MemVerge

White Paper

Software-Defined Memory: Platform for Virtual Machines

Software-defined Memory: Platform for Virtual Machines

Executive Summary

Virtual Machines (VMs) underpin a range of cloud computing services. Even in on-premises datacenters, the VM is the fundamental building block. Containers,[1] another form of virtualization, emerged later, and are now commonly deployed alongside (and even in) VMs. As workloads become more memory-intensive, DRAM capacity in servers supporting VMs and containers becomes a bottleneck. Breakthroughs in memory technology, such as Intel's Optane DC Persistent Memory (PMEM)[2] , offer a compelling increase in capacity and a cost advantage but have natively higher latency when compared to DRAM. Without being rewritten, existing applications cannot simultaneously leverage the persistence and increased capacity of PMEM. Significant code rewrites are often not feasible, especially with monolithic applications. Memory Machine, a software solution from MemVerge,[3] abstracts the memory pool to enable all applications to take advantage of PMEM without code modifications. Through sophisticated memory management, Memory Machine is able to achieve DRAM-like performance, on average, by using DRAM and PMEM in a two-tier memory hierarchy.

Persistent media modules that reside on the same memory bus as DRAM (i.e., have fast access to the CPU) open the door for innovation. Memory Machine can create snapshots of the current memory state of an application without incurring the I/O burden of an export to SSD. Such snapshots enable an application to be rolled back to a previous state or to be restarted rapidly in the case of a crash. Snapshots allow processes that rely on the same starting dataset to be replicated in seconds and to proceed in parallel, often increasing productivity significantly. Use cases have been demonstrated in areas ranging from bioinformatics to animation.

This document describes the impact of PMEM in virtualized environments and summarizes the results of tests conducted using Memory Machine supporting VM configurations. The results show that for applications that impose memory pressure, Memory Machine can achieve DRAM-like performance, or even better in some cases, while using a fraction of the available DRAM. The implication is that application density per VM, or VM density per server, can be increased by many multiples although such assertions must be made in the context of the overall system because other resources such as CPU, storage or network may become the bottleneck.

Introduction

In the modern data center, whether on-premises or in the cloud, VMs and containers are considered mature technologies and are used extensively. When compared, the received wisdom is that VMs are heavyweight (a guest OS is required for every VM) and containers are less secure (less robust isolation between processes because they share common OS resources). As deployments soared, a common set of challenges emerged, namely, the orchestration and management of thousands of instances. After a period of VM versus container as mutually exclusive options, there is now a consensus that VMs and containers will coexist for a long time, either side-by-side or as containers inside VMs.

VMs and containers differ architecturally. VMs virtualize hardware and containers virtualize the OS's user space. In real world applications, a single server may run less than a hundred VMs whereas a similar server may run thousands of containers. Considerable effort is being expended on making VMs lighter weight, i.e., quicker to start and stop, and occupying less footprint. In parallel, there are numerous initiatives to improve the overall security posture of containers.[4]

Any virtualization imposes overhead on the CPU, memory and storage provided by the underlying hardware. Often resource allocation is suboptimal because hypervisor and guest OS are unaware of each other's scheduling and memory management decisions.[5] Increasing memory capacity, such as by using DDR4-compatible PMEM modules, will improve performance but memory management is still the most important factor. Memory Machine, a software-only package from MemVerge, provides a memory virtualization and management layer that can be used to integrate DRAM with PMEM into a high capacity, two-tier memory hierarchy.

In a VM environment, Memory Machine can help scale both the number and size of VMs on a single server as well as offer flexibility in allocating DRAM and PMEM to individual VMs. The remainder of this paper is devoted to discussing the application of Memory Machine in VM environments in more detail.

Persistent Memory

Intel's second generation of Optane DC Persistent Memory[6] is available in 128GB, 256GB and 512GB capacities and can be installed alongside traditional DDR4 DIMMs. These PMEM modules are supported on 3rd Gen Intel Xeon scalable processors available in one-, two-, and four-socket platforms. Each CPU socket can contain up to six PMEM modules, providing up to 3 TB of PMEM per socket and total memory capacity of 4.5 TB per socket. Average latency (about 300 nanoseconds)[7] is close to DRAM (nominally 70 nanoseconds) and significantly faster than SSD (10's to 100's of microseconds).

