Virtualizing Performance-Critical Database Applications in VMware vSphere 6.0

Performance Study

TECHNICAL WHITE PAPER

Table of Contents

Executive Summary

The performance parity between bare-metal and virtualized servers is an accepted fact today. But what about the most demanding applications, such as monster virtual machines running databases and transaction processing applications? Experiments in this paper demonstrate that VMware vSphere® 6.0 virtual machines run out of the box at 90% of the performance of native systems even for the most demanding workloads at the highest throughput levels.

Introduction

Customers have successfully deployed database applications in virtual machines since the earliest versions of VMware ESX®, and vSphere 6.0 continues to meet the challenge of virtualizing extremely resource-intensive database workloads such as those that issue a large number of storage commands and experience high network activity to serve remote clients. New performance features in vSphere 6.0 readily handle the high consumption of CPU and memory resources of these demanding database applications.

Through performance experiments, we show how vSphere is up to the task of running robust database applications in virtual machines and answers some common customer questions that include:

- What is the performance of heavy-duty database applications in virtual machines?
- How does a virtualized environment handle high storage I/Os per second (IOPS) and high network traffic?
- Does the virtual–native performance parity extend to wide virtual machines with 64 vCPUs or more?

In this paper, we quantify the performance of a virtualized server for a very high-end Oracle database deployment (with a much larger resource footprint than customers expect to see in most production environments) and highlight the overall system performance. Our experiments show that large database applications perform well in vSphere virtual machines.

Performance Test Environment

Workload Characteristics

We derive a workload, known here as the Order-Entry benchmark, from the TPC-C workload. The Order-Entry benchmark is a non-compliant implementation of the TPC-C business model; it is not comparable to official, published TPC-C results.

The Order-Entry benchmark is an online transaction processing (OLTP) workload that runs many small transactions. Of the five transaction types, three update the database and the other two, which occur with relatively low frequency, are read-only. The I/O load is quite heavy and consists of small access sizes (2k-16k). The disk I/O accesses consist of random reads and writes with a 60/40 ratio in favor of reads. In terms of the impact on the system, this benchmark spends considerable execution time in the operating system's kernel context, which is harder to virtualize than user mode code. Specifically, how well ESXi virtualizes the hardware interrupt processing, I/O handling, context switching, and scheduler portions of the guest operating system code is critical to the performance of this benchmark. The workload is also very sensitive to the processor cache and translation lookaside buffer (TLB) hit ratios.

