2021 IEEE 10th Non-Volatile Memory Systems and Applications Symposium (NVMSA) | 978-1-6654-2375-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/NVMSA53655.2021.9628430

An Empirical Study of NVM-based File System

Hongwei Duan, Liang Shi, Qingfeng Zhuge, Edwin Hsing-Mean Sha, Changlong Li, Yujiong Liang
Big Data & Intelligent System Laboratory
East China Normal University
Shanghai, China
hwduan@stu.ecnu.edu.cn, {lshi, qfzhuge, edwinsha}@cs.ecnu.edu.cn, liclong@mail.ustc.edu.cn, lsyjfancy@gmail.com

***Abstract*—Emerging byte-addressable, non-volatile memory provides opportunities for preserving files in memory. However, there is no systematic performance study across different file systems. This paper evaluates the performance of modern NVM-based file systems (e.g., PMFS, SIMFS, and NOVA) with a series of preliminary studies. Several interesting findings are concluded with our study, which should be well considered by file system designers during the employment. Furthermore, this paper explores the visualization of NVM-based file systems and develops NVMPlayer. To the best of our knowledge, this is the first graphical tool to display the internal mechanisms of NVM-based file systems. It demonstrates how visualization can help us shed light on the complex phenomena in NVM-based file system and expose new opportunities for optimization.**

***Index Terms*—non-volatile memory, file system, in-memory file systems, performance, wear leveling, visualization**

I. Introduction

Emerging non-volatile memory (NVM) technologies such as spin-torque transfer, phase change, resistive memories [1,2,3] and 3D XPoint promise to revolutionize I/O performance. 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 [4,5]. 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) offers the possibility of storage systems that combine the best characteristics of both technologies.

Conventional file systems are not suitable for hybrid memory systems because they are built for the performance characteristics of disks and rely on disks' consistency guarantees for correctness [6]. To exploit the high performance of NVM and efficiently support more flexible access patterns, researchers and companies have developed several file systems designed specifically for NVMM. Other works have adapted existing file systems to NVMM by adding DAX support [7]. The common practice of these NVMM-based file systems are bypassing the page cache and the block-based I/O software stack, where the POSIX interface usually needs to search the metadata in DRAM. With a new mmap interface, CPU can load/store NVMM directly according to the mapping without searching the metadata. 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. However, there exists no systematic performance study across different file systems.

This paper evaluates the performance of NVM-based file systems under various real-world workloads on NVM. We selected commonly used server-class workloads such as webserver, fileserver, webproxy as they differ from each other in terms of data access patterns, metadata-data ratios, etc. We evaluated and compared the results of above workloads on NVM-based file systems - SIMFS, NOVA, PMFS. From these evaluations, several interesting observations are found, which are helpful for future system designs: (1) Some NVM-based file systems that use journaling for metadata updates may suffer performance degration with the directory width increasing. (2) Continuously increasing the number of threads, the performance will be worse, so we should limit the number of threads. (3) Different access pattens have different thread saturation point. Multithreading may improves read performance, especially sequential read, not necessarily write performance. Usually sequential read has the highest saturation point, followed by random read. (4) When access granularity is too small, the latency of NVM-based file systems increases sharply.

The increasing complexity of state-of-the-art NVM management justifies the adoption of new research and analysis techniques. Just as graphs illustrate phenomena that are hard to identify in tables, and just as one picture is said to be worth a thousand words, we claim that one video is worth a thousand histograms. To establish this claim, we developed NVMPlayer, an graphical tool for visualizing data layout and movement on NVM. This tool will give us a better understanding of how our data gets from one place to another and why. It also encouraged our use of NVMPlayer for educational purposes in undergraduate courses and projects, and for the presentation of new ideas and designs in academic conferences and industrial collaboration meetings.

In the rest of this paper, Section II presents the background and related work. In Section III, we will introduce the experimental environment and testing tools. We then summarize and analyze the key results in Section IV. We introduce NVMPlayer in Section V. Finally, section VI is the conclusion and future work.

