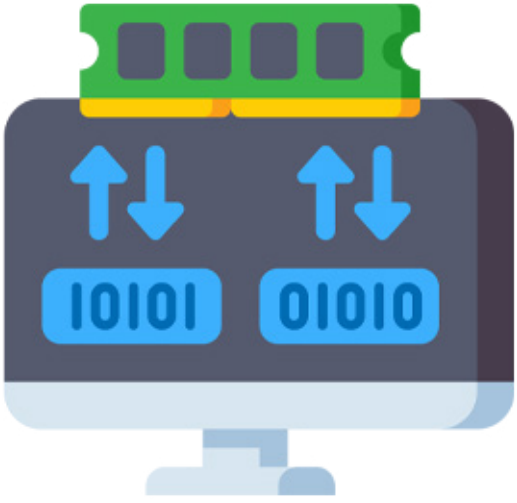
MemVerge

White Paper

Memory Machine and In-memory Databases



Memory Machine and In-memory Databases

Executive Summary

An In-Memory Database (IMDB) is a database management system that mainly stores its data collections directly in the working memory of one or more computers. IMDBs are used in applications where high-volume bursts of transactions that require microsecond responses are encountered. Persistent Memory (PMEM)[1] has larger capacity than DRAM and would seem an obvious way to improve IMDB performance but the use of PMEM is not straightforward. PMEM is slower than DRAM (nominally by a factor of three) and when DRAM is used as a global cache to improve PMEM's performance, noisy neighbor issues can occur.

Memory Machine, a software package from MemVerge,[2] can be used to manage a combination of DRAM and PMEM in such a way that the increased capacity can be used without incurring significant performance penalties. The effect of noisy neighbors can be alleviated because Memory Machine uses DRAM and PMEM in a two-tier memory hierarchy rather than as a DRAM cache in front of a PMEM pool.

Although they can be relational, many IMDBs are NoSQL. This paper describes the results obtained from testing Memory Machine with three popular NoSQL IMDBs, namely, kdb+,[3] Redis,[4] and memcached.[5] The results show that Memory Machine was able to achieve, with a fraction of the available DRAM, the same or similar results to the baseline case of all DRAM. If the DRAM was reduced further (even to zero), performance degraded. Whether the reduced performance is justified by the reduced DRAM cost is a business decision.

The results also show that the effect of noisy neighbors seen in the Intel Memory Mode[6] configuration can be avoided by using the memory management scheme implemented by Memory Machine.

Introduction

An in-memory database (IMDB) relies primarily on a computer's main memory to store data. Usually this means the entire dataset can be loaded into memory but this is not always the case. IMDBs can be relational (such as SAP HANA and Oracle TimesTen) or they can be NoSQL (such as MongoDB and Cassandra). What they do have in common is that they are optimized for high-volume, bursty transactions that require rapid responses. IMDBs find wide use in OLTP (Online Transaction Processing), on-line bidding (such as for online ad impressions), and real-time analytics (such as in gaming leaderboards).

Persistent Memory (PMEM) is a natural partner for IMDB – more memory at a lower cost – but PMEM is slower than DRAM (hundreds of nanoseconds versus tens of nanoseconds). Memory Machine, a software solution from MemVerge, can achieve DRAM-like performance from a combined pool of DRAM and PMEM. This provides applications, such as IMDB, with transparent access to larger memory capacity. DRAM can be used more efficiently so that DRAM resources can be freed up for other applications or to support additional IMDB instances.

This paper reports on results obtained from testing Memory Machine on a selection of popular NoSQL IMDBs.

Persistent Memory

Intel's Optane DC Persistent Memory is available in 128GB, 256GB and 512GB capacities and can be installed alongside traditional DDR4 DIMMs. Without Memory Machine, the simplest way to use PMEM is in Memory Mode in which DRAM is used as a cache for the slower PMEM and the OS perceives the PMEM as the only available memory.

