Rethinking Benchmarking for NVM-based File Systems

Mingkai Dong Qianqian Yu Xiaozhou Zhou Yang Hong Haibo Chen Binyu Zang

Shanghai Key Laboratory of Scalable Computing and Systems
Institute of Parallel and Distributed Systems, Shanghai Jiao Tong University

Abstract

Fast, byte-addressable NVM promises near-cache latency and near memory bus throughput for file systems, which is evident by recent development of a number of NVM file systems (NVMFS). However, a key approach to measuring their performance, file system benchmarks, remains unchanged as those for disk-based file systems (DFS). In this paper, we analyze the pitfalls of using DFS benchmarks to evaluate NVMFS and propose several changes to existing DFS benchmarks to better characterize the performance of NVMFS.

1. Introduction

NVM technologies such as PCM, STT-MRAM, Memistor, NVDIMM and Intel/Micron's 3D XPoint will likely prevail in the near future. Though different in some details and characteristics, such NVM technologies generally provide promising features such as byte-addressability, non-volatility and near-DRAM speed. Such features enable NVM to revolutionize the storage hierarchy of modern computer systems.

In response to the emergence of NVM technologies, several NVM file systems (NVMFS) have been proposed recently to exploit the advanced features of NVM. For example, BPFS [1] utilizes a tree-structured file system layout and a technique called *short-circuit shadow paging* to provide atomic, fine-grained updates to NVM; PMFS [2] exploits the processor's paging and memory ordering features for optimizations such as fine-grained logging and transparent large page supports. NOVA [15] adopts the log-structured file systems and in-DRAM index trees to further improve the file system performance. Generally, such NVMFS notably outperform traditional disk-based file systems in both throughput and latency.

APSys '16, August 04-05, 2016, Hong Kong, Hong Kong
 ISBN 978-1-4503-4265-0/16/08…$15.00
DOI: http://dx.doi.org/10.1145/2967360.2967379

However, as a key approach to measuring file system performance, file system benchmarks remain largely unchanged. NVMFS designers continuously use file system benchmarks designed for disk-based file systems (DFS) to evaluate the performance of NVMFS, which may easily lead to inaccurate results and/or conclusions.

In this paper, we conduct a measurement study on state-of-the-art NVMFS using popular DFS benchmarks. We present an analysis on the results using two standard metrics of file system benchmarking, i.e., throughput and latency. Our analysis uncovers several pitfalls of using DFS benchmarks to evaluate NVMFS, including inaccurate throughput caused by scalability issues and NUMA unawareness, as well as imprecise and varied latency.

Based on our analysis, we modify existing DFS benchmarks to mitigate some of the pitfalls, with the goal of making the benchmarks better suit the evaluation of NVMFS. We also summarize the evaluation results and provide some suggestions to NVMFS designers for more accurate evaluation results.

The rest of the paper is organized as follow: Section 2 introduces the impact of NVM on the storage hierarchy, the file systems and the benchmarking means. Section 3 then presents the evaluation results of NVMFS using DFS benchmarks, followed by by a summary of the analysis and some suggestions in section 4. Section 5 discusses the limitations of this paper and section 6 briefly discusses related work. Finally, section 7 concluded this paper.

2. What are the Differences?

The emergence of NVM technologies leads to significant changes on the storage hierarchy, the file system design and the requirements for benchmarks.

Differences in the storage hierarchy: Disks are usually treated as a secondary storage in the hierarchy of modern computer systems. As shown in Figure 1(a), accesses to disks need to invoke multiple I/O instructions to control the controllers to transfer the data from/to disks. This process is often slow due to the slow speed of disks and, especially, the lengthy seek time.

NVM, however, is a primary storage that can be directly accessed by the processors. As shown in Figure 1(b), ac-

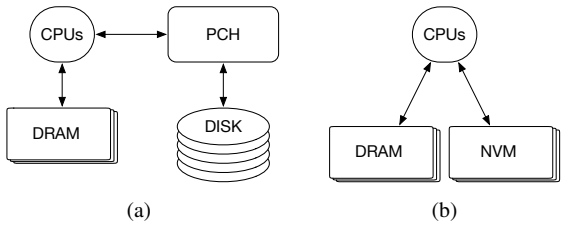


Figure 1: Architectures with disks and NVM

cesses to NVM can be simply done by issuing load/store instructions similar to accesses to DRAM, which are fast and in byte-level granularity.

