. This ensures that the users' virtual desktops are still responsive at the peak load.

We used a single-socket server with AMD EPYC 7702, 64 cores per socket, and 1TB of RAM for these tests. We used View Planner version 4.6. We configured the virtual desktop VMs to have 2 vCPUs, 4GB of memory, 50GB of disk space, and Windows 10. We were able to run 6.3% more VMs (an increase in VM density) with 7.0 U2 vs 7.0 U1. The increase in performance is due to better load balancing, which results in fewer inter-LLC migrations and reduces CPU contention.

.

vSphere with Tanzu and Weathervane 2.0

We used two identical AMD hosts as the systems under test. Each host was a Dell EMC PowerEdge R7525 featuring AMD EPYC 7742 64-core processors. We configured the hosts with 2TB of memory and 25Gb Ethernet adapters, and we used vSAN for storage with a combination of SSDs and NVMEs. We provisioned a TKG cluster with 64 worker nodes of virtual machine class type **best-effort xlarge**, each of which we configured with 4 CPUs and 32GB of memory. This configuration assigned a total of 256 vCPUs to these Kubernetes worker nodes running on the two hosts. We used the open source Weathervane 2.0 benchmark to generate measurement data at different application instance load points (number of application instances). Weathervane provides a realistic application load with a multi-tier web application, multiple services, and multiple data stores.

The throughput metric in Weathervane is the number of users, named *WvUsers*, that represent the maximum number of simulated users that could interact with the application instances without violating quality-of-service requirements for response times on its underlying operations. A higher number represents better performance.

Figure 16 below shows the out-of-the-box throughput gains achieved in 7.0 U2 relative to 7.0 U1 throughput. The maximum gain was up to 45%. Upon investigating the scheduler statistics, we observed that the NUMA rebalancing policy was not making optimal decisions in 7.0 U1. For example, in the 16-application instance case, the scheduler migrated all 32 4-vCPU VMs (on each host) to one NUMA node only due to the communication pattern of vSAN worlds. In 7.0 U2, some changes were made around the NUMA rebalancing policy to avoid such non-optimal decisions. These NUMA rebalancing policy changes were agnostic to the CPU vendor. In order to quantify the gains due to the changes made for the AMD EPYC processors, we disabled an advanced NUMA configuration option, `LocalityWeightActionAffinity`. Disabling this option means that the communication between VMs and between VMs and system worlds will be ignored for VM placement decisions across NUMA nodes.

