Virtualization Performance Insights from TPC-VMS

Wayne D. Smith, Shiny Sebastian

Intel Corporation
wayne.smith@intel.com
shiny.sebastian@intel.com

Abstract. This paper describes the TPC-VMS (Virtual Measurement Single System) benchmark that leverages the TPC-C, TPC-E, TPC-H, and TPC-DS benchmarks to provide a measure of database performance in a virtualized environment. TPC-VMS requires 3 identical TPC Benchmarks to be run in separate virtual machines, i.e. 3 TPC-C VMs, 3 TPC-E VMs, 3 TPC-H VMs or 3 TPC-DS VMs. The TPC-VMS performance metric is the minimum value of the three TPC performance metrics. During the development phase, the workload was prototyped to prove the viability of the benchmark. At first glance TPC-VMS was considered a simple benchmark; however the prototyping effort uncovered a number of performance issues intrinsic to virtualization of database applications.

1 Introduction

Cloud computing delivers virtual processors, memory and storage to a community of end-users. Key to the cloud is virtualization technology that provides a separate virtual machine environment for multiple users. A number of benchmarks currently exist to test virtualization. However there are few workloads that characterize large database performance in a virtualized environment. TPC Benchmarks are the gold standard for large database performance. This paper describes the TPC-VMS (Virtual Measurement Single System) benchmark that employs the TPC-C, TPC-E, TPC-H, and TPC-DS benchmarks to provide a measure of database performance in a virtualized environment.

2 Virtualization

Virtualization provides separate virtual machine (VM) environments for multiple users. Typically each VM includes an operating system and application code. The VM provides security and isolation from other operating systems and application code running in their own virtual machines. A hypervisor provides the virtualization of the underlying hardware by managing the virtual machines.

2.1 Virtualization Benchmarks

There are existing virtualization benchmarks such as SPECvirt_sc2010 [1], and VMmark[2]. The benchmarks have focused on smaller workloads where the virtualized environment consumes a fraction of a processor core. Thus one processor core can support several Virtual Processors (VPs) that execute the workload inside the VM. At the other extreme are database applications which may require several processor cores if not several processor sockets. There is a lack of benchmarks/workloads that characterizes large database performance in a virtualization environment. The TPC-VMS Benchmark (Virtual Measurement Single System) was created to address this issue.

3 TPC-VMS

The TPC Virtual Measurement Single System Specification [3], TPC-VMS, contains the rules and methodology for measuring and reporting TPC Benchmark metrics running in a virtualized environment. TPC-VMS leverages the TPC-C, TPC-E, TPC-H and TPC-DS Benchmarks by adding the methodology and requirements for running and reporting virtualization metrics. TPC-VMS defines four new benchmarks that are neither comparable to each other nor to the base benchmarks from which they are derived. A TPC-VMS result is a standalone TPC result. There is no requirement to publish a result of the TPC Benchmark used as the basis for the TPC-VMS result.

3.1 TPC-VMS Goals

TPC-VMS answers the basic customer question “I have a number of older databases systems, can I consolidate the database systems onto one new server”? TPC-VMS specifically addresses the consolidation issue by requiring multiple database virtual environments to be run on a single server. During the development of TPC-VMS, the overriding criterion for TPC-VMS was “time to benchmark” as the need for a database virtualization benchmark was deemed to be critical by the TPC members. Thus there was a conscious decision to keep the benchmark simple.

3.2 TPC-VMS Run Rules

The TPC-VMS Specification leverages the existing TPC Benchmarks by using the existing workloads specified by the TPC-C, TPC-E, TPC-H, and TPC-DS Specifications. Unless otherwise stated in the TPC-VMS Specification, the test sponsor must follow all requirements of the base TPC Benchmark Specification. TPC-VMS requires 3 identical TPC Benchmarks to be run in a virtualized environment, i.e. 3 TPC-C VMs, 3 TPC-E VMs, 3 TPC-H VMs or 3 TPC-DS VMs. A mixture of different TPC Benchmarks may be a more interesting workload, but was deemed too complex as the fundamental problem of "what is the metric" could not be resolved in a timely manner. The number of VMs was chosen to be 3 as the minimum number of VMs to prove the point but small enough to reduce the complex task of performing a full TPC audit for each VM. The TPC-VMS Specification includes a number of "ease of benchmarking" rules that can potentially reduce the work involved, but a full TPC audit is a non-trivial amount of work.