Differences in the file system design: The architectural differences result in different design considerations for file systems built on top of them.

Figure 2(a) shows the software stack of DFS. It is noteworthy that all accesses to the data stored in the disks must go through the block layer and the device drivers. The block layer provides the I/O scheduler and the page cache (i.e., bcache) which speeds up the read from cached blocks and helps write coalescence. The device driver layer is required to transfer data from/to disks. To achieve high performance, a DFS design tries to reduce the disk I/O as much as possible (or, at least, reduce random disk I/O by minimizing disk seeks). Hence, the main prior efforts are focused on delaying the writes [5, 6], better utilizing the page cache, and adopting disk-friendly data/metadata management and file system layout [9, 10].

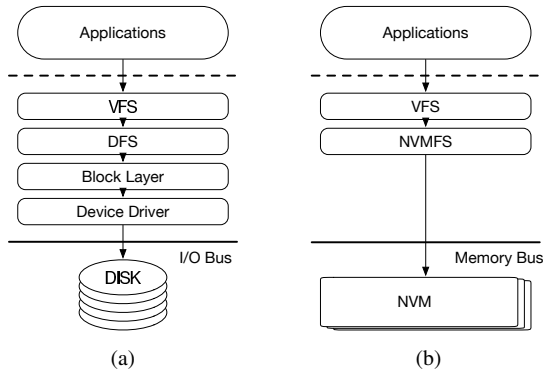


Figure 2: Software stacks with DFS and NVMFS

Thanks to the byte-addressability of NVM, NVMFS can directly access each byte in NVM without block layers or device drivers as shown in Figure 2(b). This shifts the design focus on reducing unnecessary NVM writes and expensive persistency operations (e.g. *clflush* and *pcommit*). For example, PMFS [2] condenses the size of inodes to reduce the writes in journals and NOVA [15] leverages a hybrid of in-NVM log-structured data/metadata management and in-DRAM index trees to further eliminate unnecessary writes and persistency operations.

Different requirements for the benchmarks: The evolution of storage technologies boosts the performance of file systems, which presents different requirements for benchmarking file systems.

File system benchmarks designed for DFS usually assume that the file systems have high latency and low throughput. Thus the efficiency of the benchmark itself is not very important as long as it does not affect the performance of evaluated file systems.

However, such an assumption does not hold upon NVMFS. Benefiting from the near-DRAM speed of NVM, NVMFS can usually achieve high throughput and low latency so that a benchmark designed for DFS can easily become the bottleneck in the benchmarking process and thus may report inaccurate results. Hence, the efficiency and scalability of the file system benchmark is a key to get accurate results.

3. A Case Study

NVMFS with high throughput and low latency may shift the bottleneck in the file system benchmarking. In this section, we conduct a case study by re-evaluating the performance of state-of-the-art NVMFS using DFS benchmarks (as in their original papers) and analyze the result to illustrate the pitfalls.

3.1 Setup

We choose to use PMFS [2] and NOVA [15], two state-of-the-art open-source NVMFS, to conduct the evaluation. We also evaluate EXT4 on disks with default mount options (e.g. ordered mode) to present the different performance trends between DFS and NVMFS. The DFS benchmark we use is Filebench 1.4.9.1 [3] and 1.5 [4], which are widely used in recent NVMFS evaluations [2, 7, 15]. There are four application workloads that are frequently used in NVMFS benchmarking: Fileserver, Webserver, Webproxy and Varmail. In this paper, we only show the evaluation results of Fileserver, in which each thread creates a file, writes the whole file, appends to the file, reads the whole file, and finally deletes the file. The file is closed after each operation and is opened again before the next one. The file size follows gamma distribution with 128K as the mean number. The results of other three workloads are not shown but our preliminary evaluation shows that the issues shown in this section appear in all these workloads.

