vMotion Innovations in VMware vSphere 7.0 U1

Performance Study - September 24, 2020

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® vMotion® incorporates several cutting-edge enhancements in vSphere 7.0 U1 that provide dramatic improvements in performance when migrating virtual machines (VMs). These include rearchitecting memory pre-copy with an innovative page-tracing mechanism, among other things, that greatly reduce the performance impact on guest workloads during live migration and significantly reduce vMotion duration. Performance data from Tier-1 workloads show these optimizations can scale to hundreds of vCPUs and terabytes of memory.

Introduction

vSphere vMotion enables the live migration of virtual machines from one vSphere host to another, with no perceivable impact to the end user. vMotion is a key enabler of several VMware technologies, including VMware vSphere® Distributed Resource Scheduler™ (DRS) and VMware vSphere® Distributed Power Management™ (DPM). vMotion brings invaluable benefits to administrators: it helps prevent server downtime, enables troubleshooting, and provides flexibility.

Although vMotion has been used successfully since the earliest versions of VMware ESX™, certain workloads running in large VMs—configured with more than 64 vCPUs and 512GB memory—experienced undesirable performance loss during vMotion. The new performance enhancements in vSphere 7.0 U1 let you live-migrate even monster-sized VMs running heavy-duty, enterprise-class applications, with minimal impact on performance and availability. Moreover, these enhancements result in significant performance gains even with typical-sized VMs.

This paper presents a brief description of the vMotion technology, optimizations in vSphere 7.0 U1, and performance data from a wide variety of Tier-1 application workloads and VM deployments.

Architecture

vSphere vMotion transfers the entire execution state of a running virtual machine from the source VMware ESXi™ host to the destination ESXi host over a high-speed network. The execution state primarily consists of the following components:

1. The virtual machine's virtual disks
2. The virtual machine's physical memory
3. The virtual device state, including the state of the CPU, network and disk adapters, SVGA, and so on
4. External network connections

Information about how this execution state is transferred can be found in the *VMware vSphere vMotion Architecture, Performance, and Best Practices in VMware vSphere 5* technical paper [1].

The physical memory of the virtual machine is by far the largest component of the VM's execution state that needs to be transferred during a migration. In this paper, we cover the new, innovative page-tracing mechanism, added in vSphere 7.0 U1, to improve guest performance during vMotion memory copy.

We also discuss several enhancements added to vSphere 7.0 U1 to improve the guest performance during the vMotion switch-over phase. Finally, we discuss some of the vSphere 7.0 U1 optimizations that shorten the vMotion duration.

Migration of Virtual Machine's Memory

As in vSphere 6.7, vMotion in vSphere 7.0 U1 uses essentially the same pre-copy iterative approach to transfer the memory contents. The approach is as follows:

- **[Phase 1] Guest trace phase**
  The guest memory is staged for migration during this phase. Traces are placed on the guest memory pages to track any modifications by the guest during the migration.
- **[Phase 2] Pre-copy phase**
  Because the VM continues to run and actively modify its memory state on the source host during this phase, the memory contents of the VM are copied from the source ESXi host to the destination ESXi host in an iterative process. Each iteration copies only the memory pages that were modified during the previous iteration.
- **[Phase 3] Switch-over phase**
  During this final phase, the virtual machine is momentarily quiesced on the source ESXi host, the last set of memory changes are copied to the target ESXi host, and the virtual machine is resumed on the target ESXi host.

In the next two sections, we look at how page tracing compares in vSphere 6.7 and vSphere 7.0 U1.

vSphere 6.7 Page Tracing

Page tracing is done by the Virtual Machine Monitor (VMM), which is responsible for managing the VM memory page tables. As shown in the left half of **figure 1**, the VMM installs traces by write-protecting memory pages. It sets the read-only bit of the page table entries (PTE), flushes each vCPU's translation lookaside buffer (TLB) to avoid TLB hits, and ensures the modified page table entries are honored.

As a result, when the guest attempts to write to a memory page protected by a trace, a page fault occurs. The VMM handles the page fault and notifies vMotion of the page content change (referred to as a ***trace fire***). vMotion then adds that page to the list of pages that need to be retransmitted.

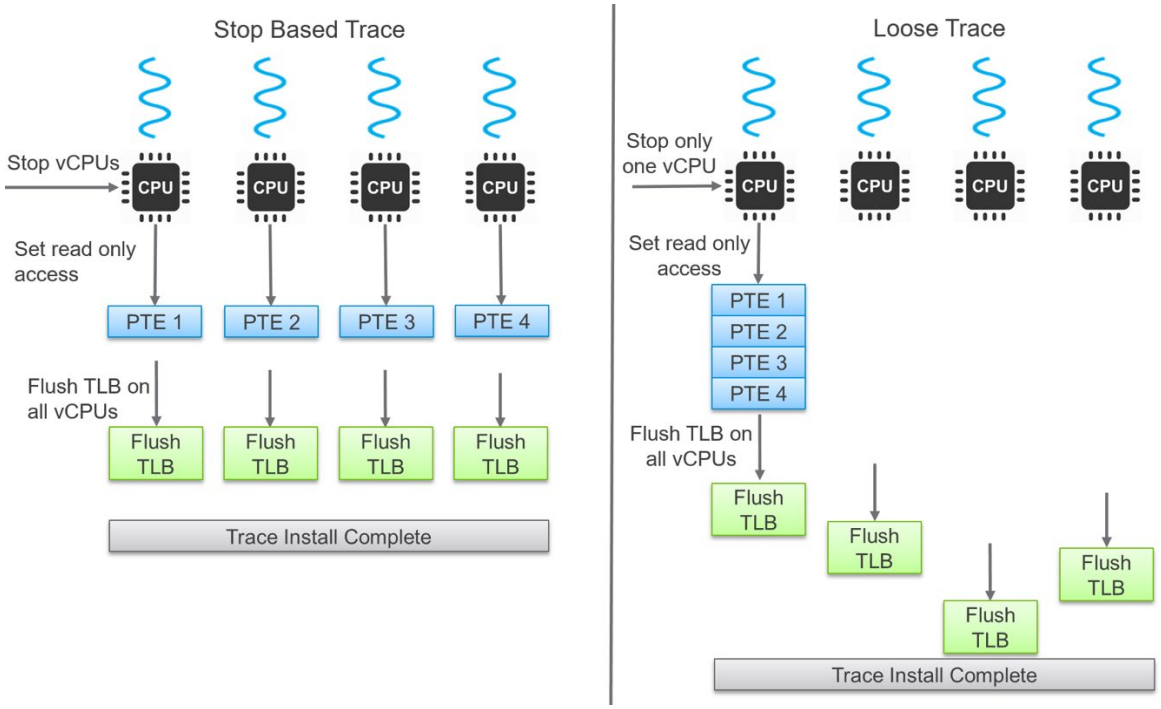

Figure 1. vSphere 6.7 stop-based tracing and vSphere 7.0 U1 loose-based tracing

The vSphere 6.7 approach for installing traces is to stop all vCPUs from executing the guest instructions and then distribute the work of installing the traces to all the vCPUs assigned to the VM (referred to as ***stop-based trace***). The virtual machine's physical memory can be as large as 6TB on vSphere 6.7, so depending on the size of the VM's physical memory and configured vCPUs, the trace-install duration can range from less than ten seconds to hundreds of seconds.

