SOLUTION BRIEF

Memory Machine and KVM

MemVerge

Memory Machine and KVM

Executive Summary

Kernel-based Virtualization (KVM)[1] is open source virtualization technology included with the mainline Linux kernel. As a result, KVM is widely distributed and continually updated. Memory Machine is a software package from MemVerge[2] that runs on Linux and allows applications to take advantage of Persistent Memory (PMEM)[3] without requiring any code to be rewritten. KVM has a unique architecture in which the user components are packaged separately from the kernel components. This enables Memory Machine to be installed within an individual virtual machine (VM) or as a single instance running underneath (and outside) all the VMs running on a given server. The latter configuration allows Memory Machine to dynamically allocate DRAM and PMEM to individual VMs based on the demands of their workloads.

Each of these VM configurations was tested by subjecting popular workloads (Redis[4] in the first configuration and MySQL[5] in the second) to industry-standard benchmarks. The results show that by using Memory Machine more efficient use can be made of the available DRAM. In the Redis case, DRAM capacity could be reduced to 2/3 the size of the in-memory database without any performance degradation. In the SQL case under memory pressure, performance degraded by less than 3% even when DRAM capacity was reduced by 75%. The extra DRAM capacity can be used to start additional database instances to increase overall performance. If reducing total cost of ownership (TCO) is the primary business objective, spending on DRAM capacity could be decreased.

Introduction

Virtual Machines (VMs) are widely deployed because of their compelling advantages – scalability, flexibility, performance, and cost efficiency. Historically, most bare-metal hypervisors were based on a version of the unix kernel although modern hypervisors usually include a specialized, lightweight kernel. Kernel-based Virtualization (KVM) is built into the Linux kernel, thus allowing any Linux kernel to become a hypervisor. Although this gives KVM a heavier footprint in comparison with other hypervisors, it also has many advantages such as distribution (included in all Linux packages), maintenance (patches included with Linux updates), and continual development (contributions by the open source community). Similarly, applications that have been validated to run on Linux in bare metal environments are highly likely to run in a KVM environment.

QEMU-KVM Architecture

Although Linux virtualization is commonly known as KVM, it is more correctly referred to as QEMU-KVM. In QEMU-KVM virtualization, the user space and kernel components are packaged independently although most Linux distributions include both. KVM has a kernel component included in mainline Linux. A user space component for hardware virtualization is included in mainline QEMU (itself a branch of Linux). This allows for two options for configuring Memory Machine in a QEMU-KVM environment.

Memory Machine installed inside VM

KVM, as a bare-metal hypervisor, ensures that a generic OS can be installed unchanged as a guest OS. Memory Machine, as a user space process that runs on Linux, will therefore also run on top of a QEMU-KVM VM (as shown schematically in Figure 1).