This work is supported by the NSFC 62072177 and 61972154, Shanghai Science and Technology Project (20ZR1417200) and the Fundamental Research Funds for the Central Universities. The corresponding author is Liang Shi (lshi@cs.ecnu.edu.cn).

978-1-6654-2375-5/21/$31.00 ©2021 IEEE

II. Background and Related Work

In this section, we will introduce some background on NVM-based file systems and related work which is close to our work.

A. NVM-based File Systems

To exploit the high bandwidth, low latency, and byte-addressable features of NVM devices, new NVM-aware file systems such as PMFS [8], SCMFS [9], Ext4-DAX , NOVA [10], SIMFS [11] and Ziggurat [12] etc, start to emerge to support the new NVM-based systems. The common practice of these NVM-based file systems is to bypass the page cache and block-based I/O software stack. This is enabled by a technique called Direct Access (DAX), which allows an application to map the pages of an NVM-backed file into its address space and then access it via load and store instructions. Ext4-DAX extend the original Ext4 file systems with DAX features, respectively. SCMFS utilizes the existing memory management module of the operating system to help mange the file system space. PMFS is a lightweight DAX file system that uses journaling for metadata updates. NOVA adopts the log-structured file system techniques to exploit the fast random access that NVMs provide. SIMFS fully utilizes the memory mapping hardware at the file access path, which is an efficient file system. Ziggurat is a tiered file system that combines NVMM and slow disks to create a storage system with near-NVMM performance and large capacity.

B. Related Work

Priya Sehgal et al. [13] concentrate on evaluating performance of the following file system configurations under different workloads on NVM: (a) default access to traditional file system, (b) fine-tuned access to traditional file systems through mount and format options, and (c) NVM-optimized file system, such as PMFS. They use various in-place-update and log-structured file systems to evaluate traditional file systems. Yang Li et al. [19] compare some NVMM file systems such as PMFS and NOVA with several mainstream traditional file systems. Then, they evaluate these file systems with specific workloads and analyze the experimental results to draw some insightful conclusions.

In 2015, Gala Yang et al. developed SSDPlayer [14], a graphical tool for visualizing the various processes that cause data movement on SSDs,

However, as far as we know, there is no systematic performance study across different emerging file systems mentioned above. Additionally, SSDPlayer is unsuitable for the study of NVM-based file system.

III. Experiment Setup

In this section, we compare three typical existing file systems, SIMFS, NOVA and PMFS on performance. First, we present the micro-operation results obtained by standard benchmark FIO [16] and Filebench [17]. Then, we show the macro-operation results including application workloads. The experimental results show that SIMFS outperforms other file systems, including NOVA and PMFS on the majority of cases. Additionally, the experiments also show that SIMFS almost reaches the memory bus bandwidth for large I/O size.

The experiments are conducted on a system equipped with 48GB DRAM and a 2.4 GHz Intel E5-2640 10-core processor. We configure 16GB DRAM acting as volatile memory for volatile data structures of the system, and use the rest of 32 GB acting as persistent memory for in-memory file systems, including PMFS, SIMFS, and NOVA. It is important to note that our experiments focus on file systems and do not evaluate the different types of NVM (e.g. PCM, STT-MRAM, ReRAM etc).

IV. Experiment Result and Observations

This section studied the access performance of three typical NVM-based file systems, those are PMFS, SIMFS, NOVA from multiple aspects, including basic I/O performance, the impact of number of threads, directory width, access granularity on performance. FIO is used to generate four different workloads, random read, sequential read, random write, and sequential write. We also use Filebench to generate different workloads, fileserver, webserver, varmail, webproxy. By changing number of threads, the size of directory width, the size of access granularity and other factors, explore the access performance of NVM-based file system under different workloads. The detailed experimental methodologies and results analysis are discussed below.