To allow a VM to continue to run during the process of trace install and to achieve the desired amount of transparency, the vCPUs alternate between guest execution and trace install. The trace install is done over multiple short trace-install phases, with guest execution interleaved between these trace-install phases. This gives the VM an illusion of continuous execution; although, in reality during each trace-install phase, all the vCPUs are stopped from executing the guest instructions and instead help vMotion trace pages.

This approach worked fine for small-to-medium-sized VMs, because stopping a few vCPUs is not too expensive, usually on the order of a few hundred microseconds. However, the cost of stopping all vCPUs is not constant. The more vCPUs the VM is configured with, the longer it takes to stop and resume them, thereby resulting in a longer disruption to the guest workloads.

vSphere 7.0 U1 Page Tracing

Loose Page-Trace Install

As previously described in the "vSphere 6.7 Page Tracing" section, the disruption to the guest workloads comes from the ***synchronous*** nature of the stop-based method. During each of the trace install phases, all the vCPUs are stopped to do the trace install work and all the vCPUs are required to flush TLBs before getting back to executing guest instructions.

With vSphere 7.0 U1, we introduce the ***loose page-trace Install*** with several optimizations to minimize the disruption to the guest workloads.

- Instead of using all vCPUs, we now have only one vCPU to perform all the tracing work. All the other vCPUs continue to execute guest instructions in parallel during the trace install process. This was achieved by carefully handling races between trace installation and trace firing. The VMM maintains the invariant that at any given time, a page is either traced or selected for transmission.
- To further minimize the workload disruption, the new model implements a ***loose*** synchronization policy that enables lazy TLB flushes. Each of the vCPUs can perform the TLB flush on their own schedule at the first opportunity (for example, during a context switch). Thus, the vCPUs are not required to immediately stop execution and flush their TLBs. The last vCPU to flush its TLB informs vMotion of trace install completion (shown in right half of **figure 1**). Overall, with ***loose page-trace install***, we greatly reduce the impact of trace on guest workload performance.

Large Trace Install

As shown in **figure 2**, in vSphere 6.7 and prior releases, traces (read-only bit) were set at a 4KB page granularity. PTEs at L2 and L3 effectively map to memory regions of larger granularity; that is, 2MB and 1GB, respectively. A 2MB large memory region encompasses 512 4KB pages. A 1GB large memory region encompasses 512^2 = 262,144 4KB pages. With vSphere 7.0 U1, the VMM will set the traces at a much larger granularity on a 1GB memory region. Instead of installing one write trace per 4KB page, VMM installs a single trace for every 262,144 4KB pages, which is more efficient and dramatically reduces the amount of trace installation time.

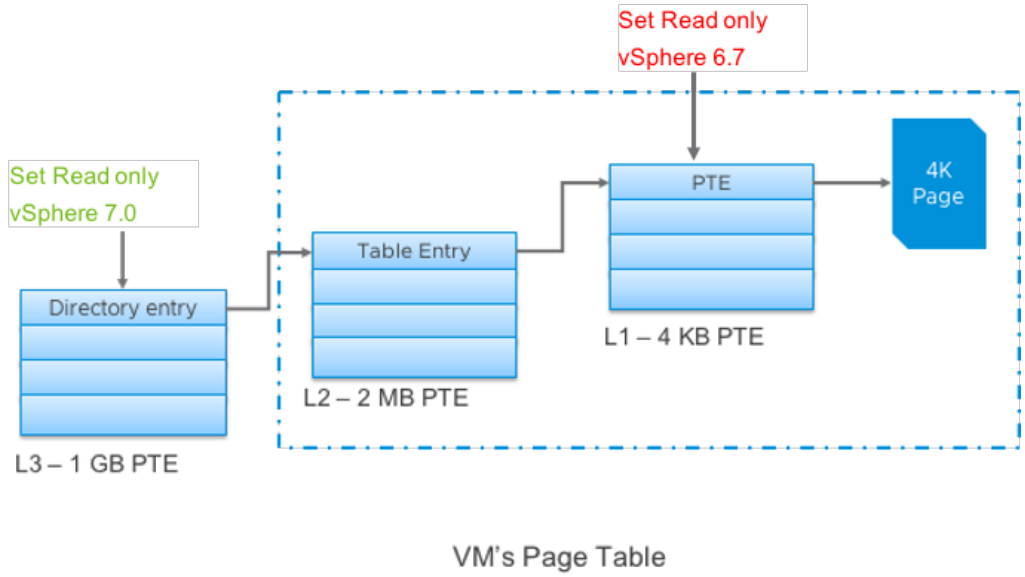

Figure 2. vSphere 6.7 4KB trace and vSphere 7.0 U1 1GB trace

To avoid data-transfer amplification, notification about the content change to vMotion is still done at the 4KB level. On a trace fire, VMM breaks the large trace (1GB memory region) to 512 L2 traces (2MB memory regions), one of which includes the 4KB page that resulted in a trace fire. That L2 trace (2MB memory region) is further broken into 512 L1 traces (4KB pages). vMotion only gets notified of the content change for that single 4KB page. Therefore, vMotion only adds that single 4KB dirty page to its list of pages that need to be retransmitted.

While we do more work on the trace-fire path, the extra cost is negligible compared to the savings from the trace-install path for several reasons.

1. The extra cost on the trace-fire path only occurs when the guest modifies a memory page and the VM's working set size is typically much smaller than the VM memory size.
2. With less trace-install duration, the guest has even less opportunity to modify memory pages.
3. The cost of trace install impacts all vCPUs, since every time we install trace, we request all the vCPUs to flush their TLBs. In contrast, the cost of trace fire impacts only one vCPU.

Overall, with this approach, we not only greatly reduce the amount of tracing work required for vMotion, we also reduce guest interruptions due to less tracing. In addition, reducing the trace duration helps cut down the overall migration time.

vMotion Switch-Over Optimizations

Memory convergence is reached when vMotion estimates the number of dirty pages left can be sent within the goal of 0.5 seconds based on the observed network bandwidth usage during precopy. vMotion then momentarily stops guest execution to switch over the VM from the source to the destination host.

vMotion switchover time is defined as the time taken to suspend the VM execution on the source host, take a checkpoint of its virtual device state, transmit both the checkpoint and the remaining dirty pages to the destination ESX host, restore from the checkpoint state, and start executing guest instructions on destination VM. The goal of vMotion is to keep the execution switchover time under one second.

