VMware vFabric Postgres 9.2

Performance and Best Practices

TECHNICAL WHITEPAPER

vmware®

Table of Contents

Introduction

VMware vFabric® Postgres (vPostgres) is a distribution of the open-source PostgreSQL [1] along with companion drivers and utilities chosen by VMware. It is supported by VMware [2].

This white paper presents the performance of vPostgres 9.2 on VMware vSphere® 5.1. We show vPostgres 9.2 in a 32-vCPU virtual machine (VM) on vSphere and a 32-core physical machine achieves the same vertical scalability as in an equally-configured native setup. Using out-of-the-box settings for both vPostgres and vSphere, we show a VM-based database consolidation approach performs on par with alternatives when memory is undercommitted and increasingly better when memory overcommitment escalates. We also show the vPostgres database memory balloon technique can help better preserve performance under memory overcommitment. We last summarize key best practices for running vPostgres on vSphere.

Vertical Scalability

Database applications benefit from the full performance capacity from each server. Meanwhile server hardware keeps improving. For example, each generation has more CPU cores and memory. Thus making database server scale up is practical and useful. Running applications on virtualization platforms offers compelling operational capability and efficiency.

vSphere is a proven platform for running many performance-critical enterprise applications, including database applications. Our study shows vPostgres on vSphere can scale up its performance as equally well as on native setup.

Consolidation

Consolidating multiple databases onto one physical machine is economical. Using virtual machines to contain the databases enables strong isolation and unique operational capabilities.

The performance needs of databases often exhibit a time-multiplexed pattern collectively. This common workload behavior presents real consolidation opportunities: achieving higher resource utilization without jeopardizing performance.

Our study shows when consolidating multiple vPostgres databases on vSphere consolidating on dedicated VMs performs better and more robust than on alternative containers.

PostgreSQL uses operating system buffer cache extensively for I/O caching and recommends configuring a moderate amount of memory (e.g., 25% of total memory [3]) for database shared buffers. This makes VMs well-suited to consolidate PostgreSQL databases out of the box: Because most of the memory in these VMs is managed by the guest operating system, the guest kernel-level memory balloon driver traditionally employed by vSphere VMs can effectively redistribute memory among the VMs without disrupting the database server's own memory management when a need for redistribution arises (e.g., when one overcommits memory to attain denser consolidation and wants simple VM provisioning).

The strong isolation between VMs makes it desirable to seek more performance out of a given amount of VM resources. Increasing the size of database server shared buffers can be a simple way to achieve this goal for some common workloads. We show by changing vPostgres server shared buffers from 25% to 75% of VM memory we gain 12-13% more throughput and reduce temporal variation of the throughput by 70-80% for a common database workload. VM-based consolidation approach also perform more robustly than alternatives under memory overcommitment and the vPostgres database balloon technique [4] can improve the robustness even further, particularly in more performance-demanding situations. Our tests show when memory is 55% overcommitted the relative performance advantage of VM-based consolidation over alternatives is 200% out of the box (25% of VM memory is used for vPostgres shared buffers.) If 75% of VM memory is used for the shared buffers, the relative performance advantage of VM-based consolidation under a 55% memory-overcommitted

situation is 60% when using only guest kernel-level balloon and 140% when using both guest kernel-level balloon and vPostgres database balloon.

Vertical Scalability Tests

Scaling up a server (e.g., with more CPUs and more memory) is useful—it reduces the need to purchase and run additional server machines—which translates to both capital and operational savings. Scaling up is also known as vertical scaling. We conduct vertical scalability tests using a vPostgres VM with 32 vCPUs on vSphere and on a physical host with 32 CPU cores. We use two workloads:

- pgbench default: pgbench [5] is a common benchmark used by the Postgres community and maintained in the PostgreSQL source tree. It runs the same sequence of SQL commands repeatedly. With the default configuration there is no delay between SQL commands and the composition of these commands is loosely based on TPC-B [6], involving five SQL commands (one SELECT, three UPDATEs, and one INSERT) per transaction.
- pgbench SELECT-only: This workload uses a non-default configuration of pgbench: only one SELECT is issued per transaction. There is no delay between the SQL commands.

Test Methodology and Settings

Figure 1 shows the test bed for vertical scalability tests exercising the vPostgres server on a single database VM on a vSphere 5.1 host.

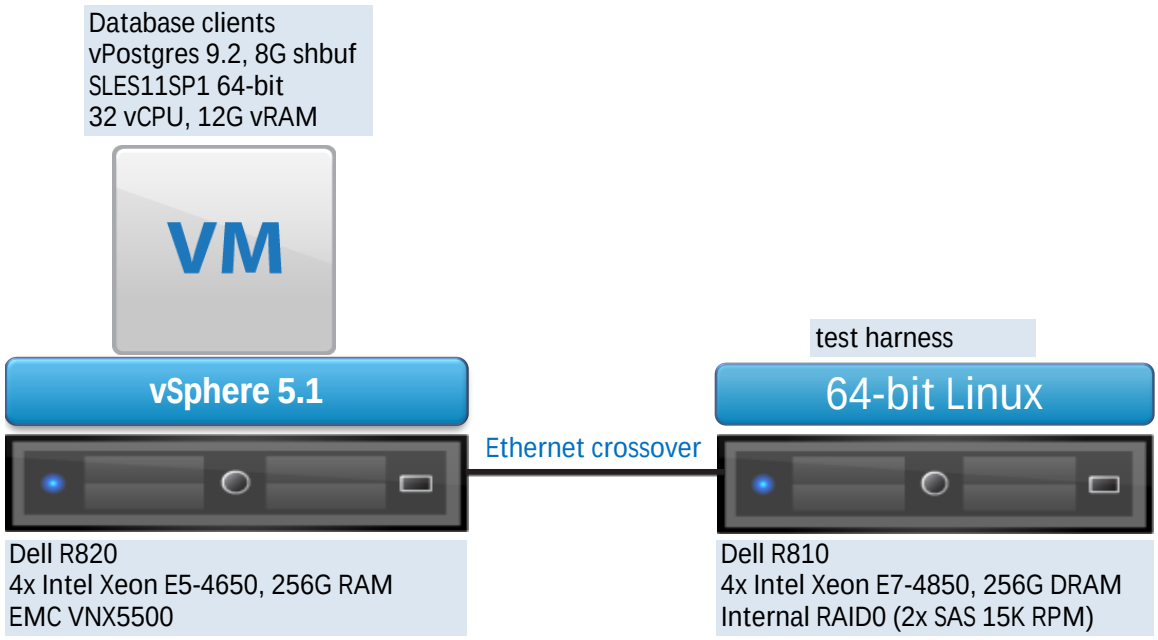

Figure 1 Experimental setup for vertical scalability tests

Because the scalability improvement of vPostgres 9.2 focuses on a pgbench SELECT-only type of workload, we also run vPostgres 9.2 in a comparable native Linux configuration using pgbench SELECT-only, following the same methods previously described. The same software stack (vPostgres 9.2, SLES11SP1) is installed natively on the same host instead of on vSphere. The vPostgres server instance is configured the same way except that the native installation uses all physical memory while the VM is configured with 12GB vRAM. We believe the advantage of the physical setup in this case is negligible, if any, given that the same and large size of database

shared buffers is used in both setups (our results indeed show that.)
Table 1 details the settings for the vertical scalability tests, both the virtual setup and the physical setup.

VIRTUAL SETUP	
Host	Dell R820, 4x Intel Xeon E5-5640 (32 cores in total), 256GB DRAM
Storage	EMC VNX5500 SAN, 3 disk array enclosures (each housing 15 SAS-II drives)
VMFS data store	3 in total, 1 per SAN disk array enclosure (DAE)
Host OS	vSphere 5.1 express patch 2 build 1021289
Virtual Machine	32 vCPUs, 12GB vRAM, 3 virtual disks (1 per VMFS data store)
Guest OS	SUSE Linux Enterprise Server 11 SP1 (SLES11 SP1) 64-bit; vPostgres DATA directory, vPostgres WAL (write-ahead log) directory and the rest (e.g., root file system and swap) mounted on a separate virtual disk
Database server	vPostgres 9.2.3
Workload	Database is populated by pgbench scale factor=100, resulting in an initial database size of about 1.3GB. It is exercised by database clients (simulated by pgbench) running on the same database VM. Each database client uses a separate database connection established on a Unix domain socket.
Tests	A series of database client counts is tested, each constituting a separate test session. Each session lasts 5 minutes. The same series is run three times back-to-back. The median reading of the three runs is reported for each database client count. The database clients drive traffic in a tight loop: no delay is used between transactions and different operations inside a transaction. This input exercises the vPostgres server in earnest.

PHYSICAL SETUP	
Host OS	The same SLES11SP1 as used as the guest OS in the virtual setup
Compute	32 CPU cores, 256GB DRAM
Storage	Three separate raw LUNs from the same three SAN DAEs as in the virtual setup
Others	Database server, workload, and test procedure are the same as those used in the virtual setup.

Table 1 Configurations in the setup for vertical scalability tests

Performance Metrics

The scalability tests are concerned with a high-watermark reading of per-server transaction throughput. Since all clients are driving transactions in earnest, how fast vPostgres server can complete the transactions solely determines the average throughput that can be achieved during the entire test session. We report the throughput in the unit of transactions per second (TPS).