A. Throughput with Single Thread

Now, we compare the performance of file systems accessed by a single thread. The experimental results are shown in Fig. 1. The ”Size of I/O requests” in the figures means the size of data requested in each I/O request issued by the benchmark. Note that from the view of FIO, it does not matter if the target file system is on hard disk or memory. Even 1 KB/2 KB requests are submitted to in-memory file systems, the sizes of I/O requests are still 1 KB/2 KB. The test file size is 512MB.

As shown in Fig. 1, for sequential read, NOVA performs best when access granularity is larger than 16KB, followed by SIMFS, then PMFS. For sequential write, SIMFS has the best performance when access granularity ranges from 16KB to 128KB , followed by NOVA and PMFS. For random read, NOVA is the best for small granularity access within 8KB, and SIMFS is better for large granularity access. For random write, the performance of SIMFS is the best under the small-grained access up to 64KB, NOVA and PMFS are about the same. With the access granularity ranging from 64KB to 256KB, NOVA is the best. What’s more, each file system has better read performance than write performance .

We also find that for each type of operation, the throughputs of all file systems increase when the size of I/O requests grows larger, as shown in Fig. 1. It is because that when the size of I/O requests gets larger, the number of I/O requests issued by benchmarks decreases and then the overhead of I/O system calls reduces.

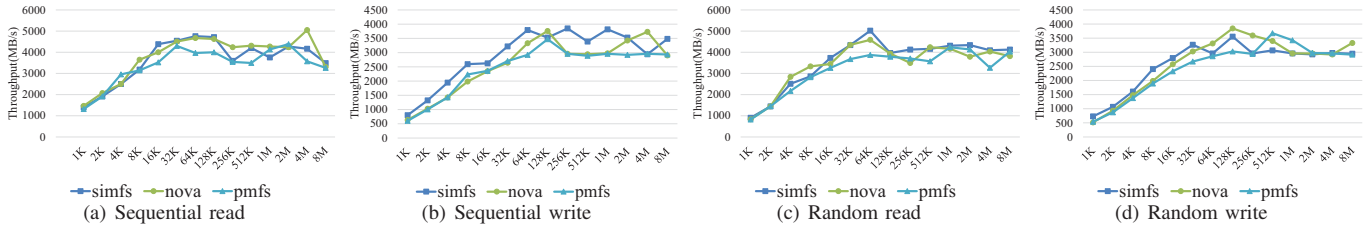


Fig. 1. Throughput comparison of three file systems with single thread.

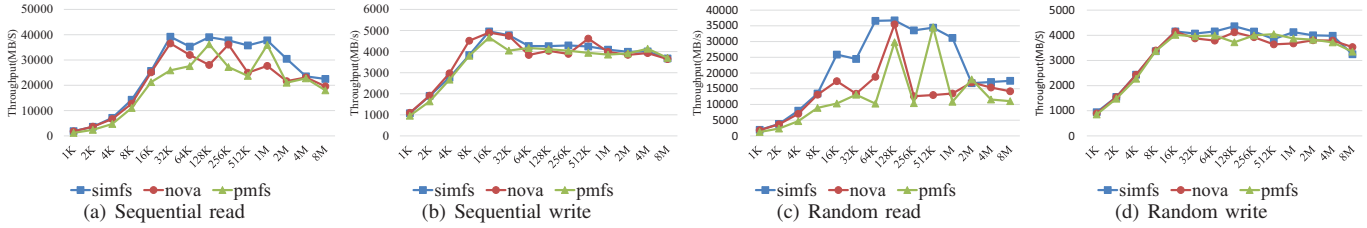


Fig. 2. Throughput comparison of three file systems with 16 threads.

B. Throughput with Multiple Threads

Fig. 2 plots the FIO throughput of different NVM-based file systems with 16 theads. As shown in Fig. 2, we find that when the number of threads increases, the performance of simfs improves sharply, especially read performance, whether sequentially or randomly, outperform NOVA and PMFS. This can be interpretated as contributed advantages of using the virtual address space of user process by multiple threads.

