NVMFS-IOzone: Performance Evaluation for the New NVMM-based File Systems

Shengke Li
School of Electronic and Computer Engineering ,
Peking University
ShenZhen, China
danya@pku.edu.cn

Dagang Li
School of Electronic and Computer Engineering ,
Peking University Shenzhen Graduate School
ShenZhen, China
dagang.li@ieee.org

Dennis Wu
Intel Asia-Pacific Research & Development Ltd
ShangHai, China
dennis.wu@intel.com

Xiaogang Chen
State Key Laboratory of Functional Materials for
Imformatics, Chinese Academy of Sciences
ShangHai, China
Chenxg@mail.sim.ac.cn

ABSTRACT

With the emerging of NVM (Non-Volatile Memories) technologies, NVMM-based (Non-Volatile Main Memories) file systems have attracted more and more attention. Compared to traditional file systems, most NVMM-based file systems bypass the page cache and the I/O software stack. With the new mmap interface known as the DAX-mmap interface (DAX: direct access), the CPU can access the NVMM much faster by loading from/storing to it directly. However, the existing file system benchmark tools are designed for traditional file systems and do not support the new features of NVMM-based file systems, so the returned results are very often not accurate. In this paper, a new benchmark tool called NVMFS-IOzone is proposed. The behavior of the tool is redesigned to reflect the new features of NVMM-based file systems. The NVM-lib from Intel is used instead of traditional `msync()` to keep data consistent when evaluating the performance of the DAX-mmap interface. Experimental results show that the new benchmark tool can reveal a hidden improvement of 1.4~2.1 times in NVMM-based file systems, which cannot be seen by the traditional evaluation tools. The data paths of direct load/store to NVMM and bypassing CPU cache are also provided to support the new features of NVMM-based file systems for multidimensional evaluation. Furthermore, embedded cleaning-ups has also been added to NVMFS-IOzone to support convenient evaluation consistency, which benefits both NVMM-based and non-NVMM-based file system benchmarking even for quick and casual tests. The whole experimental evaluation is based on real physical NVMs rather than simulated NVMs, and the experimental results confirm the effectiveness of our design.

CCS CONCEPTS

• **Information systems → performance of system[Measurement Techniques]**;

KEYWORDS

NVMFS, Evaluation, Benchmark, Non-Volatile Memory, DAX-mmap

ACM Reference Format:
Shengke Li, Dagang Li, Dennis Wu, and Xiaogang Chen. 2020. NVMFS-IOzone: Performance Evaluation for the New NVMM-based File Systems. In *The 13th ACM International Systems and Storage Conference (SYSTOR '20), June 2–4, 2020, Haifa, Israel.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3383669.3398281

1 INTRODUCTION

In recent years, various NVM technologies (Table 1), such as spin-transfer torque ram (STT-RAM), phase change memory (PCM), resistor RAM (ReRAM) and 3D XPoint memory[1][7][24][40][30][41], are emerging and developing very fast towards real-life applications. The utilization of NVMs based on these technologies as the persistent media at memory level, is attracting more and more interest from both academia and industry[14][17][27][32][28][36]. The most popular approach is to place NVMs on the processor memory bus alongside conventional DRAM, leading to hybrid volatile/non-volatile main memory architecture. Combining faster, volatile DRAM with slightly slower, denser non-volatile main memories (NVMMs) [8][29][38][42] offers the possibility of storage systems[39] that combine the best characteristics of both technologies. The 3D XPoint based NVMM such as the Intel Optane DC persistent memory is a common choice, because it is a mature technology and the performance is close to DRAM. At the same time, new NVMM-based file systems such as HiNFS[16], PMFS-new[3][21], XFS-DAX[34], NOVA[39], Ext4-DAX[11][6], etc, also start to emerge to support the new NVMM-based systems, and their layout is shown in the right half of Figure 1. The common practice of these NVMM-based file systems are to bypass the page cache and the block-based I/O software stack, where the POSIX interface usually needs to search the metadata in the DRAM. With a new mmap interface, CPU can load/store NVMM directly according to the mapping without searching the metadata, effectively exposing the NVMM to the CPU directly. This interface is also known as the DAX-mmap. The high performance makes this interface play an important role in the NVMM-based file system[19].

Apparently benchmark tools from the pre-NVM era are still used to evaluate these new NVMM-based file systems. The most commonly used evaluation tools are Filebench[35], FIO[9] and IOzone[12], etc. Filebench is a flexible framework for file system benchmarking. It can simulate specific workloads such as Web-server, Mail-server, and File-server. Unfortunately, Filebench does not support the evaluation of mmap interface. For example paper[23] used Filebench and FIO to evaluate the performance of several NVMM-based file systems, but it focused on the POSIX interface and do not involve the DAX-mmap interface. FIO and IOzone can support the mmap interface and evaluate with multiple processes and threads. IOzone focuses on the throughput of read/write and FIO focuses on the number of read and write operations per second(IOPS) or IO rate. Furthermore, IOzone can configure the CPU cache size and choose to clear the CPU cache if necessary. However, these tools evaluate the DAX-mmap interface of the NVM-based file systems following the same procedure as to benchmark the mmap interface of the traditional file system, which is as follows[9][12]:

1) Allocating a *Mainbuffer* in DRAM using `malloc()`
2) Getting the VM address (*Mmap_addr*) by `mmap()` to map the NVMM or disk space
3) Initializing *Mainbuffer*
4) Moving the *Mainbuffer* data from DRAM to Disk or NVMM at *Mmap_addr* (*Mmap_Write*) or the opposite (*Mmap_Read*) by calling `bcopy()`
5) Executing `msync()` to keep the data consistent

The corresponding data flow is shown in Figure 2(a), from which we identified the following issues:

a) In this procedure, `msync()` is used to ensure data consistency, which effectively finds dirty pages in page cache and flushes them into the storage device. However, the DAX-mmap interface actually bypasses the page cache and the CPU just accesses NVMM directly according to the mapping, so `msync()` becomes unnecessary for the DAX-mmap interface evaluation and only induce unnecessary overhead.
b) In this procedure, the data flow either starts from or ends at the DRAM (shown in Figure 2(a)). For the DAX-mmap interface, CPU can access NVMM directly without the help of the DRAM. Existing benchmark tools do not support the direct access between CPU and NVMM. (ie: CPU→CPU cache→NV-MM).
c) Since the NVMM-based file systems can achieve very low latency with the DAX-mmap interface, CPU cache now becomes a bigger contributor to the overall latency for handling cache miss and flushing the cached data into the NVMM. For data that are only processed once, bypassing CPU cache becomes a useful design option to further exploit the advantage of NVMM, but such scenarios are not supported in existing tools.
d) In many existing benchmark tools, metadata cache and page cache are not handled sufficiently before or after the tests, which may cause fluctuations in the results, especially for small file size benchmarking. This can be taken care of by external pre-test preparation and between-test cleaning-up, but integrating the right handling into the tools as an option is more efficient and convenient, especially for quick test and casual practitioners.