PMEM can be configured to operate two modes, namely:

Memory Mode

In memory mode, the DRAM in the system is used as cache and the OS perceives the PMEM as the only available memory. DRAM becomes a resource devoted to optimizing the latency of the PMEM. The advantage is that applications can run without modification. However, the PMEM is volatile because data is protected with a single encryption key that is discarded upon power down, making the data inaccessible. Performance in Memory Mode can be dependent on the behavior of concurrent applications. Performance can degrade due to excessive cache misses caused by noisy neighbors.

App Direct Mode

In this mode, applications that are PMEM-aware have visibility of two types of memory: DRAM and PMEM. Such applications can directly access the PMEM in a manner similar to DMA (thus bypassing an extra copy operation in the CPU). PMEM can be configured as a block device where a file system can be mounted or as byte-addressable memory.

Memory Machine with PMEM

Memory Machine is a software application that runs in user space of the Linux operating system and allows any application to take advantage of PMEM. The entire capacity of DRAM and PMEM can be used as memory and the persistence of PMEM is exposed. Two functions enable Memory Machine to achieve this. The first takes advantage of a facility in Linux that allows memory allocation calls to the GNU C library glibc.so to be rerouted to a custom library. In this case, Memory Machine provides the custom library. The second function is a memory management application that allocates and releases memory, manages the assignment of data to DRAM or PMEM, implements copy-on-write, maintains snapshots, and provides relevant usage statistics. An API is supported to allow developers to have fine-grained control of where data is allocated. A configuration file defines which applications will utilize Memory Machine's memory management scheme. Other applications will run as they normally do and will be unaware of Memory Machine.

Any noisy neighbor problems are alleviated because Memory Machine controls where data is located. DRAM is used as a tier of memory rather than as a cache in which data must be retrieved from memory and copied to the cache. Data that is frequently accessed is located in the low latency DRAM and less frequently accessed data is located in the slower PMEM. As the access frequency changes, data can be moved between memory tiers to maintain optimal performance.

As a result of Memory Machine's ability to manage both DRAM and PMEM, snapshots of an application's current memory state can be taken without incurring any I/O aside from the creation of a small metadata file. This enables an application to recover rapidly from a crash: all the data from the most recent snapshot is already resident in memory. An application can be rolled back to a previous state by accessing an earlier snapshot. There are other uses for snapshots, such as forensics, compliance, or in a software development environment. Snapshots allow multiple processes that start from the same state to be instantiated in seconds, for example, in what-if analyses or model training in machine learning.

Memory Machine and Bare-metal Hypervisors

As the name suggests, a bare-metal hypervisor is installed directly on the server hardware and provides a virtualized view of the underlying components (CPU, memory, storage, network interfaces, etc.) to the guest OSs running in each VM above the hypervisor. Best known of these hypervisors are VMware's vSphere Hypervisor (previously known as ESXi), Microsoft's Hyper-V, and the open source KVM. A bare-metal hypervisor requires an OS kernel to manage the underlying hardware. KVM is a component of the standard Linux kernel and that allows Memory Machine to be installed directly on KVM (see Figure 1). When used with VMware and Hyper-V, Memory Machine must be installed on Linux running as a guest OS in a VM (see Figure 2).