As shown in **figure 3** (next page), changed bitmap and swap bitmap are a function of memory size. The changed bitmap identifies the last remaining dirty pages that need to be transmitted to the destination, and the swap bitmap identifies the pages that are swapped out. Historically, vMotion transferred the entire bitmaps. This worked fine for small VMs, whose bitmaps are typically a few KBs (1 bit per page). But for large VMs, the bitmap size can be several MBs. Transferring such large bitmaps takes a few seconds and breaks the vMotion switch-over goal of 1 second. Following, we describe why vMotion needs to send these bitmaps, as well as the optimizations added in vSphere 7.0 U1 to handle these large bitmaps.

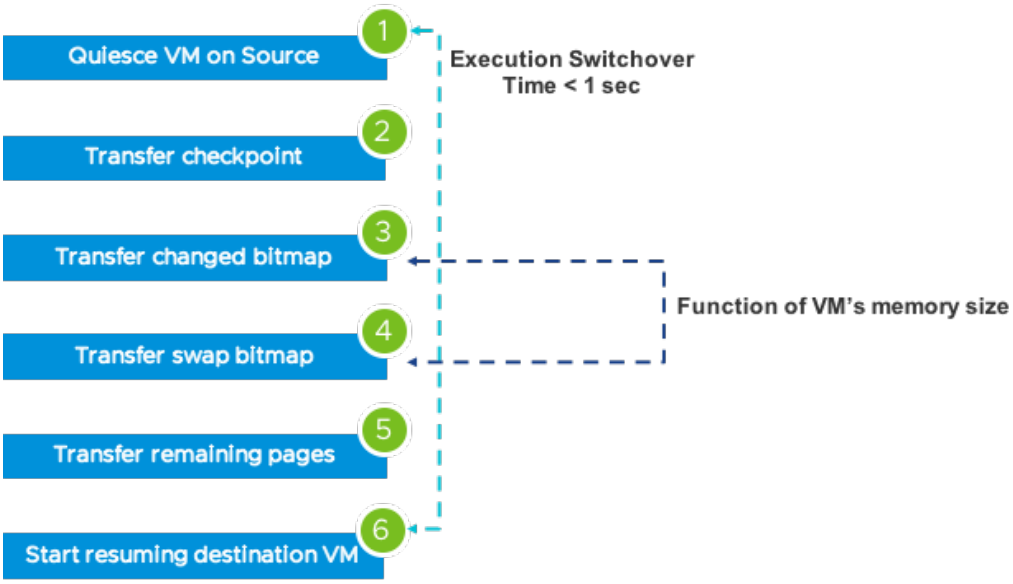

Figure 3. vSphere switch-over process

Bitmap Optimizations

To reduce the vMotion switch-over time and achieve parallelism, vMotion starts the checkpoint restore on the destination host, while some dirty pages remain to be transmitted from the source. During the VM power on, vMotion might need those pages. vMotion has a mechanism in place to mark these pages as ***remote*** on the destination, indicating those pages' contents are still on the source. This is done by transferring metadata that includes changed memory bitmap and swap bitmap during the switch-over.

Because vMotion is designed to have very few dirty pages left by the end of the precopy, we observed that the bitmap is mostly sparse with very few bits set. Therefore, instead of sending the entire bitmap, vSphere 7.0 U1 has optimizations to transfer the compacted "bitmap" with only page numbers of the last remaining pages, which drastically reduces the transfer time (see **figure 4**, next page). Similar optimizations are added to transfer the swap bitmap.

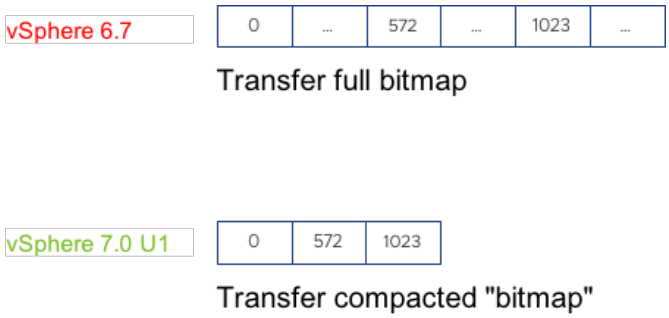

Figure 4. Bitmap transfers in vSphere 6.7 and vSphere 7.0 U1

Large Page Handling Optimizations

Having a VM's memory backed by large pages is crucial for performance, since it improves the TLB hit ratio. Therefore, every time the guest touches a memory page that is not backed by a large page, the hypervisor will try to allocate a large page and remap the existing small pages into this new, large page. However, remapping these pages can lead to longer pauses in the guest workload execution.

We identified two areas where vMotion handling of certain special pages, including ***remote*** pages and ***zero*** pages, was not optimal, as described in the next sections.

Remote Pages

For ***remote*** pages, vMotion was breaking the existing large page backings and allocating new 4KB pages on the destination to receive the latest data from the source, and later reconstructing a new, large page. We added optimizations to vSphere 7 U1 to copy the latest content of the remote pages to their original backing to avoid the cost of large page reconstruction.

Zero Pages

To reduce the memory footprint, vMotion uses page sharing for all the ***zero*** pages (pages with zero content, or uninitialized parts of large pages) for the destination VM. However, since all page sharing is at the 4K page size level, this was being done at the cost of breaking a large page. Whenever vMotion found a zero page backed by a large page, it was breaking the large page mapping to enable page sharing. We added a new zero-page policy to vSphere 7.0 U1 that minimizes breaking large pages and strikes a good balance between memory consumption and performance. With the new policy, instead of sharing all the zero content, we preserve the sharing for zero pages that used to be shared on the source host, so the memory consumption for the VM does not inflate as a result of vMotion.

With these new optimizations, we avoid the huge cost of large page reconstruction for the destination VM, which greatly improves the guest workload performance after vMotion.

Improved Heuristics to Minimize Zeroing Cost

By default, vMotion allocates large pages on the destination. In vSphere 6.7, every large page allocation on the destination is zeroed. In vSphere 7.0 U1, vMotion uses improved heuristics to conditionally skip zeroing on the destination memory allocation path if the 2MB range on the source is backed large. This not only helps the CPU cost, but also brings significant reductions in the migration duration.

Performance Test Configuration and Methodology

This section describes the testbed configuration and general testing methodology.