Since the real NVM is not currently available to us yet, we emulate it using DRAM as other standard practices (e.g., [8]). Besides, we insert 200 ns delays to emulate the latency of *pcommit* instructions and other characteristics remain unchanged as DRAM. We further discuss the emulated characteristics in Section 5.

All evaluations are conducted in Debian 8.4 with Linux Kernel 4.5.2 running on an Intel machine with two 2.3GHz

Ten-Core Intel Xeon E5 chips equipped with 128GB memory in which 62GB are used to emulate NVM[1].

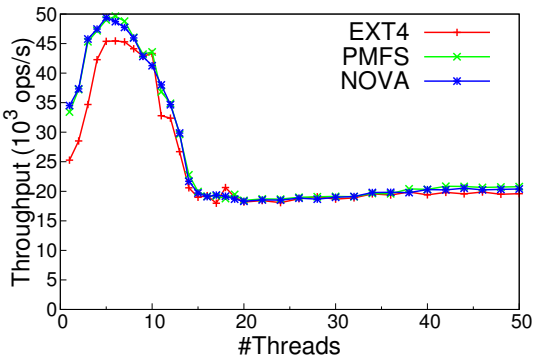


Figure 3: Scalability of Filebench 1.4.9.1

3.2 Throughput and Scalability

To evaluate the throughput and scalability, we run the Fileserver workload on PMFS, NOVA and EXT4 with different numbers of threads. Other configurations remain as default.

We first use Filebench 1.4.9.1 to perform the evaluation since it is the latest released version and has been used for many years. The result is shown in Figure 3. Surprisingly, all these three file systems achieve similar throughput and the throughput does not scale with increased thread numbers. We double-check the evaluation and find there are several scalability issues in Filebench 1.4.9.1.

We then switch to use Filebench 1.5 which is still in development and re-perform the evaluation. As shown in Figure 4 (the line indicated by "-nb"), the previous scalability issues seem to have been fixed and there is a huge performance gap between NVMFS and DFS [2]. This makes sense and matches the speed differences between NVM and disks. However, by observing the scalability of PMFS and NOVA, we find the throughput drops after 10 threads, which is far less than the default thread numbers in the workload. Thus, if the whole scalability figure is not shown, simply using the result of default configurations of the workloads will lead to inaccurate results and makes the comparison of different file systems unfair. The throughput of on-disk EXT4 does not drop with the increased number of threads [3]. This further confirms our previous conjecture that the high speed NVMFS bring anomalies to traditional slow file system benchmarks.

[1] At first, we use 64G physical memory starting from the 64G-th physical memory address. However, we find that NUMA node 0 occupies the physical address from 0 to 66G. In order to arrange all NVM to a single NUMA node, we start to emulate the NVM from 66G and thus the size is shrank to 62G for simplicity.

[2] Filebench 1.5 also improves the performance for each thread.

[3] We enlarge the file size to 1M to mitigate the effect of page cache.

We further investigate the reason for the performance drop and find that the NUMA effect is the chief culprit. The machine used for evaluation has two NUMA nodes, and the emulated NVM all reside in NUMA node 1. Thanks to the kernel NUMA balancer, a thread is more likely to be scheduled to the NUMA node it frequently accesses, i.e., NUMA node 1 in this case. Thus, if the number of threads is small, the NUMA balancer can help to reduce the number of remote memory accesses and prevent the cache line bouncing. When the number of threads exceeds the number of cores in one NUMA node, remote memory accesses and cache line bouncing are inevitable during the evaluation, which results in the collapse of the throughput.

Having known the reason, we bind the threads of Filebench to one NUMA node and perform the evaluation again. The results are shown in Figure 4 together with the result without binding (indicated by "-nb"). The line with "-b0" represents the throughput for binding all threads to NUMA node 0, and line with "-b1" indicates the result of binding to NUMA node 1.

