Persistent Memory Performance in vSphere 6.7

with Intel Optane DC persistent memory

Performance Study – April 2, 2019

Qasim Ali
Praveen Yedlapalli

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

Table of Contents

0. Introduction

Persistent memory (PMEM) is a new technology that bridges the gap between DRAM and traditional storage. It has the speed characteristics of memory, but it retains data through power cycles. While maintaining persistence of data across reboots and crashes, when compared to traditional storage, PMEM offers performance advantages such as:

- DRAM-like latency and bandwidth
- Use of regular load/store byte-addressable instructions

These characteristics make PMEM very attractive for a varied set of applications and scenarios.

Currently, there are two PMEM solutions available in the market:

- **NVDIMM-N by DELL EMC™ and HPE®:** NVDIMM-N is a type of DIMM that contains both DRAM and NAND-flash modules on the DIMM itself. Data is transferred between those two modules at startup, shutdown, or any power loss event. The DIMMs are backed by a battery power source on the mainboard in case of power loss. Currently, both HPE and DELL EMC are offering 16 GB NVDIMM-Ns [1] [2] [3] [4].
- **Intel® Optane™ DC persistent memory:** Intel Optane DC persistent memory (DCPMM) is designed to improve the overall performance of a data center system by providing large amounts of persistent storage at near memory speeds. DCPMM is not based on flash and the new material itself is persistent. The latencies of such devices are higher than DRAM, but still in the nanoseconds range. DCPMM modules are DDR4 socket compatible and will be available in sizes of 128, 256, and 512 gigabytes per module. For example, a two-socket system can have up to 6TB of DCPMM (3TB per CPU). [5] [6]

  There are two operation modes:

  - **App Direct mode:** Low latency persistent memory
  - **Memory mode:** Used as DDR4 DRAM

  We discussed the NVDIMM-N performance in the previous whitepaper [7]. In this paper, we focus solely on the DCPMM App Direct mode performance.

The rest of the paper is organized as follows:

1. We explain how PMEM can be configured and used in a vSphere environment.
2. We describe our experimental setup and various PMEM configurations used.
3. We show how applications with different characteristics can take advantage of PMEM in vSphere. Below are some of the use-cases:
   - 3.1 **Minimal Virtualization Overhead**: We demonstrate how PMEM device limits can be achieved under vSphere with little to no virtualization overhead. We show the virtual-to-native ratio along with raw bandwidth and latency numbers from **fio** I/O microbenchmark.
   - 3.2 **Relational Database Workloads**: Even without modification, traditional relational databases like **Oracle** and **MySQL** can benefit from using PMEM in vSphere.
   - 3.3 **PMEM-aware applications**: Applications get the best performance out of PMEM when modified specifically with this technology in mind. We show performance data from such applications; for example, an OLTP database like **SQL Server** and an in-memory database like **Redis.**
4. We outline some best practices on how to get the most out of PMEM in vSphere.
5. Finally, we conclude this paper with a summary.

1. vSphere PMEM

In this section, we describe how VMs can use PMEM in a vSphere environment. There are two ways of exposing PMEM to a VM:

1. **vPMEMDisk:** vSphere presents PMEM as a regular disk attached to the VM. No guest OS or application change is needed to leverage this mode. For example, legacy applications on legacy operating systems can use this mode. Note that the vPMEMDisk configuration is available only in vSphere and not in the bare-metal OS.
2. **vPMEM:** vSphere presents PMEM as an NVDIMM device to the VM. Most of the latest operating systems (for example, Windows Server 2016 and RHEL 7.4) support NVDIMM devices and can expose them to the applications as block or byte-addressable devices. Applications can use vPMEM as a regular storage device by going through the thin layer of the direct-access (DAX) file system or by mapping a region from the device and accessing it directly in a byte-addressable manner. This mode can be used by legacy or newer applications running on newer operating systems.

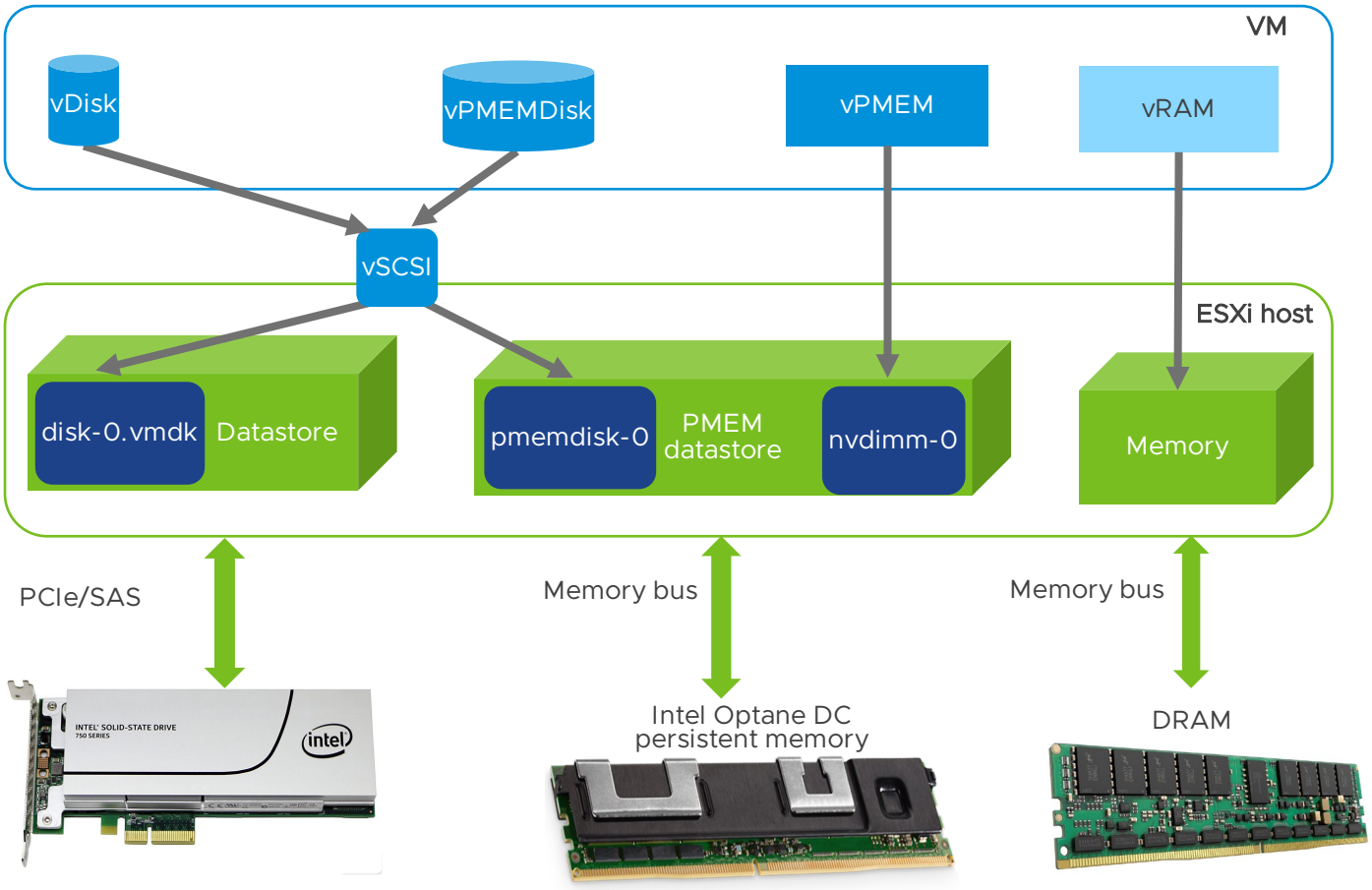

Figure 1. vSphere PMEM Architecture

Figure 1 shows the details of the PMEM architecture in vSphere. More information about vSphere PMEM can be found at docs.vmware.com [8] and storagehub.vmware.com [9].

1.1. vSphere host configuration

vSphere 6.7 identifies PMEM hardware and shows it in the *hardware resources section* of the host in vCenter as shown in figure 2. The recognized PMEM hardware is mounted as a PMEM datastore as shown in figure 3.

