A Study of Application Performance with Non-Volatile Main Memory

Yiying Zhang
University of California, San Diego
yiyingzhang@cs.ucsd.edu

Steven Swanson
University of California, San Diego
swanson@cs.ucsd.edu

***Abstract*—Attaching next-generation non-volatile memories (*NVMs*) to the main memory bus provides low-latency, byte-addressable access to persistent data that should significantly improve performance for a wide range of storage-intensive workloads. We present an analysis of storage application performance with non-volatile main memory (*NVMM*) using a hardware NVMM emulator that allows fine-grain tuning of NVMM performance parameters. Our evaluation results show that NVMM improves storage application performance significantly over flash-based SSDs and HDDs. We also compare the performance of applications running on realistic NVMM with the performance of the same applications running on idealized NVMM with the same performance as DRAM. We find that although NVMM is projected to have higher latency and lower bandwidth than DRAM, these difference have only a modest impact on application performance. A much larger drag on NVMM performance is the cost of ensuring data resides safely in the NVMM (rather than the volatile caches) so that applications can make strong guarantees about persistence and consistency. In response, we propose an optimized approach to flushing data from CPU caches that minimizes this cost. Our evaluation shows that this technique significantly improves performance for applications that require strict durability and consistency guarantees over large regions of memory.**

I. Introduction

Next-generation non-volatile memory (NVM) technologies, such as phase change memory (PCM) [14, 16], spin-transfer torque magnetic memories (STTMs) [12, 26, 36], and the memristor [39] offer low-latency access, high bandwidth, efficient fine-grain access, and persistence [33]. As a result, they have the potential to bridge the gap between slow, persistent storage (i.e., disks and SSDs) and fast, volatile memory (i.e., DRAM) [17, 18, 25, 39]. Table I summarizes the characteristics of different NVM technologies and compares them to traditional memory and storage technologies.

Attaching NVMs directly to processors will produce non-volatile main memories (*NVMMs*), exposing the performance, flexibility, and persistence of these memories to applications. NVMM will blur the line between traditional memory and storage and pose many challenges [3, 5, 6, 10, 19, 21, 27, 37, 38] to system designers.

978-1-4673-7619-8/15/$31.00 © 2015 IEEE

In order to properly design systems for NVMM, it is important to understand NVMM performance and how it will affect application performance.

In this paper, we use a commercial hardware-based NVMM emulator [10] to analyze the performance of storage applications running with NVMM. The emulator models the latency and bandwidth of NVMM as well as the cost of ensuring that data is safely stored in NVMM.

To gain insights into storage application performance with NVMM, we pose the following questions:

- How does NVMM performance affect application-level performance?
- Which aspects of NVMM performance have the largest impact on application-level performance?
- How can we minimize the cost of ensuring data is persistent in NVMM-equipped systems?

To answer these questions, we conduct experiments with microbenchmarks and a set of typical storage server applications, including a file server, a mail server, a web server, a NoSql database, a relational database, and Memcached [11]. We make several findings through our study.

- **Application performance with NVMM is much better than with SSDs and HDDs.** NVMM improves storage application performance significantly over flash-based SSDs and HDDs (up to 28×), even without any application changes. The benefit is especially big for applications with frequent writes and *fsync*s and for applications with large working sets. These results confirm that replacing SSDs and HDDs with NVMM will benefit a wide range of storage applications.
- **NVMM's lower-than-DRAM performance has only a small impact on applications.** NVMM's latency and bandwidth will likely be somewhat worse than DRAM, but the impact of this difference will be small. NVMMs with longer latencies and lower bandwidths than DRAM provide almost the same performance

	DRAM	NAND Flash	HDD	PCM	Other NVMs
Density (F^2)	6-12	1-4	2/3	4-16	4-60
Read Latency	10-50 ns	25 μs	5 ms	48-70 ns	10-100 ns
Write Bandwidth	1 GB/s per die	5-40 MB/s per die	100 MB/s per drive	100 MB/s per die	140 MB/s - 1 GB/s per die
Endurance (cycles)	$> 10^{16}$	10^4	$> 10^{16}$	10^9	$10^{12} - 10^{15}$
Byte Addressable	Yes	No	No	Yes	Yes
Volatile	Yes	No	No	No	No

TABLE I: **Comparison of Memory and Storage Technologies.** *Properties and performance comparison of memory, storage, and NVM [4, 22, 24, 25, 30, 34, 39]. Other NVMs include the Memristor, STTM, FeRAM, and MRAM.*

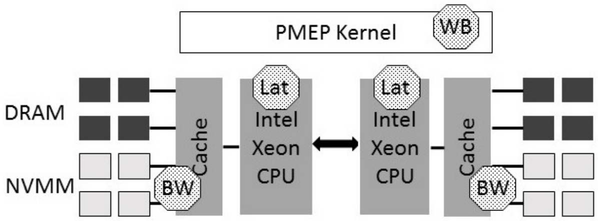

Fig. 1: **PMEP Architecture.** *Lat, BW, and WB represent the place where PMEP emulates NVMM's increased latency, reduced bandwidth, and* wbarrier *costs.*

Configuration	Description
L_{READ}	Read Latency
Th_{BW}	Read/Write Bandwidth Throttling
L_{WBAR}	Write Barrier Latency
M_{FLUSH}	*clflush* mode
M_{SYNC}	NVMM sync mode

TABLE II: **PMEP Parameters.** *The top three rows are parameters to configure the emulated hardware NVMM performance. The bottom two rows are the optimization techniques that PMEP supports and we propose.*

as idealized NVMMs with DRAM-like performance. For our applications, with the exception of Memcached, NVMM's increased latency and reduced bandwidth degrade performance by less than 10% relative to DRAM. This finding implies that further optimization of NVMM's latency and bandwidth will not be as beneficial.