The scalability of a particular vPostgres server can be gauged by the peak and the slope of the throughput curve with respect to the number of database clients: the higher the peak, the steeper the slope, the lengthier the linear scaling region, the better the scalability.

We compare the scalability between vPostgres 9.1 and vPostgres 9.2 using both pgbench SELECT-only and pgbench default workloads.

We compare the scalability between vPostgres 9.2 in the virtual setup and vPostgres 9.2 in the physical setup using pgbench SELECT-only workload.

Scalability Results: vPostgres 9.2 versus vPostgres 9.1

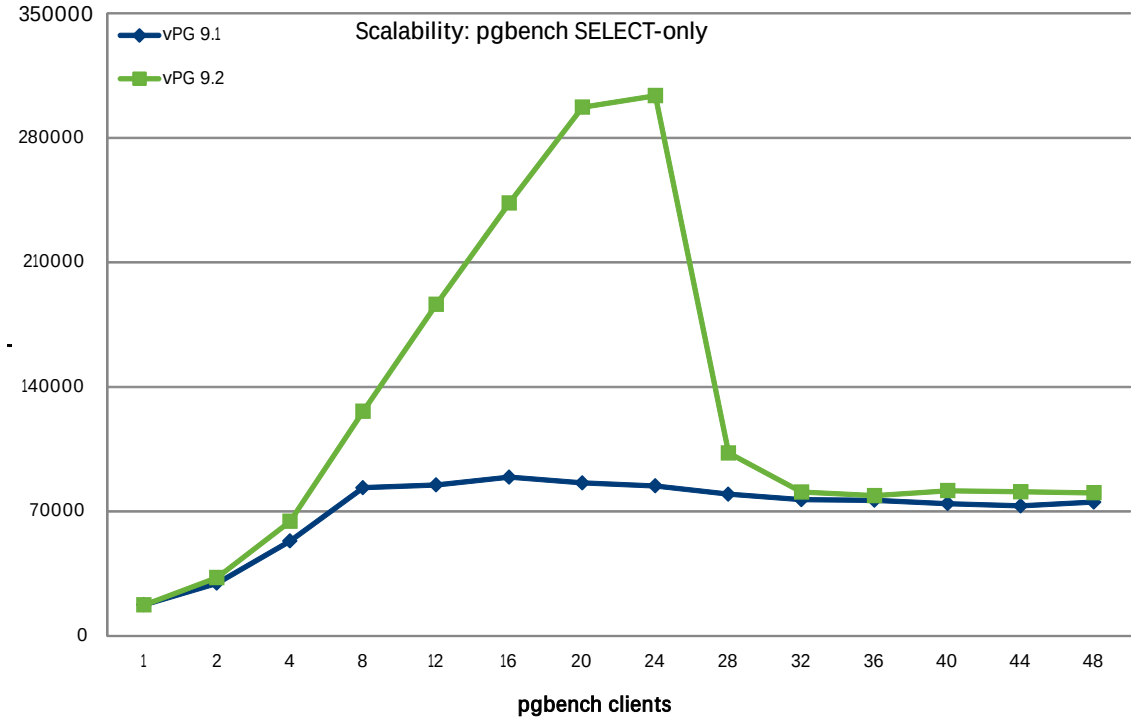

Figure 2 Scalability of pgbench SELECT-only: vPostgres 9.2 versus vPostgres 9.1

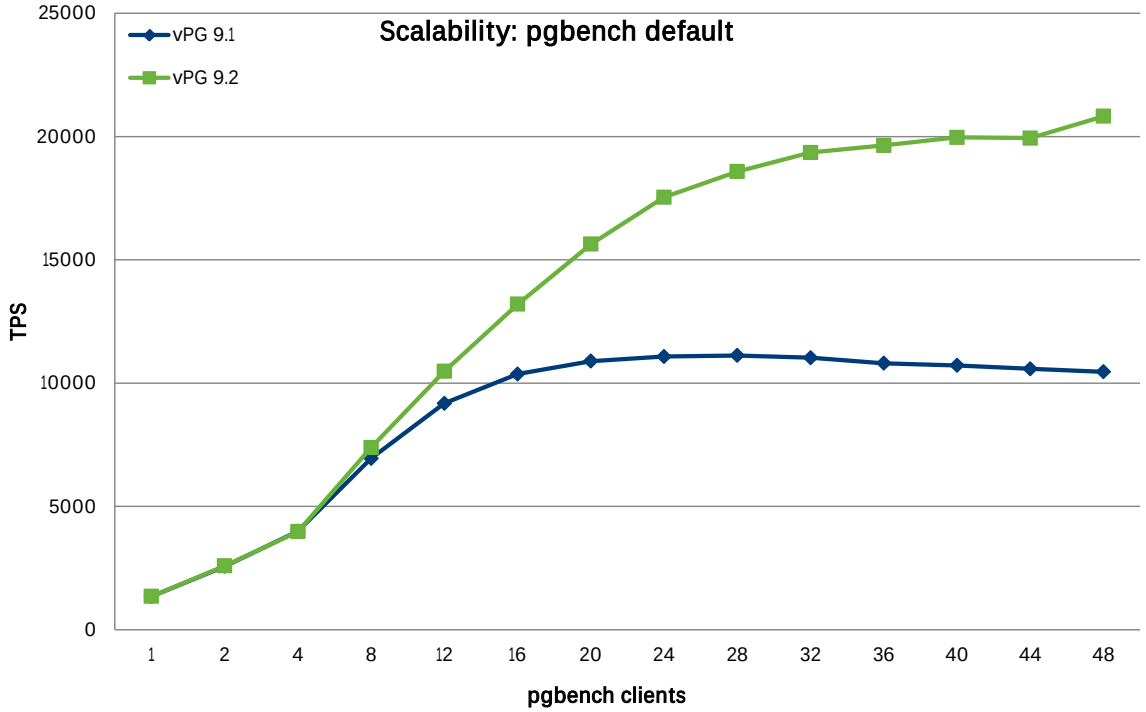

Figure 3 Scalability of pgbench default: vPostgres 9.2 versus vPostgres 9.1

Figure 2 shows the peak throughput of pgbench SELECT-only by vPostgres 9.2 is about four times that of vPostgres 9.1. The slope is steeper, and the region of linear scalability extends from up to 8 clients to up to 20 clients. This suggests that workloads featuring read-heavy and short-duration transactions will benefit greatly from vPostgres 9.2.

Figure 3 shows the scalability of pgbench default. The peak throughput of this benchmark by vPostgres 9.2 improves nearly 100% over that by vPostgres 9.1. The saturation point extends from about 20 clients to beyond 48 clients. Because this benchmark has 60% UPDATE transactions and 20% INSERT transactions in the mix, this suggests that workloads of write-heavy and short-duration transactions will also benefit greatly from vPostgres 9.2.

Scalability Results: Virtual versus Physical

Figure 4 compares the scalability of vPostgres 9.2 running in a VM versus running in a comparable native Linux system as described previously. We can see that vPostgres running in a VM scales on par with running in a native machine and fully benefits from the 9.2 scalability improvement.

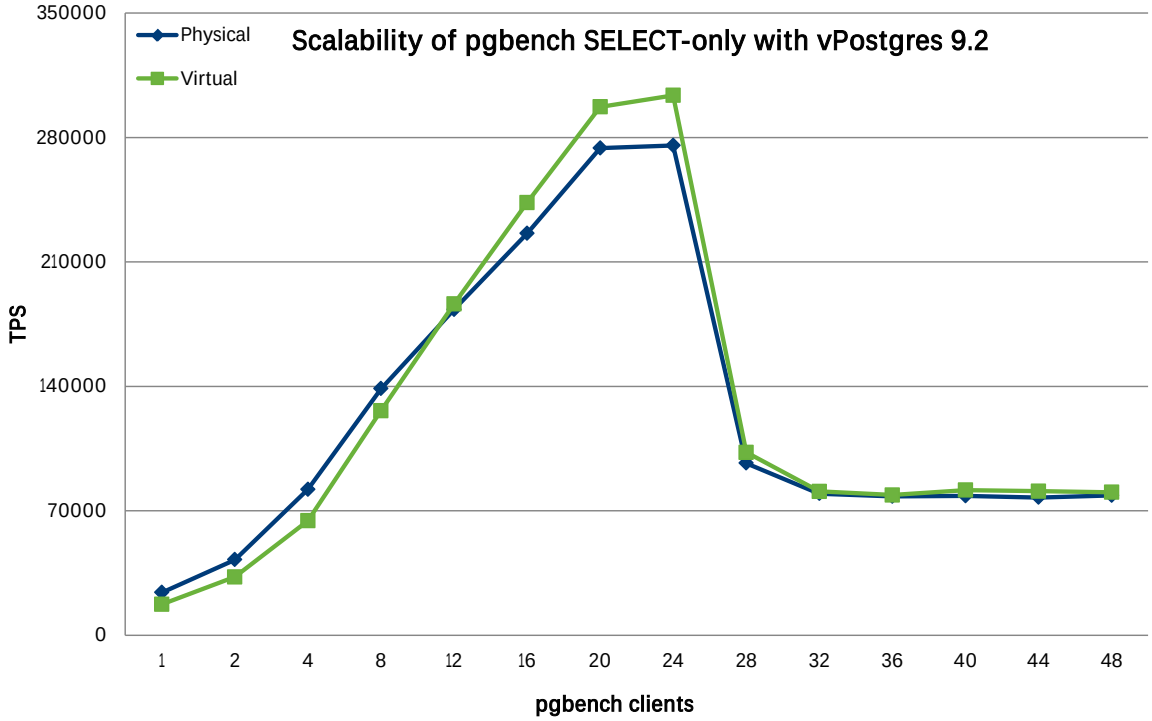

Figure 4. Scalability of vPostgres 9.2 with pgbench select-only: virtual versus physical

Consolidation Performance Tests

Consolidating multiple databases onto one physical server and attaining a higher consolidation rate are useful because it can lead to capital savings (e.g., fewer servers and licenses) and operational savings (e.g., less power, cooling, and maintenance).