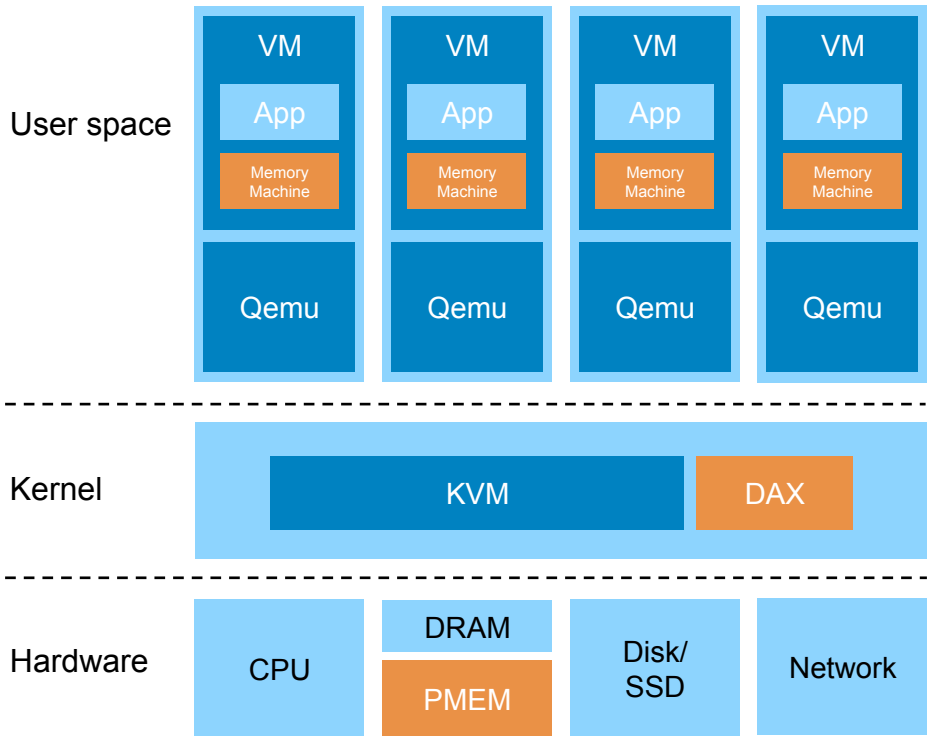

Figure 1. Memory Machine installed inside VM

Memory Machine installed underneath VM

Since QEMU-KVM is split into user and kernel modules, Memory Machine can be installed as a single instance above KVM (and below QEMU) in support of all VMs (shown schematically in Figure 2). Applications that have been qualified to run inside the VM will likely be qualified to run in this configuration because the application does not interact directly with Memory Machine (since it is installed outside of the VM).

In this configuration, Memory Machine provides further flexibility in the allocation of DRAM and PMEM resources to individual VMs. Each VM will be configured with memory capacity but the VM will not be aware of its composition i.e., the relative proportions of DRAM and PMEM. Memory Machine is able to dynamically reserve DRAM for individual VMs and optimally manage the placement of data in the large pool of PMEM available to the entire group of VMs.

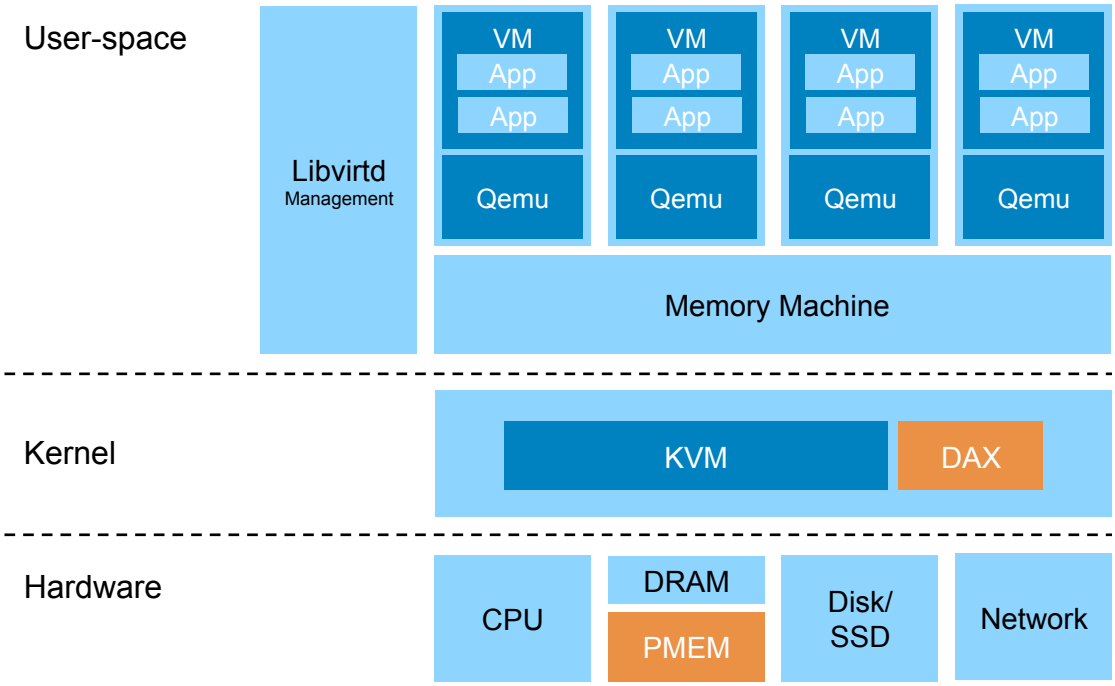

Figure 2. Memory Machine installed outside VM

Test Results using Memory Machine

Although there are specific benchmarks for hypervisor performance, it is instructive to compare the performance of applications that are used heavily in customer deployments. Two popular applications were selected. Redis (in-memory key-value database) was tested with the memtier benchmark[6] using Memory Machine installed inside the VM. MySQL (relational database that caches pages in memory) was tested with the Sysbench[7] benchmark using Memory Machine installed outside (underneath) the VM. Baseline data was captured using a VM configured with DRAM only.