Hardware Configuration

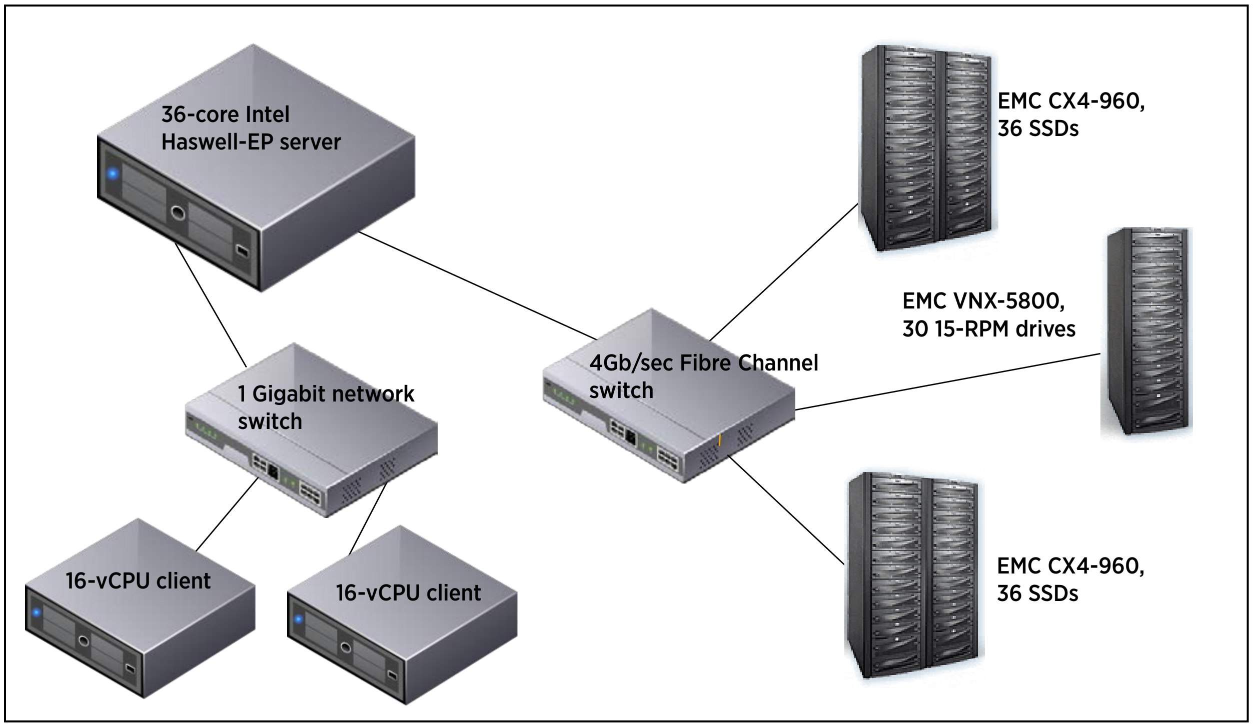

Figure 1. Hardware configuration

The testbed for the experiments consists of a server machine, two client machines used to drive the benchmark, and three large storage arrays to keep the disk latencies at acceptable levels. Figure 1 shows the server and storage components and connectivity. We use the same components in all tests by booting up the server either into ESXi or into native Linux.

Server hardware:

- Dell PowerEdge R730
- Two 18-core Intel Processors Xeon E5-2699 v3 (“Haswell-EP”) @ 2.30GHz
- 512GB memory

Server storage:

- 16 data LUNs on 72 SSDs on two EMC CLARiiON CX4-960 arrays
- 2 redo log LUNs on 30 15K-RPM disk drives on an EMC VNX-5800 array

Client machines:

- A 16-vCPU VM with 64GB of memory on a 2-socket X5550 (“Nehalem-EP”) server
- A 16-vCPU VM with 64GB of memory on a 4-socket E7-4870 (“Westmere-EX”) server

Software Configuration

Table 1 lists the software components of the testbed. The virtual machine and native machine configurations are identical in that they use the same operating system and DBMS versions, which are configured the same way. We used the same benchmarking scripts to set up and run the benchmark. We ran both native and virtual experiments against the same database that was built on the three storage arrays. In both cases, we used large memory pages to ensure optimum performance.

Software Stack	Version
ESXi	vSphere Release 6.0
Guest/native OS	RHEL 7.1 64-bit (3.10.0-229)
DBMS	Oracle 12C (12.1.0.2.0)

Table 1. Server software

Benchmark Design Considerations

With a monster virtual machine configuration at full CPU saturation such as the one we use in this study, we advise you to leave some system resources available for the hypervisor vmkernel threads. For workloads that have both high compute and high I/O requirements, you can get better performance by dividing the host system resources between the virtual machine and the hypervisor kernel. This allows the hypervisor to perform its own functions as well as the tasks it does on behalf of the virtual machine, such as I/O.

The server under test has 36 cores and 72 physical Hyper-Thread CPUs. So for the experiments on ESXi 6.0, we configure 64 virtual CPUs (vCPUs) in the virtual machine. With 72 physical CPUs (pCPUs) available, the scheduler can schedule as many as 8 auxiliary vmkernel threads on other pCPUs without impacting the guest. Note that the scheduler can still decide to allow a pCPU to be shared between a vCPU and a vmkernel thread. We retain the default setting so the scheduler still makes optimal scheduling decisions. We configure the guest with 475GB of memory.

Although this is an undercommitted configuration, the virtual–native comparisons are fair because they are comparing a 64-vCPU virtual machine on a 72-pCPU host with that same host running 72 pCPUs in the native configuration (as opposed to the unfair situation of comparing a 64-pCPU native system to a 64-vCPU virtual machine that runs on a host with more than 64 physical processors).

Database size is an important parameter in this benchmark. Vendors sometimes size their databases based on a small warehouse count, which results in a “cached” database and a low I/O rate with misleading results. We run tests with 32K warehouses, taking up 3.8TB of disk space. We size the Oracle SGA to 440GB. The large ratio between database size and SGA size gives rise to significant storage I/O activity.

Performance Results

We conduct experiments using the Order-Entry benchmark with the goal of quantifying:

- Performance gains due to enhancements built into ESXi 6.0
- Performance differential between ESXi 6.0 and ESXi 5.1
- Performance differential between ESXi 6.0 and native

The following sections detail the results of our experiments with the Order-Entry benchmark run on both native and virtual machines.