There are various approaches for consolidating databases. A critical element in any approach is the choice of database runtime container: a vPostgres server instance can hold multiple databases, an operating system can hold multiple vPostgres server instances, and a hypervisor can hold multiple VMs with each running a separate operating system instance.

From a performance point of view, challenging consolidation situations are those where the resources configured for all collocated VMs exceed the underlying host capacity (this is known as overcommitment), while runtime demands for resources from different VMs come in at different times and the aggregate demand in real time is below the capacity. In such situations, the choices of execution containers and overcommitment techniques can make a difference in redistributing resources from where they are less needed to where they are more needed and in time.

Memory often exhibits a stronger attachment to its immediate "owner" than CPU and I/O resources and thus is often the limiting factor when one aims for a denser consolidation. In studying database consolidation, we prescribe a series of memory overcommitment situations. We quantify a memory overcommitment rate using Equation 1 below.

$$MEM_{config} = The\ sum\ of\ vRAMs\ configured\ for\ database\ VMs$$

$$MEM_{avail} = The\ amount\ of\ host\ memory\ made\ available\ to\ database\ VMs$$

$$Memory\ overcommitment\ rate = 100\% - MEM_{avail}/MEM_{config}$$

Equation 1. Definition of memory overcommitment rate (for database VMs)

We send time-varying and negatively-correlated traffic streams to database VMs and make sure the memory demands by these VMs don't exceed MEM_{avail}. We acquire quantitative information of these memory demands experimentally in advance.

We use the DBT-2 benchmark [7] to populate and drive loads to the databases. DBT-2 is an online transaction processing (OLTP) performance benchmark implementing TPC-C [8]. It models a wholesale parts supplier where multiple clients access its production database, conducting five types of transactions including placing new orders, processing payments, processing order delivery, as well as checking order status and stock level. The workload generates a mixture of SELECT- and UPDATE-intensive transactions. Using DBT-2, we can control the memory demands during any period on any database by adjusting the number of database clients transacting against the database during the period.

We create memory overcommitment situations by co-locating a dummy VM and adjusting its vRAM configuration with a full memory reservation (thus the dummy VM will not be involved in memory redistribution) to attain a prescribed memory overcommitment rate.

Test Methodology and Settings

Figure 5 shows the test bed for consolidation tests. The settings are detailed in Table 2.

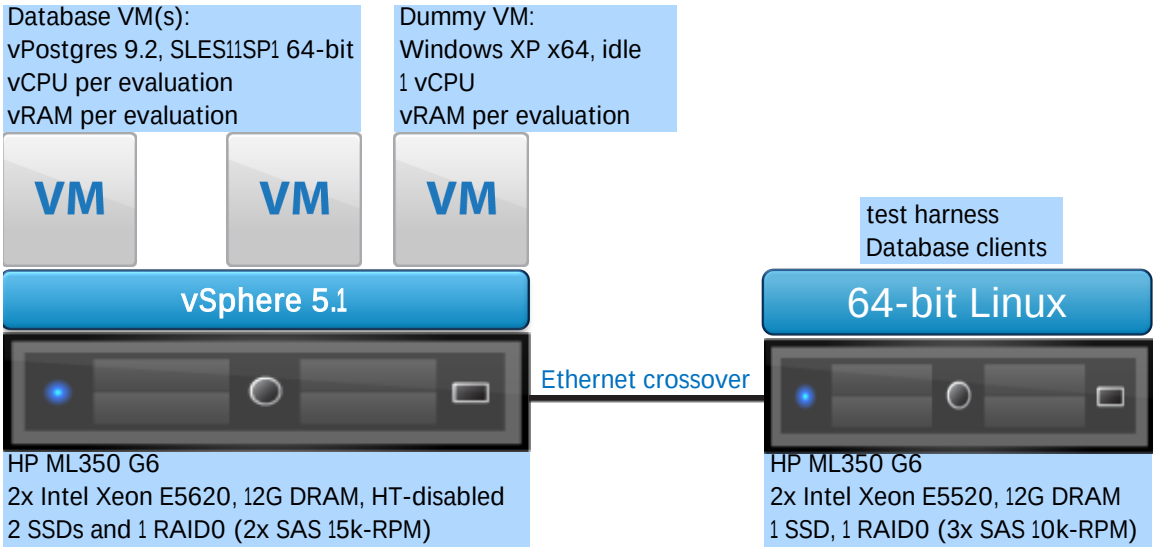

Figure 5. Experimental setup for database consolidation performance tests

Host	HP ML350 G6, 2x E5620 (8 cores in total), 12GB DRAM
Storage	2 Intel 520 SSDs, 1 RAID0 by HP P410i controller comprising 2 SAS-II 15k-rpm drives
VMFS data store	3 total, out of 2 SSDs and 1 RAID0 volume
Host OS	vSphere 5.1 express patch 2 build 1021289
Database VMs	3 vCPUs, vRAM per evaluation, 3 virtual disk files (1 per VMFS data store)
Guest OS	SUSE Linux Enterprise Server 11 SP1 (SLES11 SP1) 64-bit; vPostgres DATA directory, vPostgres WAL directory and the rest (e.g., root file system and swap) each mounted on a virtual disk backed by SSD, SSD, and RAID volume, respectively.
Database server	vPostgres 9.2.3
Workload	Two databases are consolidated (referred to as DB_1 and DB_2). Each is populated by DBT-2 warehouse=30, resulting in an initial database size of about 3GB each. They are exercised by database clients (simulated by DBT-2) running on a separate physical box. Each database client uses a separate database connection.
Tests	Against each database we send a traffic stream comprising four periods alternating between high- and low-traffic, rendered by 30 and 2 DBT-2 clients respectively (with zero think time). The two streams are complementary: during any period when DB_1 is under high traffic DB_2 is under low traffic and vice versa. Each period lasts 10 minutes for measurement and12-40 seconds for ramp-up. Figure 6 is a visual depiction of this.
Dummy VM	1 vCPU, vRAM per evaluation, Windows XP x64 as guest operating system. No active workload runs inside.

Table 2 Configurations in the setup for consolidation performance tests

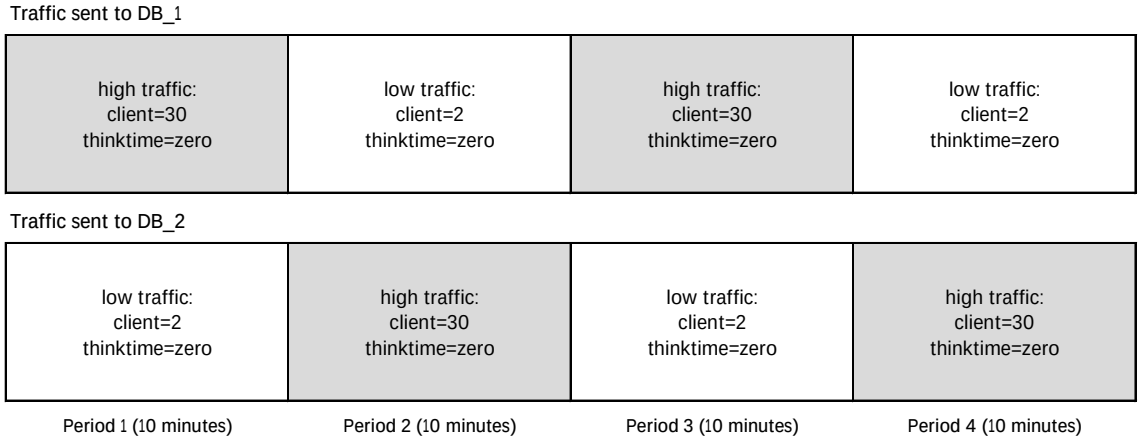

Figure 6 Streams of traffic sent to the two target databases (DB_1 and DB_2)

We compare four different consolidation schemes involving three types of containers and the vPostgres database memory balloon [4]. These schemes are:

- Consolidation scheme 1: Use one vPostgres server instance to host multiple databases. This is a **1/1/M** deployment model (1 VM, 1 vPostgres server instance, and multiple databases).
- Consolidation scheme 2: Use one Linux to run multiple vPostgres server instances each holding one database. This is a **1/M/M** deployment model (1 VM, multiple vPostgres server instances, multiple databases, and one database per server instance).
- Consolidation scheme 3: Use one VM to ultimately hold one database. This is an **M/M/M** deployment model (multiple VMs, multiple vPostgres server instances, multiple databases, and one database per server instance per VM).
- Consolidation scheme 4: This scheme uses the **M/M/M** mode and in addition the vPostgres database memory balloon [4].

