Persistent Memory Workload Characterization: A Hardware Perspective

Xiao Liu[1] Bhaskar Jupudi[2] Pankaj Mehra[3] Jishen Zhao[1]
[1]University of California, San Diego [2]Cohesity [3]Samsung Electronics
[1]{x1liu, jzhao}@ucsd.edu [3]pankaj.mehra@samsung.com

***Abstract*—Persistent memory is a new tier of memory that functions as a hybrid of traditional storage systems and main memory. It combines the advantages of both: the data persistence property of storage with the byte-addressability and fast load/store interface of memory. As such, persistent memory provides direct data access without the performance and energy overhead of secondary storage access. Being at early stages of development, most previous persistent memory system designs are motivated and evaluated by software-based performance profiling and characterization. Yet by attaching on the processor-memory bus, persistent memory is managed by both system software and hardware control units in processors and memory devices. Therefore, understanding the hardware behavior is critical to unlocking the full potential of persistent memory. In this paper, we explore the performance interaction across applications, persistent memory system software, and hardware components, such as caching, address translation, buffering, and control logic in processors and memory systems. Based on our characterization results, we provide a set of implications and recommendations that can be used to optimize persistent memory system software and hardware designs.**

***Index Terms*—Persistent memory, Characterization, Hardware metrics**

I. Introduction

Deploying byte-addressable nonvolatile memories (NVRAMs) on memory bus enable a new data storage concept, called "persistent memory" [1]. Representative NVRAMs include spin-transfer torque RAM [2], phase-change memory [3], resistive RAM [4], battery-backed DRAM [5], and 3D XPoint™ memory that is announced by Intel and Micron [6])[1] – They combine the benefits of both worlds: the byte-addressability and fast load/store interface of memory with the data persistence of storage. As such, NVRAM allows applications to directly load and store data in the main memory. Due to this game-changing feature, NVRAMs are poised to radically improve data manipulation performance and energy efficiency. NVRAM also motivates a new concept – persistent memory, which guarantees that data is recoverable (consistent and durable) through power outages and system crashes; persisting data through slow storage I/O interface is no longer required.

Traditional computer systems manage data persistence by software mechanisms of file systems and databases [8, 9]. Yet main memory access is manipulated by both system software and hardware control units, such as CPU caches, buffers, and memory controllers. As such, to fully exploit the potential of persistent memory, it is beneficial to design system software and applications with the awareness of hardware details.

However, most previous persistent memory systems are motivated and evaluated by software-based performance profiling and characterization [10, 11, 12]. These profiling schemes provide system performance implications in terms of operation throughput, software request latency, and software events (e.g., context switches and system calls). But they treat hardware components, such as CPU cache hierarchy and TLBs, as a black box. As a result, very little information about hardware components is available for persistent memory software.

In this paper, we intend to provide implications on optimizing persistent memory system designs from a hardware perspective. We analyze the performance and hardware behaviors of persistent memory systems by running various storage workloads with a variety of configurations (e.g., read/write intensity, file size, and number of threads) on traditional and persistent memory file systems. To examine the performance impact of persistent memory systems and workload configurations, we profile the statistics of a variety of software events and hardware performance counters available in modern processors. As a result, we find the following implications and recommendations that might benefit persistent memory software designs:

- We identify that persistent memory system performance is more closely correlated to the configurations and behaviors of several particular hardware components, such as last-level cache (LLC) access, and instruction and data translation look-aside buffers (TLBs) access, than others. As such, persistent memory system designers may prioritize the optimizations on the access to the hardware components with larger correlation than others.
- We show that the recently-introduced direct access (DAX) support in Linux essentially enhances file systems to benefit from the speed of NVRAMs. But the current DAX can be susceptible for the limitations on flexibility and performance offered to applications.
- Whereas NVRAM devices do not require row buffers (a set of sense amplifiers and latches that temporarily store data values) as DRAM does, buffers, if present on NVRAM chips, can significantly mitigate the performance penalties of long latency, low bandwidth, and asymmetric read/write latency in NVRAM access. As such, system software and applications with high memory access locality can exploit such buffers to optimize system performance.