- **Data persistence can be extremely costly.** Making data persistent (i.e., "syncing" it) on NVMM involves costly processor cache flush operations. This cost is especially high for applications that frequently sync large regions of memory.

Based on this last finding, we propose *Selective Persistence Flushing (SPF)*, a technique to minimize the cost of ensuring data persistence by flushing only modified data from the caches. SPF significantly improves performance for storage applications that require strict durability and consistency guarantees over large regions of memory.

The rest of the paper proceeds as follows. Section II describes the emulator we use for our study. Section III presents the performance characterization of the emulator and our optimization of the data sync process. Section IV evaluates storage applications running on NVMM. In Section V, we summarize our key observations and discuss the implications of them. Finally, we discuss related work in Section VI and conclude the paper in Section VII.

II. NVMM Emulator

To model different types of NVMMs and to study their effects in real systems, we use PMEP [10], a platform built by Intel to emulate NVMM. This section describes how PMEP emulates NVMM and its parameters. Figure 1 depicts the architecture of PMEP.

PMEP augments an off-the-shelf, dual-socket server platform with special CPU microcode and custom firmware. It partitions the system's DRAM into emulated NVMM and regular DRAM. Each processor uses four DDR3 channels. PMEP treats channels 0-1 as regular DRAM and channels 2-3 as emulated NVMM. PMEP hides the memory in channels 2-3 from the OS and exposes it as a separate memory area to the kernel.

PMEP emulates NVMM read latency, read and write bandwidth, and data persistence costs. For latency and bandwidth, PMEP modifies CPU and memory controller. It uses software to emulate data persistence costs. Table 2 summarizes the parameters of PMEP.

A. NVMM Latency

Current projections indicate that most future NVMM technologies will have higher latency than DRAM [33]. PMEP emulates NVMM read latency (L_{READ}) by increasing the number of cycles the CPU stalls for the data from NVMM on a cache miss. For example, if the CPU stalls S_{DRAM} cycles for a read from DRAM with latency L_{DRAM}, PMEP stalls the CPU for $S_{DRAM} \times L_{READ}/L_{DRAM}$ form NVMM. Since the PMEP platform uses write-back CPU caches, NVMM write latency does not directly affect the application latency. Thus, PMEP does not emulate NVMM write latency and emulates NVMM's write performance by limiting its write bandwidth.

B. NVMM Bandwidth

Due to increased latency and power limitations, NVMM will likely provide less bandwidth than DRAM [33]. PMEP's memory controller artificially restricts memory read and write bandwidth by limiting the maximum DDR transaction rate. The Th_{BW} parameter controls this setting.

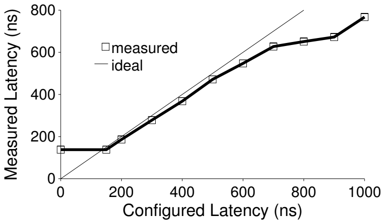

Fig. 2: **NVMM Load Latency.** *Measured NVMM load latency when varying L_{READ}. The thin line represents the expected latency with perfect emulation.*

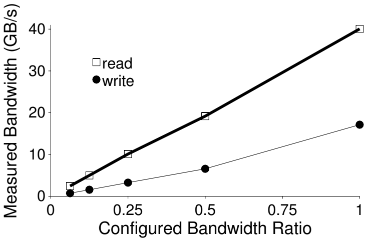

Fig. 3: **NVMM Bandwidth.** *Measured NVMM read and write bandwidth when varying Th_{BW} from 1/16 to 1× of the DRAM bandwidth.*

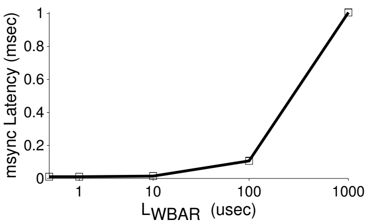

Fig. 4: ***msync* Latency with Different L_{WBAR}.** *The latency of a one-byte modification followed by an* msync *for different values for L_{WBAR}.*

C. Data Persistence

In an NVMM-equipped system, data is only persistent if it resides in the NVMM rather than the processors' volatile caches. Application code or the kernel running on PMEP can emulate the cost of making their data persistent by issuing sync calls like *msync* and *fsync*. The PMEP kernel module emulates the sync operation by flushing data from CPU cache lines using a sequence of *clflush* instructions followed by an *sfence* to enforce ordering. PMEP then issues an emulated persistent barrier, *wbarrier*, which ensures the durability of the flushed data in NVMM. To model different *wbarrier* costs (L_{WBAR}), the PMEP kernel module adds a delay before the *wbarrier*.