PMEM in Memory Mode is not persistent. Not all IMDBs require persistence (in these cases data is created anew each time the database runs). IMDBs that require non-volatile storage use background processes to write transaction log updates and database snapshots to SSD or hard disk. Memory Machine retains PMEM's persistence and this can be exploited in a number of ways, for example, to provide instant restarts of an IMDB after a crash.

Databases included in the study

Results from testing the following IMDBs are included.

- kdb+

  This is a column-oriented IMDB based on the concept of ordered lists and is particularly applicable to timeseries data (for example, a stock ticker feed). Nearly every major financial institution uses kdb+.

- Redis

  Redis is a key-value IMDB that is widely used by enterprise customers in a range of industries, from finance to retail to healthcare. Cloud-based versions of Redis are offered by all major cloud service providers.

- memcached

  Memcached is a key-value IMDB for storing small data objects and strings from results of database calls, API calls, or page rendering. Although it has many applications, it was intended to speed up web applications by alleviating database load.

Test Results: kdb+

Two types of loads (actual market data queries and a synthetic load) were applied to kdb+ running on a single server using the following set-up.

Server platform
CPU: 2 x Intel Xeon Gold 6525 @ 2.10GHz (24 cores per socket)
DRAM: 160GB

PMEM: 8 x 256GB = 2TB Intel Optane DC Persistent Memory

Software
Linux: CentOS 8.1
kdb+ 3.6

Figure 1 shows the results of submitting a set of market queries to a kdb+ instance. The baseline case using DRAM only (160GB) was compared to Memory Machine managing a small pool of DRAM (5GB) plus a large pool of PMEM. The results are expressed as a ratio of the mean time to complete query using Memory Machine configuration to the equivalent time using the baseline case. In the majority of cases, Memory Machine using 5GB DRAM achieves better performance than the baseline case.

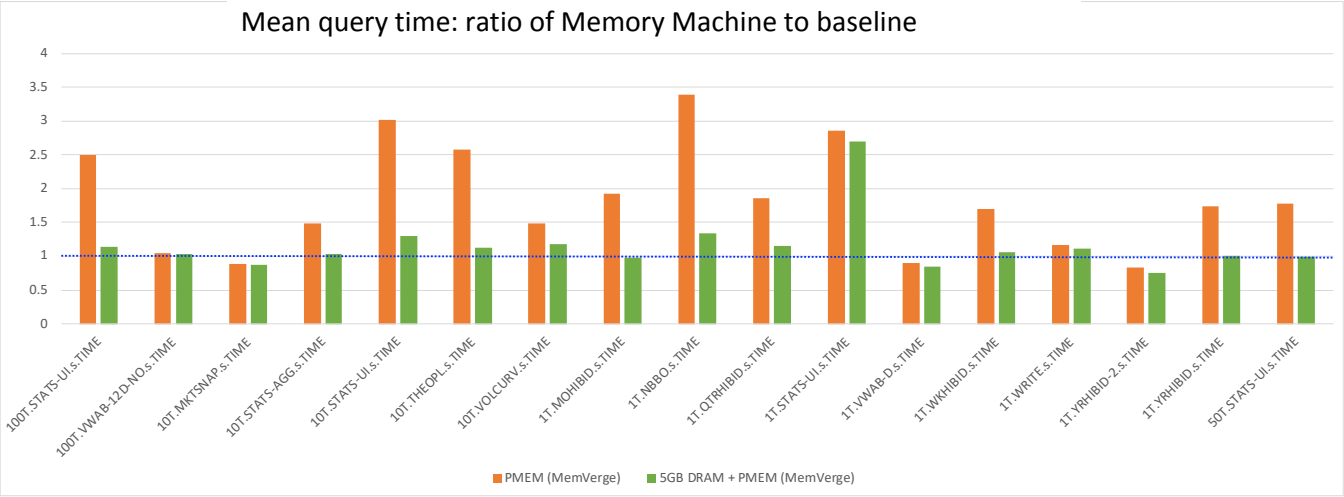


Figure 1. Mean query time for kdb+ using market data

Figure 2 shows the results where a synthetic load was used to generate bulk inserts into the database. In all but one case, the performance of Memory Machine was the same or better than the baseline case of all DRAM. As the number of inserts increased, the process became CPU-limited and the performance plateaued.