Test Results: Memory Machine installed inside VM

The results of the Redis benchmark tests are shown in Figure 3. The VM was configured on the following platform:

Hardware

- 8 vCPUs
- 32GiB DRAM, 128GiB PMEM

Software

- Linux: CentOS 8.2
- Redis 5.0.7

A database size of 6GB was created so that there was no memory pressure. The tests therefore highlight memory latency rather than cache management. Memtier has numerous options, primarily the relative frequency of SET and GET operations and the key pattern (random, sequential, etc.) in the

SET and GET operations. For example, a SET:GET ratio of 1:0 is used to populate (or repopulate) the database. In Figure 3, the notation Get50%, for example, means that SET:GET operations occur in a 1:1 proportion. If a GET operation seeks a key that does not exist in the database (no hit), a NULL result will be returned. SET operations are the most onerous (especially when the memory structures must be allocated for the initial database population) since access is blocked until the SET has been confirmed.

The case where Memory Machine is managing 6 GB of DRAM (in addition to the PMEM) is equivalent to the baseline because the entire database can be placed in DRAM. As expected, the performance was the same as the DRAM only case. When the DRAM component was reduced to 4 GB, the performance was about the same or no worse than 96% of the baseline depending on the operation. When the DRAM was reduced to 2GB, the maximum degradation measured was 16%. Some degradation is expected since 2/3 of the database was now located in the slower PMEM tier. Even when no DRAM was used, the GET operations degrade by less than 20% as a result of Memory Machine's efficient memory page allocation.

Figure 3. Redis performance measured by memtier

Since Redis is largely single threaded (background processes use multiple threads), assigning more memory does not necessarily improve performance. It is often more effective to start additional Redis instances and take advantage of additional vCPUs. The results suggest that by using Memory Machine, 8 instances of Redis using 32GB DRAM would yield the same performance (within the margin of error) as 8 instances using 48 GB DRAM, a saving of 33% in DRAM. Or equivalently, 8

instances could be run in 32 GB rather than the 5 without Memory Machine, an increase of 66% in the number of instances.

Test Results: Memory Machine installed underneath VM

The results of the MySQL benchmark tests are shown in Figure 4. In these tests, the VM, running CentOS 8.1 Linux, was configured with 8vCPUs and 16GB memory. The VM was unaware of the distinction between DRAM and PMEM because the allocation was made dynamically by Memory Machine based on the workload. Baseline case was a VM with 16GB DRAM only. Two database sizes were used: small database allowed the entire database to be cached in DRAM, and large database did not. With Memory Machine reserving 16GB of DRAM for the VM in addition to managing the PMEM, the results for the smaller database were within 2% of the baseline case, indicating that Memory Machine imposes minimal overhead. With 4 GB or 2 GB DRAM reserved, the results were about the same and within 8% of the baseline case. For the larger database (i.e., when the DRAM was under memory pressure), the results were 5% better with Memory Machine reserving 16 GB DRAM for the VM in addition to managing the PMEM. Reserving 4GB (or 2GB) of DRAM when querying the larger database caused the performance to degrade by only 3% (or 6%).

Much like for the Redis case, there is an option to start additional MySQL instances within the same VM to make full use of all vCPUs. Since Memory Machine has a view of all the memory demands from the VMs, it is also possible to support more VMs without provisioning more DRAM. The results suggest that the VM density for this application could be increased by a factor of 4 or even 8 with minimal performance loss. As with all system level performance tests, bottlenecks may occur in CPU, memory, storage or network so it is important to investigate where the congestion occurs before increasing application or VM density radically.