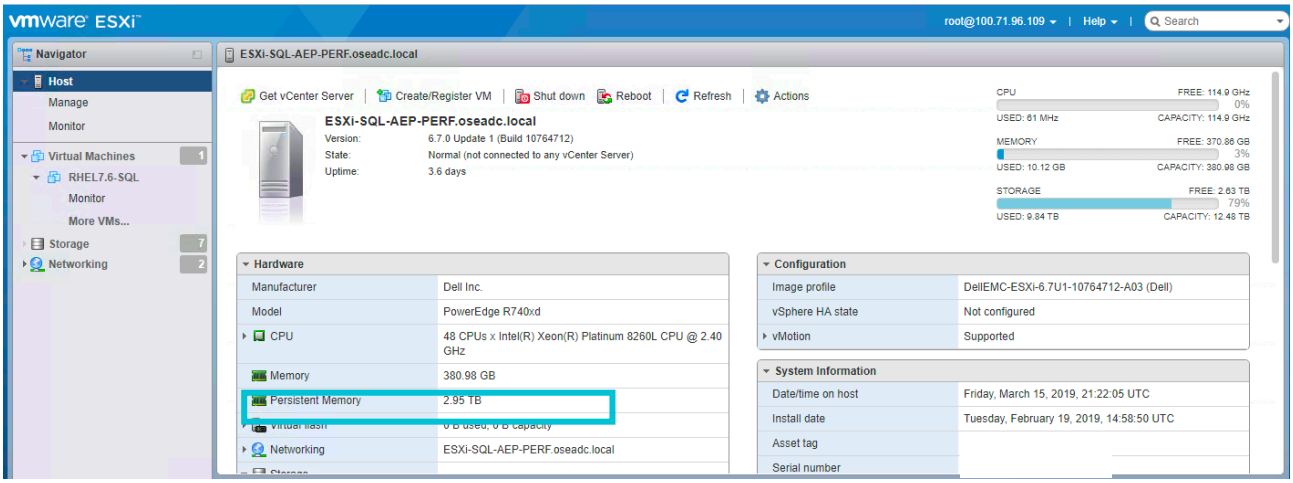

Figure 2. vSphere showing PMEM as a hardware resource

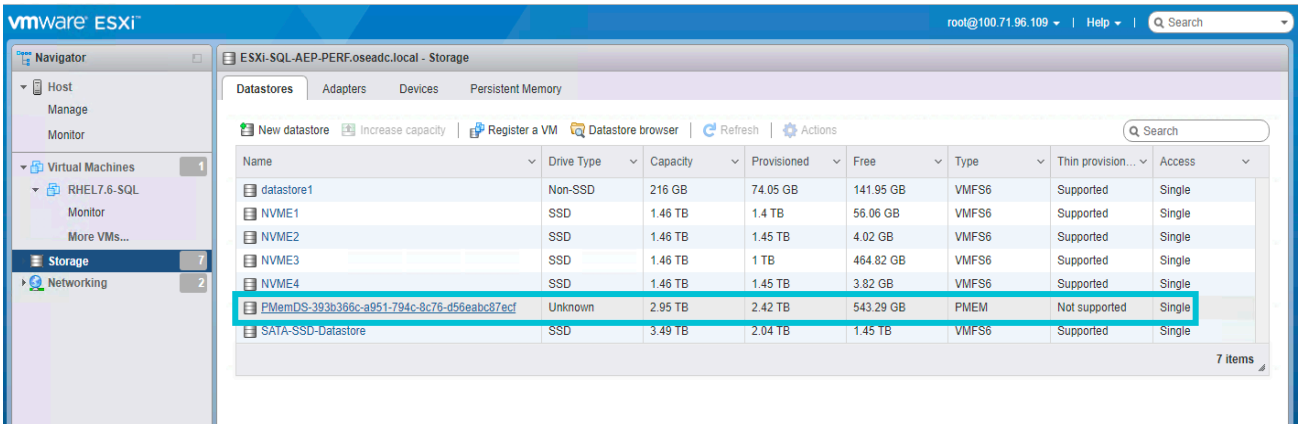

Figure 3. vSphere showing PMEM datastore

1.2. Virtual machine configuration

1. vPMEMDisk can be added to a VM like any regular disk as shown in figure 4. Make sure **VM Storage Policy** is set to **Host-local PMem Default Storage Policy**.
2. vPMEM can be added to a VM as a new NVDIMM device as shown in figure 5.

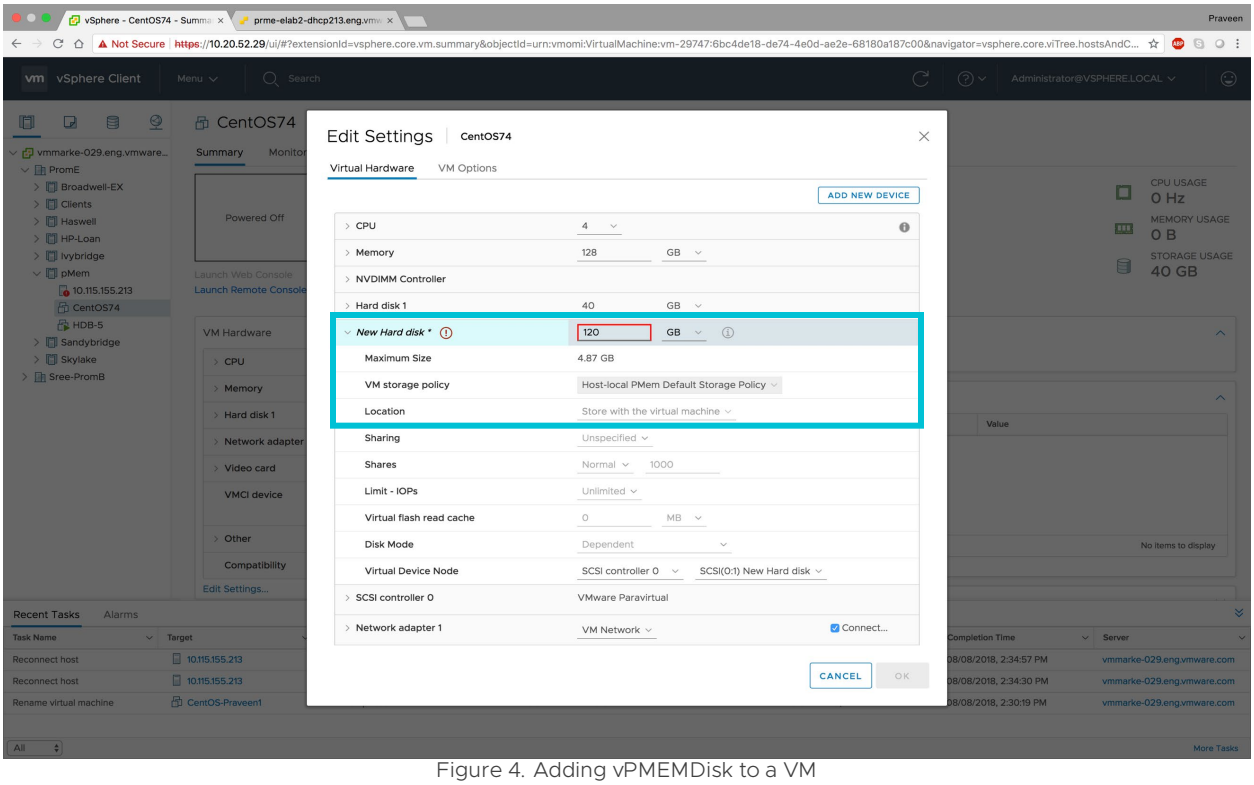

Figure 4. Adding vPMEMDisk to a VM

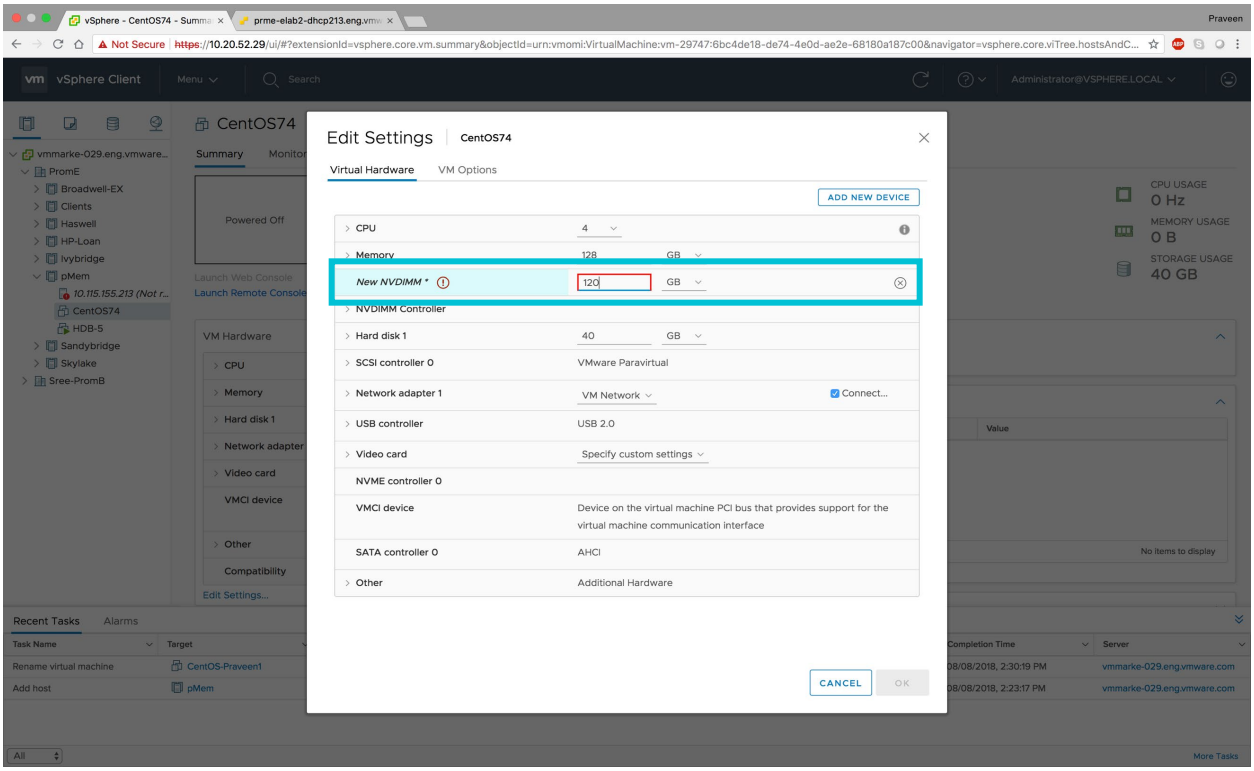

Figure 5. Adding vPMEM (NVDIMM) to a VM

2. Experimental setup

We did this performance study on two PMEM platforms: Intel PMEM platform (table 1) and Dell EMC PMEM platform (table 2). Please note that both platforms are similar.