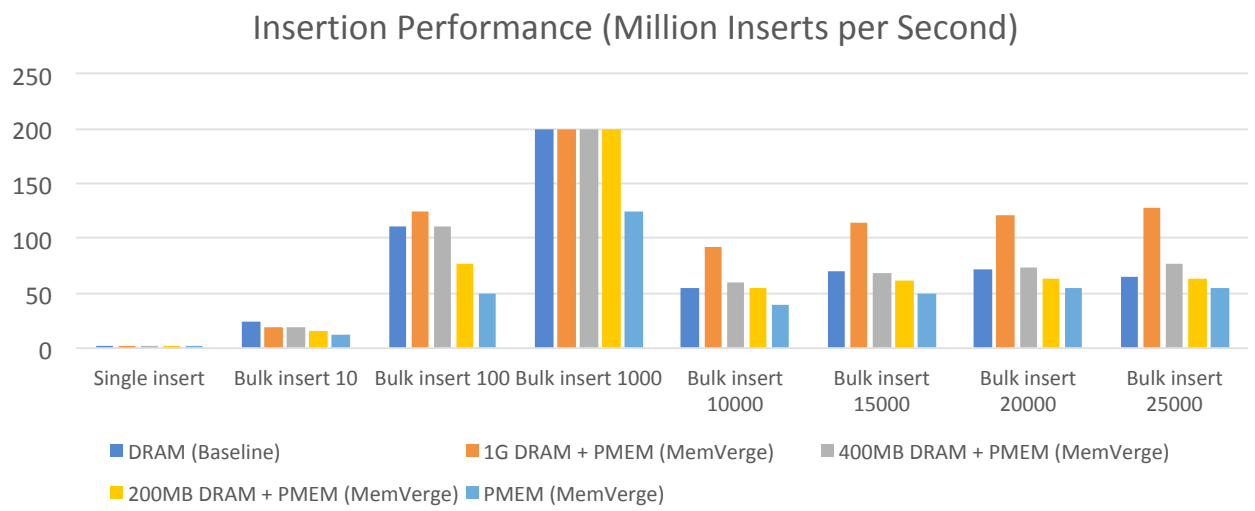


Figure 2. kdb+ Performance stress test using bulk inserts.

Memtier Benchmark

Memtier_benchmark[7] is a utility developed by Redis Labs for load generation and benchmarking NoSQL key-value databases. Redis and memcached are currently supported. Memtier has numerous options, the main variables being the relative frequency of SET and GET operations and the key pattern (random, sequential, etc.) used in the SET and GET operations. For example, a SET:GET ratio of 1:0 is used to populate (or repopulate) the database. 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.

Test Results: Redis

Memtier_benchmark tests were run on a single server using the following set-up.

Server platform

CPU: 2 x Intel Xeon Gold 8260L @ 2.20GHz (18 cores per socket)
DRAM: 12 x 16GB = 192GB DDR4
PMEM: 12 x 128GB = 1.5TB Intel Optane DC Persistent Memory

Software

Linux: RHEL Release v8.2
Redis v5.0.7

Figure 3 shows the time taken to insert data associated with random keys into an 85M key Redis database. Both 99th percentile and average latencies are shown. Baseline case was DRAM of 192GB. Also included are results obtained using Intel PMEM in Memory Mode using a DRAM cache of 192GB. Subsequent tests show results with Memory Machine using diminishing amounts of DRAM. Memory Machine with 120GB DRAM performs as well as the baseline and Memory Mode cases. Reducing the

DRAM managed by Memory Machine by roughly 70% leads to an increase in average latency of 25%. Whether incurring the increased latency to reduce the DRAM cost makes business sense would need to be evaluated by stakeholders in the enterprise.