In all schemes, the guest kernel-level balloon is used in all database VMs. By design, the vPostgres database balloon coexists and cooperates with the kernel-level balloon in scheme 4.

Performance Metrics

The performance of a database consolidation scheme is gauged by DBT-2 transaction throughput attained on both databases during all periods. We summarize the performance of each scheme by one number: the sum of DBT-2 throughput attained on the two databases, averaged across the four periods. We report DBT-2 throughput in the unit of new order transactions per minute (NOTPM) per TPC-C convention [8].

To establish a baseline, we measure the performance of these consolidation schemes under a common memory undercommitment situation.

For each consolidation scheme, we report its performance under various memory overcommitment situations relative to its baseline. The performance preservation trend under a series of progressive memory overcommitment rates characterizes the scheme's ability of capitalizing on the consolidation opportunities.

We also report on the following system metrics to provide additional perspective:

- Physical CPU usage: This is to provide a CPU cost perspective to the performance comparison between the consolidation schemes.
- vSphere and vPostgres memory statistics: We report memory ballooned by the guest kernel-level balloon drivers as well as by the vPostgres database balloon drivers in scheme 4, memory compressed by the hypervisor, and memory swapped by the hypervisor. This is to correlate with the end-to-end performance of DBT-2 throughput and to gauge how effectively the ballooning techniques redistribute memory in various database consolidation schemes to reduce the extent of more drastic hypervisor memory reclamation measures.

Memory Working Set Sizes

In order to make sure the memory overcommitment situations under study still present real consolidation opportunities, we measure the memory working set sizes (that is, the memory demands) experimentally and use this information when setting memory overcommitment rates for the consolidation performance study.

We are concerned with two separate areas of memory management: vPostgres server shared buffers and the entire VM memory managed by the guest operating system. Thus we are interested in two kinds of working set sizes: (1) working set size of the database shared buffers and (2) working set size of the VM memory all treated by the guest operating system in no discretion.

We arrive at both working set sizes by exercising a target database (sized with 30 warehouses under DBT-2) on a VM with a series of deliberately sized VM memory and vPostgres shared buffers. We conduct the sizing experiments for both traffic levels we use in consolidation performance study (that is, 30 DBT-2 clients and 2 DBT-2 clients).

We first determine the VM memory working set size: We keep the vPostgres shared buffers constant at 1GB and exercise a range of VM memory sizes. We measure DBT-2 throughput. In the resulting performance curve, the memory size reading at the knee point provides the working set size.

We then determine the working set size of the vPostgres shared buffers. We keep the difference between the VM memory size and the size of vPostgres shared buffers constant (at the value of the non-discretionary VM memory working set size determined previously subtracting 1GB). We exercise a range of vPostgres shared buffers and VM memory sizes. We measure DBT-2 throughput. In the resulting performance curve, the shared buffers size reading at the knee point provides the working set size.

Figure 7 and Figure 8 are the performance memory curves resulting from running 30 DBT-2 clients. From these figures, we read 1376MB for the non-discretionary VM memory working set size and 984MB for the shared buffers working set size. Note the performance dips around the VM memory size of 2048MB: We believe it results from the fact that PostgreSQL using operating system buffer cache extensively and thus sizing shared buffers at about half the total memory (such as in these cases) exacerbates a double-buffering phenomenon where one pays extra buffering overhead for little performance gain.

Figure 9 and Figure 10 are the performance memory curves under 2 DBT-2 clients. In this case, the traffic is light to the extent that the 1GB vPostgres shared buffers cache almost all I/O. As a result, we read no performance degradation in Figure 9 where the VM memory is sized all the way down to 1GB. We add a small safety margin and determine 1152MB to be the VM memory working set size. From Figure 10 we read 524MB for the shared buffers working set size.

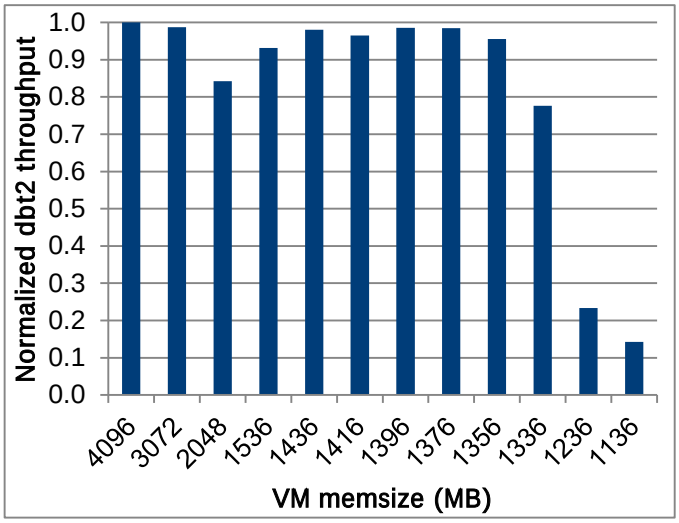

Figure 7. Normalized DBT-2 performance (30 warehouses and 30 clients) under various VM memory sizes while keeping the shared buffers constant at 1GB

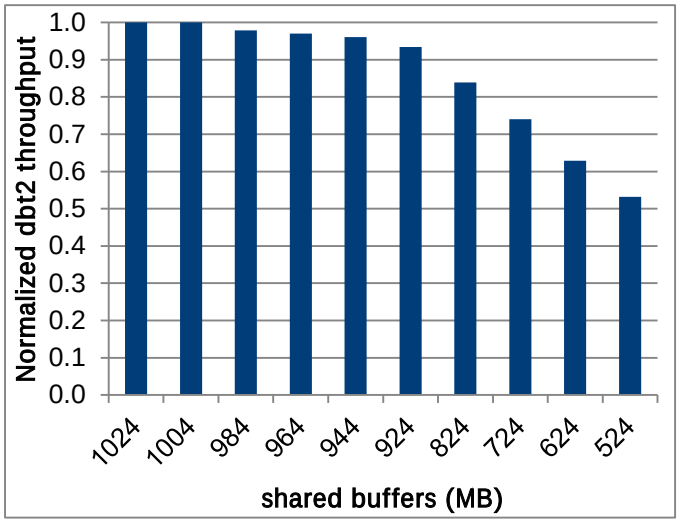

Figure 8. Normalized DBT-2 performance (30 warehouses and 30 clients) under various VM memory and shared buffers while keeping the difference between them constant at 352MB

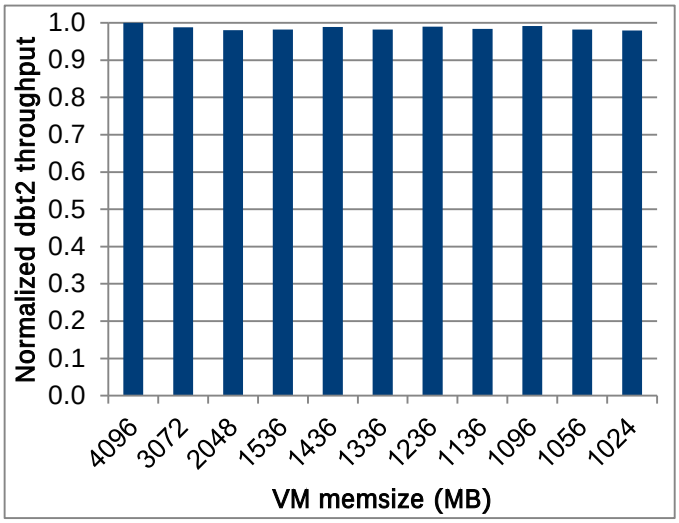

Figure 9. Normalized DBT-2 performance (30 warehouses and 2 clients) under various VM memory sizes while keeping shared buffers size constant at 1GB

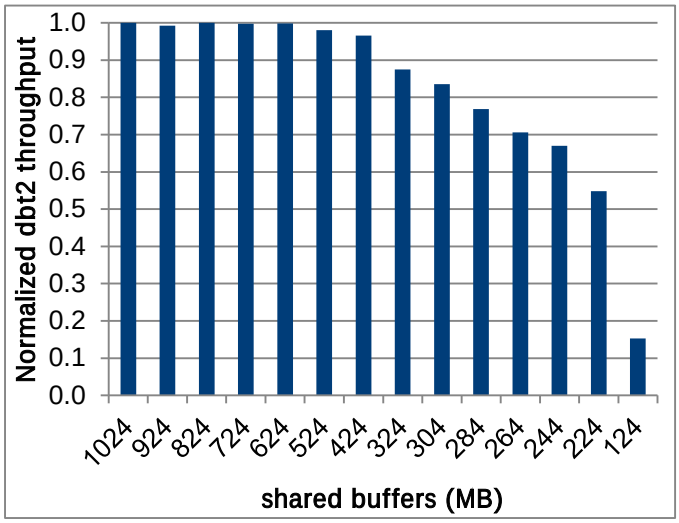

Figure 10. Normalized DBT-2 performance (30 warehouses and 2 clients) under various VM memory and shared buffers while keeping the difference between them constant at 0

In reading these working set sizes, we err on being too cautious; we make sure under all the memory overcommitment situations there is still enough host memory available to the database VMs to accommodate memory demands. We calculate that there is 5664MB memory to spare from database VMs of 8192MB total configured memory with all VM working set sizes still covered. That is, situations with memory overcommitment

rates up to 69% are still real consolidation opportunities without necessarily taking underused memory from vPostgres shared buffers to redistribute to other databases.