The metadata structure of PMFS and NOVA is quite different from that of SIMFS. PMFS organizes the file data pages by B-tree. Data accesses to a file in PMFS have large overhead on the software search of the B-tree. A NOVA inode contains pointers to the head and tail of its log. The log is a linked list of 4 KB pages, and the tail always points to the latest committed log entry. NOVA scans the log from head to tail to rebuild the DRAM data structures when the system accesses the inode for the first time. Data accesses to a file in NOVA also have large overhead on the software search. On the contrary, a file in SIMFS organizes its data pages by contiguous file virtual address space embedded into process virtual address space. Thus, the data accesses to such a file have little overhead on the search of metadata.

C. Performance of Metadata Operations

Now we measure the performance of metadata operations of NOVA, PMFS, and SIMFS using Filebench.

We use default settings for all the workloads except that the number of thread is set as 1. In the workloads, the mean number of files in the same directory is set to 100 except for listdir, which is 5.

The experimental results of metadata operations are shown in Fig. 3. We find that SIMFS outperforms NOVA and PMFS on the majority of cases. Compared with PMFS and NOVA, SIMFS also obtains much better performance on createfiles, stafiles and listdirs, and achieves about the same performance

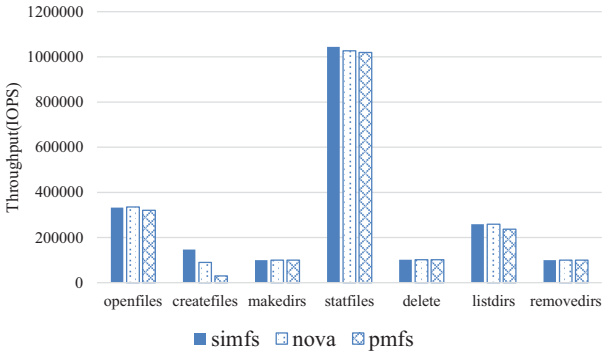


Fig. 3. Metadata operations .

on other cases except the openfiles. Compared with NOVA, the performance of SIMFS is a little weaker. It is because that SIMFS directly locates the address of inode of a file with an index, i.e., the inumber, whereas PMFS walks through the B-tree to seek an inode. Although SIMFS has to insert the file page table into the process page table in open operation, SIMFS is just slightly slower (about 0.9 percent) than NOVA on openfiles.

D. Filebench Application Load

We evaluate the performance of four multi-threaded application workloads produced by Filebench. We use the default settings for the four workloads.

Fig. 4 plots the Filebench throughput of different NVM-based file systems. As shown in Fig. 4, SIMFS performs slightly worse than NOVA under WebProxy loads and is the best under other loads. The performance of SIMFS is 1.04 times (for varmail) to 1.05 times (for webserver) , and 1.88 times (for fileserver) better than those of PMFS. Besides, the

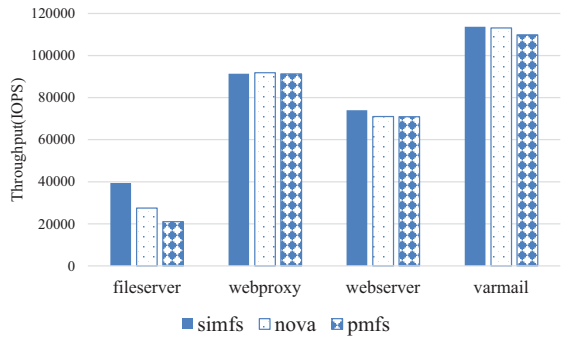


Fig. 4. Filebench application load .