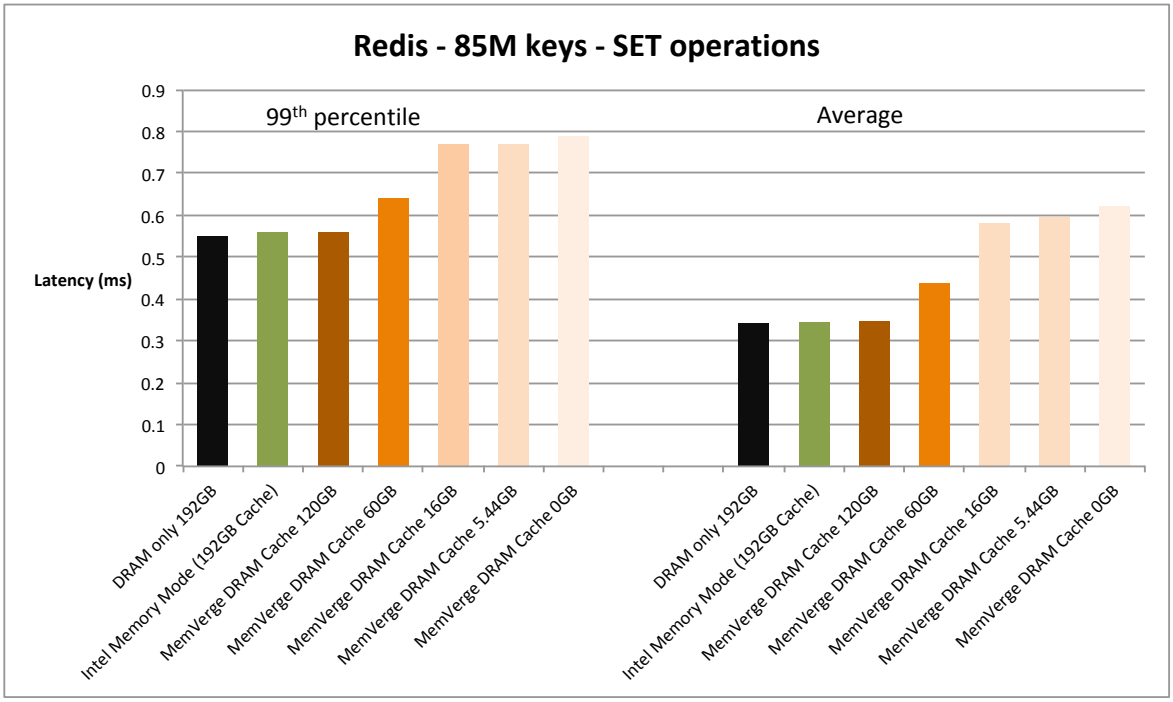


Figure 3. Latency of Redis SET operations for 85M key database

Figure 4 shows results using the same test configuration but where the metric used for comparison is operations per second. Similar conclusions can be drawn: Memory Machine with 120GB performs as well as the baseline. Reducing DRAM further (until zero) leads to increasing performance degradation.