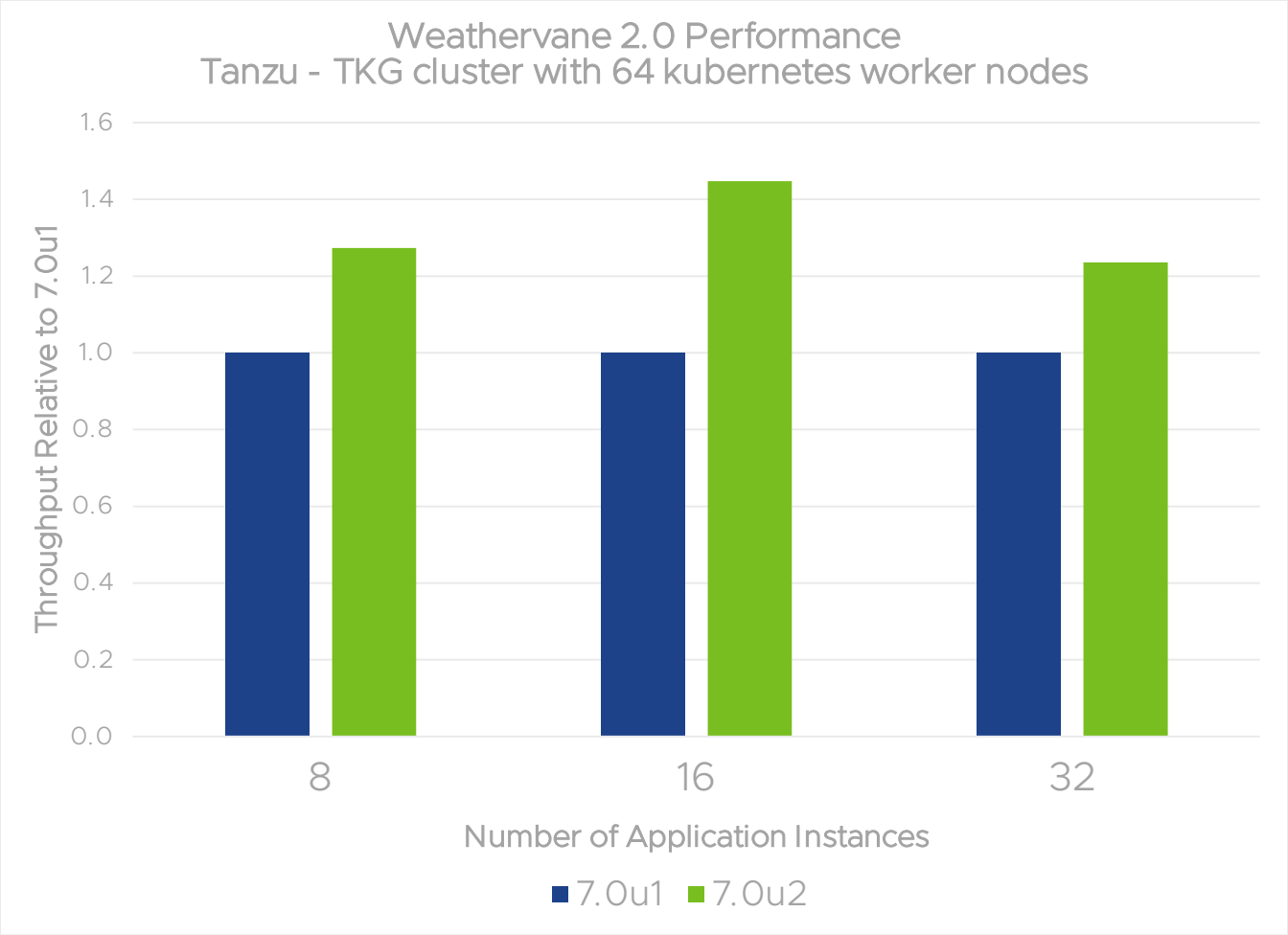

Figure 16. The vSphere 7.0 U2 CPU scheduler outperformed that in 7.0 U1 when scaling up application instances of the Weathervane application workload (with default settings).

Figure 17 below shows that with `LocalityWeightActionAffinity` disabled, we see up to 25% performance gains. The rest of the gains observed in 7.0 U2 (for example, 45%-25% = 20% additional gains in 16 application instances) are due to NUMA rebalancing policy changes not specifically tied to the AMD EPYC architecture. The `LocalityWeightActionAffinity` option had no significant impact on 7.0 U2 because there were some optimizations made around this option.