For a more reliable evaluation of the NVMM-based file system, given the above considerations, this paper mainly made the following contributions:

- The benchmarking procedure is redesigned using the NVM-lib from Intel instead of traditional `msync()` for data consistency, improving the accuracy of the evaluation results.
- Supporting the direct data path between CPU and NVMM without the interference of the *Mainbuffer* in DRAM and other software stacks.
- Adding the option of bypassing the CPU cache in the data path to support the evaluation of the new scenarios for NVMM-based file systems.
- Adding the appropriate cleaning-ups to improve consistency between evaluation trials.

Furthermore, existing researches usually use simulated NV-MM instead of physical hardware. In this paper we work with Intel and use the latest model Optane DC persistent memory in our experiments. Typical NVMM-based file systems such as Ext2-DAX, PMFS-new, XFS-DAX, NOVA and

Ext4-DAX are tested using the NVMFS-IOzone. The results show that the overhead of msync() largely under-estimate the performance of NVMM-based file system by 1.4~2.1 fold. Furthermore, benchmarking with direct load/store allows the elimination of interference from *Mainbuffer* in DRAM and other software stacks. For scenarios where data is only processed one-pass, the new option of bypassing the CPU cache reveals a further 1.78~2.3 times potential gain, which can be useful for the NVMM-based file systems. Finally, the embedded cleaning-ups effectively narrows down the fluctuation in the results.

The remainder of the paper is organized as follows: The second section focuses on the background of NVMM-based file systems and related benchmark tools. The third section provides the design details of the NVMFS-IOzone. Section 4 shows the functional experiments of the NVMFS-IOzone. The last section concludes the paper.

Table 1: Overview of NVM Technologies[23][1][7][40]

Memory Technology	Read (ns)	Write (ns)	Endurance (writes/cell)
SRAM	2~3	2~3	∞
DRAM	15	10	10^{18}
STT-RAM	5~30	10~100	10^{16}
PCM	50~70	150~220	10^{8}~10^{12}
3D XPoint	300	10~100	*
NAND Flash	2.5×10^{4}	2×10^{5}~5×10^{5}	10^{4}~10^{5}
HDD	3×10^{6}	3×10^{6}	∞

2 BACKGROUND & RELATED WORK

2.1 Non-volatile memory

Emerging non-volatile memory technologies such as spin torque transfer RAM (STT-RAM), phase change memory (PCM), SRAM, 3D XPoint, NAND Flash [7][24][40][30][41][37] are showed in Table 1. The STT-RAM can meet or surpass DRAM's latency and it may eventually appear in on-chip and last-level cache [43]. But its large cell size limits its capacity and feasibility as a DRAM replacement. PCM is denser than DRAM and can implement very large non-volatile main memories. However, its longer latency makes it unlikely to replace DRAM as main memory completely. Intel and micron's recently announced 3D XPoint memory technology is the more practical candidate. Optane is based on this technology, and its performance is 1000 times faster than NAND flash[1]. Besides, the interface of Optane can co-exist with conventional DDR4 DRAM DIMMs on the same platform[2].

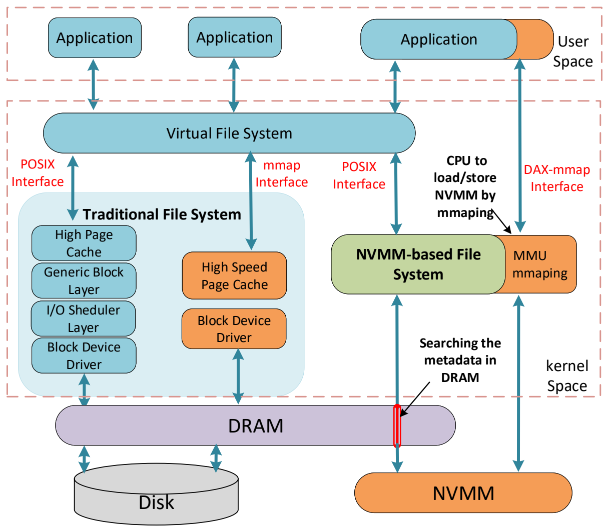

Figure 1: The layout of traditional and NVMM-based file systems.

2.2 The NVMM-based file systems

Figure 1 shows the layout of traditional and NVMM-based file systems. There are mainly two types of interfaces for file systems: POSIX and mmap/DAX-mmap. For both the POSIX and mmap interfaces in the traditional file systems, high speed page cache is used and data will be copied twice: firstly from the disk to the high speed page cache in DRAM and then to the CPU. For the new NVMM-based file systems, both its POSIX and mmap interfaces bypass the page cache and the block-based I/O software stack. The difference is that the POSIX interface still needs to use the metadata in DRAM but the mmap interface does not. With the mmap interface, CPU can access the NVMM directly according to the mapping, which effectively exposes the NVMM directly to the CPU, known as the DAX-mmap (shown in Figure 1). Most NVMM-based file systems in recent years support the DAX-mmap interface, and it is likely to be the dominant interface in the future[19]. However, the DAX-mmap interface in different NVMM-based file systems has different features, for example:

Ext2-DAX[13][6]extends the log file system Ext2 with DAX features and supports the DAX-mmap. However, huge page size cannot be aligned because of the Ext2 data structure and the layout of the file system, so huge page processing is not supported, and large files will cause pmd faults.

PMFS-new[3][21] cuts some features of Intel's PMFS and supports the DAX-mmap. Similarly to the Ext2-DAX, huge page processing is also not supported and the performance on large files is bad.

XFS-DAX[34][6] extends the XFS file system [33], a high-performance log file system originated from the SGI IRIX platform. It supports the DAX-mmap and huge page processing.

The Ext4-DAX[6][11] extends Ext4 with the DAX features. The data structure and layout of Ext4 is different from that of Ext2, so it can align huge page size and support the huge pages. Furthermore, Ext4-DAX can speed up the query of virtual space address by compressing TLB entries.

NOVA[39] adapts the conventional log-structured file system techniques to exploit the fast random access that NVMs provide. The mmap interface of NOVA implements atomic operation, which is based on the DAX-mmap interface of Ext4-DAX file system.

HiNFS[16] also supports DAX-mmap. With the DAX-mmap interface, HiNFS first flushes all dirty DRAM blocks of a file to NVMM, and then sets the state of the corresponding data blocks to Eager-Persistent. Finally, it directly maps the file to the virtual address space, and then CPU can load/store NVMM directly via the virtual address.