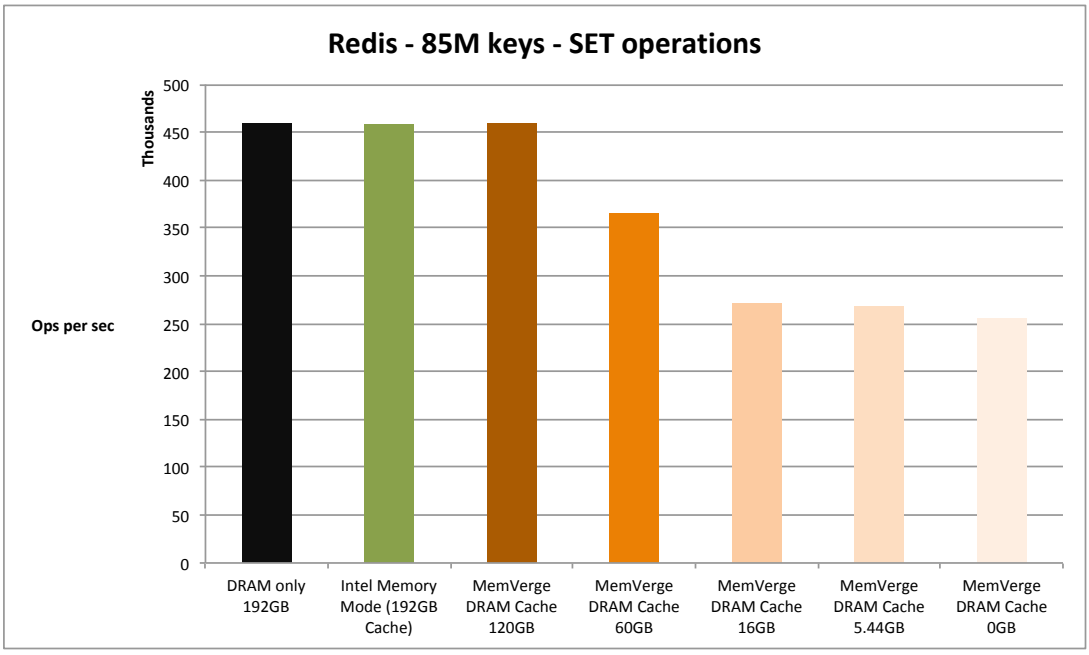


Figure 4. Operations per second for Redis SET operations for 85M key database

Figures 5 and 6 show the same tests run against a 300M key database. DRAM only case is omitted because it is equivalent to the Memory Mode case with 192GB DRAM cache. The database is too large to fit entirely into the DRAM capacity under Memory Machine management (120GB or less). Although the average latency remains the same, there is more variance. The throughput of the SET operations is lower because more of the database has to be placed in the slower PMEM tier.