The *clflush* instruction that PMEP uses by default is expensive [10] because it makes strong ordering guarantees. PMEP provides an alternative CPU cache flush mechanism, *Weakly-Ordered clflush (WOF)*. WOF emulates the performance of a weakly ordered cache flush operation by performing a read followed by a non-temporal write, which flushes and invalidates the cache line.

Besides these two methods provided by PMEP, there are also two instructions recently added by Intel for efficient cache flushes, *clflushopt* and *clwb* [8]. The *clflushopt* instruction reduces the ordering guarantee over *clflush* and only orders cache flushes by store-fence operations such as *sfence*. The *clwb* instruction also orders cache flushes only by store-fence operations, but does not force to throw away cache lines from the cache.

III. Emulator Characterization and Optimization

This section characterizes the PMEP platform, discusses how its configuration parameters affect performance, and presents an optimization to mitigate the cost of ensuring persistence in NVMM.

The PMEP platform in our experiments has two 2.6 GHz 8-core Intel Xeon processors, 40 MB of aggregate CPU cache, 8 GB of DDR3 DRAM used as normal DRAM, and 128 GB of DRAM used as emulated NVMM. The platform also has a 240 GB Intel 520-series SSD and a 7200 RPM 4 TB hard disk. The platform runs Ubuntu 13.10 with a 3.11 Linux kernel with PMEP-specific changes. The customized kernel performs *clflush*'s (or other cache flush instructions) to the data in an application *msync* call and issues an *sfence* and a write barrier after completing the *clflush*'s.

A. Load and Store

PMEP emulates NVMM's increased read latency and reduced bandwidth. We change L_{READ} and Th_{BW} and use the Intel mlc tool [13] to evaluate the resulting memory access latency and bandwidth.

Figure 2 plots the measured memory load (read) latency when changing L_{READ} from 0 ns to 1000 ns. The measured latency when L_{READ} is below 150 ns stays the same, suggesting that 150 ns is the actual DRAM read latency and the lowest latency PMEP can emulate. The measured latency stays close to the configured latency from 150 ns up to 600 ns. Since most projections of next-generation NVMs fall into this range of latency performance (1-4× latency compared to DRAM), PMEP can accurately emulate a range of potential NVMMs.

Figure 3 plots the memory load and store (read and write) bandwidth with PMEP configured to provide between 1 and 1/16 of the default DRAM bandwidth. The data shows that PMEP accurately emulates NVMM bandwidth reductions.

B. NVMM Sync

Since the caches in an NVMM-equipped systems are volatile, system designers must take special care to ensure that data the program believes is persistent actually resides in NVMM. This requires flushing the data from the caches and issuing a *wbarrier* that ensures data durability on NVMM before program execution continues. To measure the costs of ensuring data persistence, we use a microbenchmark that repeatedly writes one byte and issues an *msync* for a data area containing the byte.

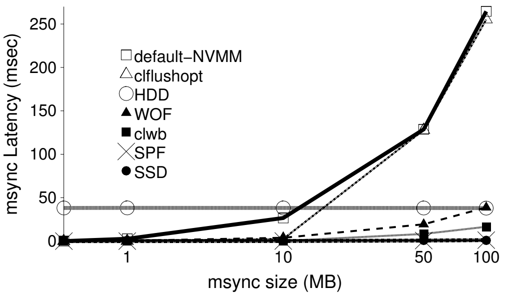

Fig. 5: **msync Performance.** *The latency of performing one-byte modification followed by an* msync *to a memory region ranging from one byte to 100 MB.*

To understand the impact of *wbarrier* latency, we change the L_{WBAR} parameter in PMEP and measure the average latency to write and then *msync* one byte. Figure 4 presents these latencies. As expected, the average latency increases with higher L_{WBAR}, since a *wbarrier* is issued for each *msync* call. Therefore, PMEP can emulate the cost of hardware *wbarrier* with its software delay.

To understand the cost of flushing data from cache, we perform a similar experiment of modifying one byte and issuing an *msync*, but we vary the size of memory region in *msync*. Figure 5 shows the average latency of this operation for NVMM, SSD, and HDD, for *msync* regions ranging in size from one byte to 100 MB. For NVMM, we test four different ways of flushing CPU caches: using the PMEP default *clflush* instruction (default-NVMM), using non-temporal write (WOF), using the *clflushopt* instruction (clflushopt), and using the *clwb* instruction (clwb).

For all these four methods, the *msync* performance cost rises linearly with the memory region's size. Default-NVMM and clflushopt both perform poorly when the memory region size is big. For memory regions larger than 10 MB, their performance can be even worse than executing *msync* on HDD. WOF and clwb perform better than default-NVMM and clflushopt. However, their *msync* latency still increase linearly with memory region size, approaching the *msync* latency of an HDD when the *msync* size is 100 MB.

The poor *msync* performance with large *msync* size is because the default implementation of *msync* in PMEP issues *clflush* for each cache line in the *msync* area. However, many of these flushes are unnecessary, since only modified data needs to be flushed from the CPU cache. The average latencies for both SSD and HDD stay constant when varying *msync* size, since, for those devices, the kernel only flushes the page containing the updated byte from the page cache.

C. *Reducing NVMM Sync Costs*

To reduce the cost of data persistence, we propose *Selective Persistence Flushing (SPF)*, an optimization technique to flush only modified cache lines rather than all of the cache lines in a sync call.