Consolidation Results: Baseline

We use performance in a common memory undercommitment situation (by configuring the dummy VM with 1GB vRAM) as the baseline. Figure 11 compares the baseline performance of the four consolidation schemes. Table 3 details the settings of these schemes.

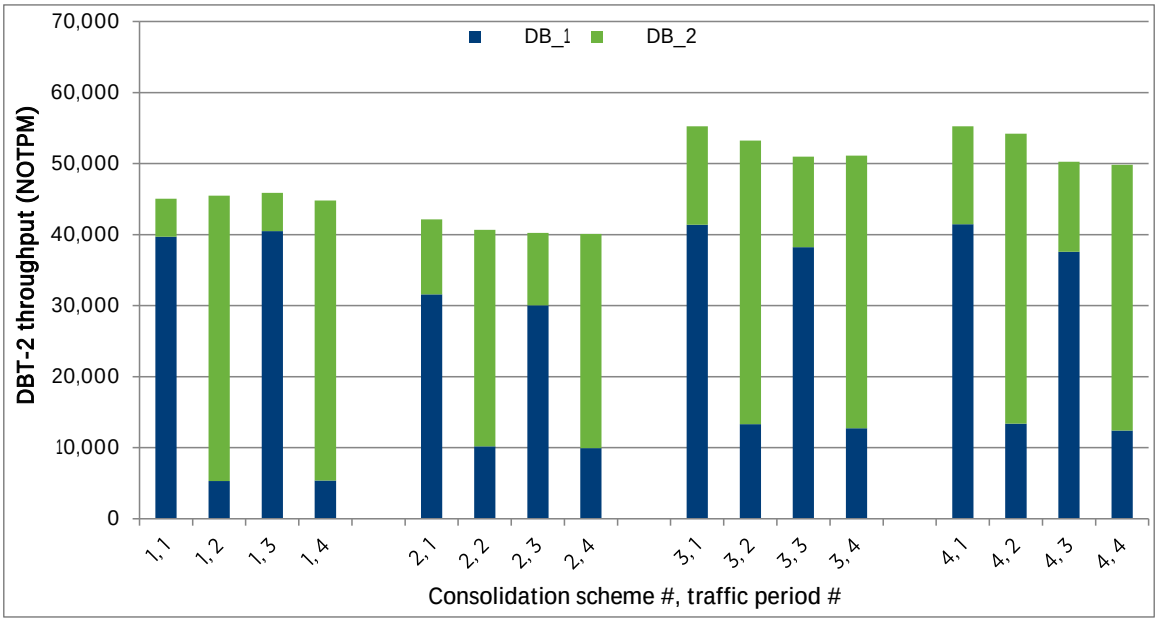

Figure 11. Performance in a memory undercommitment situation: configured per Table 3 and exercised by the traffic detailed in Figure 6. The average throughput attained on the two target databases during each traffic period are reported and stacked on top of each other.

	SCHEME 1	SCHEME 2	SCHEME 3	SCHEME 4
# database VMs, # vPostgres servers per VM, # of databases	1, 1, 2	1, 2, 2	2, 2, 2	2, 2, 2
# of vCPU per VM	3	3	3	3
vRAM per VM	8GB	8GB	4GB	4GB
shared buffers per vPostgres server	2GB	1GB	1GB	1GB
Use guest balloon	Yes	Yes	Yes	Yes
Use vPostgres balloon	No	No	No	Yes

Table 3. Settings of four database consolidation schemes under study

We weigh the following factors for an apples-to-apples performance comparison:

- Schemes 3 and 4 have three vCPUs provisioned for each of two VMs while schemes 1 and 2 have three vCPUs provisioned for one VM. Thus schemes 1 and 2 are disadvantaged in this regard.
- During any period of time one of the two databases is only lightly exercised. Thus the extra three vCPUs provisioned in schemes 3 and 4 will be lightly used per test design.

- The 4GB vRAM configured for each of the two VMs in schemes 3 and 4 is strongly isolated, in contrast to the 8GB vRAM configured for the one VM in schemes 1 and 2. Thus the busier database gains extra performance by taking advantage of the larger pool of memory in scheme 1 and to the lesser extent (as shown in Figure 11) in scheme 2. Thus schemes 3 and 4 are disadvantaged by design.

To estimate the additional throughput schemes 1 and 2 would have gained should the shortfall of CPU resources (in the amount of what is actually used in schemes 3 and 4) have been made up, we approximate it with the throughput that is already attained on the light database in the very same schemes. We add this extra throughput to schemes 1 and 2 for a baseline performance comparison: VM–based consolidations (schemes 3 and 4) perform on par with database server instance–based consolidation (scheme 1) and slightly better than operating system–based consolidation (scheme 2).

This strong isolation between VMs makes it desirable to extract more performance out of the given amount of VM resources, especially when host-wide resources are undercommitted. Figure 12 shows that, by sizing database shared buffers to be 75% of VM memory, schemes 3 and 4 perform better and more consistently: the average throughput (NOTPM) increases from 52,647 to 58,994 (12.1% more) in scheme 3 and from 52,399 to 59,296 (13.2% more) in scheme 4; the coefficient of variation (defined as standard deviation divided by mean) across the four periods decreases from 3.8% to 1.1% (71% less) in scheme 3 and from 5.2% to 1.0% (81% less) in scheme 4. Table 4 details the settings of all four schemes.

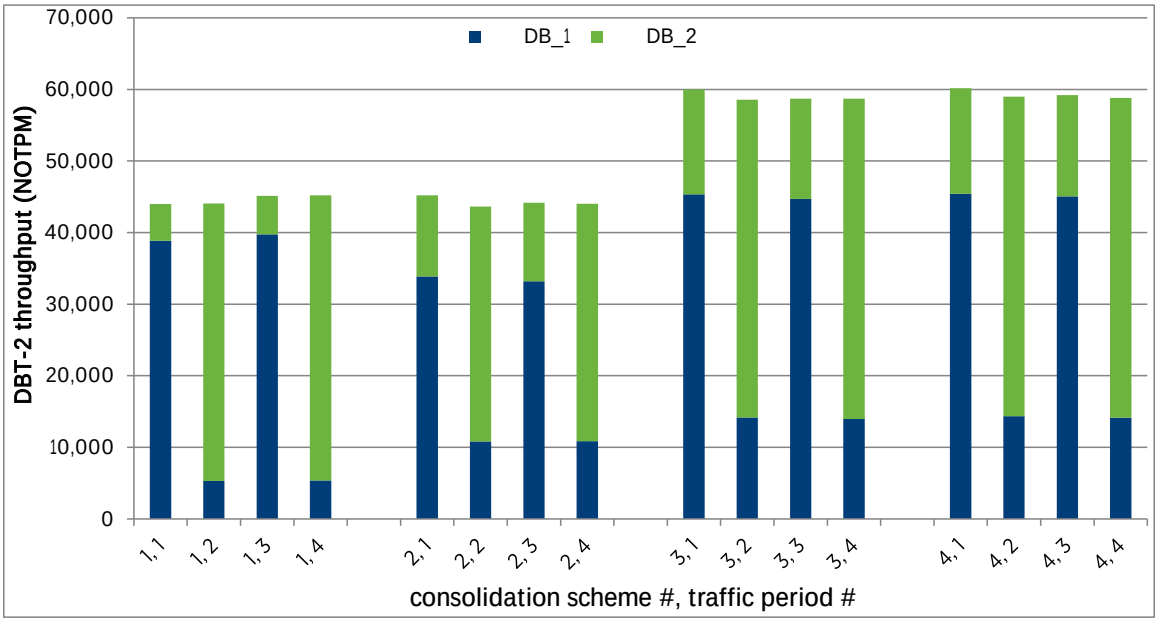

Figure 12. Performance in a memory undercommitment situation: configured per Table 4 (particularly using 75% of VM memory for database shared buffers) and exercised by the traffic detailed in Figure 6

	SCHEME 1	SCHEME 2	SCHEME 3	SCHEME 4
# database VMs, # vPostgres servers per VM, # of databases	1, 1, 2	1, 2, 2	2, 2, 2	2, 2, 2
# of vCPU per VM	3	3	3	3
vRAM per VM	8GB	8GB	4GB	4GB
shared buffers per vPostgres server	6GB	3GB	3GB	3GB
Use guest balloon	Yes	Yes	Yes	Yes
Use vPostgres balloon	No	No	No	Yes

Table 4. Settings of four database consolidation schemes using a larger portion of VM memory for database shared buffers (75% here versus 25% in Table 3)

Consolidation Results: Memory Overcommitment

While the baseline performance readings of the four consolidation schemes are roughly equal, we want to see how well they can preserve the performance in memory overcommitment situations. Figure 13 shows the performance curves under a series of escalating memory overcommitment rates. We observe that VM-based consolidation schemes are more capable of preserving the performance once the memory overcommitment rate exceeds 20%. Co-opting the vPostgres database balloon (as in scheme 4) doesn't provide extra benefit beyond using only the guest kernel-level balloon (as in scheme 3). This is not surprising as the database shared buffers only use 25% of VM memory and there is little need to enlist contribution from the underused memory in vPostgres shared buffers.