ESXi 6.0 Performance Relative to Native

Results from these experiments show that you can virtualize even the largest database applications with excellent performance. For example, with a 64-vCPU virtual machine running on a 72-pCPU ESXi host, throughput is 90% of native throughput on the same hardware platform. Table 2 summarizes statistics that give an indication of the load placed on the system in the native and virtual machine configurations. We base CPU utilizations on Non-Halted Cycles, which we measure using hardware monitoring tools, and we report the metric as the average utilization of all 36 cores/72 threads on the server. In the virtual case, the guest operating system in the 64-vCPU virtual machine reports a CPU utilization of 90.2%, and the hardware counters in the host report an average utilization of 85.1% for the 36 cores/72 threads. We used the average utilization of all 72 HT threads for an apple-to-apples comparison with the 72-thread native results.

Metric	Native System	Virtual Machine
Throughput in transactions per second	66.5K	59.5K
Average CPU utilization of 72 logical CPUs	84.7%	85.1%
Disk IOPS	173K	155K
Disk Megabytes/second	929MB/s	831MB/s
Network packets/second	71K/s receive 71K/s send	63K/s receive 64K/s send
Network Megabytes/second	15MB/s receive 36MB/s send	13MB/s receive 32MB/s send

Table 2. Comparison of native and virtual machine benchmark load profiles

Table 3 shows the corresponding guest statistics we collected while running the Order-Entry benchmark; these statistics provide another perspective on the resource-intensive nature of the workload. These common Linux performance metrics show that while the benchmark workload was heavy in terms of raw CPU demands, it also placed a heavy load on the operating system, interrupt handling, and the storage subsystem—areas that have been traditionally associated with high virtualization overheads.

Metric	Amount
Interrupts per second	327K
Disk IOPS	155K
Context switches per second	287K
Load average	231

Table 3. Guest OS statistics

CPU Utilization

According to common practice for performance testing workloads like the Order-Entry benchmark, we increase the load until it nearly saturates the CPU. Occasionally, we encounter reports that compare two throughputs without reporting the CPU utilization, or claim two systems are equivalent, even though one requires more CPU cycles to achieve the same throughput. In this report, we compare the native and virtual throughputs at the same, near-saturation CPU utilizations.

We measure the average physical CPU utilization with the hardware event counters available on Intel processors, which show that both the native and virtual configurations run at 85% average physical CPU utilization across all 72 physical CPU threads. We find that the workload cannot fully saturate the system due to long disk response times under maximum load. We could have pushed CPU utilization even higher with a more capable storage subsystem with lower latencies (even though it is uncommon for real world servers to go beyond 85% CPU utilization). The virtual server reached a maximum utilization of 85%. The native server was able to reach a higher CPU utilization of 91% at a throughput of 69.6K transactions per second due to a lower disk latency and less *waitio*. In order to provide an apple-to-apples comparison between native and virtual servers, we report the results in this paper from a native run that was also at 85% CPU utilization.

Comparison with ESXi 5.1

Experimental data comparing ESXi 6.0 with ESXi 5.1 show that high-end scale-up with ESXi 6.0 now mirrors that of native systems. To be more specific, previous tests from as far back as 2009 showed an 85% virtual–native performance ratio for 2-processor, 8-core Intel Nehalem-based servers with ESXi 4.0. However, that ratio declined at CPU counts of 32 or higher, even with ESXi 5.x.

Observe the data in Figure 2 and Figure 3. With ESXi 5.1, the Order-Entry benchmark throughput of a 64-vCPU virtual machine on a 4-socket, 32 core/64 thread E7- 4870 (Westmere) server is only 70% of the throughput of the same server in native mode when both servers are running at 77% CPU utilization (the native server reached a maximum CPU utilization of 88% and throughput of 54.8 transactions per second).