Test Configuration

vMotion tests were run on a pair of Dell R930 servers, running ESXi 7.0 U1. Each was a quad socket machine configured with 24-core 2.20 GHz Intel Xeon E7-8890 processors and 4TB of memory. Both the hosts shared a 10TB VMFS volume on a Dell EMC Unity 600 all-flash array. Each of the hosts were configured with four Mellanox 40GbE network adapters. The vMotion network was configured to use 4 vmkernel NICs with each vmkernel NIC using a unique 40GbE network adapter. The same testbed running ESXi 6.7 U3 was used as the baseline.

Here are the full specifications of the testbed:

Hardware

- **Server:** Two Dell R930 (24-core 2.20 GHz Intel Xeon E7-8890 @ 2.2 GHz, 4 TB memory)
- **Storage:** Dell EMC Unity 600 all-flash array, 10 TB VMFS volume
- **Networking:** 4 Mellanox 40GbE NICs

Software

- **VM configuration:** vCPUs varied from 12 to 192, memory varied from 12GB to 4TB, 4 vmdks (100GB system disk, 500GB database disk, 700GB backup disk, 150GB log disk)
- **Guest OS/application**: RHEL 7.6 / Oracle DB 12.2
- **Benchmark:** HammerDB using a database derived from TPC-C with a size of 3500 warehouses[1]

Measuring vMotion Performance

We used the following metrics to understand the performance implications of vMotion.

- **Migration time:** The total time taken for the migration to complete, beginning from the initiation of the migration.
- **Switch-over time:** The time during which the VM is quiesced to enable its switchover from the source host to the destination host.
- **Guest penalty:** The performance impact (latency and throughput) on the applications running inside the virtual machine during and after the migration.

vMotion Performance in a Database Environment

Database workloads are widely acknowledged to be extremely resource intensive. They are characterized by high consumption of CPU, memory, and storage resources. So, they serve as an ultimate test of vMotion performance.

This study investigates the impact of vMotion on the Oracle DB server online transaction processing (OLTP) performance.

Load-Generation Software and Testbed Configuration

We used the open-source HammerDB [2] as the client load generator with the TPC-C [3] workload profile. The performance metric we use is transactions per minute (TPM).

To get finer granular information, we also ran a script that runs in parallel with the HammerDB workload generator. The script probes the Oracle DB server every second and gets how many Oracle DB commits were performed during that one-second interval.

[1] Our test results cannot be compared to published TPC-C benchmark results because we used only the database and not the benchmark. Instead, we used HammerDB as specified.

This way, we have very granular info along with the throughput info reported by HammerDB.

Our first set of tests modeled a single virtual machine with two deployment scenarios:

Test 1: Typical-Sized VM Deployment

- VM with 12 vCPUs and 64GB of memory
- RHEL 7.6 / Oracle database server 12.2
- HammerDB workload used a TPC-C database with 3500 warehouses
- Benchmark load of 10 users (think-time: 500ms)

Test 2: Large VM Deployment

- VM with 72 vCPUs and 1TB of memory
- Identical workload as the previous scenario

We chose the **typical-sized VM deployment** based on a survey of nearly one thousand customers across all industry segments. Their VMs were typically sized with less than 12 vCPUs and 64GB of memory. The survey also revealed many deployments used OLTP applications.

VMotion Impact on Oracle DB Throughput with a Typical-Sized VM: 12 vCPUs/64GB Memory

In the typical-sized VM test scenario, a load of 10 Hammer DB users generated a substantial load on the virtual machine in terms of CPU, memory, and disk usage. vMotion was initiated during the steady-state period of the benchmark, when the CPU utilization (esxtop %USED counter) of the virtual machine was close to 650%.

Figure 5 (next page) plots the performance of the Oracle DB typical-sized VM—before, during, and after vMotion—when running vSphere 6.7. The figure shows two dips in performance. The first noticeable dip in performance is during the guest trace phase during which trace is installed on all the memory pages. The second dip is observed during the switchover phase when the virtual machine is momentarily quiesced on the source host and is resumed on the destination host. Despite these two dips, Oracle DB throughput never dropped to zero even when tracking the performance at a one-second granularity.

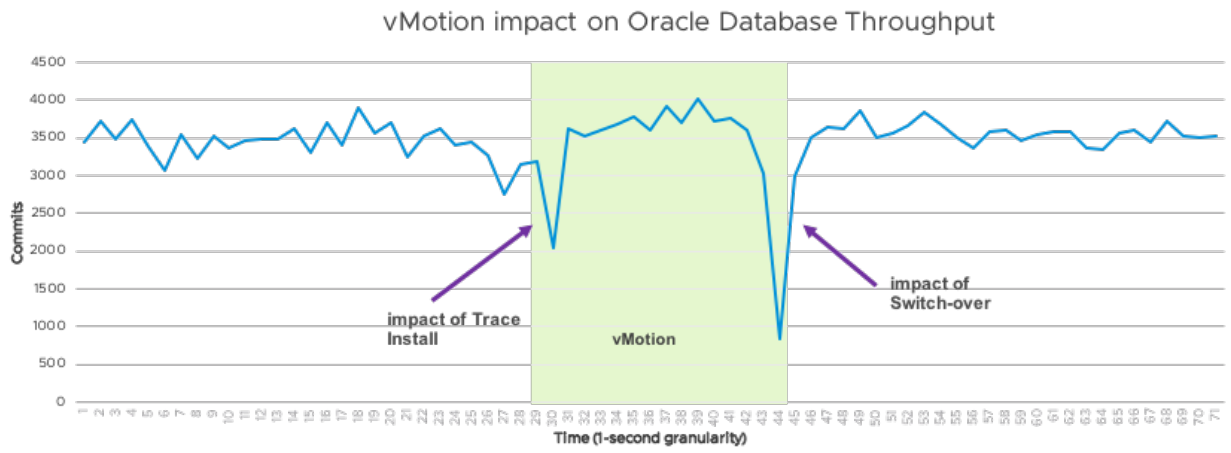

Figure 5. Oracle DB typical-sized VM performance with vMotion on vSphere 6.7 with 10 HammerDB users

vMotion Impact on Oracle DB Throughput with Large-Sized VM: 72 vCPUs/1TB Memory

Figure 6 (below) shows the impact of vMotion on Oracle DB server running inside a 72-vCPU/1TB VM with the same workload size (10 HammerDB benchmark users) used in the previous typical VM scenario. In contrast to the minimal impact seen in the typical VM scenario, we observe more disruption on workload performance in the large VM scenario.