[1]These devices are already or are expected to be available on the market in 2019 [6, 7].

TABLE I
PROFILED SOFTWARE AND HARDWARE EVENTS. (S) REPRESENTS SOFTWARE EVENTS. THE REST ARE HARDWARE EVENTS.

Cache Events	L1 data cache loads/stores/prefetches, L1 data cache load/store/prefetch misses, LLC loads/stores/prefetch, LLC load/store/prefetch misses
TLB Events	dTLB loads/stores/prefetches, dTLB load/store/prefetch misses, iTLB loads/miss
Other Events	Page faults (S), context switches (S), CPU migrations (S), branch loads, branch load misses

II. METHODOLOGY

To investigate representative file systems and storage workloads, we choose one popular traditional file system, two latest persistent memory file system implementations, and exercise them with six widely-used storage benchmarks running on a workstation configured with an emulated NVRAM.

A. System Platforms

To perform our measurements, we use a workstation equiped with Intel Xeon CPU E5-2620 v3 processor. The processor contains six 2.4GHz two-way multithreaded cores. Each core has the private 32KB 8-way set associative L1 data and instruction caches. All cores share a private 256KB 8-way set associative L2 cache and a 15MB 20-way set associative LLC. To access hardware performance counters, we use Linux `perf` utility [13] to profile various hardware events (and also software events) as listed in Table I. We ran each workload for multiple times, calculated the mean and the standard deviation of each counted software and hardware event.

The workstation runs Ubuntu with 4.4.0 Linux kernel. The workstation has four DDR4 DRAMs at 2133MHz. We emulated 12GB of NVRAM on the DRAM. To configure the NVRAM partition, we use `memmap` GRUB (boot loader) parameter to mark the specific 12GB range of DRAM as the persistent memory.

B. File Systems

Our evaluation includes three file systems. Two of them are variants of ext4, and the last is a state-of-the-art persistent memory file system [12].

Ext4. Ext4 [9] is a popular file system used in Linux. Ext4 is an extent-based file system with reduced metadata overhead. It supports three journaling modes, which provide different levels of atomicity for data and metadata. 1) The `Data-writeback` mode does not perform any data journaling, 2) the `Data-ordered` mode only records metadata in the journal, and 3) the `Data-journal` mode writes both metadata and data into the journal.

Ext4-DAX. Latest persistent memory file systems can bypass the page cache allocated in DRAM and directly access NVRAM via loads/stores using a technique called *Direct Access* (DAX) [14]. DAX maps storage components directly into userspace. Note that ext4-DAX uses journaling to persist metadata updates only; it does not support data journaling. To ensure a fair comparison between ext4 and ext4-DAX, we configured both in the `data-ordered` mode, where only metadata is journaled.

NOVA. NOn-Volatile memory Accelerated log-structured file system (NOVA) is one of the latest DAX-supported persistent memory file systems [12]. NOVA is a log-structured file system (LFS), which naturally logs the data updates. Therefore, it only journals metadata to ensure data persistence.

C. Storage Workloads

We select and use representative storage workloads for our experiments. They include three workloads from filebench suite [15], and two microbenchmarks – file compression/decompression, and FFSB [16]. Before each experiment, we exercise the DRAM and persistent memory partitions by writing, appending, and deleting files of various sizes.

Filebench. Filebench [15] is a benchmark suite designed for evaluating file systems. It can generate various workloads to emulate a variety of real-world applications. We generate three different workloads, include fileserver, webproxy, and varmail, to emulate the typical file system access patterns of the file, web proxy, and email servers.