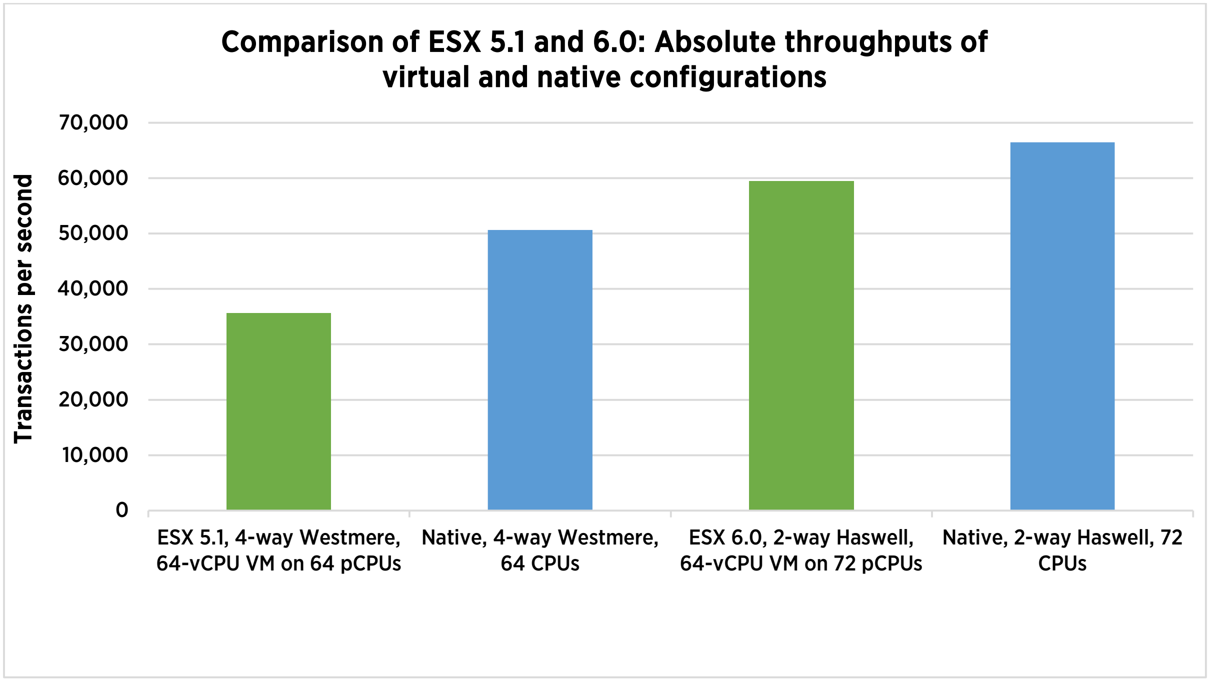

Figure 2. Absolute throughput values

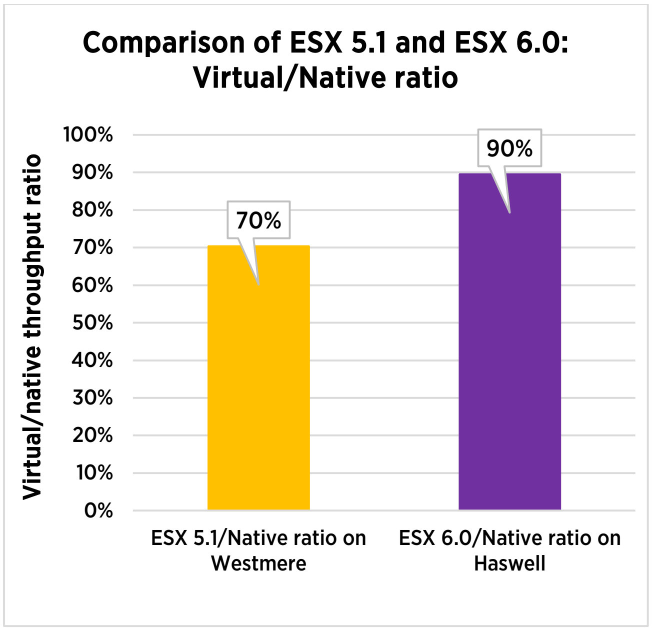

Figure 3. Relative throughput ratios

Performance Impact of New Features and Enhancements

Numerous performance enhancements have made ESXi 6.0 an even better platform for performance-critical applications compared to the ESXi 5.x releases. Consult the online document "What's New in the VMware vSphere 6 Platform" [1] for details of those enhancements. The work presented here benefits from the following optimizations:

- Storage stack optimizations
- Networking stack optimizations
- ESXi 6.0 increases the virtual machine CPU and memory sizes to 128 and 4TB, respectively. Although the virtual machine in this test is limited to 64 CPUs and 475GB, increasing the maximum limits requires improved scale-up performance, which also benefits configurations below the maximum limits.
- The choice of hardware also impacts performance. The Haswell processor has an improved TLB design that lowers the cost of TLB miss processing, especially for virtualized servers. (See "TLB Miss Processing Overhead.")

Performance Optimizations Used in the Study

One of the key performance features of ESXi 5.0 and 6.0 is out-of-the-box performance. Maximizing performance for database applications in releases before 5.0 sometimes required careful placement of vCPUs and vmkernel auxiliary threads on specific physical CPUs, as well as detailed tuning of networking and storage stacks. The appendix lists the non-default settings used in this study. Note that these settings are listed not to suggest they are best practices for performance, but as a full disclosure of all non-default settings used to maximize performance for this particular workload on this particular configuration and to highlight how *few* non-default settings were required. For the complete VMX file, see "Virtual Machine Tunable Parameters."