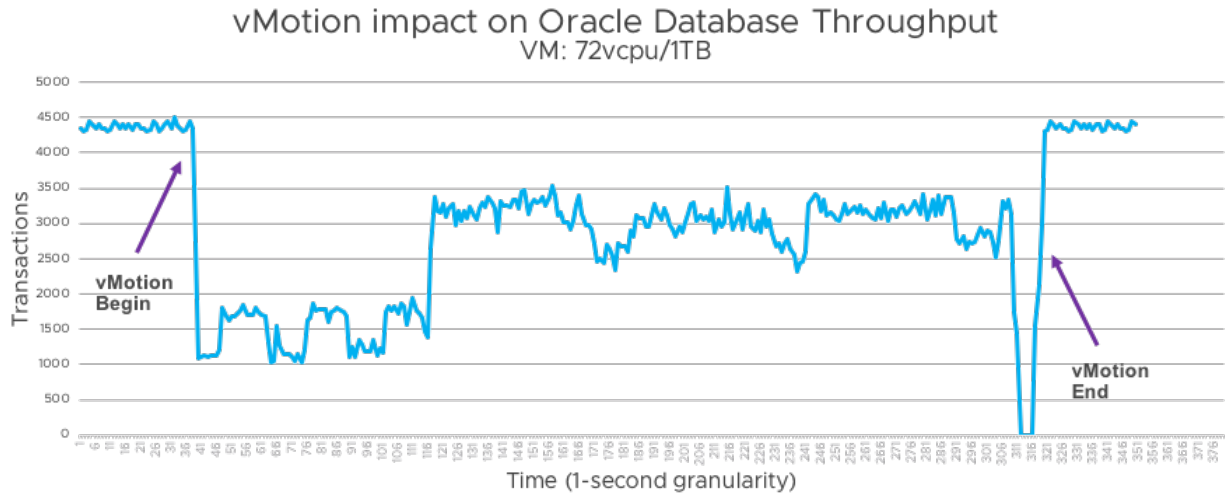

Figure 6. Oracle large-sized VM performance with vMotion on vSphere 6.7 with 10 HammerDB users

Note that the migration time during the large VM scenario is much longer than the typical VM scenario, even though the workload is identical. This is because, in the large virtual machine scenario, to simulate real-life customer deployment, we ran a microbenchmark that ensures the guest memory that is not allocated to Oracle DB had been touched and

modified before vMotion is initiated. In other words, this scenario is different from a freshly booted VM that is assigned zero pages (pages that contain nothing but zeroes). vSphere has optimizations to reduce the amount of data transferred for zero pages.

Figure 7 (below) shows a dissection of the Oracle DB throughput during a large VM vMotion scenario. The graph shows the impact on Oracle DB throughput during the initial trace phase and memory copy, and it shows the dip as the VM is suspended for switchover.

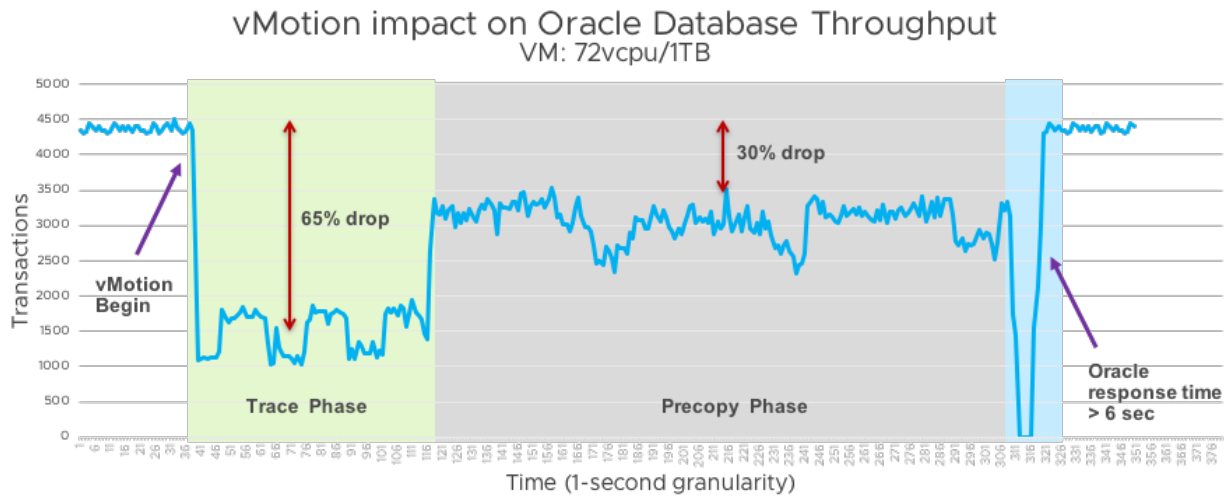

Figure 7. Various phases of Oracle DB large-sized virtual machine vMotion on vSphere 6.7

As shown in **figure 7** (above), the performance impact during the trace phase for the large VM was about 65 percent. As explained in the "Architecture" section, vCPUs alternate between guest execution and trace installation every few milliseconds. The outcome: there is a long trace phase during which the guest is operating in a degraded mode. While this approach worked for a typical-sized VM, it resulted in a tremendous amount of performance loss for large VM configurations, since the more vCPUs the virtual machine is configured with, the longer it takes to stop and resume all of them.

Tracing memory pages is not a one-time operation. Whenever vMotion transmits a dirty page, it needs to reinstall a trace on that page to track new guest modifications. Those new traces install during the memory copy phase, and trace fires that occur when the guest modifies the pages also have moderate impact on workload throughput. In our test scenario, we observed about a 30 percent drop.

Another challenge is the switchover phase. This is the phase where we transfer the last remaining dirty pages. This is the most critical phase during vMotion, because the guest is not executing any instructions during this phase. As seen in **figure 7** (above), it took Oracle DB six seconds to respond during this phase.

Figure 8 (below) compares the impact of vMotion on an Oracle DB server running inside a 72-vCPU/1TB VM on both vSphere 6.7 and vSphere 7.0 U1. Thanks to all the new performance optimizations, we observed great improvement in guest performance during the 7.0 U1 vMotion. First, by optimizing the memory allocation path on the destination, the duration of migration was cut by half. By completely reinventing hypervisor page tracing mechanisms (refer to the "vSphere 7.0 U1 Page Tracing" section), the average throughput loss of 50 percent during 6.7 vMotion was brought to less than 5 percent during 7.0 U1 vMotion. Finally, all the enhancements added to vMotion (refer to the "vMotion Switch-Over Optimizations" section) improved Oracle DB resume time during the switchover phase by an order of magnitude.