2.3 The File system benchmark tools

There are a variety of file system benchmark tools. Bonnie[4] was written in 1988, which can perform a series of benchmarking on defined file size. If the size is not defined, Bonnie defaults to 100 Mb. For each test, Bonnie can evaluate sequential write in "Per char" "Block" "Rewrite" modes, and sequential read in "Per char" "Block" modes, and random read/write in the "Seeks" mode. Unfortunately, it cannot evaluate latency. Bonnie++[18] updates from Bonnie and was released in 1999. Compared with Bonnie, the latency evaluation is supported and it can evaluate the rate of random or sequential creation and deletion of files.

Tiobench[5][26] (Threaded I/O bench) is a package of tiotest. Compared with Bonnie++, Tiobench can set the block size and support multi-threads model, but it does not support multi-processes. Sysbench[25] is a benchmark suite, which not only can evaluate file system performance but also test CPU, database, etc. For file system evaluation, the workload methods of Sysbench and Tiobench are similar. Filebench[35] is a highly flexible framework to benchmark file system. The popularity of the Filebench framework comes mainly from the fact that it is shipped with several predefined macro workloads, eg, Web-server, Mail-server, and File-server, which allows users to easily evaluate their file systems against several sufficiently different workloads.

It is notable that all of the above benchmark tools can only evaluate the POSIX interface but not the mmap interface.

FIO and IOzone can benchmark the mmap interface and their workloads are more flexible, which can simulate a given I/O workload without resorting to writing a tailored test case again and again[9][12]. They both support multi-threads, multi-processes, asynchronous I/O, synchronous I/O, etc, including a variety of evaluation operations such as *read/write, re-read/re-write, read backwards, random read/write, aio_read /aio_write*, etc. FIO focuses on the number of read and write operations per second(IOPS) or IO rate, and its other primary function is to inspect disk I/O performance besides evaluating file systems. IOzone focuses on the throughput of read/write and can configure the CPU cache size and choose to clear the CPU cache if necessary.

2.4 The mmap vs. The DAX-mmap interface

From the procedure of benchmarking the mmap interface as described in the previous section, the corresponding data flows are shown in Figure 2(a) when we use the existing tools to evaluate write performance of the mmap interface of traditional file systems and the DAX-mmap interface of NVMM-based file systems, where data is written from *Mainbuffer* in the DRAM to Disk following 1'→2'→3'→4'→5', and from *Mainbuffer* to NVMM 1→2→3→4. We call this test *Mmap_Write*, and the opposite *Mmap_Read*. The *Mainbuffer* is absolute necessary here because CPU cannot access the disks directly as a block device.

However, most NVMM-based file systems expose NVMM to CPU by the DAX-mmap interface, so CPU can access NVMM directly according to the mapping(shown in Figure 1) without relying on transit of DRAM. Clearly the traditional test procedure always involve DRAM as the source or the destination of the data flow and cannot correctly reflect the correct operation of the new NVMM-based file systems.

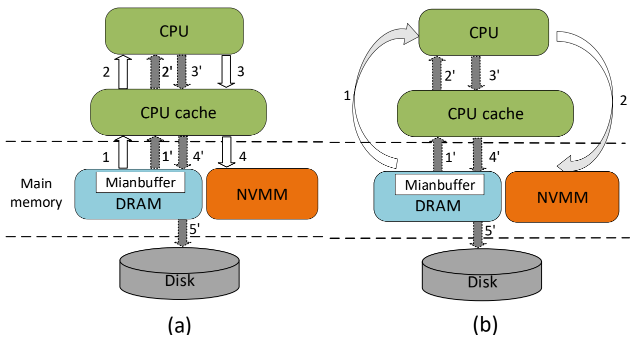

Figure 2: (a) 1→2→3→4 is the *Mainbufer* data flow for NVMM-based file systems; 1'→2'→3'→4'→5' is the *Mainbufer* data flow for traditional file systems. (b) The data flow bypassing the CPU cache.

3 THE DESIGN OF NVMFS-IOZONE

In this section, we look deeper into the benchmark process of existing tools to identify their main issues, and then propose our solutions. According to the previous discussions, mainly the following two issues are identified: the unnecessary `msync()` calls and the inappropriate data flows. Furthermore, we also add the new options of bypassing CPU cache and integrated cleaning-ups procedure to the tool to support convenient tests for potentially new features of the NVMM-based file systems. Detailed discussions will be presented in the following subsections.

Table 2: The trace of `msync()` system call.

Time	System call function
	ext4_sync_file [ext4](){
0.118us	file_write_and_wait_range(){
	mapping_needs_writeback()
	__filemap_fdatawrite_range(){
13.011us	**do_writepages()**
+16.937us	**}**
1.537us	__filemap_fdatawait_range()
	pagevec_lookup_range_tag()
	find_get_pages_range_tag()
	file_check_and_advance_wb_err()
+22.997us	}
0.923us	jbd2_trans_will_send_data_barrier[jbd2]()
	blkdev_issue_flush(){
2.586us	bio_alloc_bioset()
8.849us	submit_bio_wait()
1.863us	bio_put()
+15.997us	}
+45.855us	}

3.1 The flushing method

For both the traditional file system and the NVMM-based file system, existing evaluation tools usually call `msync()` to keep data consistent, which is a standard action for traditional file systems based on block devices. However, the `msync()` will create a store barrier, a point after which the program can assume the previous changes it made to the storage device are persistent[31]. The storage barrier requires the operating system to find dirty pages in the high speed page cache and flush them to the NVMM or disk. However for the DAX-mmap interface of the NVMM-based file system, it bypasses the page cache and the CPU can load/store NVMM directly by the mapping. Therefore, the operating system only needs to flush the CPU cache to write the changes to the NVMM.

To profile the overhead of the operation we use IOzone to benchmark the DAX-mmap interface of the Ext4-DAX as an example. Configuration of the evaluation environment is the same as the experiments in Section 4. When IOzone calls the `msync()`, we use the `kernel functional trace` to track its subfunctions and the execution time(shown in Table 2). The left side of the table is the execution time of the function and the right side is the corresponding function, where the time with symbol "+" is the execution time of the function corresponding to the brace on the right side. The function `filemap_fdatawrite_range()` is mainly to write back dirty pages in range. `do_writepages()` inside `filemap_fdatawrite_range()` will find dirty pages in the high speed page cache and then write them to the persistent memory such as disk or NVMM. It takes a lot of time and the execution time is 16.937us. However, these operations are meaningless for the DAX-mmap interface that bypasses the high speed page cache. Therefore when evaluating the DAX-mmap interface of the NVMM-based file system, it is not appropriate to call traditional `msync()`.