Achieving optimal performance requires tuning all the hardware and software in the stack, including the server hardware, processor BIOS settings, storage array settings, LUN settings, Linux operating system tunables (see

"Linux Kernel Tunables"), DBMS tunables (see "init.ora Parameters"), benchmark code and configuration, layout of benchmark data on the disks, and so on. Experience has shown that optimal performance cannot be reached without holistic tuning of the whole stack, and that was the case in our experiments, too.

We apply identical optimizations to the guest and native operating systems (for example, spreading device interrupts among multiple CPUs by manipulating the `/proc/irq/NNN/smp_affinity_list` nodes). We use the same settings (for example, the same Oracle `init.ora` file) for the database. We use the same benchmarking infrastructure (code, driver systems, and so on) and the same physical database for both cases.

TLB Miss Processing Overhead

The costs of processing translation lookaside buffer (TLB) misses can be substantial for workloads that use a lot of memory (as in database management systems or high-performance computing, for example). This overhead can grow in a virtual environment because of the *two-dimensional page walk* that is a side-effect of how modern X86 processors virtualize *virtual memory* [2]. In the early days of virtualization, the TLB miss processing overhead could sometimes double from a native server to its virtual counterpart.

In testing with ESXi 6.0 and the Haswell-EP processor, the processor spends 16% of its time in TLB miss processing in native mode and 20% in virtual. The combination of ESXi 6.0 and the most recent X86 processors make for much more efficient handling of TLB misses in a virtual environment, contributing to the virtual-native performance parity.

Conclusion

The amount of I/O and transactions per second that typical database applications generate are well below what ESXi 6.0 can handle. Experiments in this paper show that a 64-vCPU, 475GB virtual machine processes 59.5K DBMS transactions per second while issuing 155K IOPS, capabilities well above even most high-end Oracle database installations. The most demanding applications can be run with excellent performance in a virtualized environment with ESXi 6.0. Even for the applications that may require 64 or 128 vCPUs, the high-end performance boost of ESXi 6.0 over ESXi 5.x makes ESXi 6.0 an even better platform for virtualizing Oracle databases.

Disclaimers

The workload we used in our experiments is derived from the TPC-C workload. We refer to this workload as the Order-Entry benchmark, which is a non-compliant implementation of the TPC-C business model and is not comparable to official, published TPC-C results. In particular, the Order-Entry benchmark is a batch implementation and has an undersized database for the observed throughput.

Results do not represent the performance of Oracle software, neither are they meant to measure Oracle performance nor compare the performance of Oracle to another DBMS. We simply use Oracle to place a DBMS workload on ESXi to observe and optimize the performance of ESXi.