Platform	Intel
CPU	2 sockets, 48 cores (96 threads) Intel Xeon Platinum 8260L @ 2.40 GHz
Memory	768 GB
PMEM	3.0 TB
Storage	Intel P4600 NVMe SSDs
ESXi	vSphere 6.7 U.1

Table 1. Intel testbed

Platform	Dell EMC PowerEdge R740xd
CPU	2 sockets, 48 cores (96 threads) Intel Xeon Platinum 8260L @ 2.4 GHz
Memory	384 GB
PMEM	3.0 TB
Storage	Dell EMC Express Flash Mixed Use NVMe PCIe SSD PM1725a
ESXi	vSphere 6.7 U.1

Table 2. Dell EMC testbed

Note: Most of our measurements were done on the Intel platform (table 1). Only the SQL Server experiments were conducted on DELL EMC platform – table 2.

Configurations:

We compare the performance of various workloads across 3 different configurations as shown in table 3. Note that the vPMEM-aware configuration is applicable only to some workloads. Since all modern guest OSes support the most optimal vNVDIMM configuration, we focus on performance of vPMEM in this paper.

vNVMe SSD	Local NVMe SSD attached to VM via vNVMe adapter (used as baseline)
vPMEM	PMEM attached as a NVDIMM device to the VM
vPMEM-aware	Application is modified to take advantage of the new byte-addressable PMEM software paradigms [10].

Table 3. Test configurations

3. Use cases of PMEM in vSphere

This section describes some of the use cases of persistent memory in vSphere 6.7.

3.1. I/O Performance with FIO

FIO (version 3.12-109) is an I/O microbenchmark to measure I/O performance [11]. We used it to quantify the virtualization overhead and measure raw bandwidth and latency.

Highlights:

- Virtualization overhead of PMEM is less than 4%.
- The vPMEM-aware configuration can give up to 3.3x more bandwidth in a mixed read-write scenario compared to that of an vNVMe SSD.
- Application level latency with vPMEM configurations is less than 1 microsecond.

Configuration:

Table 4 gives the details of the Linux VM used for fio.

OS	RHEL 7.6
CPU	4 vCPU
vRAM	24 GB
vNVMe SSD	24 GB
vPMEM	24 GB

Table 4. FIO VM configuration

Table 5 gives the FIO parameters used.

Ioengines	libaio (default) and libpmem (in vPMEM-aware)
Test cases	random read, random read-write (50-50), random write
Threads	4 for throughput runs; 1 for latency run
File size	5 GB per thread
OIOs	16 for vNVMe SSD; 1 for vPMEM

Table 5. FIO workload configuration

3.1.1. Virtualization overhead

To quantify the virtualization overhead of PMEM, we compared FIO throughput on bare-metal installation of RHEL OS and a RHEL OS VM running on ESXi. Figure 6 shows the virtual-to-native ratio. In all the scenarios, we measured less than 4% overhead. We selected FIO to show virtualization overhead since microbenchmarks typically stress the system the most and are expected to show the maximum overhead when virtualized.

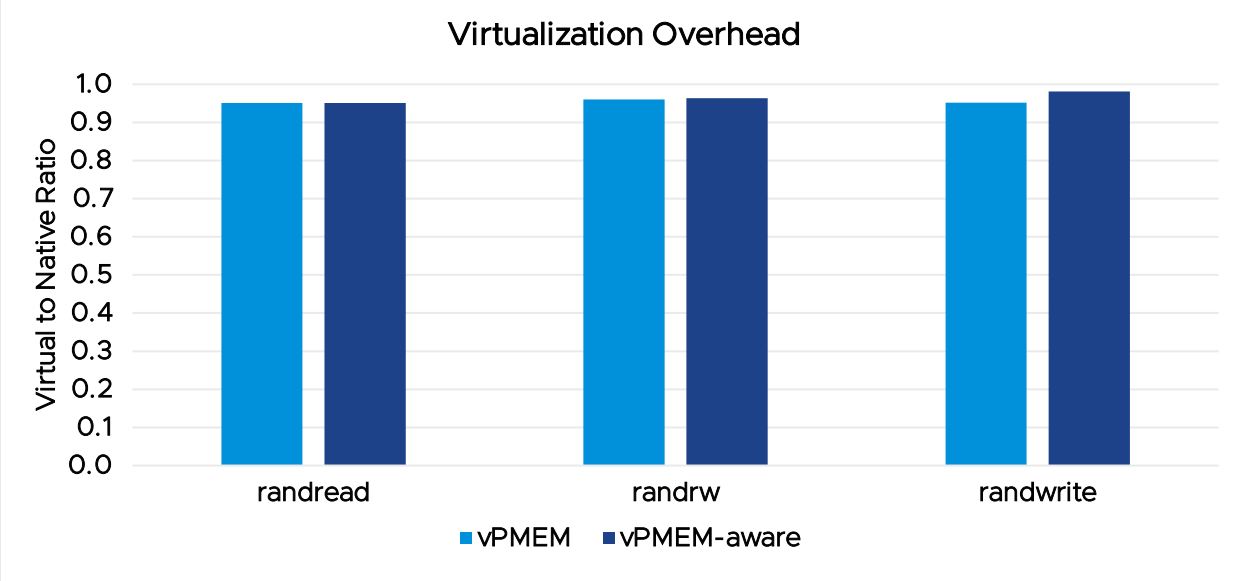

Figure 6. Virtual to native ratio (FIO 4 KB throughput)

3.1.2. Bandwidth and latency

Figure 7 shows the bandwidth measured in megabytes per second for different configurations with 4 KB I/O size. Note that the vPMEM cases are run with 1 thread to have a fair comparison with vNVMe-based configurations in which I/O is performed by one vNVMe world/thread. Since PMEM is a very low latency device, one outstanding I/O is the best choice for those configurations. In the random read case, the vPMEM-aware configuration, which uses the user-space libpmem library, yields approximately 2.4x the bandwidth compared to that of an vNVMe SSD.

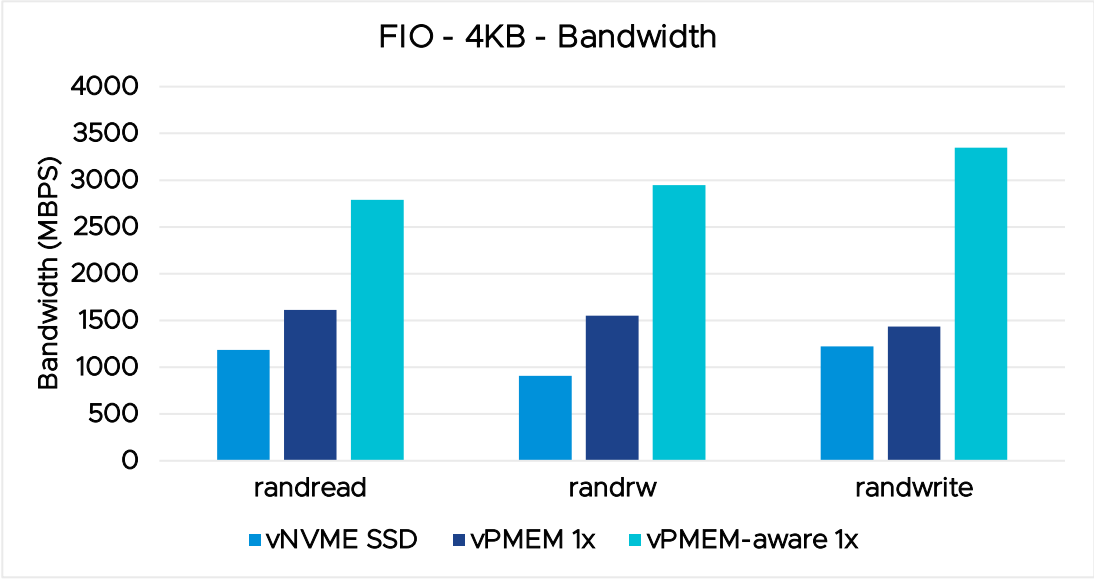

Figure 7. FIO 4 KB throughput

Figure 8 shows the bandwidth measured with 512 byte I/O. In the random write test, vPMEM-aware configuration achieved almost 1.4 GBPS bandwidth.