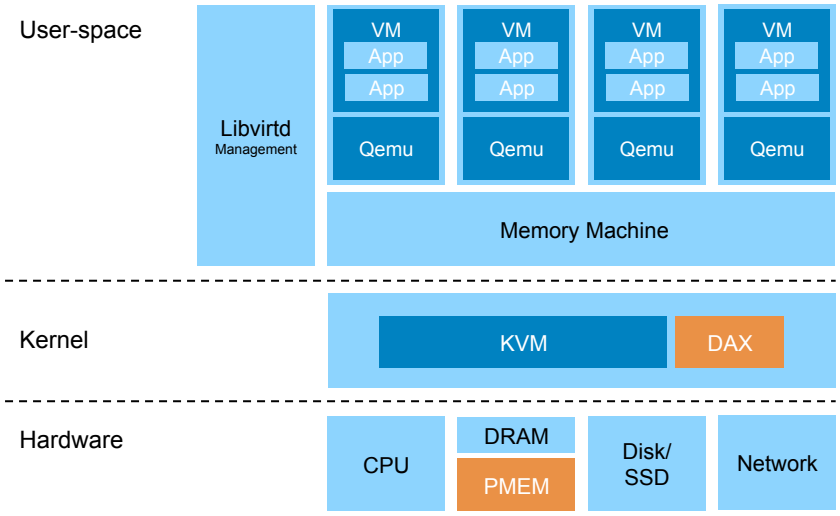

Figure 1. Memory Machine installed underneath VM

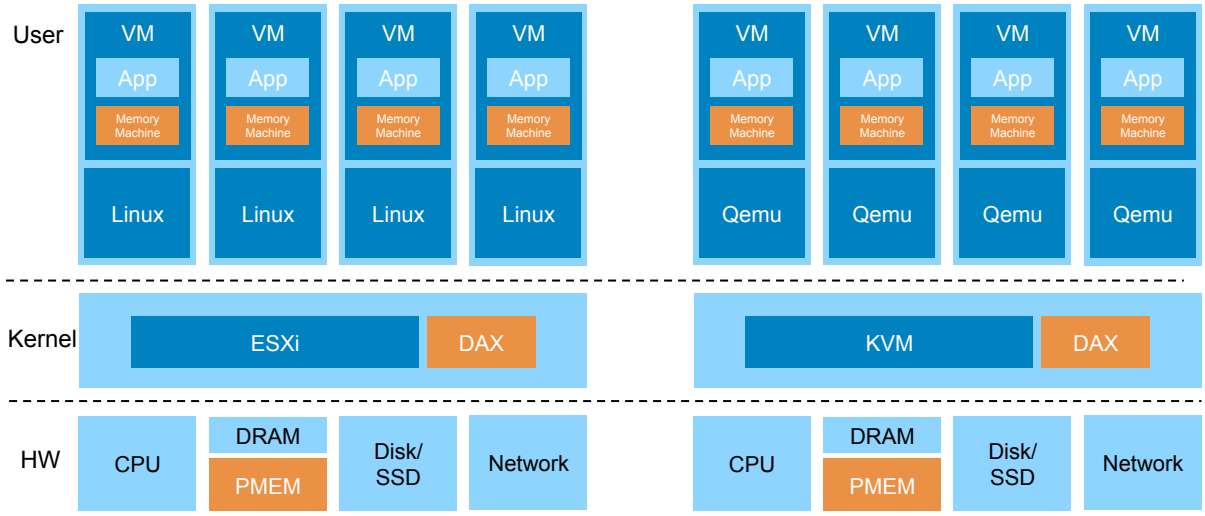

Figure 2. Memory Machine installed inside VM

PMEM-aware Virtual Machines

VM density and performance have historically been constrained by system memory. Most hypervisors now allow for dynamic memory assignment that enables the host to adjust the amount of memory assigned to a VM in accordance with actual usage. If static assignment of VM memory is used then the amount of VM memory specified in the VM configuration is always reserved.

VMware (starting with vSphere 6.7 EP10), QEMU-KVM (starting with QEMU v2.6.0) and Hyper-V (in Windows Server 2019) all support PMEM. The hypervisor will detect the presence of any PMEM resource and will expose it to any VMs on that hypervisor. PMEM can be consumed by the VM as a

byte-addressable direct access device (NVDIMM) or as a block-oriented virtual SCSI device (not currently supported in QEMU). NVDIMM can be used in Memory Mode or App Direct Mode. A guest OS that is not PMEM-aware can only use PMEM as a virtual SCSI device.