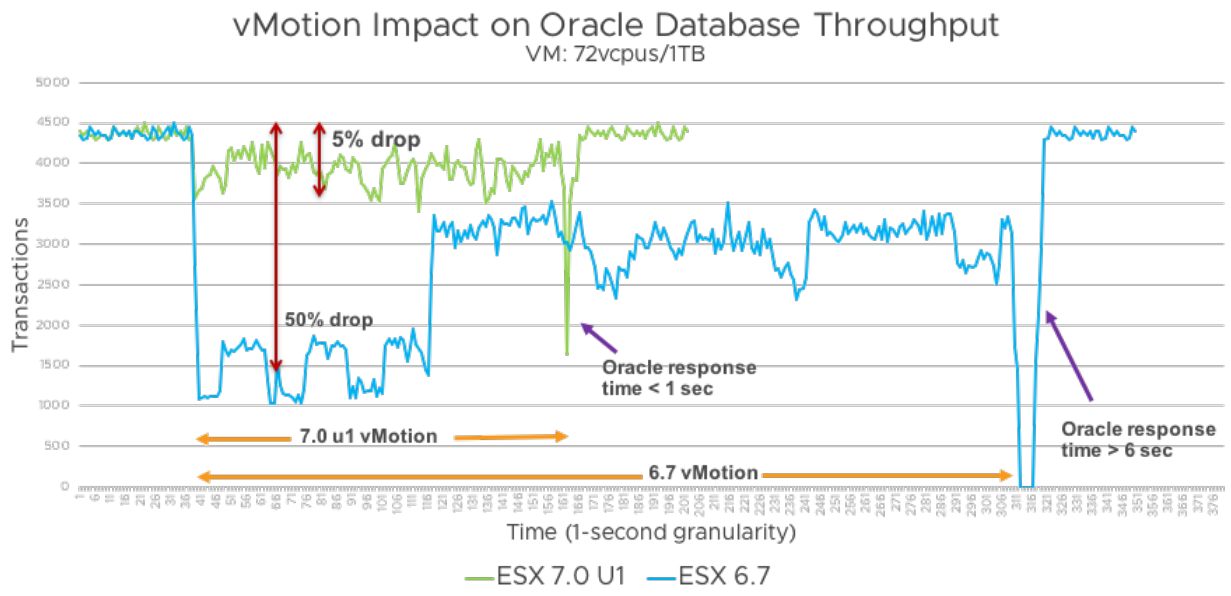

Figure 8. Oracle Large Virtual Machine Performance with vMotion on vSphere 6.7 Vs vSphere 7.0 U1

Throughput Performance of Various-Sized Oracle DB VMs

Table 1 (next page) shows another set of tests that modeled a single VM with different deployment scenarios and workload sizes. The 192 vCPUs/4TB was the largest VM configuration supported on our testbed. Note that in these scenarios, to simulate real-life customer deployments, we ran a microbenchmark that ensured all the guest memory that was not allocated to the Oracle DB had been touched and modified prior to vMotion.

Scenarios	VCPUs	Memory (GB)	Oracle SGA (GB)	HDB throughput (prior to vMotion)
Typical-Sized VM	12	64	25	270,000 TPM
Medium VM	36	128	50	350,000 TPM
Large VM	72	512	100	540,000 TPM
Very Large VM	144	1024	200	1.2 million TPM
Monster VM	192	4096	300	1.5 million TPM

Table 1. We tested Oracle DB throughput in single VMs of various sizes

Figure 9 (below) shows improvement with vSphere 7.0 U1 across the board in all the test configurations. In general, the larger the VM configuration (vCPUs, memory), the higher the performance differential between vSphere 7.0 U1 and vSphere 6.7. For monster-sized VMs, we observed as much as a 19x improvement in guest workload performance with vSphere 7.0 U1 over vSphere 6.7.

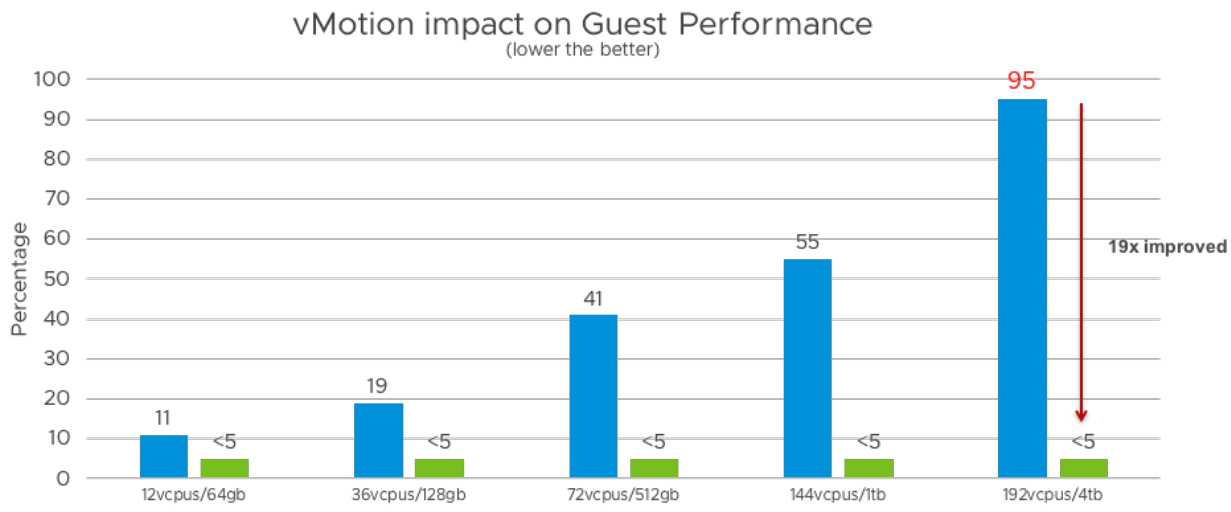

Figure 9. Oracle large VM performance with vMotion on vSphere 6.7 versus vSphere 7.0 U1

Typical-Sized VM with Half-Second Granularity: vSphere 6.7 vs vSphere 7.0 U1

Finally, we revisit the vMotion impact on a typical VM deployment used by most customers but looking at Oracle DB throughput at half-a-second granularity. The workload used 10 users with a think-time of 500 milliseconds.