When an application makes a sync call, we check the dirty bit in the page table for each memory page in the sync region. If the dirty bit is set, the page has been modified. We flush the page from the CPU cache and reset the dirty bit. For sync region that is smaller than a page, we only flush the cache lines in the region, if the page is dirty. But we do not reset the dirty bit in this case, since the page can still contain other dirty data.

Figure 5 shows the *msync* performance of SPF. SPF significantly reduces the cost of *msync* as compared to the default PMEP *msync* and the other optimized cache flushing methods, especially with large *msync* regions. SPF only flushes the modified byte instead of the whole *msync* region. Thus, its performance does not grow with increasing *msync* region size.

IV. Application Performance

This section evaluates the performance of storage applications using NVMM. Table III summarizes the applications and workloads we use in our study. These applications include a file server, a mail server, a web server, a NoSql database, a relational database, and Memcached.

Table IV summarizes the NVMM configurations we use for our application study. To provide a baseline, we start with a system that models NVMM with the same performance as DRAM (PDRAM). Since it models NVMM, PDRAM performs cache flushes and barriers that are necessary to ensure data persistence. Then, we use PMEP to test configurations that increase NVMM latency (PM-Lat), reduce bandwidth (PM-BW), and, finally, increase *wbarrier* (NVMM-Raw). NVMM-Raw models all of NVMM's projected characteristics. Next, we add two optimizations, WOF and SPF, to NVMM-Raw and measure their impact on performance. The final, fully optimized system is NVMM-Opt.

We also compare the performance of applications running on NVMM, flash-based SSD, HDD, and DRAM. For applications that run on a file system (all the applications except Memcached in Table III), we use the Persistent Memory File System (PMFS) [10] for NVMM, *ext4* for SSD and HDD, and *tmpfs* for DRAM. The DRAM configuration differs from PDRAM since it does not include cache flushes or barriers for persistence. The key difference between PMFS and conventional file systems is that physical pages of NVMM reside permanently in the kernel's address space rather than being paged between storage and the buffer cache.

Application	Type	Workloads	Data Size
FileBench	file system	FileServer, WebServer, and Varmail	1.1 GB,64 MB,700 MB
MongoDB	nosql DBMS	YCSB	170 MB
MySQL	relational DBMS	TPC-C	9.5 GB
Memcached	key-value cache	Memtier	323 MB

TABLE III: **NVMM Applications.** *NVMM applications, their types, the workloads we use in our evaluation to represent them, and the total size of these workloads. FileBench, MongoDB, and MySQL use NVMM as persistent storage, while Memcached uses NVMM as a big memory.*

Configuration	L_{READ} (ns)	Th_{BW}	L_{WBAR} (μs)	M_{FLUSH}	M_{SYNC}
PDRAM	150	1	0	default	default
PM-Lat	300	1	0	default	default
PM-BW	300	1/8	0	default	default
NVMM-Raw	300	1/8	1	default	default
NVMM-WOF	300	1/8	1	WOF	default
NVMM-Opt	300	1/8	1	WOF	SPF

TABLE IV: **Selected NVMM Configurations.** *The NVMM configurations we use for our study. Th_{BW} represents the ratio to DRAM bandwidth.*

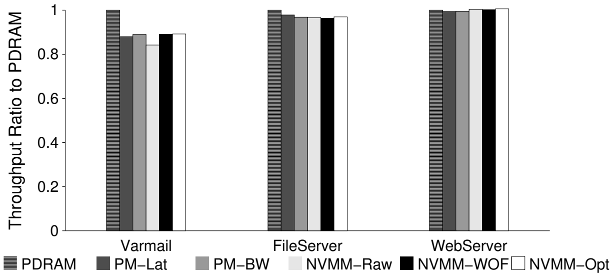

Fig. 6: **Filebench Throughput with Different NVMM Configurations.**

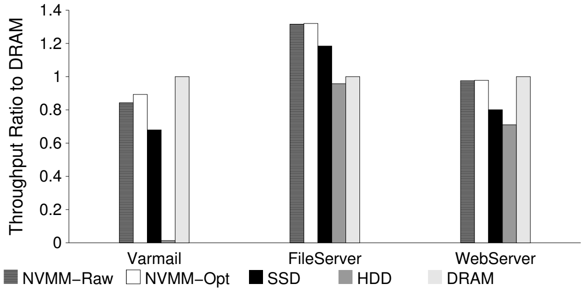

Fig. 7: **Filebench Throughput with Different Media.**

A. FileBench

We evaluate NVMM with PMFS running file system applications using the Varmail, FileServer, and WebServer workloads from the Filebench suite [31]. These workloads models typical mail server, file server, and web server access patterns.

Figure 6 plots the Filebench throughput of different NVMM configurations. Performance with the NVMM configurations are similar for FileServer and WebServer, but PM-Lat lowers Varmail performance by 12%.

Figure 7 shows Filebench throughput for NVMM, SSD, HDD, and DRAM. For all three workloads, NVMM outperforms SSD and HDD. Its performance is worse than DRAM for Varmail and similar for WebServer. Surprisingly, NVMM outperforms DRAM for FileServer by 32%. We believe this is because PMFS is more efficient than tmpfs for some operations and because PMFS directly maps NVMM without a buffer cache while tmpfs moves data between the DRAM file store and the buffer cache and doubles the memory copying costs.