Virtual Machines with Memory Machine

The benefit of a bare-metal hypervisor is that an OS can be installed unchanged as a guest OS in a VM supported by the hypervisor. Any application that runs on that OS in a bare metal environment will also run in the VM without any modification as long as the required devices are exposed by the virtualization. Memory Machine, as a user space process that runs on Linux, will also run in a VM with a Linux guest OS. All the applications supported by Memory Machine in a bare metal environment will also be supported in a VM with Memory Machine installed. This can be leveraged in devops or in cloud migration where an application running on bare metal in a development environment, say, can be easily moved into production inside a cloud-based VM.

Test Results without Memory Machine

VMware

VMware has conducted extensive performance studies of vSphere with Intel DC Optane PMEM in both Memory and App Direct modes. All tests used a DRAM:PMEM ratio of 1:4 (recommended by VMware). In Memory Mode, tests included single and multi-VM configurations subjected to representative Enterprise Java workloads as well as VMware's VMmark benchmark.[8] Without memory pressure (entire workload can fit into DRAM cache), there is no difference in performance between DRAM only cases and DRAM+PMEM cases. As expected, when memory pressure and noisy neighbors are introduced, performance degrades but the degree depends on how memory-intensive the workloads are. In a single VM configuration with a workload that cannot be cached entirely in DRAM, performance degrades by 28% from the DRAM only case.

In studies of VMs configured using vSphere in App Direct Mode,[9] standard benchmarks were used in conjunction with non-PMEM-aware applications (Oracle DB and MySQL) and PMEM-aware applications (Microsoft SQL Server 2019 and customized version of Redis). Rather than using DRAM-only configurations as a baseline, comparisons were made between non-PMEM-aware applications and PMEM-aware applications. The results show that vSphere itself adds very little overhead (less than 4%). Overwhelmingly, PMEM-aware applications performed better.

Hyper-V

Intel has reported on performance studies using Windows Server 2019/Hyper-V tested with an OLTP benchmark.[10] Baseline configuration was a server with 768GB DDR4 DRAM. Test configuration was 192GB DDR4 DRAM with 1 TB Optane DC PMEM. Results indicate that the number of VMs could be increased from 22 to 30 (36% increase).

KVM

Redis Labs has conducted performance tests of Redis running on KVM with Intel Optane Persistent Memory in Memory Mode.[11] Each Redis instance of 45GB was run in a single VM. Base case was 14 Redis instances in a server configured with 768GB DRAM. Subsequent test cases increased total memory capacity by adding PMEM and reducing DRAM. Results show that with 1.5TB of total

memory (192GB DRAM) the number of Redis instances could be doubled without increasing latency. Redis performance is highly dependent on the nature of the operations so it is not clear how generalizable these results are.

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. Memory Machine has been tested in both vSphere and QEMU-KVM environments. QEMU-KVM results (vSphere results are similar) will be reported here in order to contrast the two options for how Memory Machine can be installed. Two popular applications were selected. Redis (in-memory key-value database) was tested with the memtier benchmark using Memory Machine in the Figure 2 configuration. MySQL (relational database that caches pages in memory) was tested with the Sysbench benchmark using Memory Machine in the Figure 1 configuration. Baseline data was captured using a VM with DRAM only.

Memory Machine installed inside VM

The results of the Redis benchmark tests are shown in Figure 3. The VM was configured with 8 vCPUs, 32GiB DRAM and 128GiB PMEM. A database size of 6GB was created so that there was no memory pressure. The tests highlight memory latency rather than cache management. Memtier has numerous parameters, the primary options being 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 6GB of DRAM (in addition to the PMEM) is equivalent to the baseline because the database is 6GB. In this case, the performance was the same as the DRAM only case. When the DRAM component was reduced to 4GB, 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.