Appendix

Virtual Machine Tunable Parameters

```
.encoding = "UTF-8"
config.version = "8"
virtualHW.version = "11"
nvram = "tpcc-server-vm1.nvram"
pciBridge0.present = "TRUE"
svga.present = "TRUE"
pciBridge4.present = "TRUE"
pciBridge4.virtualDev = "pcieRootPort"
pciBridge4.functions = "8"
pciBridge5.present = "TRUE"
pciBridge5.virtualDev = "pcieRootPort"
pciBridge5.functions = "8"
pciBridge6.present = "TRUE"
pciBridge6.virtualDev = "pcieRootPort"
pciBridge6.functions = "8"
pciBridge7.present = "TRUE"
pciBridge7.virtualDev = "pcieRootPort"
pciBridge7.functions = "8"
vmci0.present = "false"
hpet0.present = "TRUE"
vmx.buildType = "stats"
svga.vramSize = "8388608"
numvcpus = "64"
memSize = "486400"
sched.mem.min = "486400"
sched.mem.minSize = "486400"
sched.mem.shares = "normal"
sched.cpu.units = "mhz"
sched.cpu.affinity = "all"
sched.mem.affinity = "all"
powerType.powerOff = "soft"
powerType.suspend = "hard"
powerType.reset = "soft"
scsi0.virtualDev = "pvscsi"
scsi0.present = "TRUE"
scsi0.intrCoalescing = "false"
scsi0.reqCallThreshold = 1
sata0.present = "TRUE"
scsi0:0.deviceType = "scsi-hardDisk"
scsi0:0.fileName = "tpcc-server-vm1-000001.vmdk"
sched.scsi0:0.shares = "normal"
sched.scsi0:0.throughputCap = "off"
scsi0:0.present = "TRUE"
ethernet0.virtualDev = "vmxnet3"
ethernet0.networkName = "VM Network"
ethernet0.addressType = "vpx"
ethernet0.generatedAddress = "00:50:56:8c:b9:85"
ethernet0.uptCompatibility = "TRUE"
```

```
ethernet0.present = "TRUE"
sata0:0.deviceType = "cdrom-image"
sata0:0.fileName = "/vmfs/volumes/4b2acea8-7cb333d8-be6d-18a9053b4890/locker/rhel-
server-7.1-x86_64-dvd.iso"
sata0:0.present = "TRUE"
floppy0.startConnected = "FALSE"
floppy0.clientDevice = "TRUE"
floppy0.fileName = "vmware-null-remote-floppy"
ethernet1.virtualDev = "vmxnet3"
ethernet1.networkName = "Private Network"
ethernet1.addressType = "vpx"
ethernet1.generatedAddress = "00:50:56:8c:6f:4f"
ethernet1.uptCompatibility = "TRUE"
ethernet1.present = "TRUE"
ethernet1.coalescingScheme = "static"
ethernet1.coalescingParams = "8,64,8"
displayName = "tpcc-server-vm1"
guestOS = "rhel7-64"
bios.bootDelay = "2000"
toolScripts.afterPowerOn = "TRUE"
toolScripts.afterResume = "TRUE"
toolScripts.beforeSuspend = "TRUE"
toolScripts.beforePowerOff = "TRUE"
uuid.bios = "56 4d 10 19 ae d3 df cb-e8 41 de 99 3d 91 ba bf"
vc.uuid = "50 0c fd d4 cd 89 03 74-0a c8 e4 4c 70 7c a4 50"
sched.cpu.min = "0"
sched.cpu.shares = "normal"
virtualHW.productCompatibility = "hosted"
sched.swap.derivedName = "/vmfs/volumes/55565160-1a8fd049-d9c3-ecf4bbd14b8c/tpcc-server-
vm1/tpcc-server-vm1-362e860f.vswp"
uuid.location = "56 4d 10 19 ae d3 df cb-e8 41 de 99 3d 91 ba bf"
replay.supported = "FALSE"
replay.filename = ""
migrate.hostlog = "./tpcc-server-vm1-9b3dc4d4.hlog"
scsi0:0.redo = ""
pciBridge0.pciSlotNumber = "17"
pciBridge4.pciSlotNumber = "21"
pciBridge5.pciSlotNumber = "22"
pciBridge6.pciSlotNumber = "23"
pciBridge7.pciSlotNumber = "24"
scsi0.pciSlotNumber = "160"
ethernet0.pciSlotNumber = "192"
ethernet1.pciSlotNumber = "224"
vmci0.pciSlotNumber = "-1"
sata0.pciSlotNumber = "33"
scsi0.sasWWID = "50 05 05 69 ae d3 df c0"
vmci0.id = "-1349589935"
monitor.phys_bits_used = "42"
vmotion.checkpointFBSize = "8388608"
vmotion.checkpointSVGAPrimarySize = "8388608"
cleanShutdown = "FALSE"
softPowerOff = "FALSE"
sata0:0.allowGuestConnectionControl = "TRUE"
```

```
tools.syncTime = "FALSE"
extendedConfigFile = "tpcc-server-vm1.vmxf"
tools.remindInstall = "FALSE"
scsi1.present = "TRUE"
scsi1.sharedBus = "none"
scsi1.virtualDev = "pvscsi"
scsi2.present = "TRUE"
scsi2.sharedBus = "none"
scsi2.virtualDev = "pvscsi"
scsi3.present = "TRUE"
scsi3.sharedBus = "none"
scsi3.virtualDev = "pvscsi"
scsi1:0.present = "true"
scsi1:0.deviceType = "scsi-hardDisk"
scsi1:0.filename = "tpcc-server-vm1_1.vmdk"
scsi1:0.mode = "independent-persistent"
scsi1:0.redo = ""
# The parameters “shares”, “throughputCap”, “present”, “deviceType”,
# “mode”, and “redo” for the remainder of the disks are identical to
# the settings for scsi1.0, and were deleted in this white paper for
# the sake of brevity
# vmdk1 to vmdk12 are data LUNs, and are striped across 3 virtual
# SCSI buses to spread the interrupt processing load of the guest OS.
# vmdk13 and vmdk14 are the redo log disks. We put them on scsi0,
# which has no other load during the benchmark run, and opt for
# low latency over CPU efficiency by setting
# scsi0.intrCoalescing = "false" and scsi0.reqCallThreshold = 1
# vmdk15 to vmdk34 hold the backup of the database, and can be
# mapped to any remaining virtual SCSI devices since they do
# not impact performance
scsi1:1.filename = "tpcc-server-vm1_2.vmdk"
scsi1:2.filename = "tpcc-server-vm1_3.vmdk"
scsi1:3.filename = "tpcc-server-vm1_4.vmdk"
scsi2:0.filename = "tpcc-server-vm1_5.vmdk"
scsi2:1.filename = "tpcc-server-vm1_6.vmdk"
scsi2:2.filename = "tpcc-server-vm1_7.vmdk"
scsi2:3.filename = "tpcc-server-vm1_8.vmdk"
scsi3:0.filename = "tpcc-server-vm1_9.vmdk"
scsi3:1.filename = "tpcc-server-vm1_10.vmdk"
scsi3:2.filename = "tpcc-server-vm1_11.vmdk"
scsi3:3.filename = "tpcc-server-vm1_12.vmdk"
scsi0:1.filename = "tpcc-server-vm1_13.vmdk"
scsi0:2.filename = "tpcc-server-vm1_14.vmdk"
scsi1:8.filename = "tpcc-server-vm1_15.vmdk"
scsi1:9.filename = "tpcc-server-vm1_16.vmdk"
scsi1:10.filename = "tpcc-server-vm1_17.vmdk"
scsi1:11.filename = "tpcc-server-vm1_18.vmdk"
scsi1:12.filename = "tpcc-server-vm1_19.vmdk"
scsi1:13.filename = "tpcc-server-vm1_20.vmdk"
scsi1:14.filename = "tpcc-server-vm1_21.vmdk"
scsi1:15.filename = "tpcc-server-vm1_22.vmdk"
scsi2:9.filename = "tpcc-server-vm1_23.vmdk"
scsi2:10.filename = "tpcc-server-vm1_24.vmdk"
```

```
scsi2:11.filename = "tpcc-server-vm1_25.vmdk"
scsi2:12.filename = "tpcc-server-vm1_26.vmdk"
scsi2:13.filename = "tpcc-server-vm1_27.vmdk"
scsi2:14.filename = "tpcc-server-vm1_28.vmdk"
scsi2:15.filename = "tpcc-server-vm1_29.vmdk"
scsi3:9.filename = "tpcc-server-vm1_30.vmdk"
scsi3:10.filename = "tpcc-server-vm1_31.vmdk"
scsi3:11.filename = "tpcc-server-vm1_32.vmdk"
scsi3:12.filename = "tpcc-server-vm1_33.vmdk"
scsi3:13.filename = "tpcc-server-vm1_34.vmdk"
scsi1.pciSlotNumber = "256"
scsi2.pciSlotNumber = "1184"
scsi3.pciSlotNumber = "1216"
scsi1.sasWWID = "50 05 05 69 ae d3 de c0"
scsi2.sasWWID = "50 05 05 69 ae d3 dd c0"
scsi3.sasWWID = "50 05 05 69 ae d3 dc c0"
scsi0:3.fileName = "tpcc-server-vm1_35-000001.vmdk"
scsi0:4.fileName = "tpcc-server-vm1_36.vmdk"
scsi0:5.fileName = "tpcc-server-vm1_37.vmdk"
numa.autosize.vcpu.maxPerVirtualNode = "32"
numa.autosize.cookie = "640001"
scsi0:6.fileName = "tpcc-server-vm1_38.vmdk"
scsi0:6.present = "TRUE"
toolsInstallManager.updateCounter = "6"
```