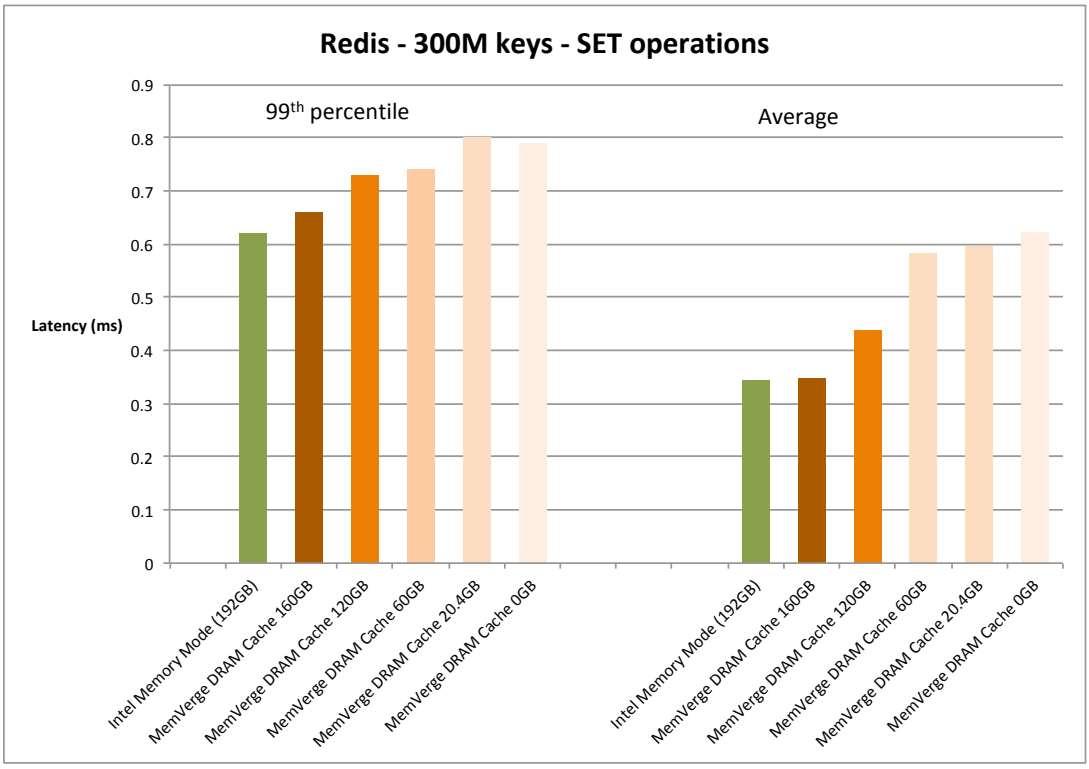


Figure 5. Latency of Redis SET operations for 300M key database

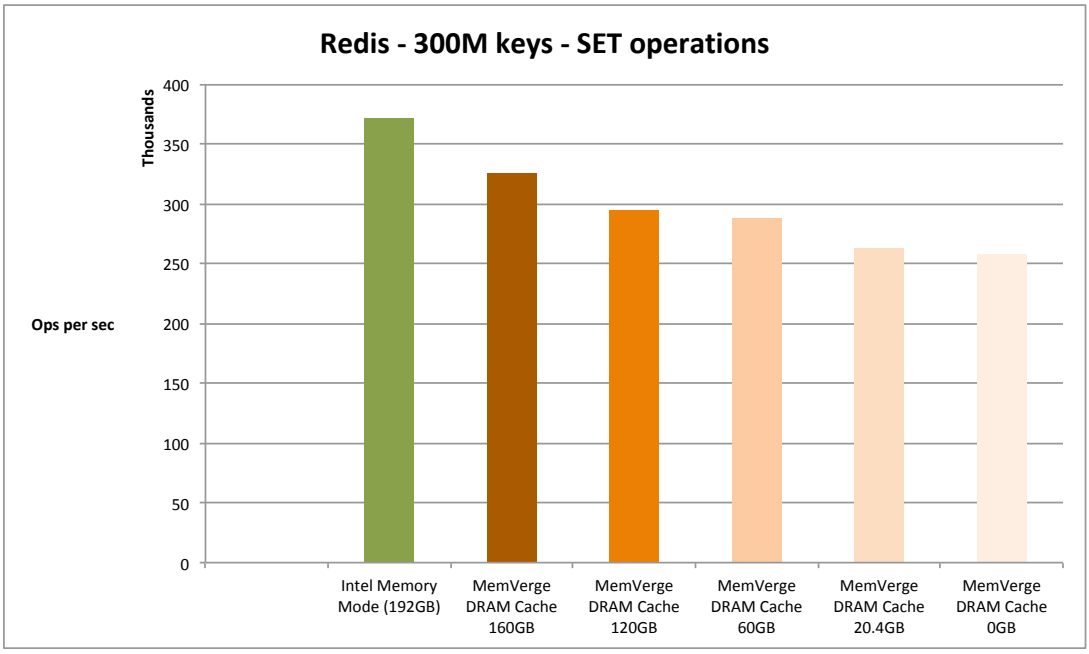


Figure 6. Operations per second for Redis SET operations for 300M key database

Test Results: memcached

Memtier_benchmark tests were run on a single server using the following set-up.

Server platform
CPU: 2 x Intel Xeon Platinum 8260L @ 2.40GHz (24 cores per socket)
DRAM: 12 x 16GB = 192GB DDR4
PMEM: 8 x 512GB = 6 TB Intel Optane DC Persistent Memory

Software
Linux: CentOS 8.1.1911
Memcached v1.5.9

Noisy neighbor effects are caused by concurrent processes competing for the same cache location. The result is an increase in the number of cache misses and a subsequent degradation in performance. Since Memory Mode uses the DRAM as a cache in front of the PMEM, Memory Mode can be susceptible. The memcached tests were used to investigate this effect.