There are direct instructions to flush the CPU cache, such as CLFLUSH, CLFLUSHOPT, CLWB, and they are supported in most modern CPUs [31][22]. Intel provides NVM libraries based on these instructions. We mainly use two libraries in the NVMFS-IOzone benchmark tool. One is the `libpmemobj` library, which allows persistent memory objects to be allocated in a safe way, and allows referring to them by Object IDs (OIDs). The other is the `libpmem` library, which contains code to detect the flush instructions supported by CPU and the best instruction options matching with the platform, which can avoid the incompatible instructions that may lead to problems such as excessive CPU temperature, overload and so on. Part of the implementation code in NVMFS-IOzone is shown in Code 1. First we check whether it is NVMM by calling `pmem_is_pmem()`, where the DAX-Mmap_addr is the address returned by the DAX-mmap interface operation and `len` is the size of every flush. Then we flush the data in the CPU cache to NVMM by calling `pmem_flush()` to keep the data consistent.

Code 1. The code lines to flush to NVMM

```
/*Judge whether it is NVMM*/
1  is_pmem = pmem_is_pmem(DAX-Mmap_addr, len);
/* flush to NVMM */
2        if (is_pmem)
/* flush the CPU cache */
3        pmem_flush(DAX-Mmap_addr, len);
/* wait for NVMM stores to drain from HW buffers */
4        pmem_drain();
```

3.2 Direct Load/Store

To get the more precise evaluation of the DAX-mmap interface without the interference of the *Mainbuffer* in DRAM, the data flow should start from the CPU to the CPU cache and then to the NVMM (the 3→4 process in Figure 2(a)) and the opposite direction. For this purpose, this part of NVMFS-IOzone is designed as following:

Code 2. The code lines to load/store NVMM

```
/*The following code is to store the data generated by RD-
RAND to NVMM. The DAX-Mmap_addr is the address
returned by DAX-mmap interface operation*/
1       asm volatile(
2           "rdrand %%rcx\n\t"
3           "mov %%rcx , %0"
4           :"=r" (*DAX-Mmap_addr)
5           :"memory"
6       );
7       DAX-Mmap_addr++;
/*The following section code is to load data from NVMM
to CPU */
8       asm volatile(
9           "mov %0,%%rcx\n\t"
10          :
11          :"r" (*DAX-Mmap_addr)
12      );
```

For the implementation of storing to NVMM, firstly we generate a series of random numbers by `RDRAND` to represent the data to be written, and put them one by one in the CPU register. Then `mov` is called to move the data from the CPU register to the target address DAX-Mmap_addr. We call this new test flow as the DAX-mmap-store, as shown in Line 1~6 of Code 2. For the DAX-mmap-load, the opposite process will occur: `mov` instruction moves the data in NVMM to the CPU register, where the data is the random number that was stored to NVMM by DAX-mmap-store, as shown in Line 8~12 of Code 2. As have been discussed in the previous subsection, for DAX-mmap-store/load we only need to flush the CPU cache to keep data consistent.

3.3 Bypassing the CPU cache

In the previous subsections, we update the methods of keeping data consistent and data access flows in the NVMFS-IOzone to support the new features of NVMM-based file systems. Compared to the traditional file systems, we found that the overhead of flushing the CPU cache in the new file systems becomes one of the main contributors to the overall latency, although its impact on traditional file systems is negligible. In pursuing for even higher performance with the DAX-mmap interface, when data in the files are only processed once, serious considerations on bypassing the CPU cache will become a reasonable design choice. To support this potential need, we add the new data path of bypassing the CPU cache NVMFS-IOzone.

Code 3. The code lines to bypass the CPU cache

```
/*DAX-Mmap_addr is an address returned by DAX-mmap
interface operation. Mainbuffer_addr is an address
returned by malloc() */
    asm volatile(
        "movdqu (%0), %%xmm0\n\t"
        "movdqu (%0), %%xmm1\n\t"
        "movdqu (%0), %%xmm2\n\t"
        "movdqu (%0), %%xmm3\n\t"
        "movntdq %%xmm0, (%1)\n\t"
        "movntdq %%xmm1, (%1)\n\t"
        "movntdq %%xmm2, (%1)\n\t"
        "movntdq %%xmm3, (%1)\n\t"
        :"=r"(DAX-Mmap_addr)
        :"r"(Mainbuffer_addr)
        :"memory"
    );
    asm volatile(
        "sfence\n"
        : :);
    DAX-Mmap_addr++;
    Mainbuffer_addr++;
```

As shown in Figure 1, the DAX-mmap interface bypasses the high speed page cache and block-based I/O software stack. The CPU can load/store NVMM directly. The data flow of the benchmark using existing evaluation tools is 1→2→3→4 (as Figure 2(a) shown). The mmap interface of traditional file systems has not only two copies of the page cache but also disk I/O time. The corresponding data flow of benchmarking is 1'→2'→3'→4'→5'(as Figure 2(a) shown). Therefore, compared with the mmap interface of traditional file systems, the DAX-mmap interface can shorten significantly the completion time of data flow and achieve low latency or high throughput even further. If it passes through the CPU cache, cache miss will be generated continuously for scenarios where data is only used once. Besides, the CPU cache also needs to be frequently flushed to keep data consistent. That overhead will not have a significant impact on the mmap interface of traditional file systems where disk I/O is the main bottleneck. However, for the DAX-mmap interface of the NVMM-based file system, the impact of that overhead will become significant. A typical example is a scenario of streaming read/ write. Streaming write only writes data to NVMM and there is no need to modify the data in the file back and forth. Such data can be called non-reused data, and caching it to the CPU cache is not only unnecessary but may also cause excessive cache exchange. Traditional tools may support directIO to bypass the page cache but cannot bypass the CPU cache.

NVMFS-IOzone use Non-temporal[20][10][15] method to bypass the CPU cache. We make use of the `XMM#` registers, which can store 128bit. There are four `XMM#` registers including `XMM0`, `XMM1`, `XMM2` and `XMM3`, so the NVMFS-IOzone sets the size of the segment for each move to be 512bit. We take Non-temporal store as an example(shown as Code 3), and the Non-temporal load is the opposite process. Firstly we move the data in DRAM (*Mainbuffer*_addr at Line 11) directly into the CPU `XMM#` register by `movdqu` instruction(shown as Line 2~5), then the `movntdq` instruction moves the data in the `XMM#` register directly into the NVMM (DAX-Mmap_addr at Line 10) (shown as Line 6~9). The entire process does not pass through the CPU cache, so the impact of cache miss and flushing CPU cache on the evaluation results can be avoided.

3.4 Embedded cleaning-ups

In order to get consistent and accurate evaluation results, we need the evaluation environment to be cleaned up for each test, or in other words to be in a "cold" state before each benchmarking. For example, there will be metadata cache (directories and nodes) and page cache in memory when benchmarking the POSIX interface of the traditional file system. For the NVMM-based file system's POSIX interface, there will be only the metadata cache because high speed page cache is bypassed. On the other hand, when benchmarking the mmap interface of traditional file system, there will be page cache in the memory; and for the DAX-mmap interface of the NVMM-based file system, we just need to take care of the CPU cache when high speed page cache and metadata are bypassed.