3.3 TPC-VMS VSUT

A TPC Benchmark typically defines a SUT or System Under Test, i.e. the hardware and software that is to be tested. For TPC-VMS the VSUT or VMS System Under Test is defined as a superset of the base TPC Benchmark SUT. The VSUT includes a Consolidated Database Server that supports the virtualization environment where the three VMs are run. To prevent a test sponsor from taking advantage of the limited number of VMs, the Consolidated Database Server is required to support a large number of VMs. Thus all I/O must be virtualized by either the hypervisor or via the I/O controllers managing the I/O devices. Without this requirement a test sponsor could simply partition the I/O by assigning an I/O device exclusively to each VM.

3.4 TPC-VMS Metric

The TPC-VMS performance metric is the minimum value of the three performance metrics for the three TPC Benchmarks run on the VSUT. The TPC-VMS performance metric is reported by prefixing a "VMS" to the TPC Benchmark performance metric, e.g. VMStpmC, VMStpsE, VMSQphDS@ScaleFactor or VMSQphH@ScaleFactor. The minimum of the three VMs was chosen as a simple means to ensure that all 3 VMs are using an equal amount of VSUT resources, i.e. the test sponsor is not gaming the results by running an asymmetric VM that ensures a higher result. The test sponsor must aim to maximize the minimum result of the 3 VMs. In order to do this the test sponsor must try to ensure that each VM uses an equal amount of VSUT resources. A secondary reason was to ensure that a TPC-VMS result would not be comparable to an underlying TPC Benchmark result, e.g. a TPC-VMS TPC-C VMStpmC result is not comparable to a TPC-C tpmC result.

4 TPC-VMS Prototype Effort

As part of the TPC development process, prototype data is generated to ensure the viability of a benchmark. The prototype results are presented to the TPC Development Subcommittee for review. The data is scrubbed of any product specific information as TPC membership is a consortium of companies that are competitors. Described here is the prototype work implemented by Intel. The prototyping effort included both the TPC-E and TPC-H benchmarks. The test configurations were similar as were the findings regarding database performance in a virtualized environment. For simplicity, only the TPC-E prototyping effort is described in this paper.

At first glance the TPC-VMS benchmark appeared to be just a simple run of 3 TPC benchmarks in a virtualized environment. However the prototyping effort uncovered a number of performance issues intrinsic to virtualization of large databases applications. The performance knowledge gained was considered an excellent ROI for Intel's investment in the TPC-VMS benchmark development.

4.1 Prototype Test Configuration

The prototype setup is shown in Figure 1. At the left are the client systems that drive the TPC-E benchmark. The VMS System Under Test (VSUT) in the center is the server that implements the virtual environment and at the right are the storage subsystems.