It is shown that there is no performance collapse after the binding. This is because binding all threads to one NUMA node prevents the cache line bouncing among different NUMA nodes.

Since the emulated NVM is on NUMA node 1, binding threads to NUMA node 1 achieves better performance than binding to NUMA node 0.

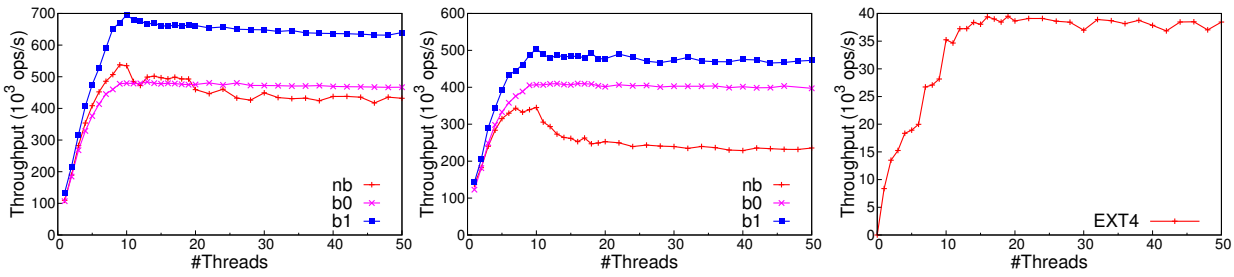
3.3 Latency

Latency is another important metric for file system benchmarking. However, evaluating the latency for NVMFS is more challenging due to the low latency of NVMFS operations.

By default, Filebench reports the latency in millisecond. However, NVMFS usually achieves operation latency in multiple microseconds, which is too small to be measured in millisecond. Thus we first modify the report unit to make it more precise before the latency evaluation.

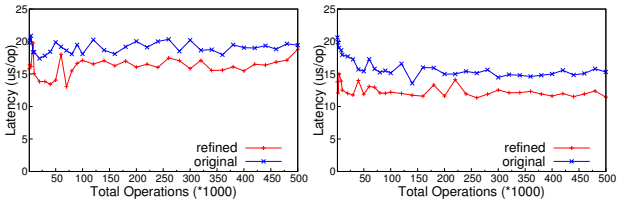
During the modification, we also find that Filebench takes the time it uses to manage its own metadata into account when testing the latency of the file systems. These include some AVL tree operations holding the locks which might be negligible when evaluating DFS operations but dominates in NVMFS. To illustrate the effect of such an omission, we refine the Filebench by excluding these irrelevant metadata operations and carry out extra experiments in which ten threads create, append and then delete files concurrently [4]. To acquire a more precise latency, the experiment is repeated multiple times with different number of operations. The result is reported in Figure 5, in which "original" indicates the latency of the original Filebench and "refined" shows the latency of the refined Filebench. We can observe that the effect is significant with respect to the low latency.

[4] For simplicity, we only measure the time of creating files.