We found that most existing benchmark tools do not provide the functionality of cleaning-ups to flush page cache, directory entire cache and inode cache, etc. It is relied on the practitioners to do the cleaning-ups who knows the file systems and what to do. But still, sometimes in order to clean these caches people may even choose to manually unmount and remount the file systems, which is unfriendly and inefficient.

For the new and still evolving NVMM-based file systems, people may not know them well as the traditional ones. Incorrect or insufficient cleaning ups may result in inconsistent or biased evaluation results, especially for small file size benchmarking. Therefore to keep the evaluation environment consistent and user-friendly, we integrate the necessary cleaning-ups as options in NVMFS-IOzone, which can selectively clean page cache, metadata cache, or all caches by writing 1,2,3 to /proc/sys/vm/drop_caches. And the options can be extended to reflect the developing of new NVMM-based file systems, so reliable and useful results can be generated conveniently even for quick tests and casual users, especially in the industry.

4 EXPERIMENTS

4.1 Experiment setup

Existing research papers on NVMM usually use various simulated NVMM, such as retaining a region of DRAM as NVMM in their experiments. Fortunately, we work with Intel and use the latest model Optane DC persistent memory–NMA1XXD128GQS in our experiments, the size is 128GB and 4 pieces are used with a total capacity of 512GB. The CPU of the test platform is Intel(R) Xeon(R) Platinum 8268 CPU @ 2.90GHz with cache size of 36608 KB. The DRAM is Micron 36ASF4G72PZ-2G3AZ with the total capacity of 128GB. The SSD is Intel S3520 Series with 1.2TB. In the following experiments, the NVMM-based file systems, traditional file systems and memory file systems are running on NVMM—Optane, SSD and DRAM respectively.

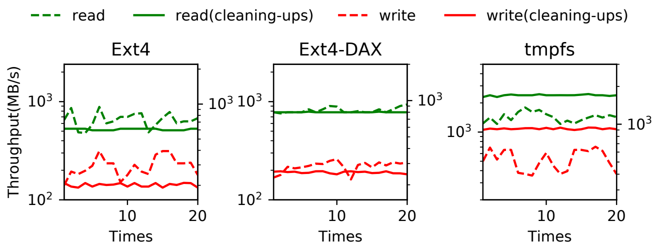

Figure 3: NVMFS-IOzone benchmarks for Ext4-DAX, Ext4 and tmpfs with and without cleaning-ups

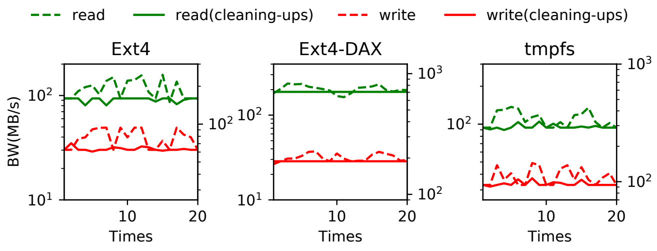

Figure 4: FIO benchmarks for Ext4-DAX, Ext4 and tmpfs with and without cleaning-ups.

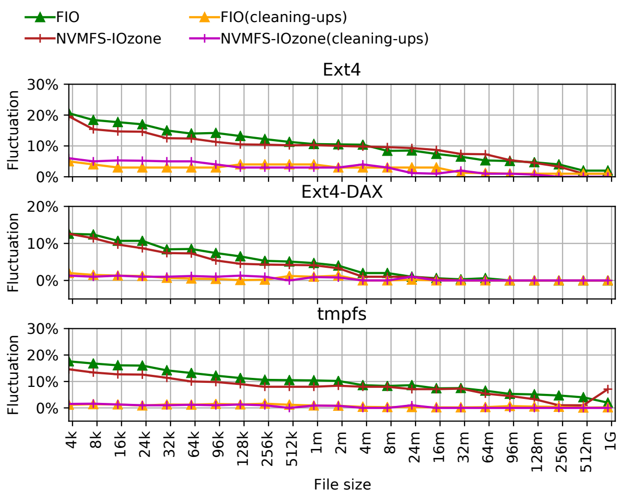

Figure 5: Evaluation for different file sizes with and without cleaning-ups. Fluctuation is represented by the relative standard deviation.

4.2 Cleaning-ups

In order to evaluate the embedded cleaning-ups function we select Ext4-DAX, Ext4, and tmpfs to represent typical NVMM-based file system, traditional file system, and memory file system respectively. As a comparison, we also test FIO alongside NVMFS-IOzone with and without proper cleaning-ups. To facilitate a fair comparison, the experiments mainly focus on the read/write performance of POSIX interface that is supported by all the three file systems. The IOzone's output is throughput(read/write rate) and FIO's is BW(I/O rate). Experiments are repeated 20 times for each workload, and the fluctuation in the results is expressed by the relative standard deviation. The results are shown in Figure 3 (NVMFS-IOzone) and Figure 4 (FIO), where the left y-axis is for read and the right y-axis is for write.

The maximum fluctuation for Ext4 with cleaning-ups option disabled is 20.6%, and for Ext4-DAX it is 12.6%. The reason that Ext4-DAX is lower is that Ext4 is affected by high speed page cache and metadata(inodes, directories) cache, but Ext4-DAX is only affected by metadata cache. Similar to Ext4, for tmpfs the maximum fluctuation is 19.3% without proper cleaning-ups. The main reason is that tmpfs lives in the page cache and on swap, so it is also affected by both page cache and metadata cache.

With cleaning-ups option enabled, the fluctuation of all the three file systems are limited to 6%. Moreover, we take the write performance evaluation as an example to show the fluctuation for different file sizes. We can see in Figure 5 that the larger file size is, the smaller fluctuation becomes. The main reason for this trend is that when the size of the file being evaluated is large, the overhead of reading from/writing to storage devices becomes the main bottleneck and less affected by the un-cleaned caches. With just an easy argument to the tool, NVMFS-IOzone can perform the appropriate cleaning-ups for the tests.

Another interesting observation can be found in the tests. For tmpfs in Figure 3 and Figure 4, the performance with cleaning-ups is higher than without when using NVMFS-IOzone, but it does not happen for FIO. The reason is that the original IOzone or NVMFS-IOzone without cleaning-ups does not empty the space occupied by the test file, so new space needs to be allocated in the next test. The embedded cleaning-ups empties the space so no new allocation is needed and the overhead is saved. On the contrary, FIO always empties the space before each new test. This shows that the original IOzone's evaluation for the memory file system is also not perfect.