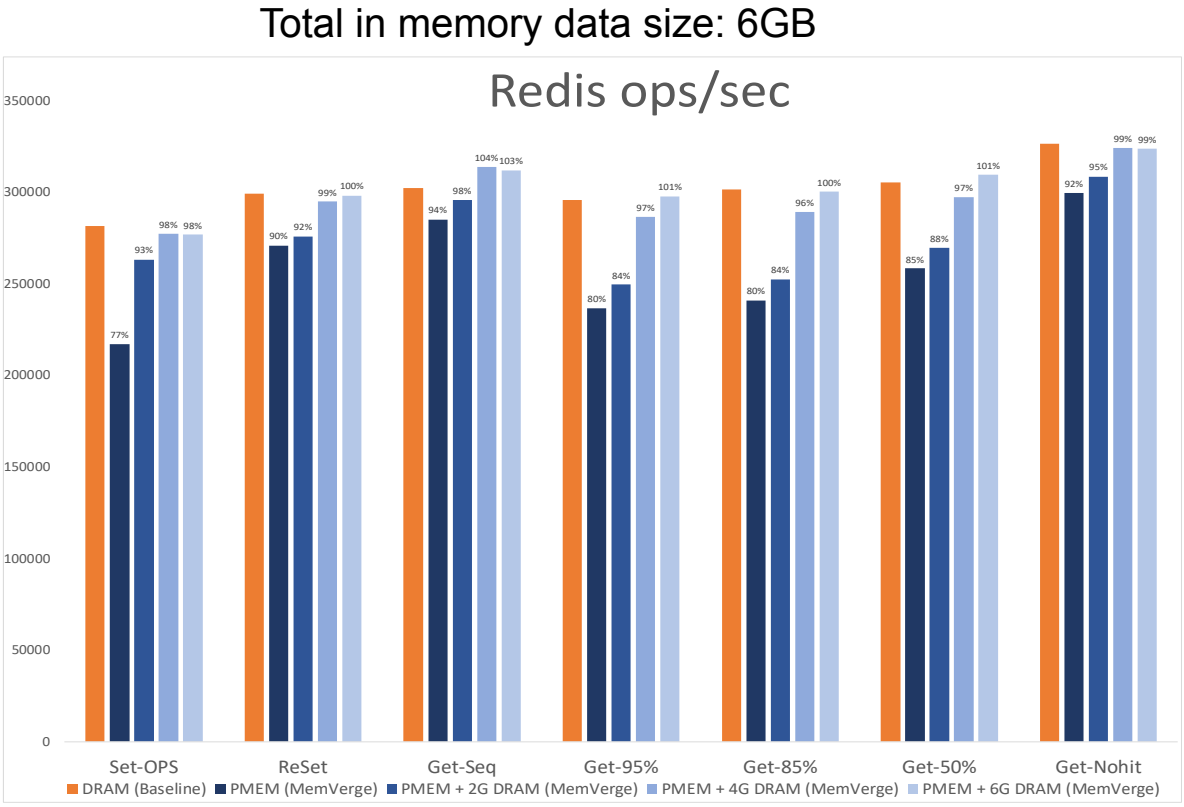

Figure 3. Redis performance measured by memtier

Since Redis is largely single threaded (background processes use multiple threads), assigning more than 6GB DRAM will not improve the performance. It would be more effective to start other Redis instances and take advantage of the 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 48GB DRAM, a saving of 33% in DRAM. Or equivalently, 8 instances could be run in 32GB rather than the 5 without Memory Machine, an increase of 66% in the number of instances.

Memory Machine installed underneath VM

The results of the MySQL benchmark tests are shown in Figure 4. In these tests, the VM 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 were within 2% of the baseline case for the smaller database, indicating that Memory Machine imposes minimal overhead. With 4GB or 2GB 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 16GB 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.