init.ora Parameters

```
compatible = 12.1.0.2.0
plsql_optimize_level=2
utl_file_dir=*
db_name = tpcc
control_files = (+DATA/control_001, +DATA/control_002)
parallel_max_servers = 0
_two_pass=false
db_files = 440
db_cache_size   = 15G
db_8k_cache_size  = 6G
db_16k_cache_size  = 120G
db_keep_cache_size  = 235G
db_recycle_cache_size  = 45G
dml_locks = 500
statistics_level = basic
processes = 1300
sessions = 1250
transactions = 1100
shared_pool_size = 14400M
db_block_size = 2048
db_writer_processes = 40
disk_asynch_io = TRUE
UNDO_TABLESPACE = undo_1
undo_management = auto
transactions_per_rollback_segment = 1
aq_tm_processes = 0
```

```
db_block_checking = false
db_block_checksum = false
fast_start_mttr_target = 0
java_pool_size = 0
log_checkpoint_interval = 0
log_checkpoints_to_alert = TRUE
log_checkpoint_timeout = 0
max_dump_file_size = 1M
timed_statistics = true
trace_enabled = false
query_rewrite_enabled = false
replication_dependency_tracking = false
resource_manager_plan = ''
_undo_autotune = false
_collect_undo_stats = false
db_cache_advice=off
log_buffer = 101580800
open_cursors = 100
sort_area_size = 2097152
_use_realfree_heap = FALSE
undo_retention = 1
```