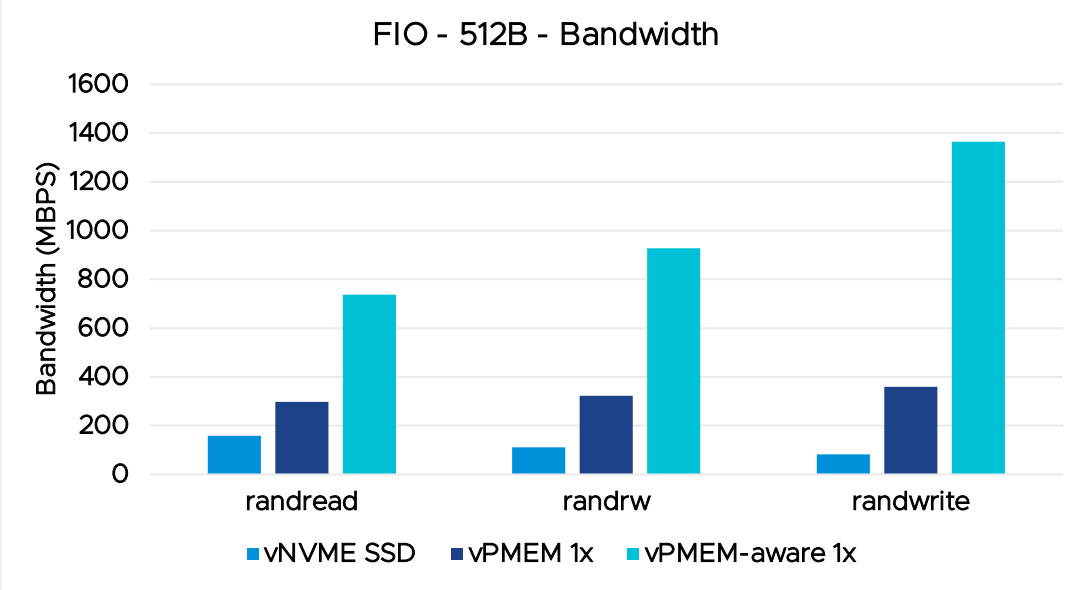

Figure 8. FIO 512 byte throughput

Figure 9 shows the bandwidth measured with 512 KB I/O. In the random write test, the vPMEM-aware case achieved more than 3 gigabytes per second of bandwidth using one thread, which is around 2.3x compared to vNVMe SSD.

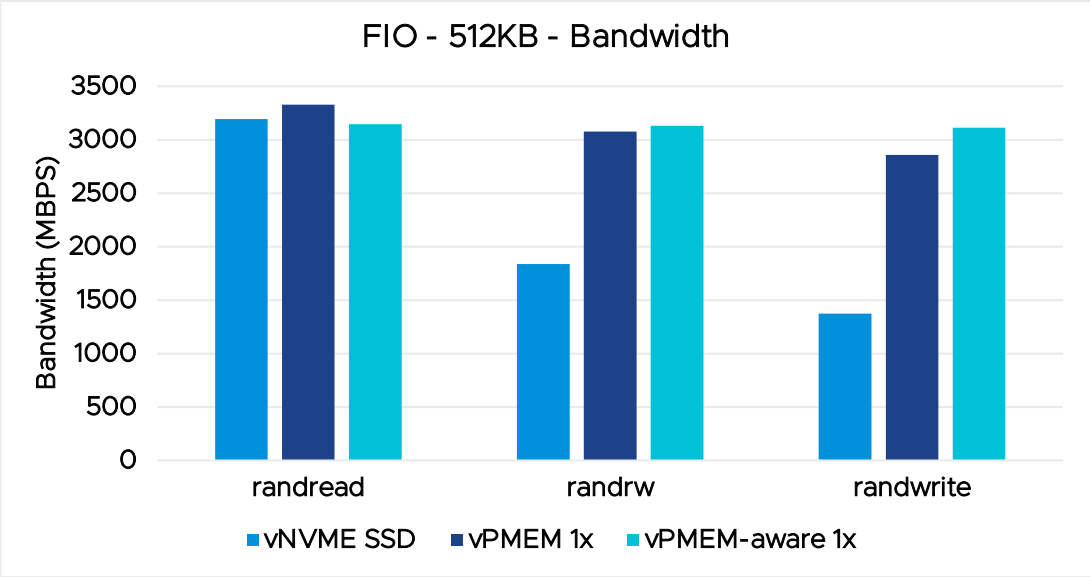

Figure 9. FIO 512 KB throughput

Figure 10 shows the 99th percentile latency in microseconds with different configurations. Both vPMEM configurations yielded sub-microsecond latencies. Note that the scale on the Y-axis is logarithmic.

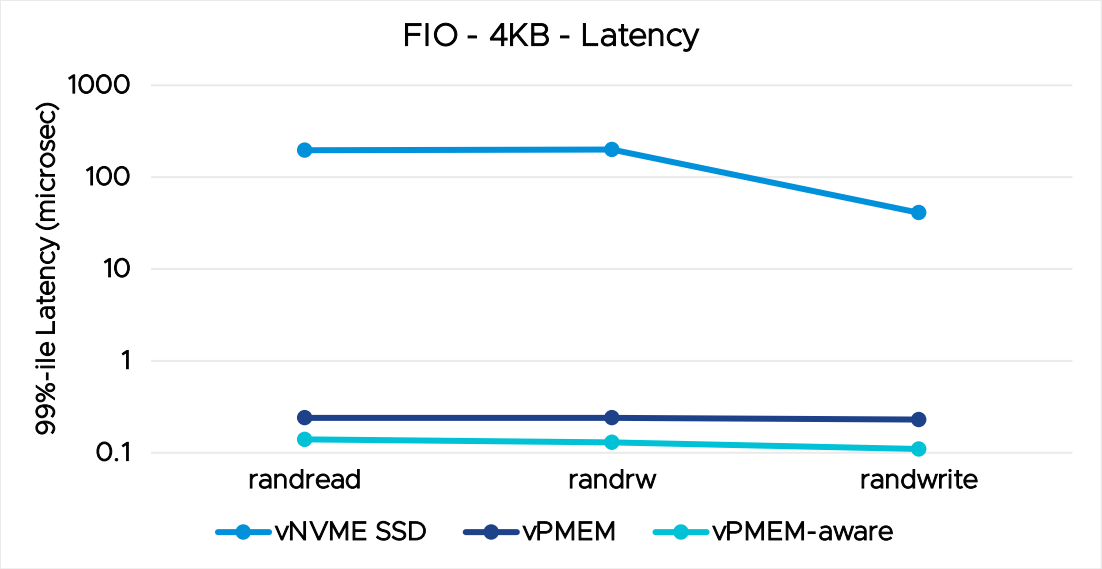

Figure 10. FIO 4 KB latency

For scenarios that demand more I/O bandwidth, we used more FIO I/O threads and observed much higher bandwidths. Using 12 threads in FIO, we measured up to **31.5 GBPS** of random read bandwidth and **12 GBPS** random write bandwidth with 512KB I/O from the vPMEM configuration. Note that these bandwidth numbers are from one socket only.

3.2. Relational database performance

3.2.1 Oracle Database

We used the hammerdb-3.1 [12] tool to benchmark Oracle Database (version 12cR2).

Highlights:

- 28% application performance improvement (HammerDB transactions per minute) with vPMEM compared to vNVME SSD
- 1.25x DB reads, 2.24x DB writes and up to 16.6x increase in DB log writes
- Up to 66x decrease in Oracle DB operations (read/write) latency

Configuration:

Table 6 shows the configuration for Oracle VM, and table 7 shows the HammerDB parameters used to test Oracle DB. Some additional configurations and parameters are mentioned in the appendix: 6.1. HammerDB Oracle.

OS	RHEL 7.6
CPU	48 vCPUs
vRAM	128 GB (SGA size = 35 GB to get the 75:25 R/W ratio)
VNVMe SSD	400 GB DB, 100 GB Logs (two 50 GB redo logs)
vPMEM	400 GB DB, 100 GB Logs (two 50 GB redo logs)

Table 6. VM configuration for Oracle

Virtual Users	70
Warehouses	3500
Warm Up Time	5 minutes
Run Time	25 minutes
Tests-Profile	TPC_C

Table 7. HammerDB parameters for Oracle

Results:

All the IOPS and latency numbers reported in figure 11 and table 8 are obtained using the iostats tool in Linux [13] [14]. More detailed iostats output is in section 6. Appendix.