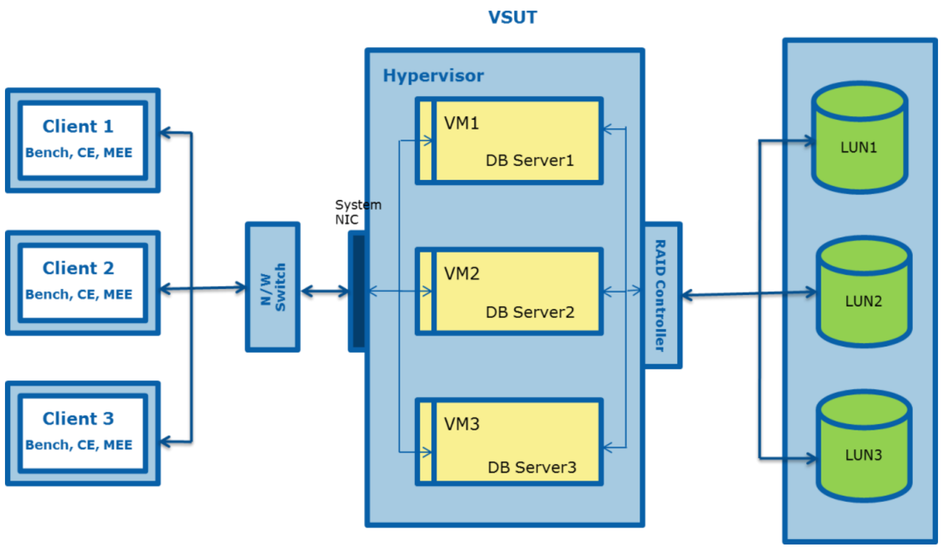

Figure 1 Prototype Setup

The VSUT server is a 2-socket 4-core Intel® Xeon X5570® (Nehalem) system. The Intel Simultaneous Multithreading feature was enabled on the server. In order to collect consistent performance data the Intel Turbo Boost Technology was disabled for all the runs described in this paper. The VSUT contained 64 gigabytes of memory. A hypervisor or in the parlance of TPC-VMS the Virtual Machine Management Software (VMMS) was used to divide the VSUT resources into multiple virtual machines (VMs), each hosting a database (DB) server. The Intel 82574 Ethernet controller in the VSUT server acts as the virtual network switch to the virtual network interface cards (VNIC) configured for each of the VMs. A RS2PI008 Intel RAID controller attaches four Newisys storage bays containing the storage devices.

The thee client machines are Intel Xeon X5570 systems, each of which run a benchmark driver (Bench), a Market Exchange Emulator (MEE) and a Customer Emulator(CE). The server and the clients are connected to each other through a 1GB network switch.

4.2 Prototype Data

As part of the prototype effort, we ran five scenarios as can be seen in Figure 2. In the first scenario one database server ran on the native system. In the second scenario one VM was configured that spanned all physical memory and virtual processors (1VMx16VP). Similarly the third, fourth and fifth scenarios have 2 VMs with 8 Virtual Processors each (2VMx8VP), 3 VMS with 6 Virtual Processors each (3VM*x6VP) and 4 VMs with 4 Virtual Processors (4VMx4VP). The VMs in each scenario are allotted an equal share of the memory, virtual processor resources and persistent storage. From the TPC-VMS benchmark perspective, only the 3VM case is of interest. In this section we discuss all the scenarios to develop a better understanding of the benchmark. Note that all the results considered here are in-spec as per the TPC-E Specification.

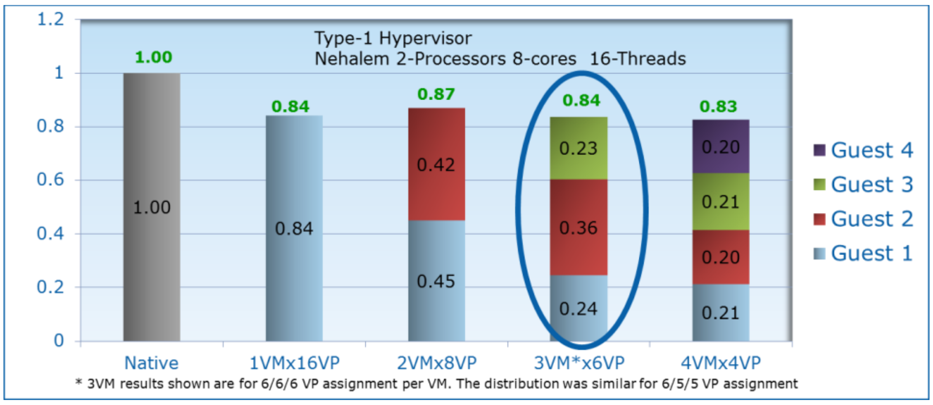

Figure 2 Relative VM vs. Native Performance

The data values in Figure 2 indicate the relative throughput results (TPC-E transactions per second or tpsE) of the benchmark on each VM in comparison to Native result. The first scenario, with the database running on the Native system, gave the highest performance result, denoted by the 1.00 value. For the second scenario (one VM, 16 virtual processors and all remaining memory), throughput dropped by 16% when compared to the Native throughput (1 – 0.84). The reasons for the performance degradation are discussed in Section 5.