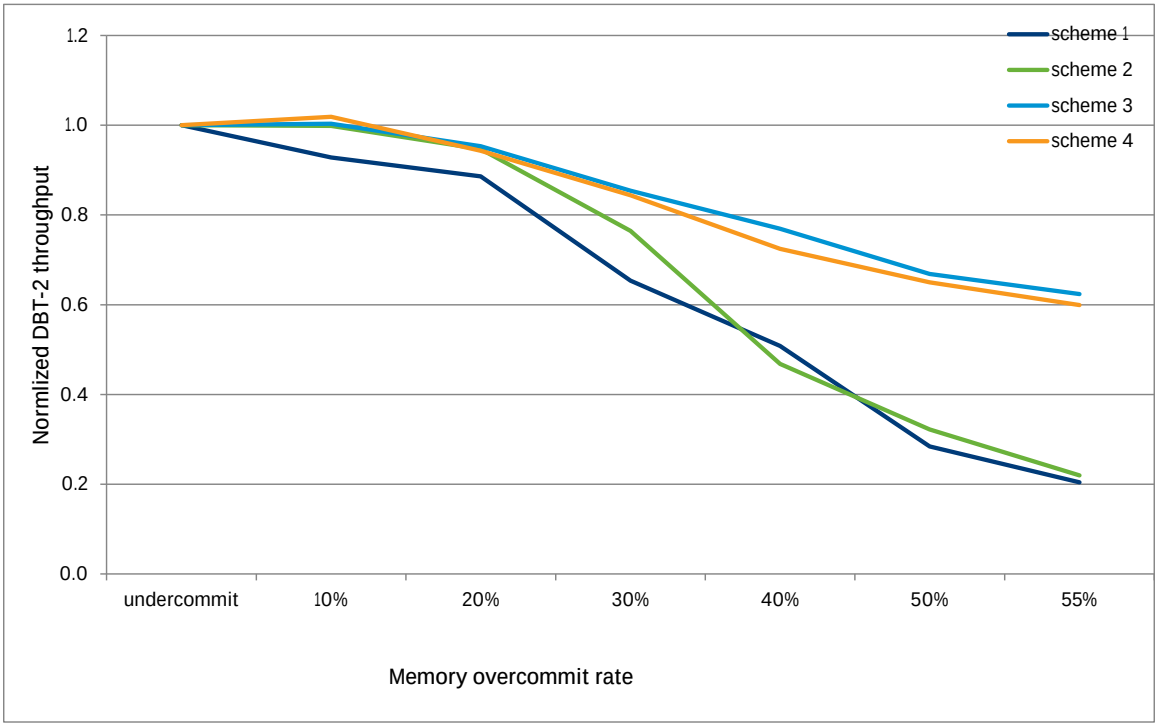

Figure 13. Performance in memory overcommitment situations: The configurations and traffic are the same as those used in Figure 11 except that the host memory available to the database VMs is adjusted to stage the prescribed memory overcommitment rates.

As discussed, for some workloads and some deployment circumstances, one might want to use unconventionally large database server shared buffers relative to VM memory. Figure 12 demonstrates the performance benefits of such cases where vPostgres shared buffers use 75% of VM memory. Figure 14 shows that the performance curves under such a setting and under various memory overcommitment situations. A VM-based consolidation scheme using only the guest kernel-level balloon (scheme 3) performs better than other consolidation approaches (schemes 1 and 2). For example, it performs 60% better under a 55% memory overcommitment rate. Co-opting the vPostgres database balloon (scheme 4) can help improve the performance by another 50% and overall performs 140% better than schemes 1 and 2.

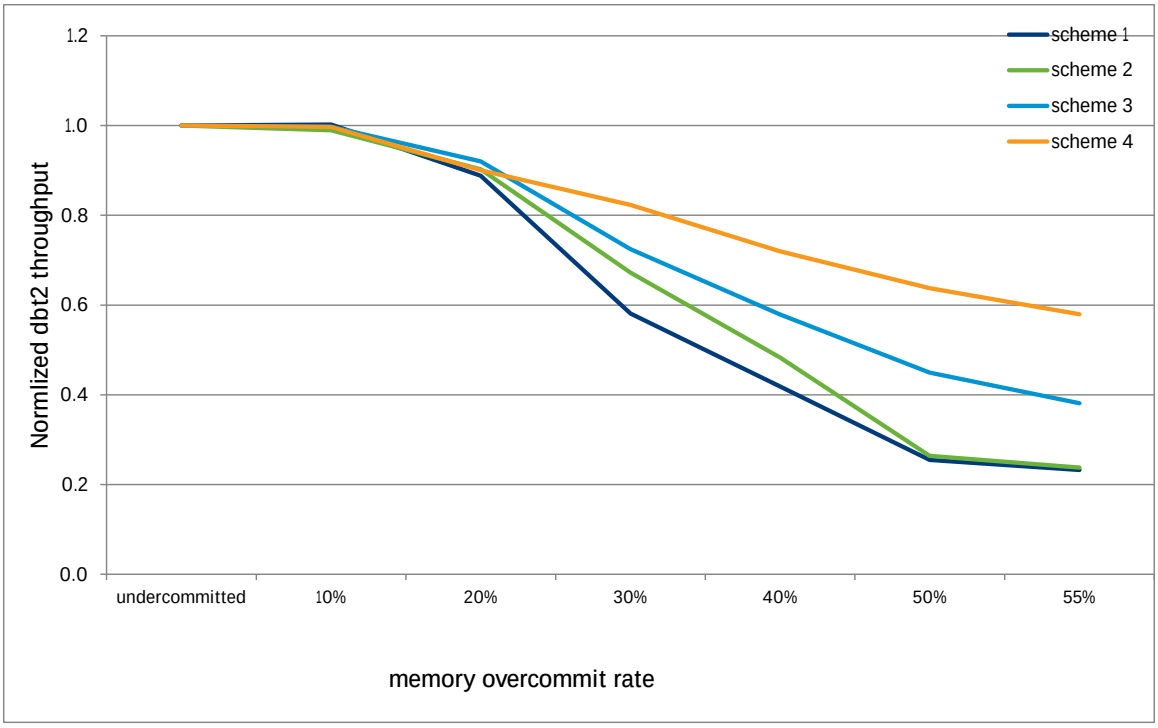

Figure 14. Performance in memory overcommitment situations—the configurations and traffic are the same as those used in Figure 12 except that the host memory available to the database VMs is adjusted to stage the prescribed memory overcommitment rates.

System Dynamics of Consolidation: CPU

While the end-to-end performance superiority of a VM-based consolidation approach is clear, extra insight into the underlying system dynamics is also useful. We want to compare CPU usage between all schemes. We also want to inspect memory reclamation data (e.g., ballooning, compression, and swapping activities) expecting that they can help explain the performance differences observed. Due to the constraint of space, we look into the system dynamics of some representative test points.

Figure 15 shows the physical CPU (PCPU) usage observed in the baseline experiments in Figure 11. Table 5 calculates CPU productivity (how much end-to-end payload throughput a fully used PCPU delivers) achieved in these experiments.

Figure 15. Physical CPU (PCPU) used by the database VMs in the baseline experiments in Figure 11: 100% PCPU used means one physical CPU is fully used.

	Scheme 1	Scheme 2	Scheme 3	Scheme 4
Average DBT-2 throughput (NOTPM)	45,303.41	40,790.49	52,647.45	52,398.71
Average PCPU used (%)	297.66	282.43	370.95	372.95
Throughput delivered per PCPU	15,219.85	14442.69	14192.60	14049.79

Table 5. CPU productivity achieved in the baseline experiments in Figure 11.

Figure 16. Physical CPU (PCPU) used by the database VMs in the baseline experiments in Figure 12

	SCHEME 1	SCHEME 2	SCHEME 3	SCHEME 4
Average DBT-2 throughput (NOTPM)	44595.37	44257.74	58993.92	59295.53
Average PCPU used (%)	296.20	293.73	383.04	389.49
Throughput delivered per PCPU	15055.83	15067.49	15401.50	15223.89

Table 6. CPU productivity achieved in the baseline experiments in Figure 12

Figure 16 shows the physical CPU usage observed in the experiments in Figure 12. Table 6 calculates CPU productivity achieved in these experiments. We see that the larger database shared buffers configured in these experiments offset the rigid memory boundary that disadvantages schemes 3 and 4: the CPU productivity of all consolidation schemes in memory-abundant baseline situations is equal, validating our previous approximation.

Comparing Figure 15 and Figure 16, we also observe more regular CPU usage in the latter. This corroborates the observation we made before that more consistent performance is attained in Figure 12 (where 75% VM memory is used for database shared buffers) than in Figure 11.

System Dynamics of Consolidation: Memory

We first inspect the memory dynamics of one set of experiments shown in Figure 13: 30% memory overcommitment rate with 25% VM memory for vPostgres shared buffers.