Figure 11 shows the breakdown of IOPS achieved by Oracle DB. The vNVMe SSD bars show the read-to-write ratio is 75:25, which makes it a typical OLTP workload. The most fascinating data point in figure 11 is the 16.6x increase in DB log writes per second to almost 96K writes per second with vPMEM. The Oracle DB writer issues smaller log writes at a high frequency because of low device latency, which results in more read/write DB operations. This translates to overall application throughput. Log write size with vNVMe SSD is 38 KB, and with vPMEM it is 3 KB (from iostats).

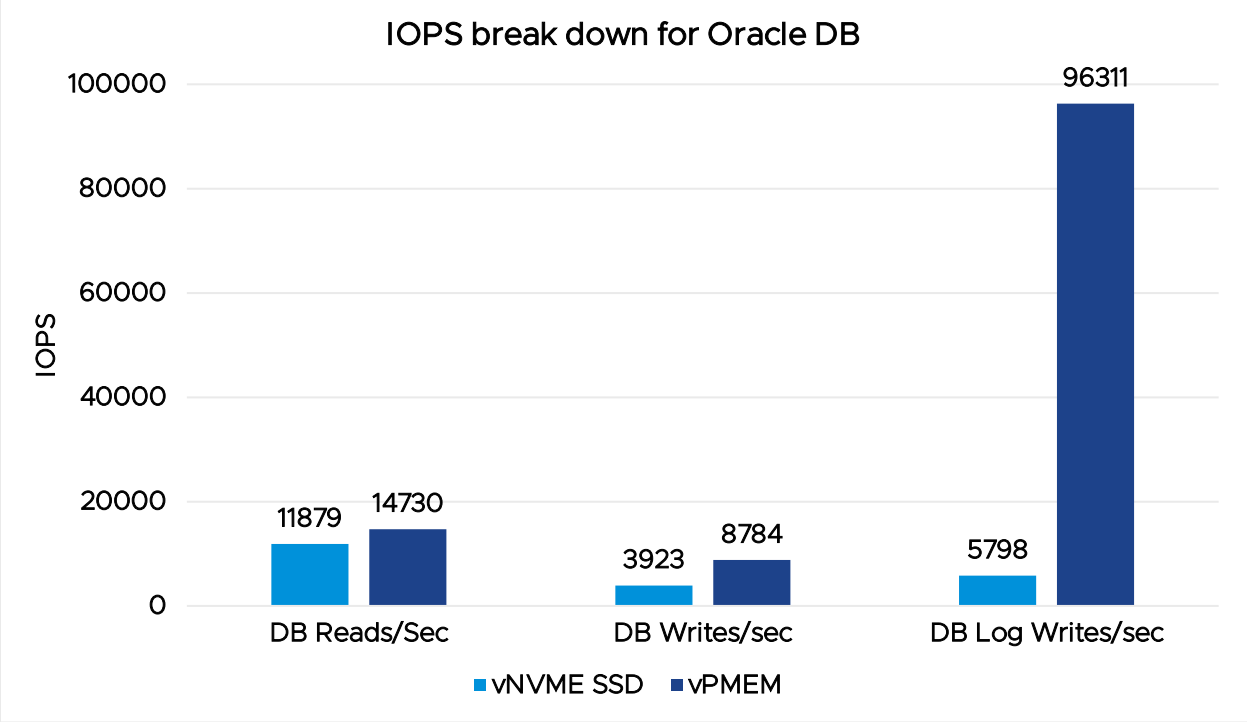

Figure 11. HammerDB with Oracle IOPS breakdown

Table 8 shows the dramatic decrease in latencies of DB read/write operations and log writes. The minimum latency reported by the iostats tool is 10 microseconds (0.01 milliseconds). We observed 0.01 in iostats for vPMEM.

Config	DB Reads (usecs)	DB Writes (usecs)	Log Writes (usecs)
vNVME SSD	330	660	90
vPMEM	~10	~10	~10

Table 8. HammerDB with Oracle DB latency breakdown (using iostat)

Figure 12 shows HammerDB throughput increases by 28%, and the server VM is close to fully utilized at 94%. (This is the guest CPU utilization obtained from mpstats/iostats.)

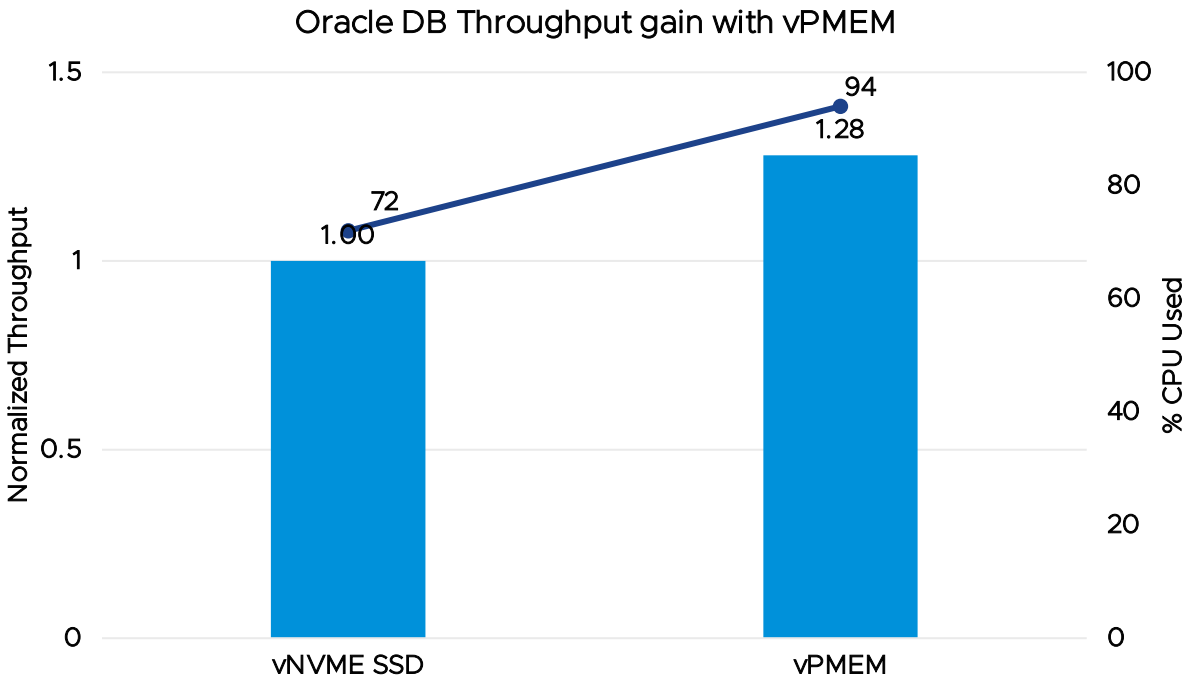

Figure 12. Oracle DB throughput gain with vPMEM using HammerDB

3.2.2 MySQL Database

This section shows how PMEM can improve the performance of a database like MySQL. We used Sysbench (version 1.1.0) [15], which is a database performance benchmark to drive the MySQL (version 8.0.15) database.

Highlights:

- vPMEM provides up to 60% better throughput and 4.6x better latency compared to vNVMe SSD.

Configuration:

Table 9 gives the details of the VM.

OS	RHEL 7.6
CPU	12 vCPU
vRAM	32 GB
vNVMe SSD	300 GB
vPMEM	300 GB

Table 9. Sysbench VM configuration

Table 10 gives the Sysbench parameters used.

DB Tables	6
Rows per table	110 M
DB Size	240 GB (24 GB Buffer pool)
Sysbench Threads	12
Tests	read_only oltp_read_write (75/25) write_only

Table 10. Sysbench workload configuration

Figure 13 shows the normalized throughput reported by Sysbench in two different configurations. The vPMEM configuration yields up to 60% better throughput compared to vNVMe SSD.