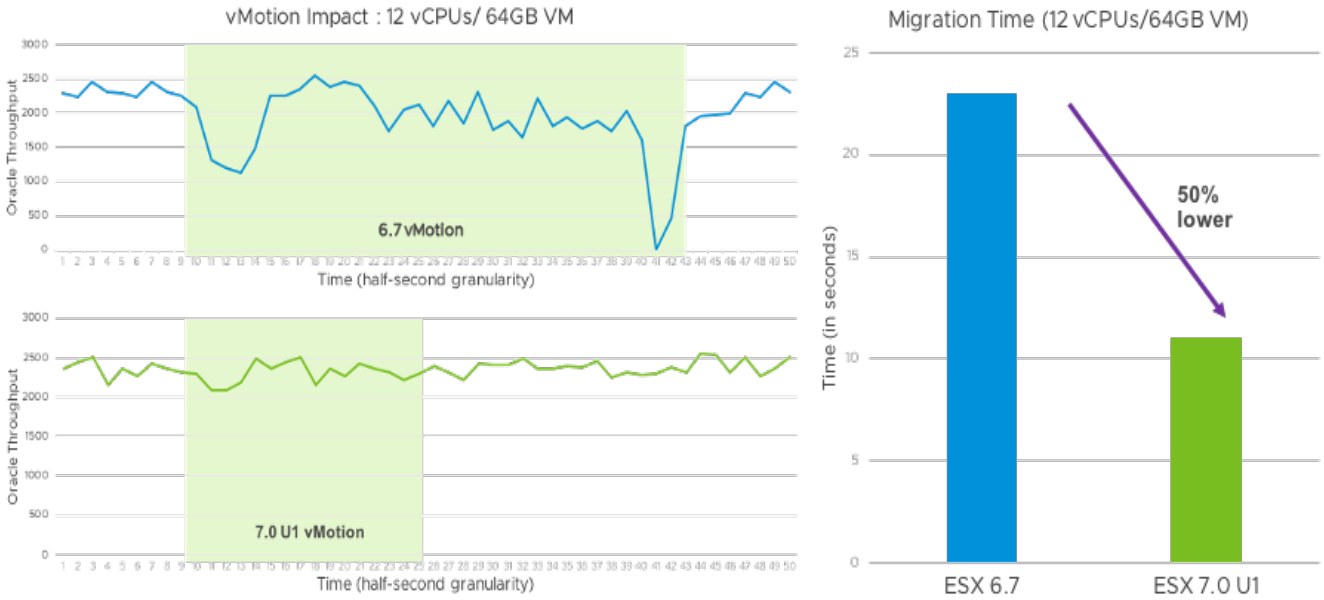

Figure 10. Oracle typical-sized VM performance with vMotion on ESXi 6.7 vs. ESXi 7.0 U1 with half-second granularity

Figure 10 shows the optimizations added to vSphere 7.0 U1 help not only large VM configurations, but also result in substantial improvements even for typical-sized VM deployments. Thanks to all the optimizations, we observed nearly 45 percent improvement in Oracle DB throughput during 7.0 U1 vMotion (average 3480 transactions per second) compared to 6.7 vMotion (average 2403 transactions per second). Notably, this improvement comes with no cost to migration time. In fact, migration time was reduced by 50 percent in 7.0 U1 vMotion, thanks to both a reduction in trace installation time and optimizations to the memory allocation path on destination.

In summary, the improvements in vSphere 7.0 U1 over vSphere 6.7 are twofold: the duration of vMotion and the impact on guest performance during vMotion.

vMotion Performance in a Hyper-Converged Infrastructure Environment

Hyper-converged infrastructures and cloud services are fast-growing environments today. They have similar characteristics, such as multiple VMs running at different load demand levels, and large fluctuations in the load level of each VM. This study investigates the impact of vMotion in an HCI environment.

Load-Generation Software and Testbed

TPC Express Benchmark HCI (TPCx-HCI) [4] measures the performance of HCI clusters under a demanding database workload. It stresses the virtualized hardware and software of converged storage, networking, and compute resources of the HCI platform.[2]

The unique characteristic of TPCx-HCI is that it's an elastic workload that varies the load delivered to each of the VMs by as much as 16x, while maintaining a constant load at the cluster level. Sustaining optimal throughput for this elastic workload on a multi-node HCI cluster would typically benefit from frequent migrations of VMs to rebalance the load across nodes. This property measures the efficiency of VM migration as well as the uniformity of access to data from all the nodes.

We configured the testbed with the following hardware and software.

Hardware

- 4 Dell PowerEdge R740xd servers
  - 2 sockets, each with a 28-core 2.5 GHz Platinum 8180 CPU
  - 768GB memory
  - 12 NVMe drives
  - 2-port Intel 40Gb 2P XL710 Adapter; one port used for vSAN traffic, one for vMotion

[2] The results here are not official TPCx-HCI results and are not meant to be compared with audited results. The benchmark was used as a tool to drive a migration load and to observe the benefits of vMotion optimizations.

Software

- 5 tiles of TPCx-HCI VMs; a total of 60 VMs
  - Each tile has 12 VMs, all different in terms of vCPU count (2 to 28), memory size (2GB to 98GB), and number of virtual disks (1 to 5). So, there is a wide range of resource usage by the VMs
- Guest OS/Application: RHEL 7.6 / PostgreSQL 10.6
- Benchmark: TPCx-HCI version 1.1.7 benchmark kit, downloaded from tpc.org

Dependence of TPCx-HCI on DRS and vMotion

TPCx-HCI has been designed to benefit from good migration performance. On a four-node cluster, all the VMs are required to start on three of the nodes. Half-way through the warm-up period, we can enable automatic load balancing to spread the VMs among the nodes. vSphere DRS is used in our case. The load directed to the 60 VMs varies every 12 minutes. So, 10 times during the two-hour run time, DRS needs to step in and migrate VMs to balance the load across the four nodes to maintain high throughput. The benchmark specification does not require rebalancing and the test sponsor may run without load balancing, albeit at less-than-optimal performance. But a good load balancer and optimized migrations will achieve much better results.

Figure 11 (next page) shows the workload profile of TPCx-HCI. The important observations are as follows:

- The benchmark has a lot of moving parts and built-in elasticity. So, the throughput and utilization curves have a lot of fluctuations. That's typical of this benchmark.
- The solid black line is the throughput. The distribution of load among the 60 VMs changes every 12 minutes, but the overall load is kept constant. So, if the load balancer does its job well, the throughput will remain relatively constant.
- The four dotted lines show the CPU utilizations of the four nodes. As required, node 4 starts out with no VMs and has a utilization of about 20% until the middle of the warm-up period when DRS kicks in and moves VMs to node 4 to balance the load. Throughout the run, the load varies every 12 minutes, the nodes go out of balance, and DRS activates to rebalance them.
- The green line on the bottom shows the total vMotion migrations for each minute. During warm-up, moving VMs to the idle node 4 requires about 20 migrations; fewer migrations are needed for the rest of the run. There are a total of 158 migrations during the two-hour run.