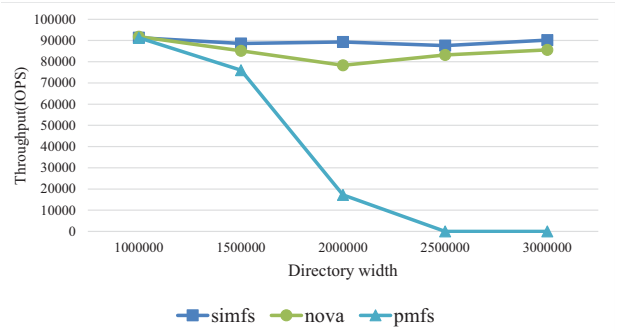


Fig. 5. Webproxy .

performance improvement of SIMFS over NOVA ranges from 5 percent (for webserver) to 43.3 percent (for fileserver).

The performance improvement of SIMFS obtained from the simplified metadata structures and the use of file virtual address space and MMU, which can be refered in paper [11].

The variation of performance improvement over different workloads is caused by the characteristics of the workload. For example, the performance improvement of fileserver and webserver is higher than that of varmail. It is because that fileserver and webserver commits more data accesses and less metadata operations than varmail does, and SIMFS gains significant performance improvement from data accesses. Webproxy has a deep directory depth, and the metadata management strategy that NOVA designs for NVM can help it to find the target files much faster.

E. The Impact of Directory Width on Performance of File System

Fig. 5 plots the Filebench throughput of different NVM-based file systems with different directory width.

As shown in Fig. 5, with the directory width increasing, the throughput of SIMFS change a little, followed by NOVA, while the throughput of PMFS decreases sharply. We can conclude that SIMFS is the least sensitive to increasing directory width, with little performance change, and PMFS performs the worst. This is because that with the directory width increasing, the directory depth decreases. When the directory depth is less than 1, all the files are contained within one large directory. The performance of such a fileset depends on how lookup and other metadata operations are handled in large directories. We found that SIMFS and NOVA use hash tree (indexed directory) to store directory entries while PMFS do not, and hence they perform badly.

Finding 1: *Some NVM-based file systems that use journaling for metadata updates may suffer performance degration with the directory width inereasing.*

F. The Impact of Number of Threads on Performance of File Syetem

In this subsection, we show the performance of in-memory file systems accessed by multiple threads. The experiments are conducted by FIO benchmarks, where threads may read/ write the same file. The sizes of the files are set to 512MB in the experiments.

Fig. 6 and Fig. 7 show the experimental results of SIMFS and NOVA with multiple threads respectively, where the vertical axes show the aggregated throughputs of threads. The number of threads in the tests is from 2 to 32. As each thread has its own submission queue (SQ) and completion queue (CQ), we use the terms ”threads” and ”queues” interchangeably when no confusion occurs. As shown in the Fig. 6 and Fig. 7, usually 16 or 32 is the sufficient number of threads to support a NVM-based file system, as the throughput of DRAM-like NVM saturates with a very low queue depth.

One interpretation is that interrupt service routine (ISRs) are invoked too frequently when increasing the number of threads (queues) till 32. As NVMe supports 65536 queues, this ISR overhead may not be acceptable in modern systems and will be a serious issue for future NVM technologies.

As shown in Fig. 6a and Fig. 7a, we can observe that sequential read performance improves as the numbers of threads in the tests increase. As shown in Fig. 6c and Fig. 7c, we can observe that the random read performance of SIMFS improves as the number of threads in the tests increases to 16, while NOVA's is only 8. As shown in Fig. 6b, Fig. 6d, Fig. 7b, Fig. 7d, as the number of threads in the tests increases, write performance isn't improving.

We can draw the following conclusions: (1) Different file systems have different thread saturation point, usually the thread saturation point of SIMFS is higher than NOVA. (2) Different access pattens have different thread saturation point, the thread saturation point of sequential reads have the highest saturation point, followed by random reads.