As for the remaining cases, the 2VMx8VP scenario showed the best result of all runs in a virtualized environment. Note that for scenarios with an even number of VMs running on this 2-socket system (i.e., 2VMx8VPs and 4VMx4VPs scenario) throughput reported by each VM is almost identical to the other VMs in the system. However in the case of an odd number of VMs, (3VMx6VP scenario), we notice a high variation of throughput reported by each of the VMs. The second VM reported the highest relative throughput of the 3 VMs (0.36 of the Native), with the 1st and 3rd VMs reporting almost similar relative performance, 0.23 and 0.24 respectively. As described in 3.4, the TPC-VMS benchmark requires the test sponsor to report the lowest throughput of the 3 VMs, 0.23 in this case. As noted in Figure 2, we conducted the 3 VM experiments with both over subscription - 6/6/6 VPs over 16 logical processors and equal subscription - 6/5/5/ VPs over 16 logical processors. Similar results were obtained in both the cases. The reasons for the asymmetrical performance are discussed in Section 5.

5 Performance Issues

The majority of the virtualization benchmarks existing in the industry run VMs with a small memory footprint and virtual processors that only utilize a fraction of a single processor core. The performance penalty of the virtualization layer was generally observed to be 3-4%. However for large database applications, the memory footprint is larger, memory accesses are more randomized and the virtual processors consume all of the compute resources of a processor core. TPC-VMS uncovered a few performance issues for large database applications running in a virtualized environment.

5.1 Memory Address Translation Overhead

In a virtualized environment, the hypervisor is the control system that runs on the host and abstracts the system's physical resources (processor, memory and I/O) to the guest operating systems running in the virtual machines while retaining full control of the platform. Each guest OS thinks it is running on its own individual machine. Virtualization of the memory is slightly more complicated than in a native environment. The hypervisor must handle three address spaces:

- Host physical address (HPA): Physical address space of the physical system managed by the hypervisor
- Guest physical address (GPA): The physical address space of the guest OS in the VM
- Guest linear address (GLA): The virtual address space of the applications running under the control of the guest OS

The hypervisor manages two levels of address translation. First is the translation from GLA space to GPA space. And second is the translation from GPA space to HPA space. To accelerate the address translation, hypervisors implement in software shadow page tables that combine the two levels of address translation. When a guest OS modifies a TLB that specifies the GLA to GPA translation, the hypervisor must substitute the shadow page HPA into the TLB thus ensuring the GLA to HPA translation. The shadow page table HPA substitution occurs on a page fault. Having the hypervisor code handle each and every page fault can be a major performance issue for applications that randomly access large amounts of memory.

With the Intel VT-x/i technology [4], the above mentioned memory translation overhead is significantly reduced using hardware assisted memory management Extended Page Tables (EPT). When EPT is in use, memory addresses that would normally be treated as HPA without virtualization are now treated as GPA addresses. These GPA addresses are translated to HPA addresses in turn by traversing a set of EPT paging structures as shown in Figure 3 [5].