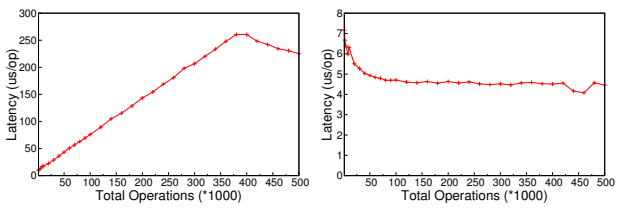
(a) NOVA (b) PMFS (c) EXT4

Figure 4: Scalability of Filebench 1.5



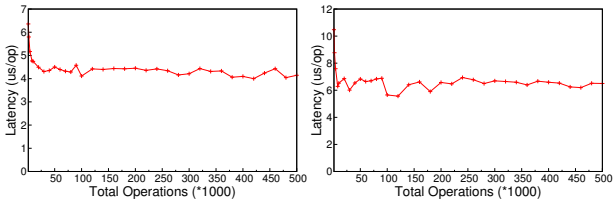
(a) PMFS (b) NOVA

Figure 5: Latency of default and refined Filebench



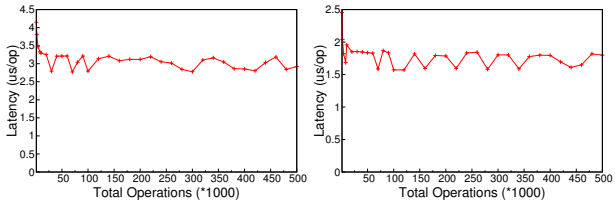
(a) PMFS (b) NOVA

Figure 6: Latency trends without deletion



(a) PMFS (b) NOVA

Figure 7: Latency trends with deletion



(a) PMFS (b) NOVA

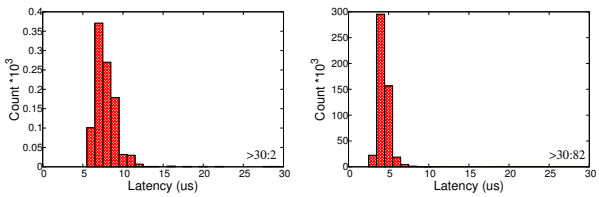
Figure 8: Append latency with deletion

We then use the refined Filebench to test the latency of creating files in two approaches. The first approach, *the non-deletion approach*, is to continuously create a file and append 4KB data for 16 times without deletion, thus the size of the directory keeps increasing during the test. The second approach, *the deletion approach*, is to create a file in an empty directory, append 4KB data for 16 times and then remove it before creating another file. The deletion approach aims to measure the shortest time of creating files since this minimizes the effect of directory organization.

We run these tests using one thread and vary the number of creations from 1K to 500K to reveal the influence of testing iterations. The results are shown in Figure 6 and Figure 7.

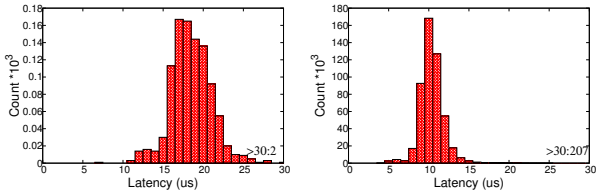
As expected, for the non-deletion approach, the latency increases with the increasing number of operations as shown in Figure 6(a). This makes sense since the price of organizing files in the directory increases when there are more files. This is evident for PMFS which uses array-based dentry management. For NOVA (Figure 6(b)), which uses in-DRAM radix trees to manage a directory, such a latency increase is negligible.

However, in the deletion approach (Figure 7), the latency is high when tested with less operations and it drops with the increase of total operation numbers. This is because the latency of NVMFS is so low that it can be easily disturbed. With the increase of tested operations, the disturbance is gradually eliminated from the average. The latency for append (shown in Figure 8) presents the same trend.



(a) 1K operations (b) 500K operations

Figure 9: Latency distributions for PMFS



(a) 1K operations (b) 500K operations

Figure 10: Latency distributions for NOVA

When we conduct the evaluation for latency, we find that although the latency always decreases in one run, it is unstable across different runs.

Figure 9 and Figure 10 show the distributions of latency of creating files for PMFS and NOVA respectively in one run of the deletion approach. The notation in the right bottom corner presents the number of operations that takes longer time than 30 us.

We choose two scenarios to show the distribution, one with 1K operations, the other 500K.

In the 500K scenario, the average latency is more concentrated and shorter than that in 1K scenario. In both scenarios, latency varies greatly, from a few microseconds to thousands of microseconds.

We end this section by summarizing the modifications in Filebench during the evaluation.

- We add options to the Filebench workloads to bind all the threads to specific NUMA nodes or CPUs to evaluate the influence of NUMA.
- We reformat the output of the Filebench to report the latency in higher precision for NVMFS.
- We remove Filebench metadata operations from the latency measurement to make the benchmarking much more precise.

4. Summary and Suggestions

In this section, we summarize the pitfalls of benchmarking NVMFS using DFS benchmarks and give some suggestions for future works.

4.1 Summary

With the memory hierarchy goes up, we need to rethink the benchmark design for NVMFS.