Finding 2: *We observe that, continuously increasing the number of threads can significantly hurt the average throughput. When the thread count goes beyond the number of threads that saturate the throughput, the performance rapidly degrades, since the saturation point means that the bandwidth of the given PCIe bus has already run out.*

This is because more threads generating small I/O requests are allowed to run concurrently under the block NVM. However, the throughput brought by the DRAM-like NVM by

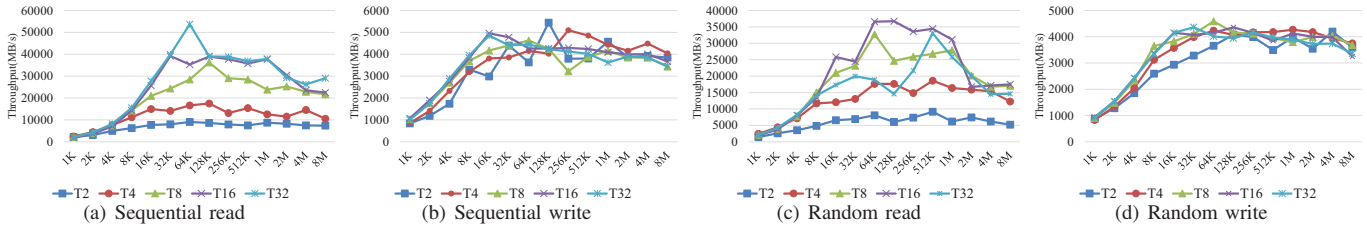


Fig. 6. Throughput comparison of SIMFS with varying number of threads.

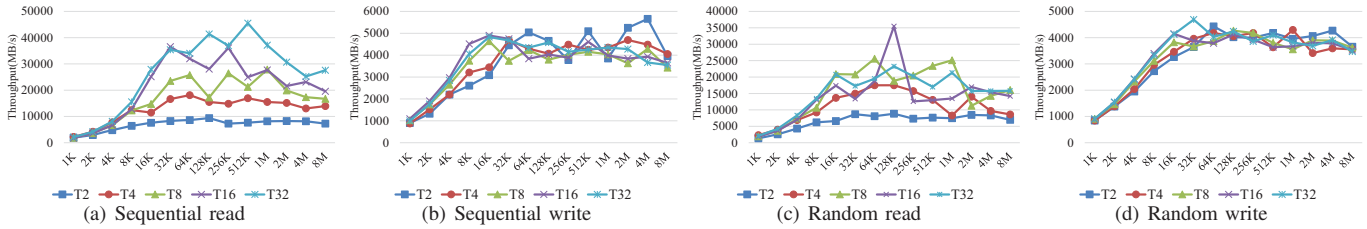


Fig. 7. Throughput comparison of NOVA with varying number of threads.

doubling the number of threads is far less than the PCIe bus bandwidth capacity.

Systems should minimize the number of concurrent threads targeting a single DIMM simultaneously. An Optane DIMM's limited store performance and limited buffering at the iMC, and on the DIMMs combine to limit its ability to handle accesses from multiple threads simultaneously. We have identified two distinct mechanisms that contribute to this effect.

Finding 3: *Different access pattens have different thread saturation point. Multithreading improves read performance, especially sequential reading, not necessarily write performance. Usually two threads are enough to saturate NVM write bandwidth. More threads may degrade the performance.*

This can be interpreted that multiple readers can read at the same time without locking, while multiple writers can't write at the same time and need locking, which increases the software overhead.

Also we find in most situations, the performance of the NVM device saturates with 16 or 32 threads, even with the high-speed next generation NVMe.

G. The Impact of Access Granularity on Performance of File Syetem

In this subsection, we evaluate the impact of access granularity on performance of file syetem by single thread. The experiments are conducted by FIO benchmarks, where a thread may read/ write the same file. The sizes of the files are set to 512MB in the experiments.

Fig. 8 shows the latencies of sequential and random reads (or writes) for three file systems with the size of IO request of 64B, 128B, 256B, 512B, 1KB, 2KB, . . . , 512KB and 1MB respectively. As shown in Fig. 8, by this test, we can conclude that when access granularity is lower than 256B, the latency increases sharply. Internally, Optane DIMMs update Optane contents at a 256B granularity. This granularity, combined with a large internal store latency, means that smaller updates are inefficient since they require the DIMM to perform an internal read-modify-write operation causing write amplification. The less locality the accesses exhibit, the more severe the performance impact.