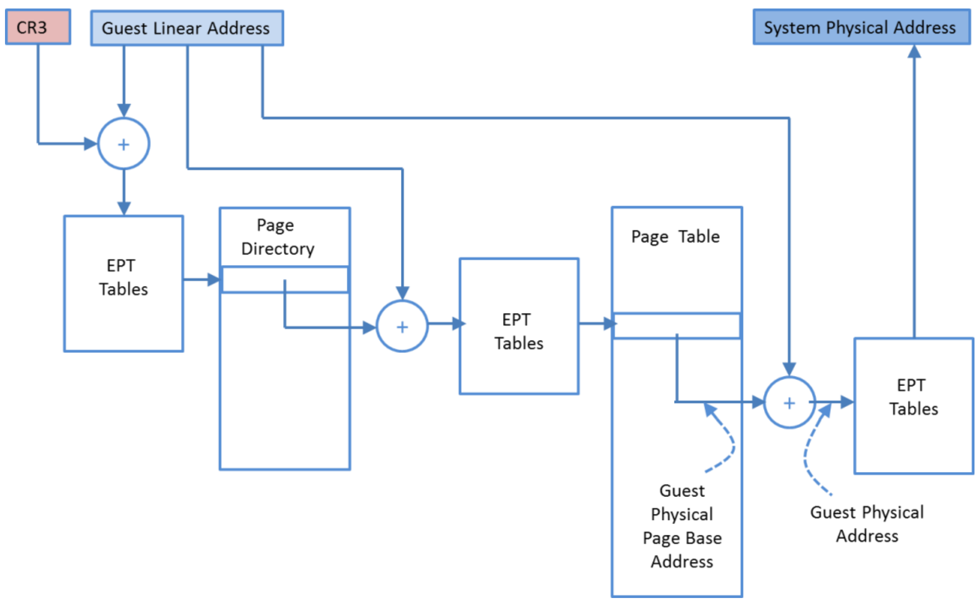

Figure 3 EPT page table walk

EPT is optionally activated on VM entry and deactivated on VM exit. The advantages of EPT are multifold. EPT handles memory management for VMs and hence avoids the need for the hypervisor to sync the shadow page tables. Previously shadow page tables had to be maintained for each guest process for every guest. Now only one EPT is required per VM. Hence the benefit of EPT scales as the number of VMs increases. However multiple EPT tables are traversed during each address translation, thus overhead due to a TLB miss is increased. Also, the increase in the paging structures will cause an incrementing stress to the caching system, i.e. additional cache misses.

For the Intel Xeon X5570 processor the EPT translation increases TLB miss latency by 1.9X. In the Native scenario we observed 3.6 TLB (both instruction and data) misses per 1,000 instructions executed. In the 1VMx16VP scenario we observed 4.2 TLB misses per 1,000 instructions executed. Not only is the TLB miss latency longer due to the EPT translation, but the overall TLB miss rate increases due to the additional pressure on the cache caused by the EPT paging structures, hypervisor code and hypervisor data. This in turn causes additional instruction and data cache misses for the overall application.

5.2 Database Memory Allocation

When we analyzed TPC-E performance in 1, 2, 3 and 4 VM environments, we observed that as VMs were added the total free memory increased. As VMs are added the amount of memory available to each VM should be the total memory minus any hypervisor memory divided by the number of VMs. In each VM, memory is consumed by the OS, any background applications and the database. The database was configured to pre-allocate all available memory using 2 megabyte large pages in order to minimize memory address translation overhead (Section 5.1). The database memory was pre-allocated to reserve a small amount of memory for the guest OS to allocate to any new applications. We observed that as VMs were added, the sum of all of the free memory in the VMs was increasing with the number of VMs. For instance, for 3 VMS nearly 10 gigabytes cumulative memory was left unused in the system that contained 64 gigabytes of memory.

An in-house tool was used successfully in all VM scenarios to acquire almost all the VM's memory. Thus the virtualization environment was ruled out as cause of the ever increasing free memory. The issue was eventually traced to a problem with the database memory allocation algorithm. The database software allocation algorithm in each VM was leaving memory free culminating in an ever increasing total amount of free memory as the number of VMs increased. The issue was overlooked in the Native scenario but easily identified in the virtualization scenarios. The issue has been reported to the database software vendor.

5.3 NUMA Memory Latency

In a multi socket system, memory is distributed among processors such that latency of a memory access is determined by the memory location relative to a processor. Memory local to a processor is accessed faster by the processor than memory local to another processor also called remote memory and hence the term Non Uniform Memory Access (NUMA). For Nehalem processors latency due to a remote memory access is roughly 1.7 times the latency for a local memory access.

The percentage of local memory accesses versus all memory access is defined as the NUMA locality for an application. For the Native result in Figure 2 the NUMA locality is 80%, indicating that on average 80% of the processor memory accesses were local. Each processor is accessing its local memory 80% of the time while 20% of the memory accesses are remote. For the 1 VM result in Figure 2, the NUMA locality is 48.5%, indicating that 51.5% of the memory accesses are remote. Thus the memory access in the 1VMx16VP case will on average be significantly slower than in the Native case.

For the Native case, the high percentage of NUMA locality is due to man-years of effort to optimize the database software for NUMA, i.e. ensure the software is NUMA aware. The same effort must now be applied to the hypervisor and VMs to ensure that they are NUMA aware in regards to the placement of virtual processors and virtual memory. For small VMs that do not span multiple processors the issue is easily resolved. However, latency due to poor NUMA locality is more conspicuous in cases where a VM spans multiple sockets. The VM must be NUMA aware to ensure a Virtual Processor's memory is allocated locally to the processor executing the VP. The hypervisor must now consider NUMA as a factor to the VP scheduling policy such that a VP is not migrated from a processor that has high NUMA locality to a remote processor that will have poor NUMA locality.