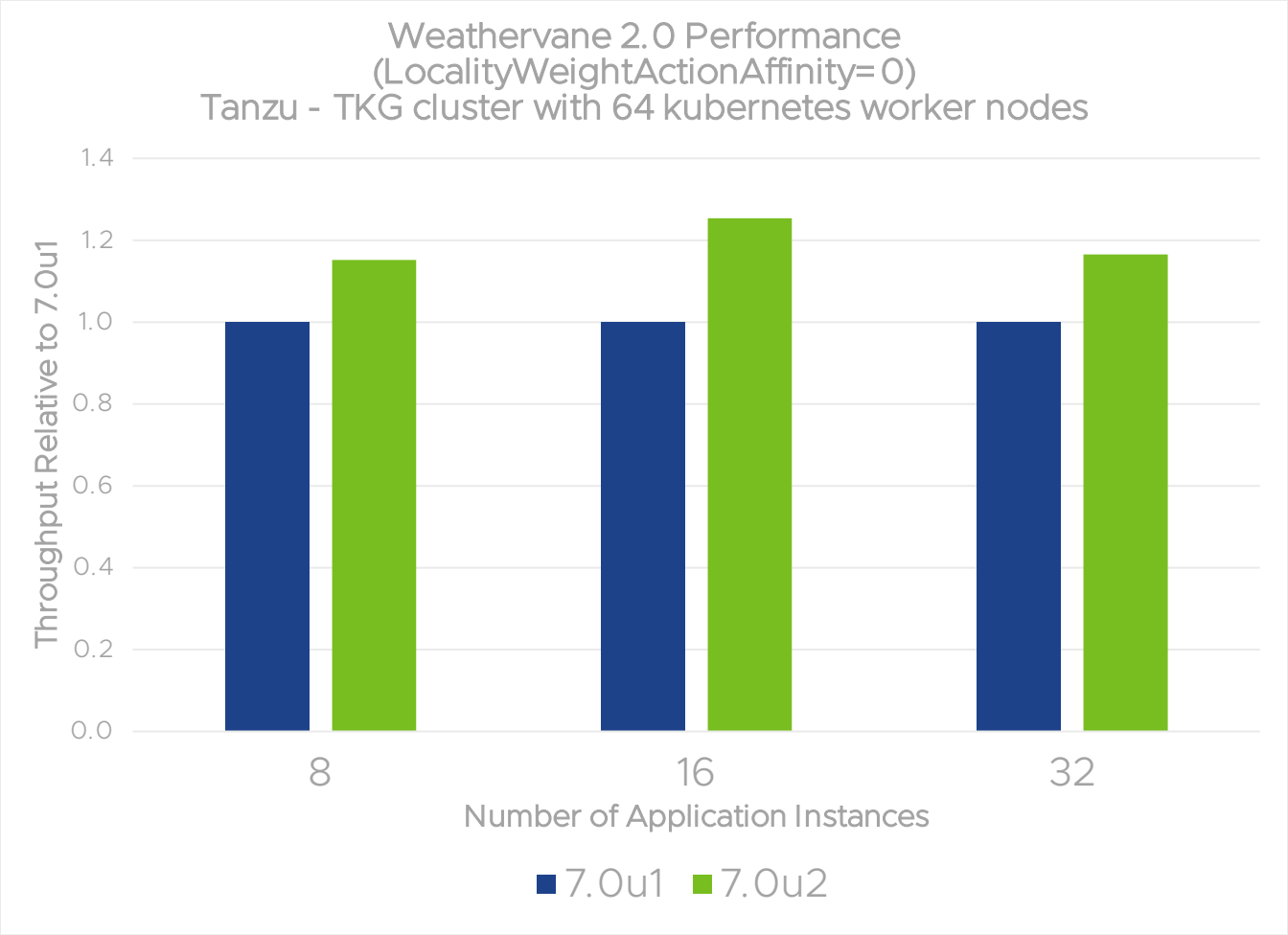

Figure 17. Weathervane 2.0 benchmark performance results for the vSphere CPU scheduler; with `LocalityWeightActionAffinity` disabled, we observed up to 25% better performance in 7.0 U2 relative to 7.0 U1.

The magnitude of the gains for different application instances are different because it depends on how the vCPUs are placed in 7.0 U1. The 4-vCPU VMs are nicely aligned and placed on a single LLC in 7.0 U2 in all application instance scenarios. For example, if the vCPUs are spanning more LLCs and more vCPUs are interfering with each other's LLCs in 7.0 U1, the more gain 7.0 U2 will see in performance because the vCPUs are well-aligned in their own LLCs. That is the reason the 16-application instance case sees the biggest gain of 25%, followed by 8 and 32 application instances.

Microbenchmark Performance

We evaluated many different microbenchmarks covering CPU, memory, storage, and networking. Examples include CoreMark, REDIS, SPECjbb, fio, and Netperf using single VMs with varying vCPU sizes and multiple VMs.

CPU performance using CoreMark

CoreMark® is a benchmark designed to test the CPU performance on embedded platforms. Coremark contains implementations like list processing, matrix manipulation, state machine, and cyclic redundancy check (CRC). This microbenchmark spends most of its cycles in the CPU core and core caches.

Table 1 below shows the gains achieved on Rome hardware on the CoreMark microbenchmark. We used a variety of configurations: 1X16 vCPU means a single 16-vCPU VM, 1x64 vCPU means a single large 64-vCPU VM that fits within a single socket, 8x16 vCPU means eight 16-vCPU VMs, and finally sixteen 16-vCPU VMs, for a total of 256 vCPUs on 256 PCPUs. Most of the gain is due to better load balancing that eliminates the hyper-threading contention. The gains were up to 30%.

VM configuration	Performance delta between 7.0u1 vs 7.0u2
1X16 vCPU	No change
1x64 vCPU	+30%
8x16 vCPU	+21%
16x16 vCPU	+3%

Table 1. CoreMark with an AMD EPYC Rome host showed a 30% performance gain for a single 16-vCPU VM. Other VM configurations showed some improvement or no change.