Workload	Read	Update	Scan	Insert	Read&Update
A	50	50	-	-	-
B	95	5	-	-	-
C	100	-	-	-	-
D	95	-	-	5	-
E	-	-	95	5	-
F	50	-	-	-	50

TABLE V: **YCSB Workload Properties.** *The percentage of different operations in each YCSB workload.*

B. MongoDB

MongoDB [20] is a NoSql database that stores its data in memory-mapped files and use memory loads and stores for data access. By default, MongoDB uses its JOURNALED mode, in which it logs data to a journal file so it can respond to the client quickly. Then, it lazily checkpoints the data to a global data file. MongoDB also supports a FSYNC_SAFE mode, which only uses the data file without journaling.

YCSB [7] is a benchmark that evaluates key-value stores. YCSB includes six workloads that imitate web application data access patterns. Table V summarizes the YCSB workloads. Each workload performs 1000

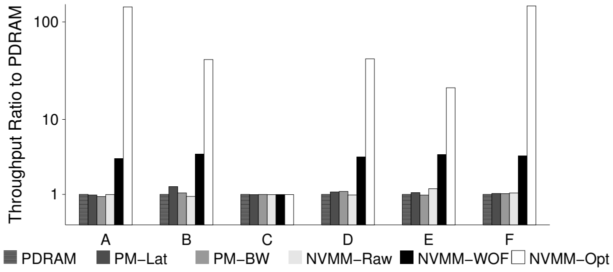

Fig. 8: **YCSB Throughput with Different NVMM Configurations on FSYNC_SAFE MongoDB.**

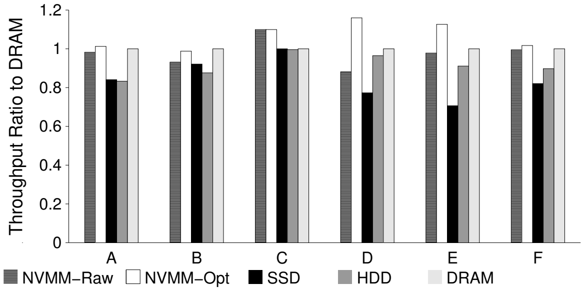

Fig. 10: **YCSB Throughput with Different Media on JOURNALED MongoDB.**

operations on a database with 1000 1 kB records.

Figure 8 presents the YCSB throughput of different NVMM configurations using the MongoDB FSYNC_SAFE mode. For the five workloads that contain writes, NVMM-WOF provides a small improvement over NVMM-Raw, and NVMM-Opt has a significant improvement (notice the log scale on the Y axis). MongoDB's FSYNC_SAFE mode *fsync*'s the MongoDB data file after each write operation and blocks the client call until this operation completes. Without SPF, the system performs *clflush* to the whole data file, while SPF only flushes the changed data and thus enables significantly improved performance.

Figure 9 plots the YCSB throughput of NVMM, SSD, HDD, and DRAM using MongoDB FSYNC_SAFE. NVMM-Opt outperforms SSD and HDD significantly, while NVMM-Raw is worse than SSD and HDD for write-intensive workloads.

We also evaluate NVMM performance with MongoDB's JOURNALED mode. Figure 10 plots the comparison of NVMM, SSD, HDD, and DRAM. The performance of NVMM-Opt is close to or better than DRAM (1% worse to 16% better). NVMM's improvement over SSD or HDD with JOURNALED is smaller than with FSYNC_SAFE, because with JOURNALED, MongoDB does not perform *fsync* and only waits for the write to the journal file, which is small enough to fit in the page cache.

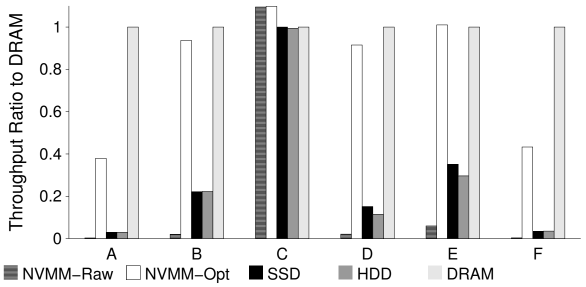

Fig. 9: **YCSB Throughput with Different Media on FSYNC_SAFE MongoDB.**

C. MySQL

MySQL [32] is a widely-used relational database. To exercise MySQL, we use TPC-C [35], a benchmark that models database online transaction processing. Our configuration uses a middle-size database with 100 warehouse, 100 clients, and 9.4 GB of data.

Figure 11 plots the TPC-C throughput when storing the database on NVMM with different configurations, SSD, HDD, and DRAM. Overall, NVMM delivers better performance than SSD and HDD by 1.8× and 28×, and is only 10% slower than DRAM. Across the NVMM configurations, bandwidth has the biggest impact on TPC-C throughput, followed by read latency. SPF improves the throughput by 7% over NVMM-Raw.

From the results of MySQL and MongoDB, we find SPF to be most effective with database applications, since database workloads often sync data frequently to ensure strict durability and consistency.