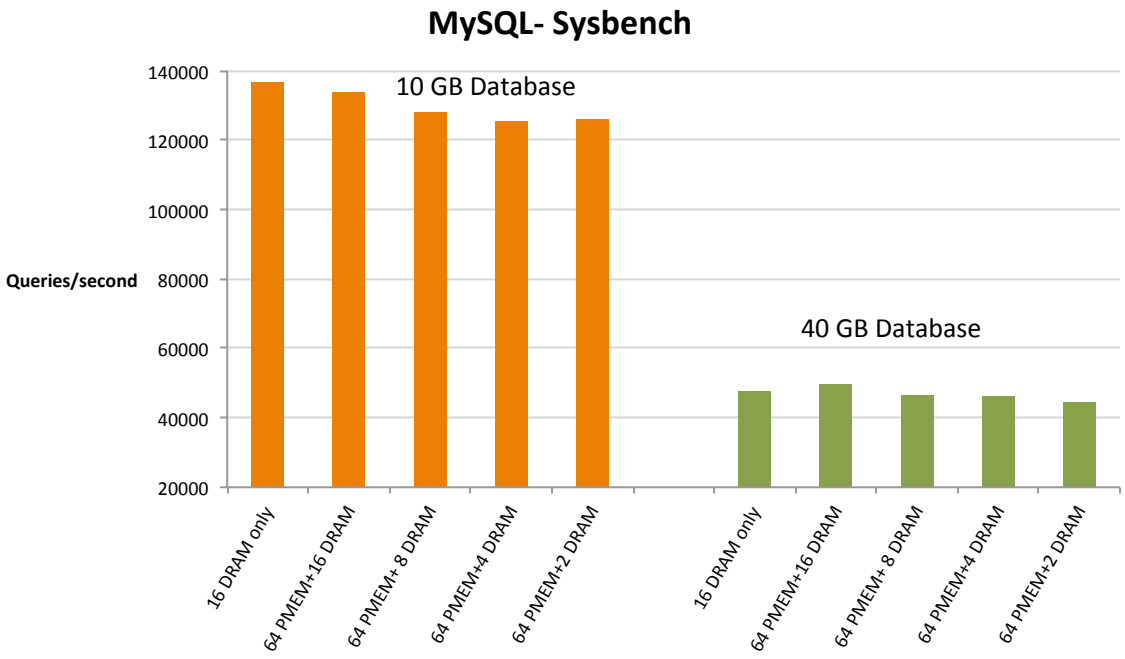

Figure 4. MySQL performance measured by Sysbench

Business Impact

Increasing VM density per server or increasing application density per VM will lead to reductions in TCO. Fewer servers translate into easier administration, lower capex, and savings in power, space and cooling.[12] By virtualizing memory, Memory Machine provides an additional component to the software defined data center (currently, storage, compute, networking and management) and contributes to its economic advantages.

MemVerge's benchmark tests suggest that application or VM density could be increased by as much as 8x without incurring significant performance degradation. Care must be taken in production environments because other system factors (non-memory related) may influence the overall scaling.

Features that utilize PMEM's persistence have demonstrable productivity impacts. Crash recovery time can be greatly reduced. A restoration that previously took 30 minutes now takes seconds. Reloading data in an iterative machine learning pipeline is often reduced from 15-20 minutes to 1-2

seconds. Application instances can be replicated in seconds allowing analyses or development to proceed in parallel.

Certain software packages are licensed per physical CPU. Examples are VMware vSphere and Oracle Enterprise Edition database (in the Oracle case, the licensing is based on the number of cores times the number of CPUs). Servers have a limited number of memory slots per CPU (typically 12 in a high end server) and there are strict rules for how the slots can be populated. When the memory limit per CPU is reached, additional CPU sockets must be populated. By using software-defined memory, more usable memory may be configured per CPU socket, potentially avoiding licensing costs for additional CPUs.

Conclusion

Memory-intensive applications benefit from more memory. Merely adding PMEM is insufficient. Memory Mode removes persistence and is susceptible to noisy neighbor problems in the DRAM cache. App Direct mode is more robust but can suffer performance issues and requires code rewrites. Through its sophisticated memory management, Memory Machine maintains persistence, achieves DRAM-like performance, alleviates noisy neighbor problems, and does not require code rewrites. Additional features, such as memory snapshots, can improve productivity.