We also ran CoreMark on AMD EPYC Milan hardware by using a single VM with varying vCPU sizes, all the way up to the maximum supported vCPU size of 256 vCPUs for a single VM. The gains here are also due to a better load balancer in 7.0 U2 vs 7.0 U1 that reduces CPU contention—specifically hyper-thread contention when the CPU scheduler schedules vCPUs on two hyper-threads of a core when free full cores are available. We observed similar gains of up to 31% in this case as well.

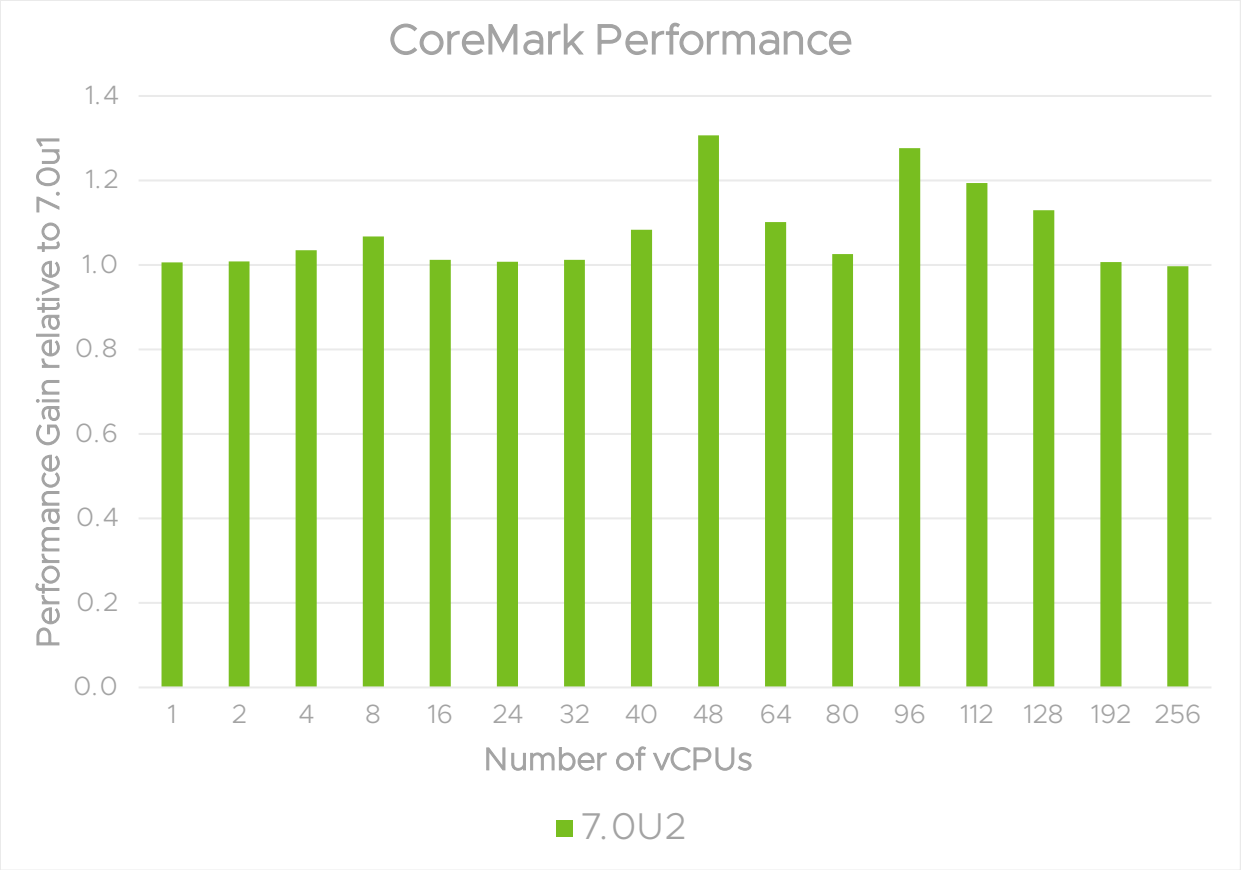

Figure 18. We ran the CoreMark workload in a single VM while we scaled up the number of vCPUs. There were performance gains in several of the vCPU configurations.

In-memory database performance with Redis

Redis is a popular open source, in-memory key-value store. Redis can be used as a database, cache, or message broker. We used Redis as a cache in these experiments. We configured VM with 4-vCPUs and 4GB to host the Redis Server and another 4-vCPU 4GB VM to host the memtier benchmark as a client on the same host. We used a data size of 4KB. We ran three configurations with all GET operations, all SET operations, and 50% GET and 50% SET operations. We gained up to 25% performance in a single VM case and up to 12% in multiple VM tests as shown in Table 2. The gains are due to better vCPU placements aligning with the LLC such that all 4 vCPUs from the same VM reside on a single LLC.