D. Memcached

Memcached [11] is an in-memory key-value store used widely to cache data for large web sites. Memcached uses NVMM as an extension to main memory that provide more capacity, but lower performance. To evaluate NVMM's impact on Memcached performance, we use Memtier [28], a benchmark for key-value stores. We use set-to-get ratios of 1:6 and 1:30 [1, 2], 10-500 byte data size, and 100000 requests.

Figure 12 plots the throughput of Memcached with different NVMM latency and bandwidth configurations. As expected, the Memcached throughput drops with higher read latency and lower bandwidth. Compared to DRAM, a typical NVMM configuration (NVMM-Raw) causes 23% performance overhead. Even though NVMM performs worse than DRAM, it is still useful as main memory, since it offers more capacity than DRAM, which is crucial for large-scale applications [23].

V. Summary and Discussion

This section summarizes our findings from our evaluation and discusses their systems implications.

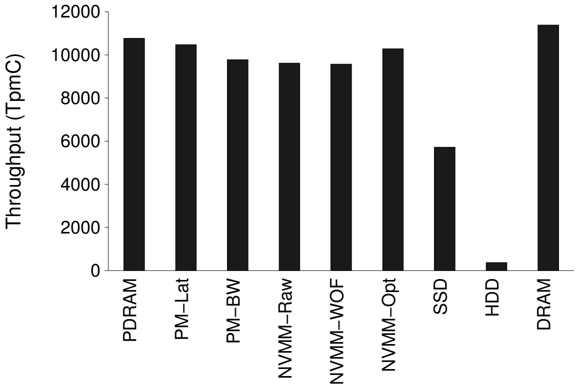

Fig. 11: **MySQL TPC-C Throughput.** *Throughput is measured in Transactions per Minute (TpmC).*

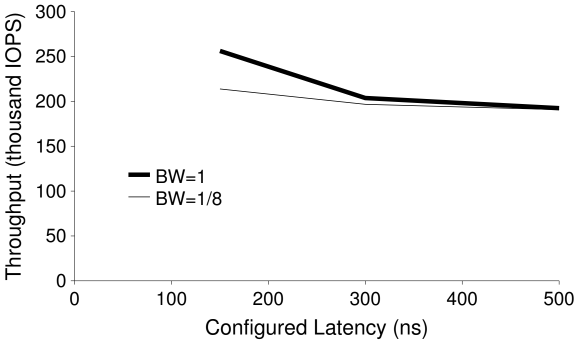

Fig. 12: **Memcached Throughput.**

First, NVMM improves storage application performance significantly over SSD and HDD. NVMM's higher latency and lower bandwidth than DRAM does not have big impact on storage applications. In most cases, storage application performance with NVMM configuration is very close to the performance of an idealized NVMM with DRAM configuration. As described in a recent survey of non-volatile memory technology trends [33], the density of promising NVM technologies continue to go up and some of them are approaching the density of NAND flash. Therefore, NVMM could serve as a very fast storage device for storage applications.

Second, the cost of data persistence can be a major factor in NVMM performance degradation. Both the performance of flushing data to NVMM and the amount of data to flush can affect storage application performance. Optimized cache flush instructions help reduce the cost of making data persistent. Our selective cache flushing technique significantly reduces the amount of data to flush. These optimizations are especially effective with database applications, since they often sync data frequently for strong consistency and durability.

Finally, when using NVMM as a big memory for memory-intensive applications, we see a small performance drop over DRAM. This performance drop is still acceptable for many applications. Therefore, NVMM will also be useful for applications that need a large memory or require lower energy consumption.

VI. Related Work

Recent years have seen increased interest in NVMM [3, 5, 6, 10, 19, 21, 25, 27, 37, 38]. This section discusses previous NVMM evaluations and how they differ from our work.

Simulation is a common method used in NVMM-related research [6, 15, 19, 25, 27]. Simulators can only run traces but not real-time workloads and benchmarks. Moreover, they often lack or simplify real system properties and effects. We use a hardware emulator to study storage application performance. Emulation provides a more realistic environment by using real system software and hardware.

Another method used in previous research [6, 29, 38] is to treat unmodified DRAM as NVMM. These works do not model NVMM's higher latency, lower bandwidth, or data persistence costs. We study these properties and their effects on storage applications.

Several works on NVMM [4, 5, 9, 10, 37] used emulation for their evaluation. Mnemosyne [37] only emulates NVMM's slower writes and write barrier but not reads. PMBD [4] uses a similar method to model NVMM latency, but does not model its data persistence costs. NV-Heaps [5] use a sampling-based emulation approach combined with a processor simulator to estimate performance costs. The emulator we use models NVMM's latency, bandwidth, and data persistence costs with customized hardware and software.

Two recent works [9, 10] also use the PMEP system. PMFS [10] uses PMEP to evaluate their proposed NVMM-oriented file system. DeBrabant *et al.* [9] evaluated NVMM performance for database systems with MySQL and an NVMM-optimized database. Their study only uses two NVMM configurations with increased latency and compare them with DRAM. We offer a detailed analysis of the impact of various NVMM configurations on storage applications, databases, and Memcached. We also evaluate several different optimizations of the data persistence process.