4.3 The flushing method

Among the existing popular evaluation tools, only FIO and IOzone support mmap interface benchmarking. FIO focuses on the number of read and write operations per second(IOPS), and IOzone focuses on the throughput of read/write, which is the same as NVMFS-IOzone.

Therefore to facilitate comparison, we take IOzone to represent traditional benchmark tools and use Ext2-DAX, PMFS-new, XFS-DAX, NOVA and Ext4-DAX as evaluation objects. The purpose is to compare the `msync()` method with the new NVM-lib method used in NVMFS-IOzone. Given that the main function of `msync()` and NVM-lib is to ensure data flushed to the NVMM, the experiments focus on DAX-mmap write operation, and the results are shown in Figure 6. We can find that the throughput of Ext2-DAX, PMFS-new, XFS-DAX and Ext4-DAX with NVM-lib are higher than that with the traditional `msync()`, with an average increase of 1.4 times, and a maximum increase of 2.1 times, such as XFS-DAX and Ext4-DAX. These results confirm the effectiveness of replacing the `msync()` mechanism with the NVM-lib method.

An interesting observation can be seen with NOVA as shown in the fourth diagram of Figure 6, where its performance with `msync()` is already very close to that with the new NVM-lib method. We find out that NOVA has changed the `msync()` behavior in its implementation, where it defines functions `_mm_clflush()` and `_mm_clflushopt()` that can flush CPU cache without those unnecessary actions[39].

In conclusion, the overhead of the traditional `msync()` will largely under-evaluate the real performance of the NVMM-based file system, and by switching to NVM-lib, NVMFS-IOzone can reveal the actual performance. One side note that in Figure 6 there is a sharp increase in throughput after 1MB file size for XFS-DAX and Ext4-DAX, this is because

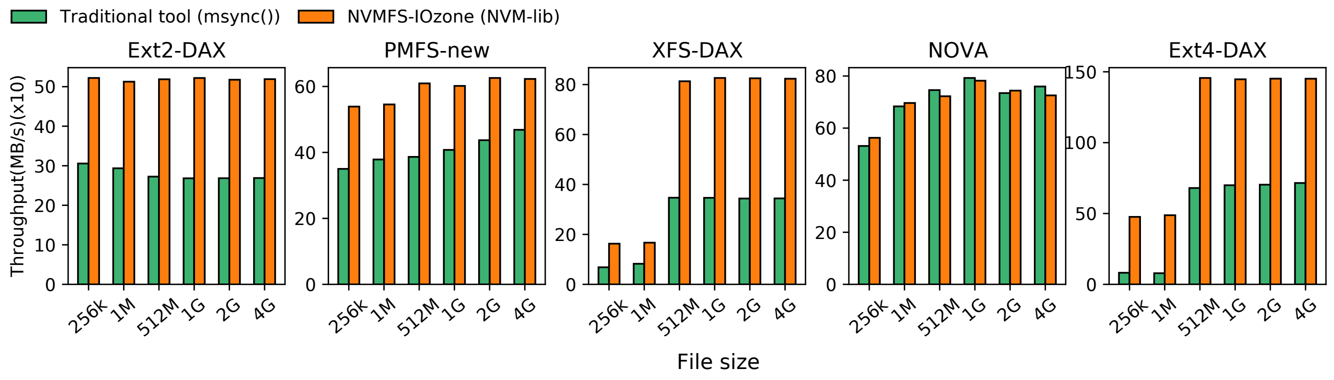

Figure 6: Comparison of traditional benchmark tools using `msync()` and NVMFS-IOzone using NVM-lib.

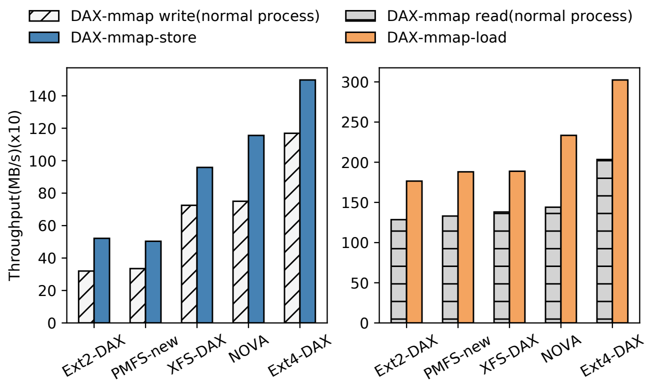

Figure 7: Comparison of DAX-mmap-load/store and the normal data flow (1→2→3→4 in Figure 2(a))

they support huge pages. We will see similar behavior also in the following experiments.

4.4 Direct Load/Store

The DAX-mmap interface allows the CPU to load/store NVMM directly. In order to prove the effectiveness of DAX-mmap-store/load, we compare it with the normal process (1→2→3→4 as shown in Figure 2(a)). The file size is set to 1GB. The results are shown in Figure 7. For all the file systems, we can see that the throughput of DAX-mmap-store/load is always higher than that of the normal process, because DAX-mmap-store/load avoid the *Mainbuffer* in DRAM and only involve the part of 3→4. Interestingly we can see that with the normal read and write process, XFS-DAX and NOVA have very similar throughput performance. However for DAX-mmap-store/load, the throughput of NOVA is higher than that of XFS-DAX. The fact is that the DAX-mmap of NOVA compresses the TLB entry compared to that of the XFS-DAX, which should lead to a better performance [11][39]. This is in consistent with the results of DAX-mmap-store/load but not visible with the normal read/write process.

In conclusion, the normal process used in existing evaluation tools may not reveal all the performance details of NVMM-based file systems because the overhead of passing the *Mainbuffer* of DRAM and other software stacks may overcast the real performance. DAX-mmap-store/load in NVMFS-IOzone let the CPU store/load NVMM directly so we can see more pure and actual performance of NVMM-based file systems without the relevant interference.

4.5 Bypassing CPU cache

In this subsection, we try the new method of bypassing the CPU cache(1→ 2 in Figure 2(b)) in comparison with the traditional method that doesn't bypass CPU cache(1→2→3→4 in Figure 2(a)). The write and read throughput of NVMM-based file systems using DAX-mmap and traditional file systems using mmap are shown in Figure 8 and 9, respectively.