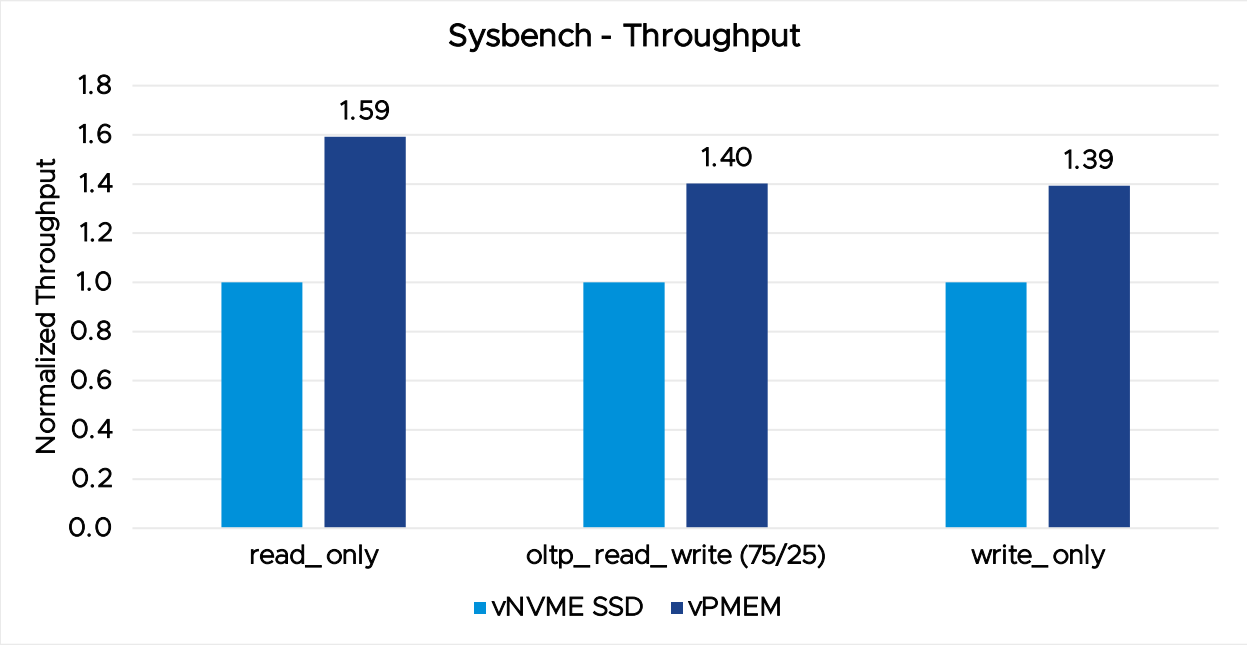

Figure 13. Sysbench with MySQL throughput

Figure 14 shows the normalized application-level 95th percentile latency reported by Sysbench in different configurations. The vPMEM case yields up to 4.6x better latency compared to vNVMe SSD.

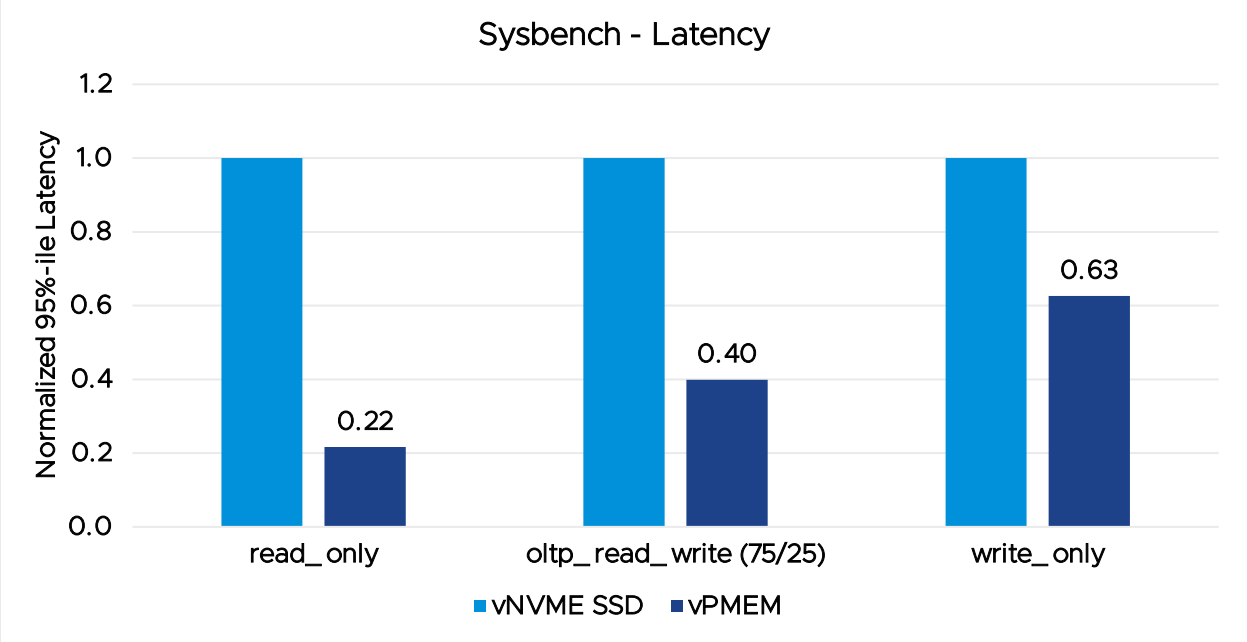

Figure 14. Sysbench with MySQL latency

3.3. PMEM-aware applications

We believe that, in order to achieve the maximum possible benefit from PMEM devices, applications need to be modified. PMEM-aware applications work in conjunction with direct access (DAX) mode where the NVDIMM device is mounted in the guest OS with the DAX filesystem option. Applications can use vPMEM by mapping a region from the device and accessing it directly in a byte-addressable manner.

Examples of such modified applications are Microsoft's SQL Server 2019 [16] [17] [18]and VMware's modified Redis [19].

3.3.1. Microsoft's SQL Server 2019

SQL Server 2019 preview extends support for PMEM. On Linux, there is full enlightenment of data and transaction log files placed on PMEM. Enlightenment refers to the method of access to the storage device using efficient user-space `memcpy()` operations. Rather than going through the file system and storage stack, SQL Server leverages DAX support on Linux to directly place data into devices, which reduces latency [18].

We used a workload derived from TPC-C to test SQL Server 2019 CTP 2.3 performance. As such, the results obtained with the derived workload are not compliant results and are not comparable to published TPC-C results. Tables 11 and 12 show the parameters we used for these experiments.

Highlights:

- Up to 2.7x increase in application performance using vPMEM-aware SQL 2019 when compared with vNVME SSD.

Configuration:

OS	RHEL 7.6
CPU	48 vCPU
vRAM	350 GB (SQL.Max.Memory = 315 GB)
vNVMe SSD	1.3 TB DB, 200 GB logs (on two separate SSDs)
vPMEM	1.3 TB DB, 200 GB Logs

Table 11. SQL Server VM configuration

Num_threads	100
Warehouses	14,000
Run Time	30 minutes
Tests-Profile	OLTP with a workload derived from TPC-C

Table 12. Derived TPC-C benchmark parameters for SQL Server

We used two configurations to quantify the performance gains:

1. vNVME SSD – DB and logs are on two separate NVME SSDs
2. vPMEM – DB and logs on vPMEM

Results:

Figure 15 shows that by moving the DB and logs to vPMEM, the fully vPMEM-aware (or in other words fully enlightened SQL 2019) can improve the throughput (transactions per minute) by 2.7x.

> **Note**: We believe that if you partition your DB tables across multiple NVME SSDs, then the baseline performance (NVME SSD) can improve.

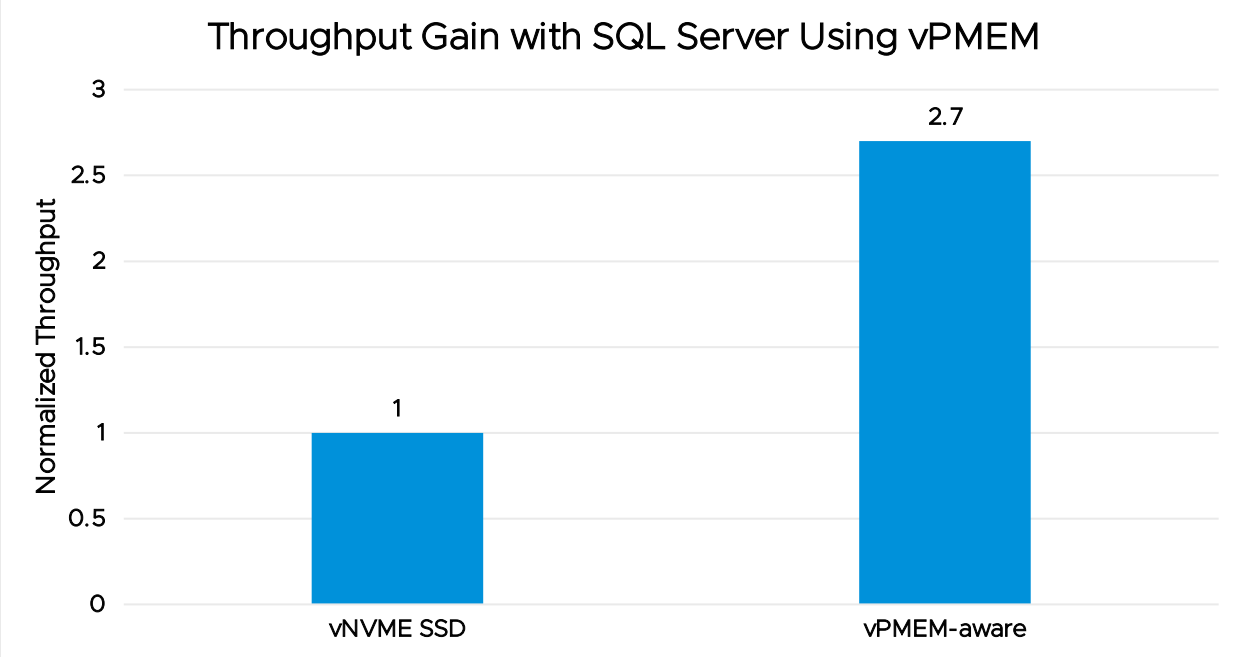

Figure 15. SQL Server application throughput gain: vNVMe SSD vs. vPMEM

3.3.2. VMware modified Redis

Redis is a popular open source, in-memory key-value store [20]. Redis can be used as a database, cache, or message broker. In this instance, we used Redis (version 3.2.9) as a database with perfect consistency (sync every SET operation to backing storage), and compared it to the PMEM-aware Redis that is developed in-house by VMware, using the new PMEM software libraries [19]. PMEM-aware Redis offers perfect consistency by default because both SET and GET operations are done in PMEM as opposed to in DRAM as in non-modified Redis.