Linux Kernel Tunables

```
kernel.sem = 250 32000 100 128
kernel.shmall = 134217728
kernel.shmmax = 549755813888
kernel.shmmni = 4096
kernel.panic_on_oops = 1
fs.file-max = 6815744
fs.aio-max-nr=1048576
net.ipv4.ip_local_port_range = 9000 65500
net.core.rmem_default = 4194304
net.core.rmem_max = 4194304
net.core.wmem_default = 262144
net.core.wmem_max = 1048586
kernel.core_uses_pid = 1
kernel.wake_balance = 0
vm.hugetlb_shm_group = 54322
vm.legacy_va_layout = 0
vm.nr_hugepages = 225500
kernel.perf_event_paranoid = -1
```

Non-Default Performance Optimizations

- We used virtual RDM disks (see VMware KB 2009226 [3] and VMDK vs. RDM [4]). Although Virtual RDM might have a slight performance advantage over VMFS, we employed it to allow the use of the same exact physical database for both native and virtual experiments.
- To reduce the CPU consumption of virtual network interrupt processing, we set the virtual network interrupt coalescing mode to static by manipulating the `ethernet1.coalescingScheme` and `ethernet1.coalescingParams` parameters (see the "Performance Best Practices Guide" [5]). This makes sense for this workload, which is much more sensitive to the CPU utilization of processing networking packets than it is to networking latency. Do not use the static setting for latency-sensitive applications (see the "Best Practices for Performance Tuning of Latency-Sensitive Workloads in vSphere VMs" [6]).
- Databases are sensitive to the latency of redo log writes. To ensure low latencies, we avoid interrupt coalescing on the virtual SCSI bus that holds the redo log by not putting any data LUNs on that bus and by setting `scsi0.intrCoalescing` to `"false"` and `scsi0.reqCallThreshold` to `1` (see the "Performance Best Practices Guide" [5]).

References

[1] VMware, Inc. (2015, August) What's New in the VMware vSphere 6.0 Platform.
https://www.vmware.com/files/pdf/vsphere/VMW-WP-vSPHR-Whats-New-6-0-PLTFRM.pdf

[2] VMware, Inc. (2009) Performance Evalutation of Intel EPT Hardware Assist.
http://www.vmware.com/pdf/Perf_ESX_Intel-EPT-eval.pdf

[3] VMware, Inc. (2015, October) Difference Between Physical Compatibility RDMs and Virtual Compatibility RDMs (2009226). http://kb.vmware.com/kb/2009226

[4] Mark Achtemichuk. (2013, January) vSphere 5.1 - VMDK versus RDM.
https://blogs.vmware.com/vsphere/2013/01/vsphere-5-1-vmdk-versus-rdm.html

[5] VMware, Inc. (2015, June) Performance Best Practices for VMware vSphere 6.0.
http://www.vmware.com/files/pdf/techpaper/VMware-PerfBest-Practices-vSphere6-0.pdf

[6] VMware, Inc. (2013, March) Best Practices for Performance Tuning of Latency-Sensitive Workloads in vSphere VMs. https://www.vmware.com/files/pdf/techpaper/VMW-Tuning-Latency-Sensitive-Workloads.pdf

About the Author

At VMware since 2007, Reza Taheri has worked on tuning vSphere for Tier 1 applications, especially databases and transaction processing workloads, to close the performance gap with native systems for these most challenging workloads to below 10%. He enjoys working on the interactions of the many components of a system, from low level processor characteristics, to database query optimization, to storage array caching policies. He started his career at Bell Labs for 5 years, then he was at HP for 20 years. He chairs the industry-standard committee that is developing a new TPC performance benchmark for virtualized databases.

Acknowledgements

The author would like to thank the following people for their contributions to this paper: Priti Mishra, Bruce Herndon, Todd Muirhead, David Morse, Tariq Magdon-Ismail, Seongbeom Kim, Lenin Singaravelu, Qasim Ali, Son Luong, Haoqiang Zheng, Mark Achtemichuk, and Julie Brodeur.