VM configuration	Performance delta between 7.0u2 vs 7.0u1
1x4 vCPU client and server	Up to +25%
32 client-server pairs	Up to +12%

Table 2. Redis performance results show up to 25% gain for the Redis client and server installed on one 4-vCPU VM, and up to 12% gain for 32 client-server pairs.

SPECjbb – Java server workload

We ran the SPECjbb® 2015 Java server workload in a single 16-vCPU VM (with varying vCPU configurations), but the performance of this configuration regressed by about 7% because this workload is memory-intensive and benefits from more cache bandwidth rather than locality. For multiple instances of 16 vCPU SPECjbb workload (8 and 16), performance was about the same. Performance test results were similar on both AMD EPYC Rome and Milan processors.[4]

Storage and networking microbenchmarks

We used fio to test the VMFS I/O performance for the two versions of CPU schedulers. We did a variety of single and multiple VM fio tests using different storage adapters and different underlying hardware: SSDs, SATA, and NVMEs. We observed up to 50% improvement in CPU efficiency (cycles/IO) and up to 9% throughput gain in many configurations; we did not observe any regressions.

In networking tests using Netperf, we observed performance improvements of up to 50% for 7.0 U2. However, some single-VM tests regressed where the VM size could not fit the LLC. In such scenarios, the networking worlds (TX/RX) were not able to fit on the same LLC as the vCPUs (for better LLC hit rates) because fewer LLCs were packed with vCPUs. This is expected because the vSphere 7.0 U2 CPU scheduler optimizes the locality of vCPUs. In 7.0 U1, since vCPUs were more or less spread across the entire socket, networking worlds had an opportunity to be placed with the vCPUs.

Best Practices

Based on our extensive testing using the vSphere 7.0 U2 CPU Scheduler, we recommend the following best practices:

- Use out-of-the-box BIOS configurations (NPS-1 and CCX-as-NUMA disabled) and vSphere settings on AMD EPYC processors to get the best performance on vSphere 7.0 U2. For example, VMmark performance is up to 12% better (without any QoS failures) when out-of-the-box options are used—that is, the default setting for NPS-1 is used and CCX-as-NUMA is disabled.

[4] Our results are not comparable to SPEC's trademarked metrics for this benchmark.

- You may test and use the NPS-4 configuration if your application satisfies the below conditions:
  - It requires very high bandwidth (for example, high performance computing applications).
  - It requires the lowest possible memory latency.
  - It is highly NUMA optimized.
  - You can afford to pin/affinitize the VMs.

We believe that for most of the applications, using out-of-the-box BIOS configurations and vSphere settings will give optimal performance on vSphere 7.0 U2.

Conclusion

This paper describes the vSphere CPU scheduler performance optimizations specific to the AMD EPYC architecture. These changes, which are extensions to the existing vSphere resource management stack, leverage the LLC locality of AMD EPYC processors to provide near optimal performance for most types of benchmarks and applications. An extensive evaluation shows that the vSphere 7.0 U2 CPU scheduler can achieve **up to 50% better performance** on AMD EPYC processors with out-of-the-box BIOS options and vSphere settings.

References

- AMD EPYC Rome Application Performance on vSphere Series: Part 1 – SQL Server 2019
- AMD 2nd Gen EPYC (Rome) Application Performance on vSphere Series: Part 2 – VMmark
- AMD 2nd Gen EPYC (Rome) Application Performance on vSphere Series: Part 3 – View Planner Virtual Desktops
- AMD 2nd Gen EPYC (Rome) Application Performance on vSphere Series: Part 4 – STREAM and Java EE

About the Authors

Qasim Ali is the lead author of this paper. Other authors include **Todd Muirhead**, **Jim Hsu**, **James Zubb** and **Benjamin Hoflich**.

Acknowledgments

We thank AMD for their support and collaboration; we thank Xunjia Lu and Yifan Hao from the vSphere resource management team for implementing the CPU scheduler optimizations and helping review this paper. We also thank Peter Jonasson's vSphere performance automation team for conducting thousands of performance automation tests to quantify the vSphere 7.0 U2 CPU scheduler.