Finding 4: *When access granularity is too small, usually when access granularity is lower than 256B, the latency of NVM-based file systems increases sharply.*

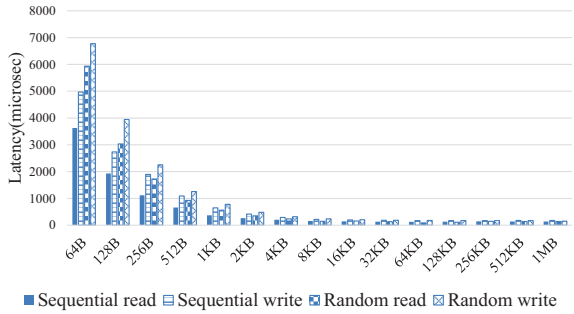
V. NVMPLAYER

Additionally, we explore the visualization of NVM-based file systems, and developed NVMPlayer, a graphical tool for visualizing the various processes in NVM-based file systems.

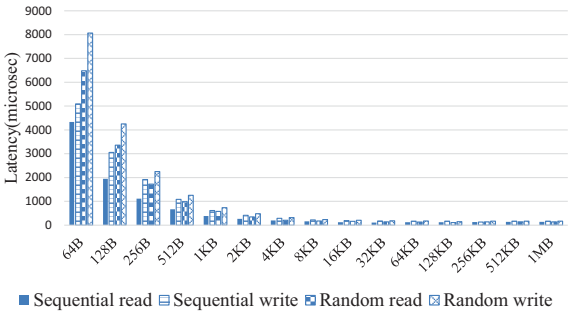
Many wear leveling optimizations [15] [18] incur additional internal data movement. Quantifying the write amplification is important for analyzing the effect of such optimizations on the performance and durability of the NVM device. However, doing so is not always trivial and requires a deep understanding of the interacting causes of data movement within each device.

Currently available simulators output internal state and statistics in the form of lists, tables and histograms, from which deriving internal processes is cumbersome and requires a great deal of skill and imagination.

In Fig. 9, the logical page (or inode) number is written on the left side, and the corresponding write number is written on the right side. NVMPlayer separates pages into partitions according to their temperature. It is used with traces in which each input write request is tagged by a temperature tag. The user can specify the number of partitions. For example, there are ten temperature tags. When the temperature of one page is in the range [1,2], one can specify it is a cold page, warm in [3,5], and hot when the temperature is above 6, and the migration threshold is 10. The cold page, warm page, hot page are taged with blue, orange, red respectively. The migration method is that you first find the free page, then find



(a) The latency of NOVA



(b) The latency of SIMFS

Fig. 8. Read/write latency under different access granularity.

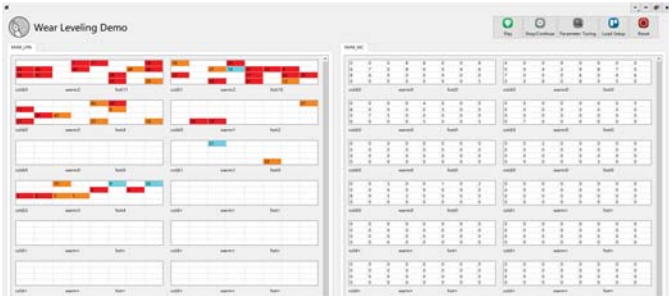


Fig. 9. Wear leveling demo.

the coldest page for exchange if you cannot find one. When the temperature of one page change, its colour may change correspondingly, so we can clearly see the wear leveling process. We believe much more complicated phenomena can be identified and analyzed as visualization becomes a standard research tool.