PMEM-aware Redis offers the following benefits:

- **Better performance** in most cases with persistence and perfect consistency for all operations. This makes a case for using PMEM-aware Redis as a primary database.
- **Instant recovery** from crash. PMEM-aware Redis does all operations to and from PMEM, which persists through a crash. It does not need to load any data from disk to memory at initialization. In contrast, the original Redis must load the database from disk to memory at initialization, which can take up to several minutes.

We used **memtier_benchmark** (version 1.2.15), which is an open-source benchmarking tool for key-value stores to drive Redis [21].

Highlights:

- Redis write throughput is 2x better in the vPMEM-aware configuration compared to vNVMe SSD.
- The vPMEM-aware configuration yields 8.6x better latency.
- The vPMEM-aware configuration performs much better than vPMEM, making a case to modify applications.

Configuration:

Table 13 gives the details about the Redis VM.

OS	RHEL 7.4
CPU	4 vCPUs
vRAM	370 GB
vNVMe SSD	400 GB
vPMEM	400 GB

Table 13. Redis VM configuration

Table 14 shows the Redis parameters used.

DB Size	80 million keys (360 GB)
Data Size	4 KB
Driver	memtier_benchmark
Throughput parameters	pipeline=6; clients=50; threads=4
Latency parameters	pipeline=1; clients=1; threads=1
Tests	0% Writes; 30% Writes; 70% Writes; 100% Writes

Table 14. Redis workload configuration

Results:

Figure 16 shows the normalized throughput reported by memtier_benchmark in different configurations. The 100% writes case stresses the storage most; in this case, the vPMEM-aware configuration provided 2x better throughput. Moreover, vPMEM-aware throughput is 1.4x compared to vPMEM.

Redis is an in-memory database, and all the data is cached in the DRAM with writes persisted to the backing storage: either SSD or PMEM. In the 0% writes case, all the reads are coming from DRAM in the vNVMe SSD and PMEM configurations, and they show the same performance. But in the PMEM-aware configuration, there is no DRAM involved, and the reads are coming from PMEM, which is slower than DRAM; hence, the 33% drop in read-only performance. As the percentage of writes increase, the Redis throughput in the vPMEM-aware configuration stays consistent, while the performance on the vNVME SSD drops significantly due to an increased number of writes to the SSD and fewer DRAM reads.

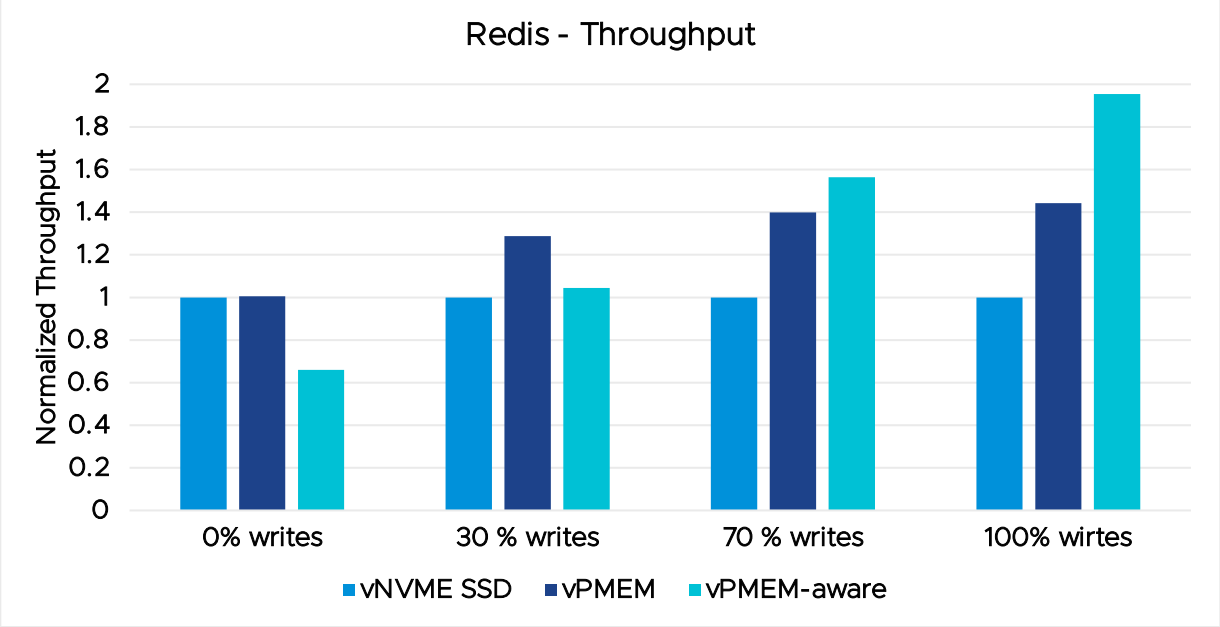

Figure 16. Redis throughput

Figure 17 shows the latency per Redis operation reported by memtier_benchmark. As the number of writes increase, the filesystem is stressed more, and latency increases accordingly in both the vNVME SSD and vPMEM configurations. The latencies in vNVME SSD are higher than vPMEM because the device latency is higher in vNVME SSD. In the 100% writes case, latency with vPMEM-aware is 8.6x better than NVMe SSD and 3.8x better than vPMEM.

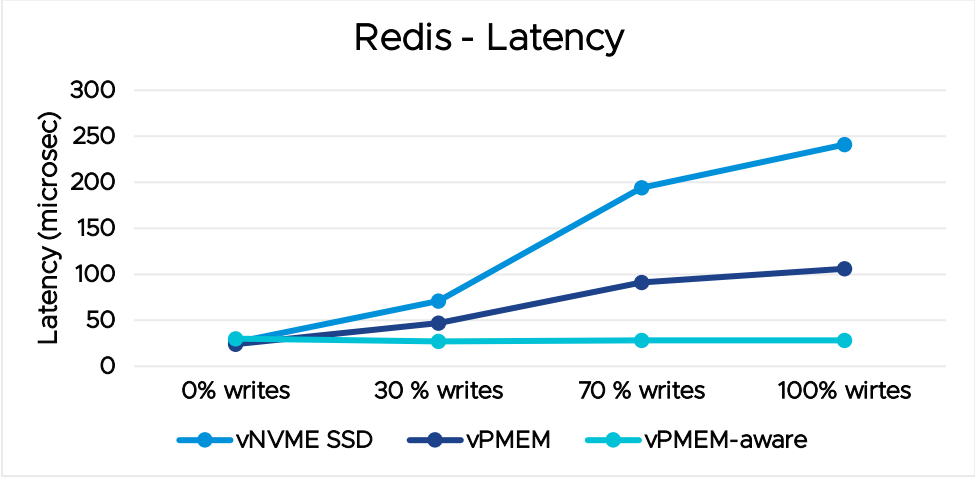

Figure 17. Redis latency

Figure 18 shows the crash recovery time in seconds for the 360 GB database. Note that vPMEM-aware Redis recovers instantly, while non-modified Redis takes more than 12 minutes to reload the DB from disk to memory.