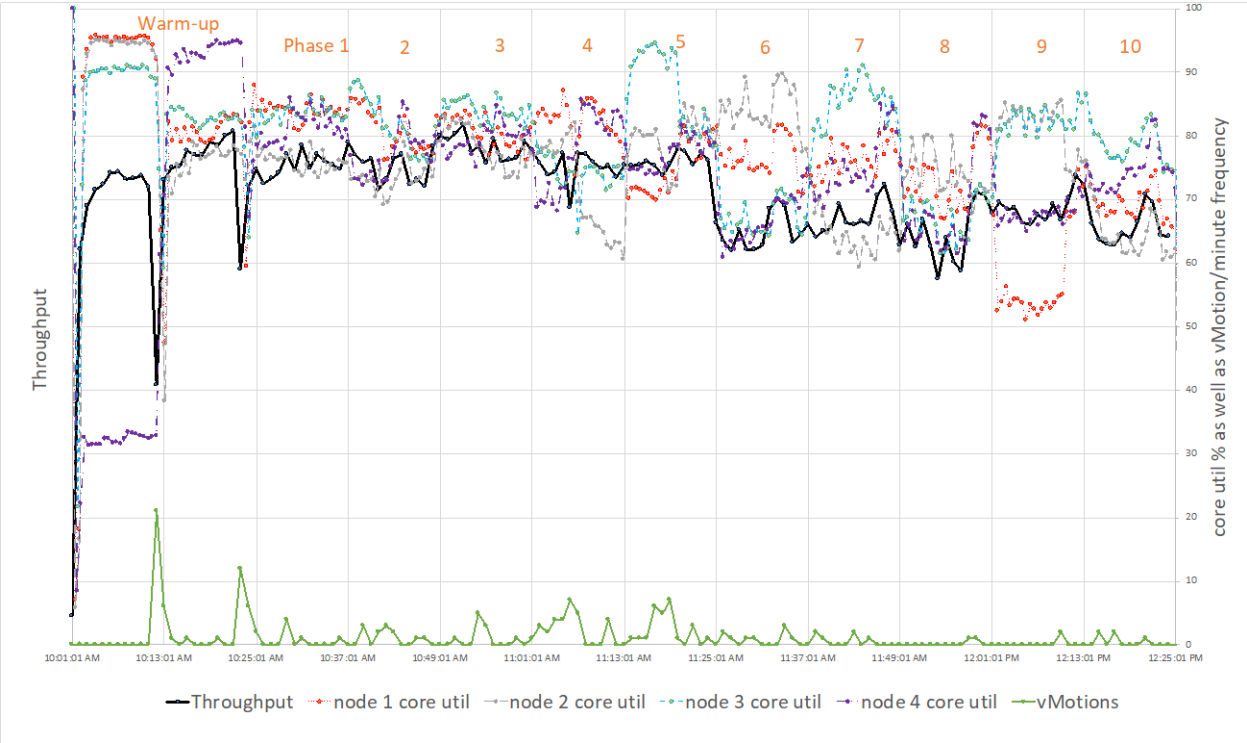

Figure 11. Throughput, CPU utilization, and migrations for a 2-hour TPCx-HCI run

Impact of vMotion Performance on TPCx-HCI

TPCx-HCI is a great benchmark for measuring vMotion performance because TPC benchmarks have always had a 90% response-time criteria for database transactions. For TPCx-HCI, these 90% response-time limits vary from 1 to 3 seconds. In practice, we meet these requirements very easily, with the 90th percentile response times in the tens of milliseconds. However, TPC benchmarks always place another restriction on response time: the average response time must be lower than the 90th percentile. This requirement is violated if there are long-tail response times in the mix. There are 2,000 response-time checks for our configuration, checking the response times for every transaction type for every VM for each of the ten 12-minute phases.

It is not hard to see how an inefficient vMotion can lead to a failed average < 90th % check. With response times in the 10-millisecond range, if 1% of transactions experience a 1-second downtime due to vMotion, we can end up with an average > 90th %. So TPCx-HCI is a great tool to study vMotion downtime with its elastic, unpredictable workload. With 2,000 different checks, unless vMotion downtime is truly unnoticeable, we have a good chance of failing one or more average < 90th % checks. A single such failure makes the benchmark run invalid.

Due to the elastic, non-deterministic nature of the benchmark, we took 18 runs at each data point to get a statistically significant measure of long-tail response times. Also, we set the DRS ***Migration Threshold*** to the maximum level of 5 to force a high migration rate to study the effects of vMotion on long-tail response times.

When we started with 6.7 U3, we had:

- 14 of 18 runs had average > 90th % and invalidated the run.
- There were a total of 148 failed checks over the 18 runs.
- This was due to occasional vMotion downtimes of several seconds, as described in the earlier sections.

With vSphere 7.0 U1, we see:

- 2 of 18 runs had average > 90th % and invalidated the run (an 86% reduction in failures compared to vSphere 6.7).
- There were a total of 4 failed checks over the 18 runs (a 97% reduction in failed checks compared to vSphere 6.7).

Conclusion

vMotion is one of the most popular features of vSphere. It provides invaluable benefits to administrators of virtualized datacenters, it enables business continuity and flexibility even in the face of physical server downtime due to patch updates or troubleshooting, and it enables load balancing.

vSphere 7.0 U1 includes several performance enhancements to make it easier than ever to enable vMotion on even monster-sized virtual machines running heavy-duty, enterprise-class applications, with minimal overhead on guest workloads.

Test results show the following:

- Improvements in vSphere 7.0 U1 over vSphere 6.7 are twofold: the duration of vMotion and the impact on application performance during vMotion.
- The completely re-architected hypervisor page tracing results in dramatic improvements in guest workload performance (for example, Oracle DB throughput improved by a 19x factor) in vSphere 7.0 U1 over vSphere 6.7.
- There are consistent reductions in the range of 50% in vMotion duration on vSphere 7.0 U1 over vSphere 6.7, due to the improved heuristics and optimizations introduced in vSphere 7.0 U1 vMotion.

References

[1] Setty, Sreekanth; VMware. (2011) VMware vSphere vMotion Architecture, Performance, and Best Practices in VMware vSphere 5. http://www.vmware.com/techpapers/2011/vmotion-architecture-performance-and-best-practi-10202.html

[2] HammerDB benchmarking and load testing software. https://hammerdb.com

[3] TPC. TPC-C online transaction processing benchmark. http://www.tpc.org/tpcc/

[4] TPC. TPC Express Benchmark HCI (TPCx-HCI). http://www.tpc.org/tpcx-hci/

About the Author

Sreekanth Setty is a staff member with the performance engineering team at VMware. His work focuses on investigating and improving the performance of VMware's virtualization stack, most recently in the live-migration area. He has published his findings in many technical papers and has presented them at various technical conferences. He has a master's degree in computer science from the University of Texas, Austin.

Acknowledgements

The author would like to sincerely thank Arun Ramanathan, Yury Baskakov, Yanlei Zhao, Anurekh Saxena, and Reza Taheri for their invaluable contributions to the paper. He also would like to extend thanks to the members in his team including Tony Lin and Julie Brodeur for their reviews.