- *Fileserver* emulates the basic file server access pattern, including a sequence of creates, deletes, appends, reads, writes, and metadata operations. These operations are performed by multiple "user" threads.
- *Webproxy* emulates a plain web proxy server. This workload is characterized by a fairly flat namespace hierarchy with a directory width of 1,000,000. We create 10,000 files with an average size of 1MB in the directory. We employ 100 threads to perform a mix of create, append, read, and delete operations over these files with a 5:1 read/write ratio.
- *Varmail* emulates the access pattern of a mail server. The operations of this workload include create-append-synchronization, read-append-synchronization, read, and delete. We configure a directory that stores 1000 files with a median file size of 100MB. This workload has 16 threads with a 1:1 read/write ratio.

File Compression and Decompression. We employ `zip`, and `unzip` to investigate the performance of file compression and decompression in persistent memory systems. The zip operation compresses the archives, while the unzip operation extracts the archives. We use them to compress and decompress two MPEG-4 files; each has a size of 2.3GB.

FFSB. The Flexible File System Benchmark (FFSB) [16] generates customizable workloads to measure file system performance. It employs *pthreads* to support multiple groups of threads that can access multiple files simultaneously. We employ 20 threads to perform random read and write operations over 110 files, the size of a single file is 100MB.

III. THE IMPACT OF NVRAM ACCESS LOCALITY

In this section, we examine the performance impact of adopting hardware buffers in NVRAM components in per-

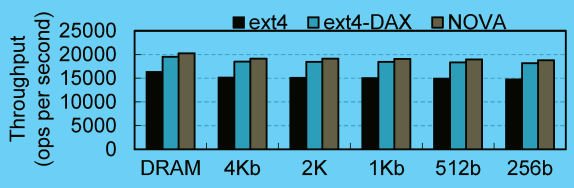

Fig. 1. System throughput over the DRAM and the NVRAMs with row buffer sizes ranges from 256b to 4Kb. The workload has 60% sequential read operations.

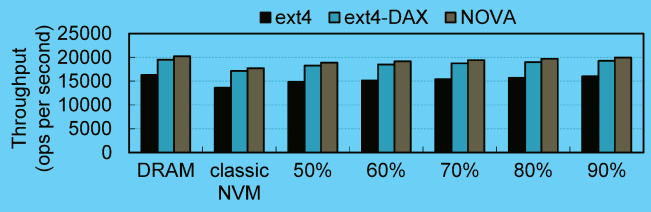

Fig. 2. System throughput under various memory models. The *y*-axis represents the throughput. The *x*-axis represents memory models, which include DRAM, classic NVRAM with no buffer, and NVRAMs with buffer that have the buffer hit rates range from 50% to 90%.

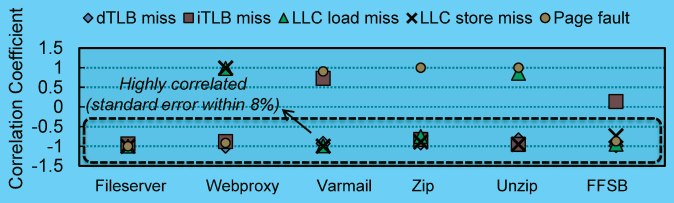

Fig. 3. Correlation coefficients between workload performance and five hardware metrics. The *x*-axis represents each benchmark. The *y*-axis is the correlation coefficient between throughput and each metric.

sistent memory. Row buffer is the cache for memory rows in DRAM. In the future NVRAM, there will be a similar structure, we name it "buffer" in this paper. Like row buffer in the DRAM, buffer acts as a cache to the NVRAM rows. The size of NVRAM's buffer could be different from DRAM row buffer size because of the different write powers [17]. Therefore we conduct two studies to explore the impacts of the buffer size and the buffer hit rate on the performance.

Buffer size. We present result of a sensitivity study of buffer size in Figure 1. We estimate the performance based on the NVRAM architecture in [17]. We evaluate with various memory models with the fileserver workload. Our evaluation configures the fileserver with 60% sequential read operations.