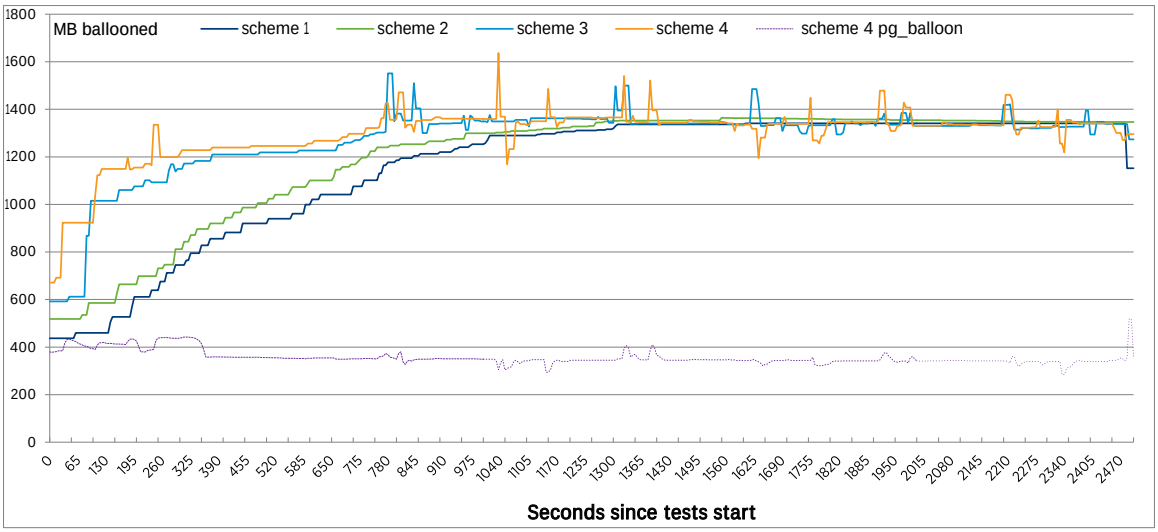

Figure 17. Memory ballooned by guest kernel-level balloon drivers and vPostgres database balloon drivers, configured per Table 3 and exercised by the traffic detailed in Figure 6 and subjected to a 30% memory overcommitment rate.

Figure 17 shows the amount of ballooned memory by kernel-level balloon drivers in all consolidation schemes and in addition to that by vPostgres database balloon drivers in scheme 4. On average 8% more memory is consistently ballooned in scheme 3 and 4 than in schemes 1 and 2. Note we only need to look at the memory ballooned by the kernel-level balloon driver for this comparison because the vPostgres database balloon driver by design cooperates with the guest operating system memory management regime and thus the memory ballooned by the vPostgres database balloon drivers will, in time, be reflected in the memory ballooned by the guest kernel-level balloon drivers.

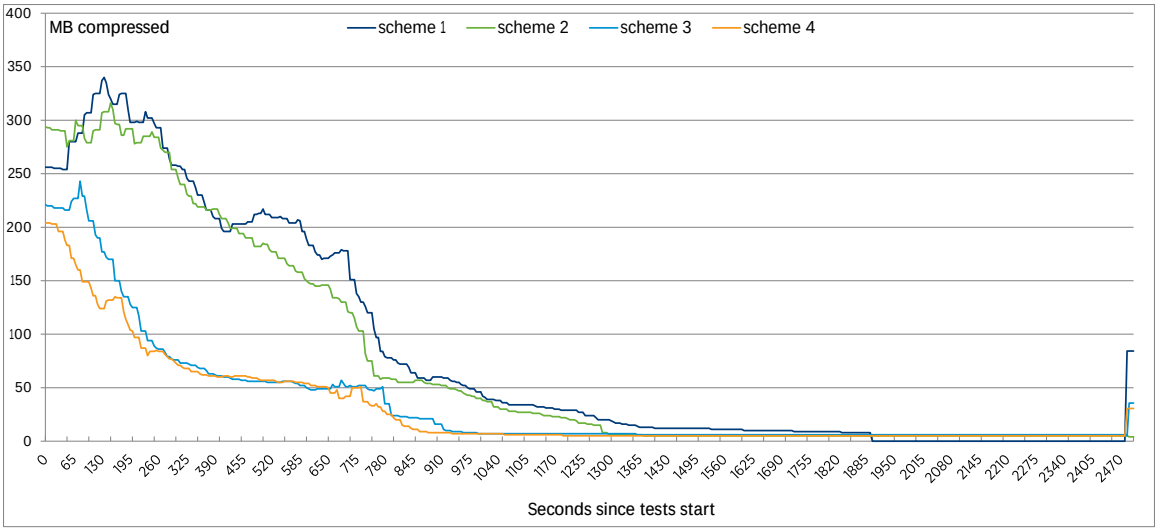

Figure 18. Memory compressed by hypervisor, configured per Table 3 and exercised by the traffic detailed in Figure 6 and subjected under a 30% memory overcommitment rate.

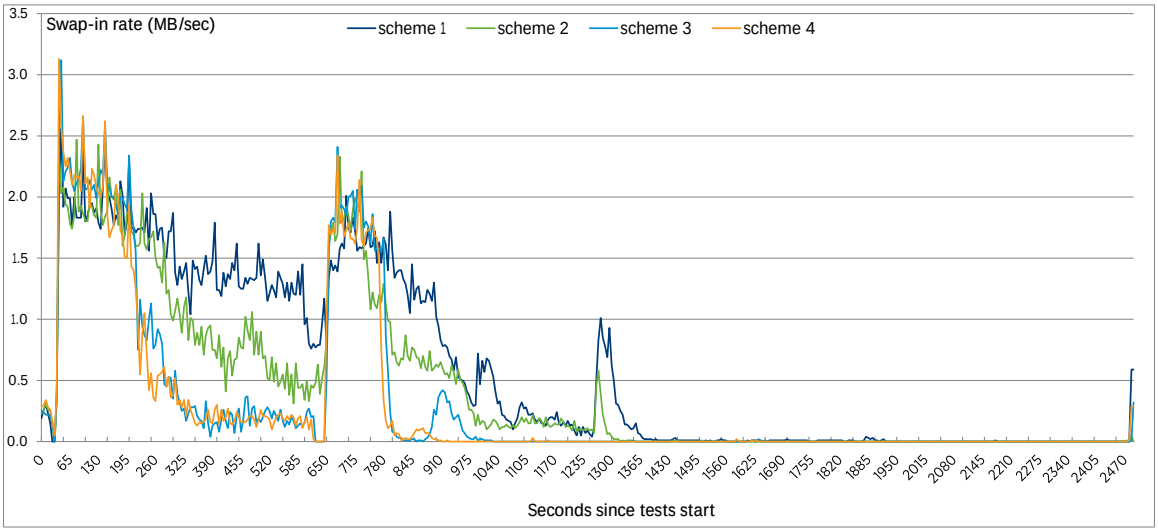

Figure 19. Memory swap-in rate by hypervisor, configured per Table 3 and exercised by the traffic detailed in Figure 6 and subjected under a 30% memory overcommitment rate.

Figure 18 and Figure 19 show hypervisor-level memory compression and swapping activities, respectively. On average, schemes 3 and 4 reduce the extent of these more drastic memory reclamation measures by more than 50% compared with schemes 1 and 2.

We then take a look at the memory dynamics of one set of experiments shown in Figure 14, that of 30% memory overcommitment rate with 75% VM memory for vPostgres shared buffers.

Figure 20, Figure 21, and Figure 22 show the ballooning, compression, and swapping activities in these experiments, respectively. Besides a similar pattern we saw in Figure 17, Figure 18, and Figure 19, scheme 4 now clearly benefits from the vPostgres database balloon: compared to scheme 3, scheme 4 reduces the extent of hypervisor-level compression and swapping by about 20%.

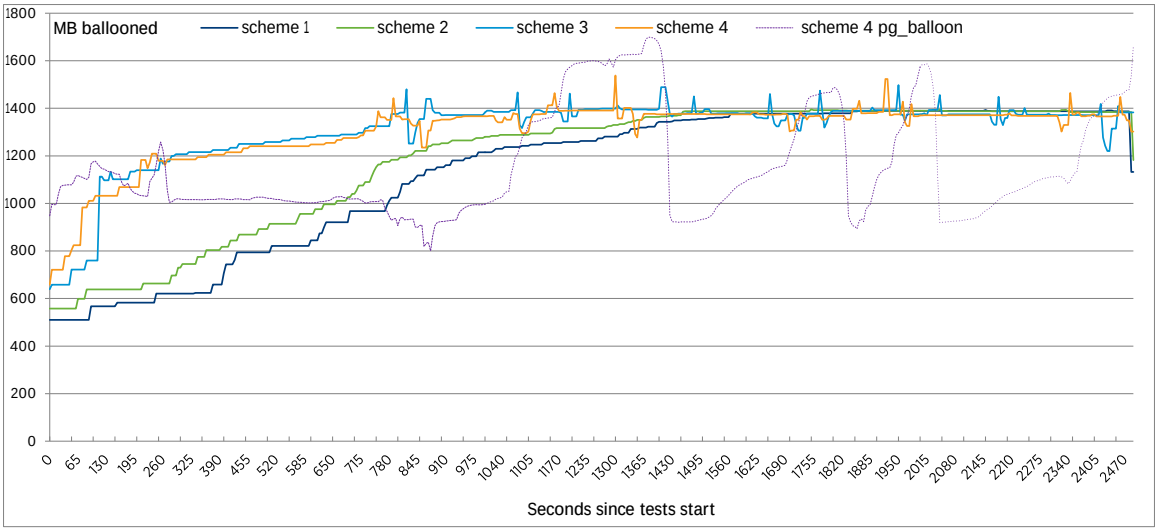

Figure 20. Memory ballooned by the guest kernel-level balloon drivers and vPostgres database balloon drivers, configured per Table 4 and exercised by the traffic detailed in Figure 6 and subjected to a 30% memory overcommitment rate.