VI. Conclusions

The ever-increasing complexity of NVM-based systems and their management makes it increasingly difficult to analyze related new methods and optimizations. In this paper, we evaluate the performance of three typical state-of-the-art NVM-based file systems, and several interesting observations are concluded from the empirical evaluations, which will help systems designers during the development of NVM-based file systems. We also showed that a graphical illustration of data movement processes on NVM can facilitate a much deeper understanding of their causes and effects. We thus believe that visualization should be a standard mechanism in the tool box of every NVM oriented research or development team.

References

[1] A. Akel, A. M. Caulfield, T. I. Mollov, R. K. Gupta, and S. Swanson. Onyx: A protoype phase change memory storage array. In Proceedings of the 3rd USENIX Conference on Hot Topics in Storage and File Systems, HotStorage'11, pages 2-2, Berkeley, CA, USA, 2011. USENIX Association.

[2] T. Kawahara. Scalable Spin-Transfer Torque RAM Technology for Normally-Off Computing. Design & Test of Computers, IEEE, 28(1):52-63, Jan 2011.

[3] S. Raoux, G. Burr, M. Breitwisch, C. Rettner, Y. Chen, R. Shelby, M. Salinga, D. Krebs, S.-H. Chen, H. L. Lung, and C. Lam. Phase-change random access memory: A scalable technology. IBM Journal of Research and Development, 52(4.5):465-479, July 2008.

[4] 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.

[5] 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, nonvolatile memories. ACM Sigplan Notices 47, 4 (2012), 105–118.

[6] T. S. Pillai, V. Chidambaram, R. Alagappan, S. Al-Kiswany, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. All File Systems Are Not Created Equal: On the Complexity of Crafting Crash-Consistent Applications. In 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14), pages 433–448, Broomfield, CO, Oct. 2014. USENIX Association.

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

[8] S. R. Dulloor, S. Kumar, A. Keshavamurthy, P. Lantz, D. Reddy, R. Sankaran, and J. Jackson, System software for persistent memory,in Proc. 9th ACM Euro. Conf. Comput. Syst., 2014, pp. 1-15.

[9] WU, X., AND REDDY, A. Scmfs: a file system for storage class memory. In Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (2011), ACM, p. 39.

[10] Jian Xu and Steven Swanson. NOVA: A Log-structured File System for Hybrid Volatile/Non-volatile Main Memories. In 14th USENIX Conference on File and Storage Technologies (FAST 16), pages 323-338, Santa Clara, CA, February 2016. USENIX Association.

[11] Sha H M , Chen X , Zhuge Q , et al. A New Design of In-Memory File System Based on File Virtual Address Framework. IEEE Transactions on Computers, 2016, 65(10):1-14.

[12] Zheng S, Hoseinzadeh M, Swanson S. Ziggurat: A Tiered File System for Non-Volatile Main Memories and Disks. 17th USENIX Conference on File and Storage Technologies (FAST 19). 2019: 207-219.

[13] Sehgal P , Basu S , Srinivasan K , et al. An empirical study of file systems on NVM. Mass Storage Systems & Technologies. IEEE, 2015.

[14] Gala Yadgar, Roman Shor, Eitan Yaakobi, and Assaf Schuster. 2015. It's not where your data is, it's how it got there. In Proceedings of the 7th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage'15).

[15] Liu Q , Varman P . Ouroboros Wear Leveling for NVRAM Using Hierarchical Block Migration. Acm Transactions on Storage, 2017, 13(4):1-31.

[16] (2014). Fio: flexible I/O tester [Online]. Available: http://freecode.com/projects/fio

[17] (2014). Filebench [Online]. Available: http://filebench.sourceforge. net

[18] Chen X , Sha H M , Zeng Y , et al. Efficient wear leveling for inodes of file systems on persistent memories. 2018 Design, Automation Test in Europe Conference Exhibition (DATE). 2018.

[19] Yang, L. , Fang L , Nong X, et al. "File System for Non-volatile Main Memories: Performance Testing and Analysis." File System for Non-volatile Main Memories: Performance Testing and Analysis 0.