We make two observations from the result. First, the throughput increases with the row buffer size. When buffer size increases, random memory accesses have a higher chance of hitting the address range within a buffer size. Hence, the buffer hit rate increases with the buffer size. The increased buffer hit rate reduces the memory access cycles and results in a higher throughput. Second, the shrunk row buffer size could potentially hurt the NVRAM performance in the future. The throughput drops 2% when the buffer size shrinks 50%. For the future NVRAM, the buffer size might be $1/2$ or $1/8$ of the row buffer size in DRAM [17], which could lead to a 2%–8% performance loss. NVRAM customized eviction and prefetch strategies could compensate for this performance loss.

Buffer hit rate. We present the result of a sensitivity study of buffer hit rate in Figure 2. We use the same performance model from the buffer size study. The result includes two baselines: DRAM and classic NVRAM. The classic NVRAM ignores the existence of buffer. We studied the buffer hit rate ranges from 50% to 90%; the row buffer size is set to 2Kb across the experiment.

We make two observations from the results. First, the system throughput increases rapidly as the buffer hit rate increases. Each time hit rate increases 10%, NOVA throughput increases 1.9% while ext4 and ext4-DAX throughput increase 1.4%. Second, the NVRAM with 90% buffer hit rate has a minor performance difference to the DRAM performance. We discover that the performance difference between the DRAM and the NVRAM with 90% buffer hit rate are within 2% among all the file systems.

Certain types of workloads can achieve 90% buffer hit rates. We conduct a further investigation with zip workload, a typical stream application. We employ Pin [18] to measure the row buffer locality. Because of the similarity between the page size and the buffer size, we use Pin measured page locality to estimate the buffer hit rate. The result shows that the buffer hit rates are above 98% across three file systems. When running these workloads, we can expect similar performance behaviors between NVRAM- and DRAM-based main memories.

IV. Microarchitecture Analysis

Furthermore, we examine the relationship between persistent memory system performance and various hardware and software event counters. Our experiments across six workloads identify that four hardware counters and one software event are closely correlated to system throughput. Figure 3 plots the statistical correlation between throughput and these five metrics across six benchmarks. We use measured results from multiple continuous executions to calculate the correlation coefficient. Because these metrics are usually considered as the major source of performance overhead. We only consider metrics that negatively correlate to the throughput are justifiable. We make three observations from the results.

First, dTLB miss shows a high negative correlation with the throughput among all the benchmarks. This indicates that address cache miss contributes more to the performance overhead than the data cache miss. Increase the size of the TLB cache could provide performance gain for the persistent memory system.

Second, the page fault is least correlated to the performance. As the Figure 3 shows, the correlation between page fault and throughput varies drastically among different workloads. Higher page faults do not necessarily lead to bad performance. The reason is that all the page faults collected by *perf* are reported as minor page faults in the DAX featured file systems. DAX generated page faults lead to a much less performance penalty than non-DAX generated page faults.

Third, LLC load and store miss show an insignificant correlation to the performance. This relates to the limitation of the perf tool. All the LLC misses visit DRAM in a DRAM-only main memory. However, LLC misses could access NVRAM in a hybrid main memory. Even though we partition DRAM to simulate the hybrid memory, the perf tool cannot distinguish the LLC misses lead to NVRAM access or DRAM access. Therefore, the perf tool measured LLC miss is not equal to the NVRAM access. To make LLC load and miss usable for persistent memory, the perf tool should be extended to support profiling for the hybrid memory.

V. Implications on Persistent Memory System Design

Based on our observations, we provide the following implications and recommendations for persistent memory design for reaping the full potential of this new data storage component.

First, *persistent memory system performance is highly correlated to the configuration and behavior of a small number of hardware components.* As shown in Section IV, across various workloads, persistent memory system performance is always closely related to the configuration and behavior of certain hardware components, such as LLC and instruction/data TLBs. The rationale behind is that these hardware components are the most closely coupled with memory access among the components in the processor. **Recommendation:** In order to fully optimize persistent memory performance, system designers may consider optimizing the access to these highly correlated hardware components instead of others.