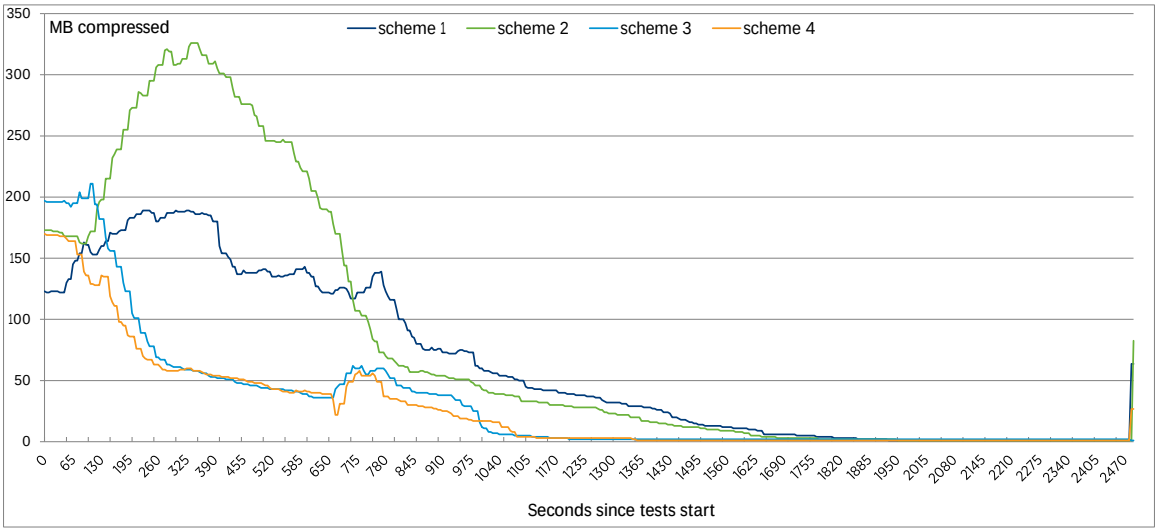

Figure 21. Memory compressed by hypervisor, configured per Table 4 and exercised by the traffic detailed in Figure 6 and subjected under a 30% memory overcommitment rate.

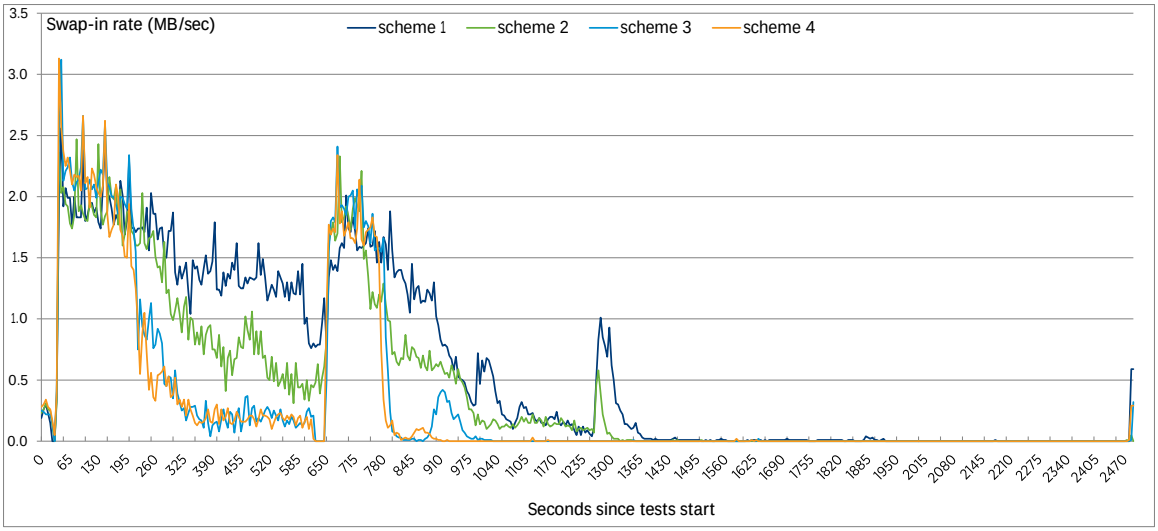

Figure 22. Memory swap-in rate by hypervisor, configured per Table 4 and exercised by the traffic detailed in Figure 6 and subjected under a 30% memory overcommitment rate.

Performance Best Practices

vPostgres 9.2 inherits all the performance improvements made in PostgreSQL 9.2 from upstream and features good out-of-box performance in most situations. vSphere is known for great out-of-box performance for common enterprise workloads. As a result, we refer to the performance best practices for vSphere [9] and PostgreSQL [3] for general performance tuning efforts. We point out what are most relevant to database workloads below:

- Configure the VM to use the VMXNET3 NIC driver.
- Configure the VM to use the PVSCSI disk driver and use thick and eager-zero virtual disk formats.
- When using Linux as a guest operating system, use the noop disk scheduler for guest disks. Refer to VMware KB 2011861 [10].

We recommend the following performance best practices specifically for vPostgres on vSphere:

- Dedicate a VM and a vPostgres server to any database of nontrivial performance concern.
- Put the vPostgres database cluster's DATA and WAL directories each on a dedicated virtual disk (VMDK) file and separate them from other entities such as operating system code and data.
- Test your application by experimentally sizing vPostgres server shared buffers beyond the conventional recommendation to seek the best performance out of fixed VM memory resources and use the highest load levels anticipated for your database application to conduct such tests.
- Avoid sizing vPostgres shared buffers of around 50% of VM memory. For example, size vPostgres shared buffers outside the region of 45-55% of VM memory.
- Enabling the vPostgres database balloon is recommended when (even occasional) memory overcommitment is anticipated, particularly if you size shared buffers unconventionally large relative to VM memory size. To use this feature, add the following line to the `postgresql.conf` of vPostgres server instance and then reload `postgresql.conf` (refer to Tuning Your PostgreSQL Server [3] for how to do this).

  ```
  enable_pgballoon = on
  ```

- Always enable the guest kernel-level balloon driver. Refer to the "Guest Operating System Consideration" section in vSphere Performance Best Practices [9].

Conclusion

This paper presents performance data and best practices for vPostgres 9.2 on vSphere 5.1.

We demonstrate vPostgres 9.2 features significant scalability improvement over the previous version. We also demonstrate vPostgres 9.2 on vSphere 5.1 achieves a vertical scalability equal to what is achieved on a native setup.

We show consolidating vPostgres databases on dedicated VMs performs superior to consolidating them on a database server instance or on a guest operating system instance with multiple server instances.

We show that one can use an unconventionally larger sizing for vPostgres shared buffers to gain better and more consistent performance out of a fixed amount of VM resource when using dedicated VMs to contain databases.

We show that under memory overcommitment situations, vPostgres database contained in dedicated VM perform more robustly than that contained in other types of containers and even more so with the vPostgres database balloon, particularly in more performance-demanding situations.

References

[1] The PostgreSQL Global Development Group, "PostgreSQL," http://www.postgresql.org/.

[2] VMware, Inc., "VMware vFabric Postgres," http://www.vmware.com/products/vfabric-postgres/.

[3] G. Smith, C. Browne and R. Treat, "Tuning Your PostgreSQL Server," http://wiki.postgresql.org/wiki/Tuning_Your_PostgreSQL_Server. [Accessed 13 Sept. 2013].

[4] S. Schneider, "VMware vFabric Blog: 4 Ways VMware Transforms Postgres for the Cloud," VMware, Inc., 17 May 2012. http://blogs.vmware.com/vfabric/2012/05/what-is-new-in-vmware-vfabric-postgres-91.html.

[5] The PostgreSQL Global Development Group, "PostgreSQL Documentation: pgbench," http://www.postgresql.org/docs/9.2/static/pgbench.html.

[6] TPC: Transaction Processing Performance Council, "TPC-B," http://www.tpc.org/tpcb/default.asp.

[7] M. Wong, "Database Test Suite: DBT-2," http://sourceforge.net/apps/mediawiki/osdldbt/index.php?title=Main_Page#dbt2.

[8] TPC: Transaction Processing Performance Council, "TPC-C," http://www.tpc.org/tpcc/default.asp.

[9] VMware, Inc., "Performance Best Practices for VMware vSphere 5.1," 10 Sept. 2012. http://www.vmware.com/resources/techresources/10329.

[10] VMware, Inc., "KB 2011861: Linux 2.6 kernel-based virtual machines experience slow disk I/O performance," 4 Feb. 2013. http://kb.vmware.com/kb/2011861.

About the Authors

Dong Ye is a Staff Engineer in the Performance Group at VMware. He works on vPostgres performance. Previously he has worked on VMware Mobile Virtualization Platform performance, vSphere ESX/ESXi performance, and JVM performance on vSphere.

Rishi Bidarkar is a Sr. Manager in the Performance Group at VMware. He leads the Solutions Performance team. He has filed several patents in the area of VDI performance and display benchmarking.

Acknowledgements

Thanks to Heikki Linnakangas, Jignesh Shah, Reza Taheri, Michael Paquier, and Priti Mishra for their constructive comments. Thanks to Julie Brodeur for her editorial help.

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