Scalability dominates. DFS benchmarks seem not to have paid much attention to the scalability of the benchmark itself, since DFS are often the bottleneck and the scalability of benchmarks does not matter. This might explain why Filebench 1.4.9.1, which has severe scalability issues, has been frequently used for a long time. The emergence of high speed NVMFS raises higher requirements for the benchmarking that the scalability of benchmarks should not affect the evaluation results, which most of current DFS benchmarks will do. More efforts are needed in either optimizing existing DFS benchmarks to suit NVMFS benchmarking or designing new benchmarks for NVMFS.

NUMA matters. Benchmarks designed for DFS do not need to consider the influence of NUMA since it barely affects the evaluation result. However, in the benchmarking of NVMFS, the influence of NUMA is significant as shown in Section 3.

Latency varies. Previous work has pointed out that the working set size impacts the latency significantly [12]. The precondition is further released for NVMFS. Even with the same workload, the latency can vary a lot in one run. It's more difficult to measure the low latency of NVMFS precisely since a subtle disturbance will affect the latency significantly. One possible solution is to iterate the test more times to expect the latency to get stable. This works in some of the tests, but is not feasible for more complex evaluations such as the impact of directory structures.

4.2 Suggestions

We suggest that NVMFS benchmarks should be designed and implemented with great care. Different factors, such as NUMA, scalability, and the low latency of NVMFS should be taken into consideration. In the workload design, some other metrics, such as NUMA-friendliness and latency with high precision, that are meaningless and often ignored in DFS should also be measured for NVMFS.

For the designers of NVMFS, if the DFS benchmarks are used to evaluate NVMFS, we strongly recommend testing more aspects to make sure that the DFS benchmarks are not the bottlenecks and the results reflect the true performance of NVMFS. For example, we recommend to present the whole scalability figure rather than the throughput of a fixed number of threads. If there are scalability issues, the possible reasons might be the scalability of the benchmark itself and cache line bouncing among NUMA nodes. When the latency is evaluated, multiple runs are needed to get a relatively accurate latency for NVMFS. Besides, as mentioned in previous work [12], the average latency sometimes is not a good metric to reflect the true latency. Hence, we encourage the designers to report a histogram of latency distribution.

5. Limitation and Discussion

Other NVM characteristics: There are some other characteristics of NVM that are different from DRAM, including extra write latency, lower bandwidth and endurance issues. These characteristics are not emulated in this paper for two reasons. First, these characteristics vary among different NVM techniques and thus a simple emulation according to one specific NVM technique does not help in generality. Secondly we do not have the real NVM available and emulating some of these characteristics is difficult without special hardware platforms. Nevertheless, we do think these characteristics are important in the evaluation of NVMFS and thus, if possible, should be considered and evaluated in the benchmarking.

Other dimensions of file system benchmarking: As described in previous work [12], there are many dimensions in the benchmarking of file systems. Although not all these dimensions are covered in this paper, we believe that this paper is conducive to the benchmarking and development of other dimensions.

File systems, benchmarks and applications: The relationships among the file systems, benchmarks and real applications are interesting. File systems are designed according to the requirements of the applications, which are indirectly reflected by some of the benchmarks. Benchmarks, which are bonds between file systems and applications, need to both expose the different characteristics among file systems and mimic the behaviour of real applications to provide an approximation of performance in real product environments. The applications, to achieve better performance, might adjust their logic to suit the characteristics of underlaying file systems exposed by the results of benchmarking. These three influence and promote each other and a change in file systems will be reflected in benchmarks and applications.

6. Related Work

To the best of our knowledge, this is the first paper to discuss NVMFS benchmarking using DFS benchmarks. However, benchmarking of traditional file systems has been discussed for a long time. Tang et al. [11] criticizes the file system benchmarks used in 1994. Traeger et al. [14] discusses the difficulties in benchmarking the file systems and introduces benchmarks, industry experiences and benchmarking guidelines at that time. Traeger et al. [13] surveys 415 file system and storage benchmarks from 106 papers and finds that most popular benchmarks are flawed and many papers fail to provide true performance. The authors also provide guidelines to improve future performance evaluations. Tarasov et al. [12] proposes several dimensions of file system benchmarking and reviews widely used benchmarks. A conclusion is drawn by experiments that many benchmarks can be fragile and produce inaccurate results.