Second, *DAX substantially enhances file systems to leverage the performance benefit of persistent memory, yet may not be the only option.* Our results show that file systems with DAX (ext4-DAX and NOVA) provide much better performance than traditional ext4 file system by bypassing the page cache. However, current DAX design has limitations. DAX needs to be configured as a global parameter of file systems, regardless of the workload running on top of it. This can significantly reduce the flexibility of DAX. **Recommendations:** Persistent memory system design needs to provide a flexible interface for workloads to enable/disable DAX. The performance and flexibility of DAX design can also benefit from the automatic tuning of system configurations.

Third, *buffers on NVRAM devices can substantially impact persistent memory system performance.* Whereas NVRAM devices do not require row buffers as DRAM does, the performance of persistent memory systems can substantially benefit from adopting buffers on NVRAM devices. Such buffer can significantly mitigate the performance penalty of the long latency, low bandwidth, and asymmetric read/write latency in NVRAM access. **Recommendations:** NVRAM hardware design can improve system performance by incorporating buffers on NVRAM devices. System software and application design can leverage such buffers by exploiting the locality in data access at the granularity of the buffer size.

VI. Acknowledgment

We thank the anonymous reviewers for their valuable feedback. This paper is supported in part by NSF grants 1829524, 1829525, 1817077, and SRC/DARPA Center for Research on Intelligent Storage and Processing-in-memory.

References

[1] P. Mehra *et al.*, "Fast and flexible persistence: the magic potion for fault-tolerance, scalability and performance in online data stores," in *IPDPS*, 2004.

[2] R. Patel *et al.*, "Reducing switching latency and energy in STT-MRAM caches with field-assisted writing," *IEEE Transactions on VLSI*, 2016.

[3] V. Sousa, "Phase change materials engineering for reset current reduction," in *Workshop on Innovative Memory Technologies*, 2012.

[4] C. Cagli, "Characterization and modelling of electrode impact in HfO2-based RRAM," in *Workshop on Innovative Memory Technologies*, 2012.

[5] T. C. Bressoud *et al.*, "The design and use of persistent memory on the DNCP hardware fault-tolerant platform," in *DSN*, 2001.

[6] "Intel Optane DC Persistent Memory," 2019, https://www.intel.com/content/www/us/en/architecture-and-technology/optane-dc-persistent-memory.html.

[7] HP, "SanDisk and HP Launch Partnership to Create Memory-Driven Computing Solutions," 2015, https://www8.hp.com/us/en/hp-news/press-release.html?id=2099577.

[8] C. Mohan *et al.*, "ARIES: A Transaction Recovery Method Supporting Fine-granularity Locking and Partial Rollbacks Using Write-ahead Logging," *ACM TODS*, 1992.

[9] "Ext4," 2018, https://www.kernel.org/doc/Documentation/filesystems/ext4.txt.

[10] P. Sehgal *et al.*, "An empirical study of file systems on NVM," in *MSST*, 2015.

[11] Y. Zhang *et al.*, "A study of application performance with non-volatile main memory," in *MSST*, 2015.

[12] J. Xu *et al.*, "NOVA: A Log-structured File System for Hybrid Volatile/Non-volatile Main Memories," in *FAST*, 2016.

[13] "Perf wiki," 2015, http://perf.wiki.kernel.org/.

[14] M. Wilcox, "Add support for NV-DIMMs to ext4," 2014, https://lwn.net/Articles/613384/.

[15] "Filebench," 2014, http://filebench.sourceforge.net.

[16] "Flexible file system benchmark," 2013, https://sourceforge.net/projects/ffsb/.

[17] B. C. Lee *et al.*, "Architecting Phase Change Memory As a Scalable DRAM Alternative," in *ISCA*, 2009.

[18] C.-K. Luk *et al.*, "Pin: Building customized program analysis tools with dynamic instrumentation," in *PLDI*, 2005.