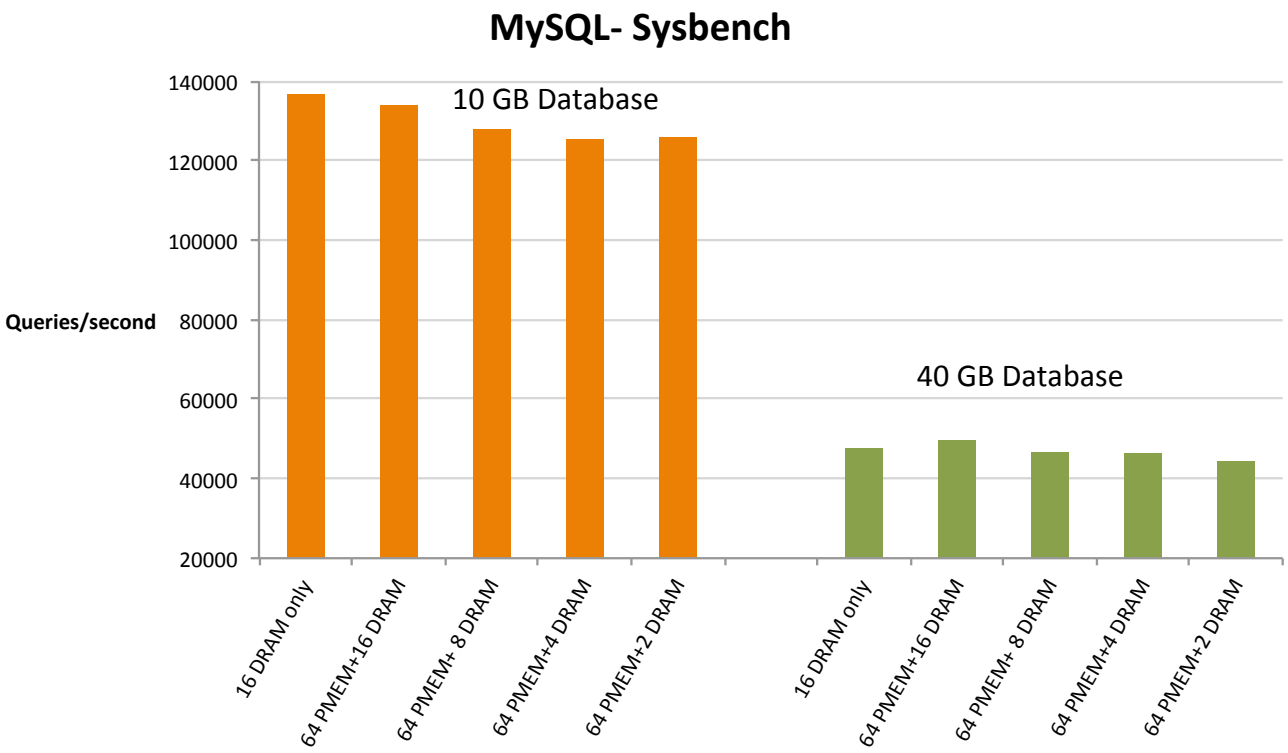

Figure 4. MySQL performance measured by Sysbench

Conclusion

In a QEMU-KVM environment, Memory Machine can be installed as an independent instance in each VM or as a single instance running as a platform underneath (outside) all VMs. In the latter configuration, individual VMs are unaware of Memory Machine. All applications that have been qualified to run in the VM will likely be qualified to run in this configuration.

Memory-intensive applications, such as an in-memory DB or a SQL DB, running in a VM will benefit from more memory. This was demonstrated when Memory Machine was used to manage a pool of PMEM added to the existing DRAM. If cost management is the primary concern, Memory Machine can achieve the same performance as a DRAM only case but with much more efficient use of the DRAM in a combined pool of DRAM and PMEM. If performance is the primary concern, the unused DRAM can be used to start additional application instances to avoid overloading a single threaded process (such as Redis) or a multi-threaded application that runs as a single process (such as MySQL).

[1] https://www.redhat.com/en/topics/virtualization/what-is-KVM
[2] https://memverge.com
[3] https://www.intel.com/content/www/us/en/products/docs/memory-storage/optane-persistent-memory/optane-persistent-memory-200-series-brief.html
[4] https://redislabs.com

[5] https://www.mysql.com
[6] https://redislabs.com/blog/memtier_benchmark-a-high-throughput-benchmarking-tool-for-redis-memcached
[7] https://github.com/akopytov/sysbench