Finally, concurrently to this work, Sehgal *et al.* conducted a study of file system performance on NVMM [29]. Their focus is on evaluating different file systems and various file system configurations and they only use DRAM as the stand-in for NVMM. We study a wide range of applications and use a hardware emulator to evaluate the effect of various NVMM performance characteristics.

VII. Conclusion

We present a study of NVMM for storage server applications using a hardware NVMM emulator. Our evaluation results show that NVMM improves the performance of storage applications over SSD and HDD. However, the cost of making data persistent on NVMM can significantly reduce the performance of applications that require strict durability and consistency guarantees over large regions of memory. Our proposed optimization technique solves this problem by selectively flushing data. The resulting system has similar or even better performance than DRAM for certain storage applications.

Acknowledgments

We thank the anonymous reviewers for their enormously valuable feedback and comments, which have substantially improved the content and presentation of this paper. We also thank Dulloor Subramanya, Jeff Jackson, and the vLab team from Intel Corp. for their help with the PMEP platforms. Finally, we thank the members of the NVSL research group for their insightful comments.

This work was supported in part by the Center for Future Architectures Research (C-FAR), one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of C-FAR or other institutions.

References

[1] D. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan. FAWN: A Fast Array of Wimpy Nodes. In *Proceedings of the 22nd ACM Symposium on Operating Systems Principles (SOSP '09)*, Big Sky, Montana, October 2009.

[2] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny. Workload Analysis of a Large-scale Key-value Store. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '12)*, London, United Kingdom, June 2012.

[3] K. Bailey, L. Ceze, S. D. Gribble, and H. M. Levy. Operating System Implications of Fast, Cheap, Non-volatile Memory. In *Proceedings of the 13th USENIX Conference on Hot Topics in Operating Systemsi (HotOS '13)*, Napa, California, May 2011.

[4] F. Chen, M. P. Mesnier, and S. Hahn. A Protected Block Eevice for Persistent Memory. In *Proceedings of the 2014 IEEE Symposium on Mass Storage Systems and Technologies (MSST '14)*, Santa Clara, California, June 2014.

[5] J. Coburn, A. M. Caulfield, A. Akel, L. M. Grupp, R. K. Gupta, R. Jhala, and S. Swanson. NV-Heaps: Making Persistent Objects Fast and Safe with Next-generation, Non-volatile Memories. In *Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '11)*, New York, New York, March 2011.

[6] J. Condit, E. B. Nightingale, C. Frost, E. Ipek, D. Burger, B. C. Lee, and D. Coetzee. Better I/O through Byte-Addressable, Persistent Memory. In *Proceedings of the 22nd ACM Symposium on Operating Systems Principles (SOSP '09)*, Big Sky, Montana, October 2009.

[7] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking Cloud Serving Systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC '10)*, New York, New York, June 2010.

[8] I. Corp. Intel Architecture Instruction Set Extensions Programming Reference. https://software.intel.com/sites/default/files/managed/0d/53/319433-022.pdf.

[9] J. DeBrabant, J. Arulraj, A. Pavlo, M. Stonebraker, S. Zdonik, and S. Dulloor. A Prolegomenon on OLTP Database Systems for Non-Volatile Memory. In *Proceedings of the 5th International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures (ADMS '14)*, 2014.

[10] S. R. Dulloor, S. Kumar, A. Keshavamurthy, P. Lantz, D. Reddy, R. Sankaran, and J. Jackson. System software for persistent memory. In *Proceedings of the EuroSys Conference (EuroSys '14)*, Amsterdam, The Netherlands, April 2014.

[11] B. Fitzpatrick. Distributed Caching with Memcached. *Linux Journal*, 2004(124):5, August 2004.

[12] M. Hosomi, H. Yamagishi, T. Yamamoto, K. Bessho, Y. Higo, K. Yamane, H. Yamada, M. Shoji, H. Hachino, C. Fukumoto, et al. A Novel Nonvolatile Memory with Spin Torque Transfer Magnetization Switching: Spin-RAM. In *Electron Devices Meeting, 2005. IEDM Technical Digest. IEEE International*, pages 459–462, 2005.

[13] Intel Corporation. Intel Memory Latency Checker. https://software.intel.com/en-us/articles/intelr-memory-latency-checker.

[14] B. G. Johnson and C. H. Dennison. Phase Change Memory, September 2004. US Patent 6,791,102.

[15] S. Kannan, A. Gavrilovska, and K. Schwan. Reducing the cost of persistence for nonvolatile heaps in end user devices. In *Proceedings of the 20th IEEE International Symposium on High*

Performance Computer Architecture (HPCA '14), Orlando, Florida, February 2014.

[16] B. C. Lee, P. Zhou, J. Yang, Y. Zhang, B. Zhao, E. Ipek, O. Mutlu, and D. Burger. Phase-change technology and the future of main memory. *IEEE micro*, 30(1):143, 2010.

[17] M.-J. Lee, C. B. Lee, D. Lee, S. R. Lee, M. Chang, J. H. Hur, Y.-B. Kim, C.-J. Kim, D. H. Seo, S. Seo, et al. A Fast, High-Endurance and Scalable Non-Volatile Memory Device Made from Asymmetric Ta2O(5-x)/TaO(2-x) Bilayer Structures. *Nature materials*, 10(8):625–630, 2011.