7. Conclusion

This paper analyzed the pitfalls of using DFS benchmarks to evaluate NVMFS and concluded that as the memory hierarchy goes one level up, we need to rethink the benchmarking for NVMFS. Based on the analysis, this paper proposed and evaluated several changes to existing DFS benchmarks to better characterize NVMFS. Besides, this paper also provided suggestions to help improve the evaluation of NVMFS.

Acknowledgements

We thank Jian Xu from UC San Diego for pointing out the scalability issue of Filebench 1.4.9.1 and suggesting the development version. We also thank our shepherd Jian Ouyang and the anonymous reviewers for their insightful comments and suggestions. This work is supported in part by China National Natural Science Foundation (61572314), the Top-notch Youth Talents Program of China, Zhangjiang Hi-Tech program (No. 201501-YP-B108-012), and Singapore NRF (CREATE E2S2).

References

[1] CONDIT, J., NIGHTINGALE, E. B., FROST, C., IPEK, E., LEE, B., BURGER, D., AND COETZEE, D. Better i/o through byte-addressable, persistent memory. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles* (2009), ACM, pp. 133–146.

[2] DULLOOR, S. R., KUMAR, S., KESHAVAMURTHY, A., LANTZ, P., REDDY, D., SANKARAN, R., AND JACKSON, J. System software for persistent memory. In *Proceedings of the Ninth European Conference on Computer Systems* (2014).

[3] Filebench-1.4.9.1. `https://sourceforge.net/projects/filebench/files/filebench/filebench-1.4.9.1/`, 2011.

[4] Filebench-1.5. `https://sourceforge.net/p/filebench/code/ci/filebench-1.5/tree/`, 2015.

[5] GANGER, G. R., MCKUSICK, M. K., SOULES, C. A., AND PATT, Y. N. Soft updates: a solution to the metadata update problem in file systems. *ACM Transactions on Computer Systems (TOCS) 18*, 2 (2000), 127–153.

[6] GANGER, G. R., AND PATT, Y. N. Metadata update performance in file systems. In *Proceedings of the 1st USENIX conference on Operating Systems Design and Implementation* (1994), USENIX Association.

[7] OU, J., SHU, J., AND LU, Y. A high performance file system for non-volatile main memory. In *European Conference on Computer Systems* (2016).

[8] How to emulate persistent memory. `http://pmem.io/2016/02/22/pm-emulation.html`, 2016.

[9] RODEH, O., BACIK, J., AND MASON, C. Btrfs: The linux b-tree filesystem. *ACM Transactions on Storage (TOS) 9*, 3 (2013), 9.

[10] ROSENBLUM, M., AND OUSTERHOUT, J. K. The design and implementation of a log-structured file system. *ACM*

Transactions on Computer Systems (TOCS) 10, 1 (1992), 26–52.

[11] TANG, D., AND SELTZER, M. Lies, damned lies, and file system benchmarks. Tech. rep., Technical Report TR-34-94, Harvard University, 1994.

[12] TARASOV, V., BHANAGE, S., ZADOK, E., AND SELTZER, M. Benchmarking file system benchmarking: It* is* rocket science.

[13] TRAEGER, A., ZADOK, E., JOUKOV, N., AND WRIGHT, C. P. A nine year study of file system and storage benchmarking. *ACM Transactions on Storage (TOS) 4*, 2 (2008), 5.

[14] TRAEGER, A., ZADOK, E., MILLER, E. L., AND LONG, D. D. Findings from the first annual file and storage systems benchmarking workshop. In *Initial workshop report* (2008), vol. 6, Citeseer.

[15] XU, J., AND SWANSON, S. Nova: A log-structured file system for hybrid volatile/non-volatile main memories. In *USENIX Conference on File and Storage Technologies* (2016), pp. 323–338.