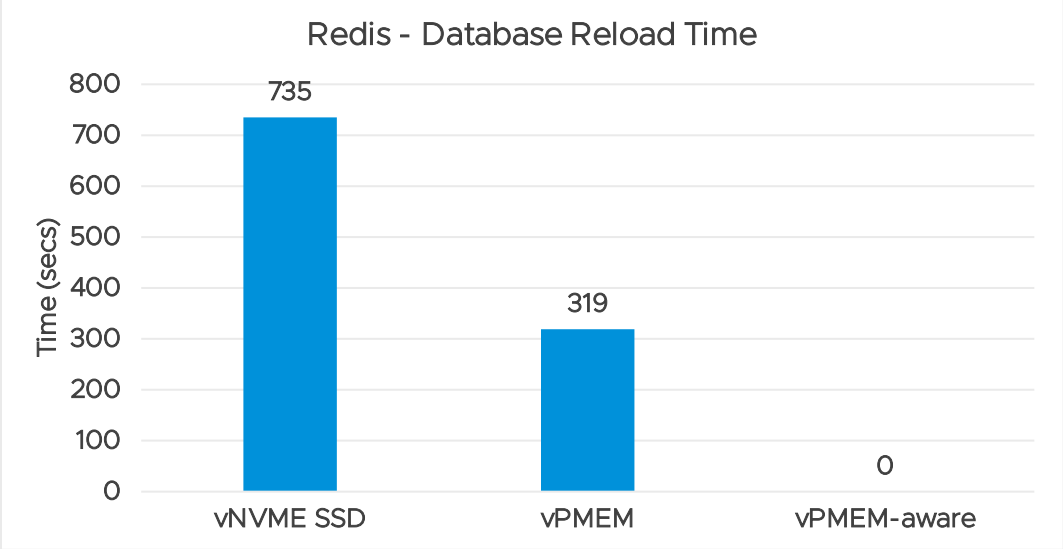

Figure 18. Redis crash recovery time

4. Best Practices

Here are some of the best practices based on our performance testing:

- Use vNVDIMM to get the best performance out of DCPMM.
- To get the best application performance, the workload/application needs to be modified to directly map the DCPMM in user-space and use `memcpy()` calls or libpmem libraries instead of I/O calls. For example, the way Microsoft SQL 2019 has been modified.
- Caching hot and predominantly ready-only data in DRAM still makes sense even when full dataset is stored in DCPMM. For example, Redis results with read dominant configurations illustrates the benefits of DRAM caching.

5. Conclusion

In summary:

- vSphere can achieve close to PMEM device bandwidth and latency.
- The virtualization overhead of PMEM in vSphere is less than 4%.
- vSphere PMEM gives up to 3.3x throughput improvement in micro benchmarks, and up to 2.7x improvement in tier-1 workloads compared to vNVMe SSD.

6. Appendix

6.1. HammerDB Oracle

We used large pages at system boot and THP in Linux was disabled. We also set priority for LGWRT and DB WRT using this command:

```
chrt --rr -p 83 -$P
```

We determined that this command was needed to get the best performance because, when CPU is near saturation, the log writer does not get scheduled optimally, resulting in subpar performance.

HammerDB client and Oracle server are in the same VM for the results in section 3.1.

Iostats for PMEM devices can be enabled using the following command:

```
$ echo 1 > /sys/block/pmem<n>/queue/iostats
```

I/Os that go through DAX paths (mounted using the -o dax option) are not counted in iostat output. In this experiment, the -o dax option did not make a difference to performance, so we mounted the PMEM devices without the -o dax option to collect detailed iostats for more insights.

The device nvme0n1/pmem0 have Oracle DB tables. The device nvme1n1/pmem1 has the Oracle redo logs.

```
Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s   wkB/s avgrq-sz  avgqu-sz   await r_await w_await
nvme1n1           0.00     9.6     0.00  5798.00   0.00   214156     73.86     0.60     0.10    0.00   0.10
nvme0n1           0.00     0.20   11879  3923.20  95033  54834.40    18.97     6.52    0.41    0.33    0.66

Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await r_await w_await
pmem1             0.00     0.00     0.00   96311     0.00   280275    5.82     0.14     0.00    0.00    0.00
pmem0             0.00     0.00    14730   8784  117845 59906    15.12     0.18    0.01    0.01    0.01
```

7. References

[1] Hewlett Packard Enterprise. (2018) HPE NVDIMMs. https://www.hpe.com/us/en/product-catalog/servers/persistent-memory/pip.hpe-persistent-memory.1008649009.html

[2] Hewlett Packard Enterprise. (2018) https://www.hpe.com/us/en/servers/persistent-memory.html

[3] Dell. Dell EMC NVDIMM-N Persistent Memory User Guide. https://www.dell.com/support/manuals/us/en/04/poweredge-r940/nvdimm-n_ug_pub/introduction?guid=guid-8884370c-5553-4089-b613-a3c570b56f0e&lang=en-us

[4] Dell. (2018, April) Dell EMC NVDIMM-N Persistent Memory User Guide. https://topics-cdn.dell.com/pdf/poweredge-r740_users-guide3_en-us.pdf

[5] WikiChip. Cascade Lake - Microarchitectures - Intel. https://en.wikichip.org/wiki/intel/microarchitectures/cascade_lake

[6] Intel Corporation. Intel Optane DC Persistent Memory. https://www.intel.com/content/www/us/en/architecture-and-technology/optane-dc-persistent-memory.html

[7] VMware, Inc. (2018, August) Persistent Memory Performance on vSphere 6.7. https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/performance/pmem-vsphere67-perf.pdf

[8] VMware. (2018, March) What's New in Performance for vSphere 6.7? Persistent Memory. https://docs.vmware.com/en/vSphere/6.7/solutions/vSphere-6.7.2cd6d2a77980cc623caa6062f3c89362/GUID-B65C7840D2D42343A7630B03A47BF4FB.html

[9] VMware. (2018) vSphere support for Persistent Memory (PMem) NVDIMM. https://storagehub.vmware.com/t/vsphere-storage/vsphere-6-7-core-storage-1/pmem-persistant-memory-nvdimm-support-in-vsphere/

[10] pmem.io. Persistent Memory Development Kit. https://pmem.io/pmdk/

[11] GitHub: fio. https://github.com/axboe/fio

[12] HammerDB. (2018) HammerDB benchmark. http://www.hammerdb.com/

[13] Michael Kerrisk. (2018, April) Linux User's Manual. http://man7.org/linux/man-pages/man1/iostat.1.html

[14] Sebastien Godard. iostat(1) - Linux man page. https://linux.die.net/man/1/iostat

[15] GitHub. (2018) Sysbench. https://github.com/akopytov/sysbench

[16] Microsoft. (2019) How It Works (It Just Runs Faster): Non-Volatile Memory SQL Server Tail Of Log Caching on NVDIMM. https://www.microsoft.com/en-us/sql-server/sql-server-2019

[17] Microsoft. (2018, July) Quickstart: Install SQL Server and create a database on Red Hat. https://docs.microsoft.com/en-us/sql/linux/quickstart-install-connect-red-hat?view=sqlallproducts-allversions

[18] Microsoft. (2018, November) How to configure persistent memory (PMEM) for SQL Server on Linux. https://docs.microsoft.com/en-us/sql/linux/sql-server-linux-configure-pmem?view=sqlallproducts-allversions

[19] VMware. (2018) What Is Persistent Memory? https://code.vmware.com/persistent-memory-initiative

[20] Redis Labs. Redis. https://redis.io/

[21] Redis Labs. (2018) GitHub: memtier_benchmark. https://github.com/RedisLabs/memtier_benchmark

[22] Hewlett Packard Enterprise. (2018, June) HPE Scalable Persistent Memory User. https://support.hpe.com/hpsc/doc/public/display?docId=emr_na-a00038915en_us

[23] John McCalpin. STREAM: Sustainable Memory Bandwidth in High Performance Computers. https://www.cs.virginia.edu/stream/

[24] Microsoft. (2017, March) SQL Server, Databases Object. https://docs.microsoft.com/en-us/sql/relational-databases/performance-monitor/sql-server-databases-object?view=sql-server-2016

[25] Violin Systems. Violin 6600 All Flash Array. https://www.violinsystems.com/products/violin-6600/

[26] Microsoft. (2018, October) DiskSpd: A Robust Storage Performance Tool. https://gallery.technet.microsoft.com/DiskSpd-A-Robust-Storage-6ef84e62

[27] Sreekanth Setty. (2011) vMotion Architecture, Best Practices, and Performance in vSphere 5.0. https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/vmware-vmotion-performance-vsphere5.pdf

About the Authors

Qasim Ali is a Staff Engineer II in the Performance team at VMware. He graduated with a PhD from Purdue University in 2009. Since then, he has been working on tuning vSphere's hypervisor and VMkernel stack (CPU, memory, and power). Currently he is focused on persistent memory (PMEM) performance in vSphere. He enjoys the challenges of optimizing vSphere using the latest hardware enhancements and technologies. He is the author of many conference papers, patents, and VMworld presentations.

Praveen Yedlapalli is a Sr. MTS in the vSphere Performance team. He focuses on the CPU and memory management in the ESXi kernel. He is actively working on quantifying and improving PMEM performance in vSphere. Praveen has a PhD from Pennsylvania State University (2014), where he focused on improving CPU-memory bottlenecks. He is the author of many CPU/memory publications and has given talks at various academic and VMworld conferences.

Acknowledgements

We would like to thank Intel and Dell for the hardware, and Microsoft for helping with SQL Server 2019 experiments. Special thanks to Jamie Reding from Microsoft and Syama Poulari from Dell EMC.