For the write performance of all NVMM-based file systems (NOVA, PMFS-new,XFS-DAX, Ext4-DAX) in Figure 8, the performance gap of the two methods is large. The throughput of bypassing the CPU cache is on average 1.78 times higher than the traditional method, and the maximum is up to 2.3 times. Because the NVMM can be accessed directly by mapping and there is no overhead of disk I/O, so the overheads of cache miss and flushing will cause significant impact on the throughput. On the other hand, for read performance of all the NVMM-based file systems (XFS-DAX, Ext4-DAX, NOVA, PMFS-new) in Figure 9, we can observe that the performance gap is smaller. The main reason is that DAX-mmap read does not need to flush CPU cache to NVMM, so it is only the overhead of cache miss that causes the performance gap. For both the read and write performance of traditional file systems (XFS, Ext4), on the other hand, we can see in Figure 8 and Figure 9 that there is almost no performance

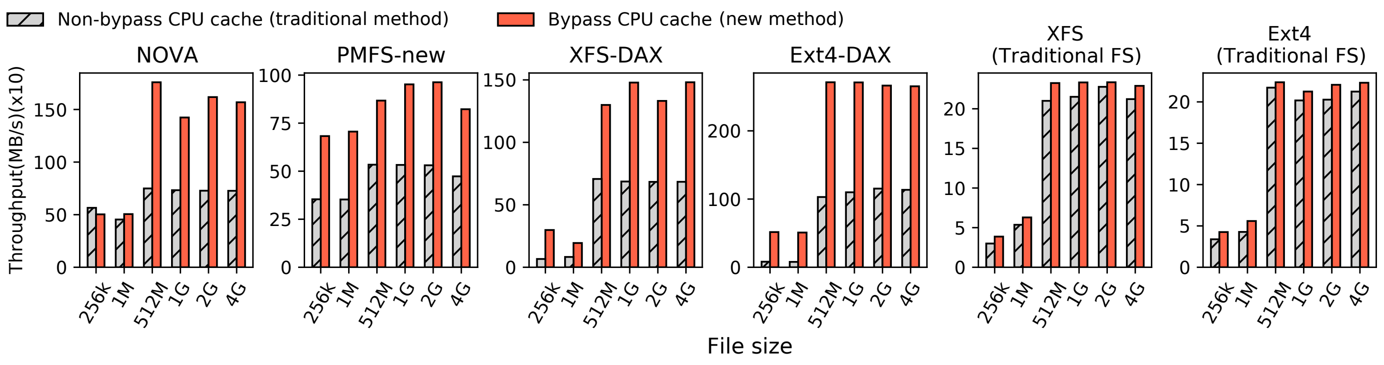

Figure 8: mmap/DAX-mmap WRITE performance comparison between traditional method and the new method (bypassing CPU cache) in NVMFS-IOzone.

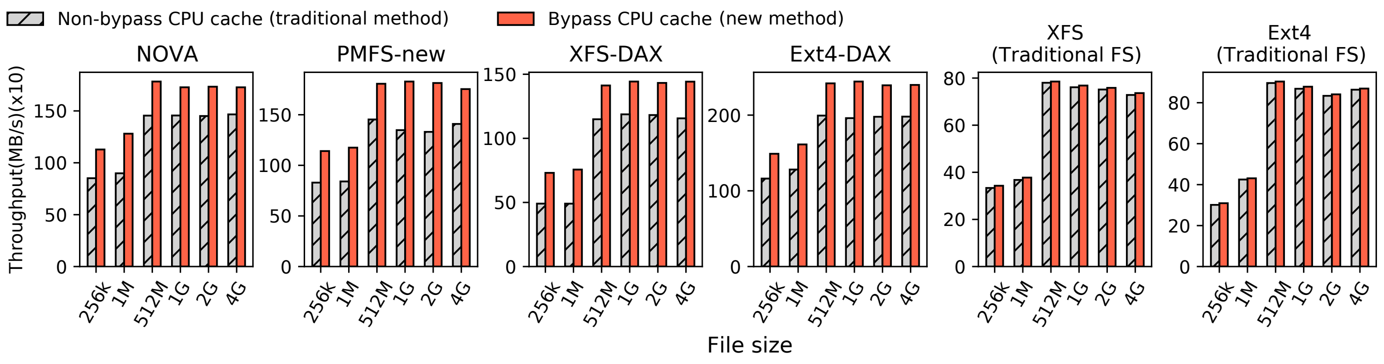

Figure 9: mmap/DAX-mmap READ performance comparison between traditional method and the new method (bypassing CPU cache) in NVMFS-IOzone.

gap between the two methods, because for them the disk I/O is the main bottleneck.

In conclusion, bypassing CPU cache is meaningless for traditional file systems because the bottleneck there is the disk I/O. However, for NVMM-based file systems, whether bypassing CPU cache or not has a big impact on performance, so it becomes a viable technique in pursuing for even higher performance for NVMM-based file system design. Making it an option in our new tool can be of big help to test those scenarios.

5 CONCLUSIONS

In this paper, a new evaluation tool called NVMFS-IOzone is proposed to meet the new requirements of conveniently and accurately benchmarking the new NVMM-based file systems. The read/write and synchronization mechanisms are redesigned based on the new features of the DAX-mmap interface, and their effectiveness is verified with physical NVM hardware. The evaluation results show that our new benchmark tool can reveal the real performance not seen by traditional evaluation tools. Furthermore, the new tool also supports test with bypassing CPU cache, which may become useful for the new NVMM-based file systems. Integrated cleaning-ups options are also provided in the new tool to support convenient and user-friendly evaluations.

ACKNOWLEDGMENTS

Corresponding author: Dagang Li. We thank the reviewers for their constructive comments that have significantly improved the paper. This work was supported by Shenzhen Research Program under Grant KQJSCX20180323174744219 and the internship provided by Intel Asia-Pacific Research & Development Ltd.

REFERENCES

[1] [n.d.]. https://newsroom.intel.com/news-releases/intel-and-micronproduce.
[2] [n.d.]. optane persistent memory brief. https://www.intel.com/content/www/us/en/products/docs/memory-storage/optane-persistent-memory/optane-dcpersistent-memory-brief.html..
[3] [n.d.]. PMFS-new is baesd the PMFS provided by intel. https://github.com/NVSL/PMFS-new.
[4] 1988. Bonnie measures the performance of Unix file system operations. http://www.textuality.com/bonnie/.
[5] 1999. tiobench - Threaded I/O bench. https://linux.die.net/man/1/tiobench.