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.

MemVerge is at the cutting edge of software-defined memory. As new memory technologies emerge, MemVerge will continue to innovate and the feature set of Memory Machine will be enriched, especially as requirements from demanding production environments are incorporated.

Appendix

This appendix provides additional background for interested readers.

Under the hood: Hypervisors

Types

A Type 1 Hypervisor (or bare-metal hypervisor) is installed directly on the server hardware and virtualizes the underlying components. A Type 2 Hypervisor is installed on an OS already running on the host computer and allows the guest OS to run as a process on the host OS (hence the view sometimes expressed that a container is akin to a Type 2 hypervisor). Type 1 hypervisors dominate data center environments. In reality, the difference between Type 1 and Type 2 is less distinct: even a Type 1 hypervisor needs an OS kernel to manage the interactions with the hardware. A Type 1 hypervisor packages a specialized OS kernel (rather than a generic OS as in Type 2) with the hypervisor.

X86 Virtualization

Full virtualization allows a guest OS to run unmodified on the hypervisor. Since the guest OS acts as if it has privileged access to the hardware, the hypervisor must intercept some kernel calls and limit their scope. For example, a reboot command from the guest OS does not mean reboot the host OS. In mainframe computers, virtualization was achieved by using a "trap and emulate" approach but quirks in Intel's x86 architecture disqualified this approach.

In 1998 VMware cracked this problem[13] by creating a hypervisor composed of a *vmkernel* (containing boot loader, device drivers, schedulers, etc.) and a *Virtual Machine Monitor (VMM)*. A VMM is a process running in user space and a separate VMM instance is created for each guest OS. The VMM can intercept any kernel call made by the guest OS and translate it into suitable instructions for the vmkernel. This is known as binary translation (BT) because binary x86 code is being translated, not source code.

Over time, both Intel and AMD provided hooks to support hardware-assisted virtualization. This relieved the VMM of much of the BT burden and hypervisor performance improved considerably.

VMs *and* Containers: Best of both worlds?

As the virtualization market has matured, VMs have become more specialized. It is now common to see discussions of choosing the right VM for the workload.[14] Although containers are often run in generic VMs, the notion of running containers inside specialized VMs is gaining traction. For example, VMware's vSphere Integrated Containers[15] allows developers to develop in containers and deploy them in specialized VMs alongside traditional VM-based workloads, all managed from the same vSphere platform.

Firecracker from AWS is an example of a lightweight VM known as a *microVM*.[16] [17] MicroVMs reduce memory footprint (to about 5 MB) and speed up start times (to fractions of a second) by eliminating unnecessary functionality in the VMM such as device drivers for USB, etc. Tools such as Ignite[18] allow containers to be managed just like Docker but to be run in a Firecracker VM, thus combining the convenience of containers with the security of VMs.

Unikernels are another approach to combine the benefits of VMs and containers but remain controversial.[19] In the Unikernel approach, a single application is compiled along with a streamlined OS, thus eliminating the distinction between the kernel and user space. The result can be deployed as a VM or run on bare metal.

Intel is supporting a development effort to develop a PMEM interface for containers. Known as Intel PMEM-CSI, it is a container-storage interface for container orchestrators like Kubernetes. It makes local PMEM available as a file system volume to container applications.

Under the hood: Persistent Memory

Flash memory is composed of an array of memory cells made from floating gate transistors. Electrons can be injected into (or released from) the floating gate thereby changing the threshold voltage required to make current flow through the transistor. The electrons are trapped in the floating gate even if the power is removed, thus making flash memory persistent.