5.4 Asymmetrical 3 VM Results

The experimental setup described in this paper contains 2 nodes (i.e., processor sockets) with 4 cores each. Intel Hyper Threading technology feature was turned on and hence we had 16 logical processors (LP) in total. For the 3VMx6VP scenario in Figure 2, each VM is assigned 6 virtual processors. When the first VM is started, it is scheduled to be run on the first processor acquiring 6 out of 8 LPs on the processor. Processor 0 is now the home node of the first VM. When the second VM is started, it is scheduled to be run on processor 1 acquiring 6 LPs on the processor. Now that we have only 4 LPs left unallocated in the system, starting a third VM with 6 VPs can cause resource contention issues. The third VM is scheduled by the hypervisor to be run on the processor 0 following a round robin policy. Since processor 0 is the home node for both VM1 and VM3 and only VM2 runs on processor 1, processor 0 is oversubscribed and processor 1 is undersubscribed. Contention for processor 0 cycles will cause the hypervisor to migrate VPs of VM1 and VM3 to processor 1.

A problem arises due to the hypervisor optimization to move all VPs to their home node or processor. For small VMs where the VPs consume a fraction of a processor core, this optimization is valid to ensure the highest performance as the VPs will benefit from shared cache and local memory accesses. For large VMs that span multiple processors the optimization to move all VPs to their home node introduces a performance degradation as the VPs are continuously consolidated onto the home node and then moved off the home node to balance processor utilization. In the case of the 2VMx8VP versus the 3VMx6VP scenarios in Figure 2, there is a fivefold increase in the number of VP migrations per second. The high number of VP migrations also impact NUMA locality (Section 5.3) as the migrated VP's memory accesses continually change from local to remote. Compared to the 2VMx8VP and 4VMx4VP scenarios, the 3VMx6VP scenario reported 10% more remote accesses.

For the 3VMx6VP case, only VM2 is scheduled to be run on processor 1. Since VM2 runs undisturbed, it generates the highest relative performance (the 0.36 shown in Figure 2). However since VM1 and VM3 are both scheduled on processor 0, they compete for resources. The processor itself is oversubscribed when all VPs of VM3 are migrated to their home node. Hence VM1 and VM2 generate relatively lower and similar relative performance results (0.23 and 0.24 respectively) causing overall asymmetric results.

Migrating VPs to their home node is a valid performance optimization for small VMs. However for large VMs that spans multiple processors migrating to the home node can cause performance issues. The hypervisor scheduling algorithms should be changed to identify the high VP migration and back off on the continual migration of the VPs to their home node.

6 Summary

The TPC-VMS benchmark leverages the TPC-C, TPC-E, TPC-H, and TPC-DS benchmarks to provide a measure of database performance in a virtualized environment. TPC-VMS requires 3 identical TPC Benchmarks to be run in a virtualized environment, i.e. 3 TPC-C VMs, 3 TPC-E VMs, 3 TPC-H VMs or 3 TPC-DS VMs. At first glance TPC-VMS may be considered a simple benchmark; however the prototyping effort uncovered a number of performance issues. The underlying problem is that virtualization environments have been optimized for VMs with small memory footprints and VPs that consume a fraction of a processor core. Database applications require VMs that have a large memory footprint, access memory randomly and VPs that consume an entire processor core.

Acknowledgements

The TPC-VMS benchmark is a collaborative of many individuals from several leading companies in the computer industry. The material presented here is the result of the work of the committee members as a whole rather than two authors. We would like to acknowledge the contributions of Dave Raddatz, John Fowler, Karl Huppler, Jamie Reding, Rick Freeman, Andy Bond, Reza Taheri, Charles Levine and Vish Viswanathan.

References

1. SPEC Virtualization Committee, http://www.spec.org/virt_sc2010
2. VMware Inc., VMmark: A Scalable Benchmark for Virtualized Systems, http://www.vmware.com/pdf/vmmark_intro.pdf
3. TPC-VMS Specification, http://www.tpc.org/tpcvms/spec/tpc-vms-v1_1_0.pdf
4. Intel Software Development Manual 2013, http://download.intel.com/products/processor/manual/325462.pdf
5. Enabling Intel Virtualization Technology Features and Benefits, http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/virtualization-enabling-intel-virtualization-technology-features-and-benefits-paper.pdf