[6] 2019. Direct Access for files. https://www.kernel.org/doc/Documentation/filesystems/dax.txt.
[7] Ameen Akel, Adrian M Caulfield, Todor I Mollov, Rajesh K Gupta, and Steven Swanson. 2011. Onyx: A Prototype Phase Change Memory Storage Array. *HotStorage* 1 (2011), 1.
[8] Joy Arulraj, Andrew Pavlo, and Subramanya R Dulloor. 2015. Let's talk about storage & recovery methods for non-volatile memory database systems. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 707–722.
[9] Jens Axboe. [n.d.]. fio - Flexible I/O tester rev. 3.16 - FIO's documentation. https://fio.readthedocs.io/en/latest/fio_doc.html.
[10] Kumud Bhandari, Dhruva R Chakrabarti, and Hans-J Boehm. 2012. Implications of CPU caching on byte-addressable non-volatile memory programming. *Hewlett-Packard, Tech. Rep. HPL-2012-236* (2012).
[11] Mingming Cao, Suparna Bhattacharya, and Ted Ts'o. 2007. Ext4: The Next Generation of Ext2/3 Filesystem.. In *LSF*.
[12] Don Capps and William Norcott. 2008. IOzone filesystem benchmark.
[13] Rémy Card. 1993. second extended file system is a file system for the Linux kernel. https://en.wikipedia.org/wiki/Ext2.
[14] Adrian M Caulfield and Steven Swanson. 2013. Quicksan: a storage area network for fast, distributed, solid state disks. In *ACM SIGARCH Computer Architecture News*, Vol. 41. ACM, 464–474.
[15] Feng Chen, Michael P Mesnier, and Scott Hahn. 2014. A protected block device for persistent memory. In *2014 30th Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 1–12.
[16] Youmin Chen, Jiwu Shu, Jiaxin Ou, and Youyou Lu. 2018. HiNFS: A persistent memory file system with both buffering and direct-access. *ACM Transactions on Storage (TOS)* 14, 1 (2018), 4.
[17] Joel Coburn, Adrian M Caulfield, Ameen Akel, Laura M Grupp, Rajesh K Gupta, Ranjit Jhala, and Steven Swanson. 2012. NV-Heaps: making persistent objects fast and safe with next-generation, non-volatile memories. *ACM Sigplan Notices* 47, 4 (2012), 105–118.
[18] Russell Coker. 1999. Bonnie++ is for Unix-like operating systems. https:https://linux.die.net/man/1/tiobench.
[19] Jonathan Corbet. 2017. the futher of DAX. https://lwn.net/Articles/717953/.
[20] Part Guide. 2011. Intel® 64 and ia-32 architectures software developer's manual. *Volume 3B: System programming Guide, Part* 2 (2011).
[21] Intel. [n.d.]. PMFS is a file system for persistent memory, developed by Intel. https://github.com/linux-pmfs/pmfs.
[22] Intel. May,2019. Intel 64 and IA-32 Architectures Software Developer's Manual. https://software.intel.com/sites/default/files/managed/39/c5/325462-sdm-vol-1-2abcd-3abcd.pdf.
[23] Joseph Izraelevitz, Jian Yang, Lu Zhang, Juno Kim, Xiao Liu, Amirsaman Memaripour, Yun Joon Soh, Zixuan Wang, Yi Xu, Subramanya R Dulloor, et al. 2019. Basic performance measurements of the intel optane DC persistent memory module. *arXiv preprint arXiv:1903.05714* (2019).
[24] Takayuki Kawahara. 2010. Scalable spin-transfer torque ram technology for normally-off computing. *IEEE Design & Test of Computers* 1 (2010), 52–63.
[25] Alexey Kopytov. 2012. Sysbench manual. *MySQL AB* (2012), 2–3.
[26] Mika Kuoppala. 2002. Tiobench-Threaded I/O bench for linux.
[27] David E Lowell and Peter M Chen. 1997. Free transactions with rio vista. In *SOSP*, Vol. 97. 92–101.
[28] Dushyanth Narayanan and Orion Hodson. 2012. Whole-system persistence. *ACM SIGARCH Computer Architecture News* 40, 1 (2012), 401–410.
[29] Moinuddin K Qureshi, Vijayalakshmi Srinivasan, and Jude A Rivers. 2009. Scalable high performance main memory system using phase-change memory technology. In *ACM SIGARCH Computer Architecture News*, Vol. 37. ACM, 24–33.
[30] Simone Raoux, Geoffrey W Burr, Matthew J Breitwisch, Charles T Rettner, Y-C Chen, Robert M Shelby, Martin Salinga, Daniel Krebs, S-H Chen, H-L Lung, et al. 2008. Phase-change random access memory: A scalable technology. *IBM Journal of Research and Development* 52, 4.5 (2008), 465–479.
[31] Andy Rudoff. 2017. Persistent memory programming. *Login: The Usenix Magazine* 42 (2017), 34–40.
[32] Mahadev Satyanarayanan, Henry H Mashburn, Puneet Kumar, David C Steere, and James J Kistler. 1994. Lightweight recoverable virtual memory. In *ACM SIGOPS Operating Systems Review*, Vol. 27. ACM, 146–160.
[33] Inc Silicon Graphics. [n.d.]. XFS_Papers_and_Documentation. http://xfs.org/index.php/XFS_Papers_and_Documentation.
[34] Adam Sweeney, Doug Doucette, Wei Hu, Curtis Anderson, Mike Nishimoto, and Geoff Peck. 1996. Scalability in the XFS File System.. In *USENIX Annual Technical Conference*, Vol. 15.
[35] Vasily Tarasov, Erez Zadok, and Spencer Shepler. 2016. Filebench: A flexible framework for file system benchmarking. *login: The USENIX Magazine* 41, 1 (2016), 6–12.
[36] Haris Volos, Andres Jaan Tack, and Michael M Swift. 2011. Mnemosyne: Lightweight persistent memory. In *ACM SIGARCH Computer Architecture News*, Vol. 39. ACM, 91–104.
[37] Chundong Wang and Weng-Fai Wong. 2013. SAW: System-assisted wear leveling on the write endurance of NAND flash devices. In *Proceedings of the 50th Annual Design Automation Conference*. ACM, 164.
[38] Michael Wu and Willy Zwaenepoel. 1994. eNVy: a non-volatile, main memory storage system. In *ACM SIGOPS Operating Systems Review*, Vol. 28. ACM, 86–97.
[39] Jian Xu and Steven Swanson. 2016. {NOVA}: A Log-structured File System for Hybrid Volatile/Non-volatile Main Memories. In *14th {USENIX} Conference on File and Storage Technologies ({FAST} 16)*. 323–338.
[40] Wei Xu, Hongbin Sun, Xiaobin Wang, Yiran Chen, and Tong Zhang. 2009. Design of last-level on-chip cache using spin-torque transfer RAM (STT RAM). *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 19, 3 (2009), 483–493.
[41] Wei Xu, Hongbin Sun, Xiaobin Wang, Yiran Chen, and Tong Zhang. 2009. Design of last-level on-chip cache using spin-torque transfer RAM (STT RAM). *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 19, 3 (2009), 483–493.
[42] Yiying Zhang and Steven Swanson. 2015. A study of application performance with non-volatile main memory. In *2015 31st Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 1–10.
[43] Jishen Zhao, Sheng Li, Doe Hyun Yoon, Yuan Xie, and Norman P Jouppi. 2013. Kiln: Closing the performance gap between systems with and without persistence support. In *2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 421–432.