NAND flash technology has improved in both capacity and performance primarily by extending the memory cell grid into 3D and by encoding 3 or 4 bits per cell. The highest performing NAND flash-based SSDs are packaged as modules that connect via the NVMe (Non-volatile Memory express) interface (a PCIe bus for connecting non-volatile media). Such modules can have capacities up to 15TB with average latencies < 100 microseconds.[20]

Intel and Micron jointly developed 3D Xpoint technology in order to decrease latencies and increase the endurance of NAND flash. 3D Xpoint memory is non-volatile but uses different semiconductor behavior than that used in NAND flash. Currently, Micron markets a 3D Xpoint-based SSD with a capacity of 805GB and average latency of about 10 microseconds.[21]

Intel's announcement in early 2019 of Optane DC Persistent Memory signified another major innovation in non-volatile memory (it is different from 3D Xpoint). Packaged as a DIMM that could populate a standard DDR4 slot, Optane DC Persistent Memory was offered with storage options (128GB, 256GB, and 512GB) much denser than the typical 32GB DDR4 DRAM module (although larger DRAM capacities exist). In mid-2020 Intel announced the second generation of Optane DC Persistent Memory that offers improved I/O performance.

---

[1] https://www.docker.com/resources/what-container#/package_software
[2] https://www.intel.com/content/www/us/en/architecture-and-technology/optane-dc-persistent-memory.html
[3] https://memverge.com
[4] https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-190.pdf
[5] https://helda.helsinki.fi/bitstream/handle/10138/165920/penberg-thesis.pdf
[6] https://www.intel.com/content/www/us/en/products/docs/memory-storage/optane-persistent-memory/optane-persistent-memory-200-series-brief.html
[7] https://arxiv.org/pdf/1903.05714.pdf
[8] https://www.vmware.com/techpapers/2020/pmem-balanced-profile-perf.html
[9] https://www.vmware.com/techpapers/2018/optane-dc-pmem-vsphere67-perf.html
[10] https://www.intel.com/content/www/us/en/products/docs/memory-storage/optane-persistent-memory/optane-dc-persistent-memory-brief.html
[11] https://object-storage-ca-ymq-1.vexxhost.net/swift/v1/6e4619c416ff4bd19e1c087f27a43eea/www-assets-prod/summits/26/presentations/23794/slides/redis-with-Optane-Persistent-memory-and-recorded-demo.pdf
[12] https://www.intel.com/content/dam/www/public/us/en/documents/success-stories/verizon-media-article-1-summary.pdf
[13] https://course.ece.cmu.edu/~ece845/sp18/docs/vmware-evolution.pdf
[14] https://azure.microsoft.com/en-us/services/virtual-machines/
[15] https://docs.vmware.com/en/VMware-vSphere-Integrated-Containers/1.5.6/rn/VMware-vSphere-Integrated-Containers-156-Release-Notes.html
[16] https://firecracker-microvm.github.ioh/
[17] https://arxiv.org/pdf/2005.12821.pdf
[18] https://github.com/weaveworks/ignite
[19] https://www.hpe.com/us/en/insights/articles/what-is-a-unikernel-and-why-does-it-matter-1710.html
[20] For example https://media-www.micron.com/-/media/client/global/documents/products/product-flyer/9300_ssd_product_brief.pdf?la=en&rev=b6908d03082d4fd7b022a2f40d1b731e

[21] https://media-www.micron.com/-/media/client/global/documents/products/product-flyer/x100_product_brief.pdf?rev=5117c6eb09514eae998f5f420f89b0ce

Learn More

Big Memory

IDC Big Memory Definition and PMEM Forecast Presentation

IDC Big Memory Definition and PMEM Forecast Video

The Next Platform: The Era of Big Memory is Upon Us

Webinar: Breakthroughs in Big Memory

Intel Podcast: Big Memory Software Defined Controller

Memory Machine Software

The Skinny on Memory Machine

1-Page Memory Machine Data Sheet

Demo: Creating Clones of Redis VMs in Microsoft Azure

Demo: Memory Machine Software Capabilities: Memory Snapshots and Managing from GUI and Command Line

Demo: Cloning an 800GB kdb+ Database in Seconds

Demo: See kdb+ in-memory on AWS run faster with Memory Machine software

Intel Optane Persistent Memory

Intel Optane Persistent Memory

MemVerge

MemVerge Corporate Brochure

MemVerge

@memverge

@memverge