Sixteen instances of memcached were started. For each test, four instances, bound to the same CPU core, were selected and subjected to a load generated by memtier (all SET operations using random keys). Figure 7 compares the results using DRAM capacity (384GB) greater than the capacity required by the four memcached instances (160GB). This means that all operations could be conducted in DRAM and any noisy neighbor issues exposed.

Each test was performed using Memory Machine followed by Intel Memory Mode. Then the test repeated with another group of four memcached instances. In Figure 7, the results labeled Set 1 using Memory Machine (mm) should be compared with Set 1 using Memory Mode (Intel). Similar comparisons should be made for Sets 2 through 6.

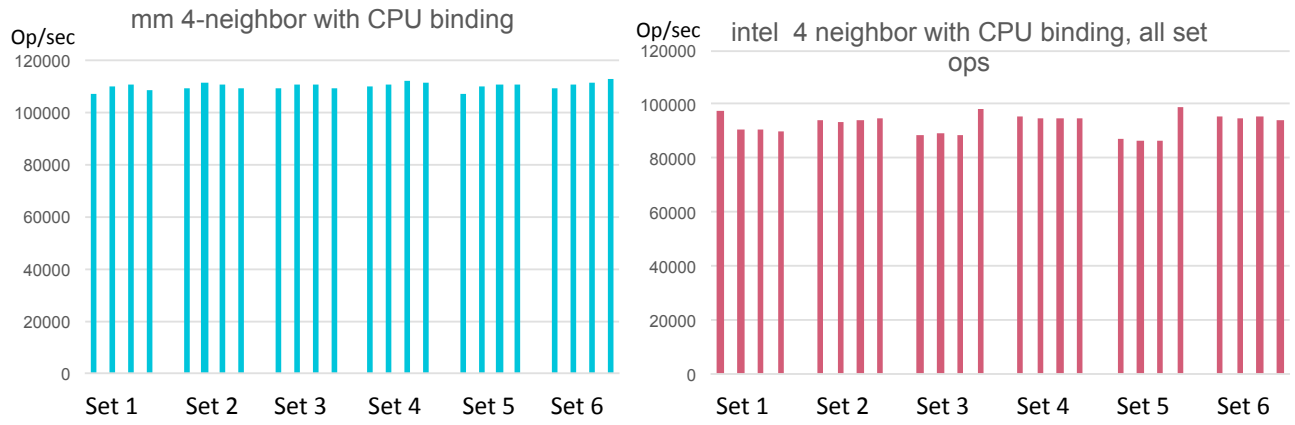


Figure 7. Operations per second for memcached SET operations for 160GB database

The results show that when using Memory Machine the number of SET operations per second is always above 100k whereas in Memory Mode the number of operations per second is always less than 100k. This shows that Memory Machine alleviated the noisy neighbor issues.

Conclusion

The test results show that for the three IMDBs investigated, Memory Machine was able to achieve, with a fraction of the available DRAM, the same or similar results to the baseline case of all DRAM. If the DRAM was reduced further (even to zero), performance was reduced also. Whether the reduced performance is justified by the reduced DRAM cost is a business decision.

The results show that the effect of noisy neighbors seen in the Memory Mode DRAM cache can be alleviated by using the memory management scheme implemented by Memory Machine.

---

[1] https://www.intel.com/content/www/us/en/products/docs/memory-storage/optane-persistent-memory/optane-persistent-memory-200-series-brief.html
[2] https://memverge.com
[3] https://kx.com/platform/#timeseries
[4] https://redislabs.com
[5] https://www.memcached.org
[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://redislabs.com/blog/memtier_benchmark-a-high-throughput-benchmarking-tool-for-redis-memcached/

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