[18] Micron Technology Inc. P8P Parallel Phase Change Memory (PCM). http://www.micron.com/~/media/Documents/Products/Data%20Sheet/PCM/p8p%_parallel_pcm_ds.pdf.

[19] J. C. Mogul, E. Argollo, M. Shah, and P. Faraboschi. Operating System support for NVM+DRAM Hybrid Main Memory. In *Proceedings of the 12th Workshop on Hot Topics in Operating Systems (HotOS '09)*, Monte Verita, Switzerland, May 2009.

[20] MongoDB Inc. MongoDB. http://www.mongodb.org/.

[21] I. Moraru, D. G. Andersen, M. Kaminsky, N. Tolia, P. Ranganathan, and N. Binkert. Consistent, Durable, and Safe Memory Management for Byte-addressable Non Volatile Main Memory. In *Conference on Timely Results in Operating Systems (TRIOS '13)*, Farmington, Pennsylvania, November 2013.

[22] H. Nazarian. Crossbar Resistive Memory: The Future Technology for NAND Flash. http://www.crossbar-inc.com/assets/img/media/Crossbar-RRAM-Technology-Whitepaper-080413.pdf.

[23] R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab, D. Stafford, T. Tung, and V. Venkataramani. Scaling Memcache at Facebook. In *In Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI '13)*, Lombard, IL, April 2013.

[24] Numonyx Inc. Phase Change Memory. http://www.pdl.cmu.edu/SDI/2009/slides/Numonyx.pdf.

[25] M. K. Qureshi, V. Srinivasan, and J. A. Rivers. Scalable High Performance Main Memory System Using Phase-change Memory Technology. In *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA '09)*, Austin, Texas, June 2009.

[26] D. Ralph and M. D. Stiles. Spin Transfer Torques. *Journal of Magnetism and Magnetic Materials*, 320(7):1190–1216, 2008.

[27] L. E. Ramos, E. Gorbatov, and R. Bianchini. Page Placement in Hybrid Memory Systems. In *Proceedings of the International Conference on Supercomputing (ICS '11)*, Tucson, Arizona, 2011.

[28] Redis Labs. Memtier Benchmark. https://github.com/RedisLabs/memtier_benchmark.

[29] P. Sehgal, S. Basu, K. Srinivasan, and K. Voruganti. An Empirical Study of File Systems on NVM. In *Proceedings of the 2015 IEEE Symposium on Mass Storage Systems and Technologies (MSST '15)*, Santa Clara, CA, June 2015.

[30] H. Shiga, D. Takashima, S. Shiratake, K. Hoya, T. Miyakawa, R. Ogiwara, R. Fukuda, R. Takizawa, K. Hatsuda, F. Matsuoka, Y. Nagadomi, D. Hashimoto, H. Nishimura, T. Hioka, S. Doumae, S. Shimizu, M. Kawano, T. Taguchi, Y. Watanabe, S. Fujii, T. Ozaki, H. Kanaya, Y. Kumura, Y. Shimojo, Y. Yamada, Y. Minami, S. Shuto, K. Yamakawa, S. Yamazaki, I. Kunishima, T. Hamamoto, A. Nitayama, and T. Furuyama. A 1.6GB/s DDR2 128Mb Chain FeRAM with Scalable Octal Bitline and Sensing Schemes. In *IEEE International Solid-State Circuits Conference, (ISSCC '09)*, San Francisco, California, February 2009.

[31] Sun Microsystems. Solaris Internals: FileBench. http://www.solarisinternals.com/wiki/index.php/FileBench.

[32] Sun Microsystems. MySQL White Papers, 2008.

[33] K. Suzuki and S. Swanson. A Survey of Trends in Non-Volatile Memory Technologies: 2000-2014. In *the 7th International Memory Workshop (IMW'15)*, Monterey, CA, May 2015.

[34] R. Takemura, T. Kawahara, K. Miura, H. Yamamoto, J. Hayakawa, N. Matsuzaki, K. Ono, M. Yamanouchi, K. Ito, H. Takahashi, S. Ikeda, H. Hasegawa, H. Matsuoka, and H. Ohno. 32-mb 2t1r spram with localized bi-directional write driver and '1'/'0' dual-array equalized reference cell. In *2009 Symposium on VLSI Circuits (VLSI '09)*, Kyoto, Japan, June 2009.

[35] Transaction Processing Council. TPC Benchmark C Standard Specification, Revision 5.11. Technical Report, 2010.

[36] A. Tulapurkar, Y. Suzuki, A. Fukushima, H. Kubota, H. Maehara, K. Tsunekawa, D. Djayaprawira, N. Watanabe, and S. Yuasa. Spin-Torque Diode Effect in Magnetic Tunnel Junctions. *Nature*, 438(7066):339–342, 2005.

[37] H. Volos, A. J. Tack, and M. M. Swift. Mnemosyne: Lightweight persistent memory. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '11)*, New York, New York, March 2011.

[38] X. Wu and A. Reddy. Scmfs: A file system for storage class memory. In *International Conference*

for High Performance Computing, Networking, Storage and Analysis (SC '11), Nov 2011.

[39] J. J. Yang, D. B. Strukov, and D. R. Stewart. Memristive devices for computing. *Nature nanotechnology*, 8(1):13–24, 2013.