Benchmarking File System Benchmarking: It *IS* Rocket Science

Appears in the proceedings of the 13th USENIX Workshop in Hot Topics in Operating Systems (HotOS XIII)

Vasily Tarasov, Saumitra Bhanage, and Erez Zadok
Stony Brook University

Margo Seltzer
Harvard University

Abstract

The quality of file system benchmarking has not improved in over a decade of intense research spanning hundreds of publications. Researchers repeatedly use a wide range of poorly designed benchmarks, and in most cases, develop their own ad-hoc benchmarks. Our community lacks a definition of *what* we want to benchmark in a file system. We propose several dimensions of file system benchmarking and review the wide range of tools and techniques in widespread use. We experimentally show that even the simplest of benchmarks can be fragile, producing performance results spanning orders of magnitude. It is our hope that this paper will spur serious debate in our community, leading to action that can improve how we evaluate our file and storage systems.

1 Introduction

Each year, the research community publishes dozens of papers proposing new or improved file and storage system solutions. Practically every such paper includes an evaluation demonstrating how good the proposed approach is on some set of benchmarks. In many cases, the benchmarks are fairly well-known and widely accepted; researchers present means, standard deviations, and other metrics to suggest some element of statistical rigor. It would seem then that the world of file system benchmarking is in good order, and we should all pat ourselves on the back and continue along with our current methodology. We think not.

We claim that file system benchmarking is actually a disaster area—full of incomplete and misleading results that make it virtually impossible to understand what system or approach to use in any particular scenario. In Section 3, we demonstrate the fragility that results when using a common file system benchmark (Filebench [10]) to answer a simple question, "How good is the random read performance of Linux file systems?". This seemingly trivial example highlights how hard it is to answer even simple questions and also how, as a community, we have come to rely on a set of common benchmarks, without really asking ourselves *what we need* to evaluate.

The fundamental problems are twofold. First, accuracy of published results is questionable in other scientific areas [8], but may be even worse in ours [11, 12]. Second, we are asking an ill-defined question when we ask, "Which file system is better." We limit our discussion here to the second point.

What does it mean for one file system to be better than another? Many might immediately focus on performance, "I want the file system that is faster!" But faster under what conditions? One system might be faster for accessing many small files, while another is faster for accessing a single large file. One system might perform better than another when the data starts on disk (e.g., its on-disk layout is superior). One system might perform better on meta-data operations, while another handles data better. Given the multi-dimensional aspect of the question, we argue that the answer can *never* be a single number or the result of a single benchmark. Of course, we all know that—and that's why every paper worth the time to read presents multiple benchmark results—but how many of those give the reader any help in interpreting the results to apply them to any question other than the narrow question being asked in that paper?

The benchmarks we choose should measure the aspect of the system on which the research in a paper focuses. That means that we need to understand precisely what information any given benchmark reveals. For example, many file system papers use a Linux kernel build as an evaluation metric [12]. However, on practically all modern systems, a kernel build is a CPU bound process, so what does it mean to use it as a file system benchmark? The kernel build does create a large number of files, so perhaps it is a reasonable meta-data benchmark? Perhaps it provides a good indication of small-file performance? But it means nothing about the affect of file system disk layout if the workload is CPU bound. The reality is that it frequently reveals little about the performance of a file system, yet many of us use it nonetheless.

We claim that file systems are multi-dimensional systems, and we should evaluate them as such. File systems are a form of "middleware" because they have multiple storage layers above and below, and it is the interaction of all of those layers with the file system that really affects its behavior. To evaluate a file system properly we first need to agree on the different dimensions, then agree on how best to measure those different dimensions and finally agree on how to combine the results from the multiple dimensions.

In Section 2 we review and propose several file system evaluation criteria (i.e., a specification of the various dimensions) and then examine commonly used benchmarks relative to those dimensions. In Section 3 we examine 1–2 small pieces of these dimensions to demonstrate the challenges that must be addressed. We conclude and discuss future directions in Section 4.

Related Work. In 1994 Tang et al. criticized several file system benchmarks in wide-spread use at that time [11]. Surprisingly, some of these benchmark are still in use today. In addition, plenty of new benchmarks have been developed, but quantity does not always mean quality. Traeger and Zadok examined 415 file system benchmarks from over 100 papers spanning nine years and found that in many cases benchmarks do not provide adequate evaluation of file system performance [12]. Table 1 (presented later in Section 2) includes results from that past study. We omit discussing those papers here again, but note that the quality of file system benchmarking does not appear to have improved since that study was published in 2008. In fact, this topic was discussed at a BoF [13] at the FAST 2005, yet despite these efforts, the state of file system benchmarking remains quite poor.

2 File System Dimensions

A file system abstracts some hardware device to provide a richer interface than that of reading and writing blocks. It is sometimes useful to begin with a characterization of the I/O devices on which a file system is implemented. Such benchmarks should report bandwidth and latency when reading from and writing to the disk in various-sized increments. IOmeter [9] is an example of such a benchmark; we will call these *I/O benchmarks*.

Next, we might want to evaluate the efficacy of a file system's on-disk layout. These should again evaluate read and write performance as a function of (file) size, but should also evaluate the efficacy of the on-disk meta-data organization. These benchmarks can be challenging to write: applications can rarely control how a file system caches and prefetches data or meta-data, yet such behavior will affect results dramatically. So, when we ask about a system's on-disk meta-data layout, do we want to incorporate its strategies for prefetching? They may be tightly coupled. For example, consider a system that groups the meta-data of "related files" together so that whenever you access one object, the meta-data for the other objects' meta-data is brought into memory. Does this reflect a good on-disk layout policy or good prefetching? Can you even distinguish them? Does it matter? There exist several benchmarks (e.g., Filebench [10], IOzone [2]) that incorporate tests like this; we will refer to these benchmarks as *on-disk benchmarks*. Depending on how it is configured, the Bonnie and Bonnie++ benchmarking suites [1, 4] can measure either I/O or on-disk performance.

Perhaps we are concerned about the performance of meta-data operations. The Postmark benchmark [7] is designed to incorporate meta-data operations, but does not actually provide meta-data performance in isolation; similarly, many Filebench workloads can exercise meta-data operations but not in isolation.

As mentioned above, on-disk meta-data benchmarks can become caching or in-memory benchmarks when file systems group meta-data together; they can also become in-memory benchmarks when they sweep small file sizes or report "warm-cache" results. We claim that we are rarely interested in pure in-memory execution, which is predominantly a function of the memory system, but rather in the efficacy of a given caching *approach*; does the file system pre-fetch entire files, blocks, or large extents? How are elements evicted from the cache? To the best of our knowledge, none of the existing benchmarks consider these questions.

Finally, we may be interested in studying a file system's ability to scale with increasing load. This was the original intent behind the Andrew File System benchmark [5], and while sometimes used to that end, this benchmark, and its successor, the Linux kernel compile are more frequently cited as a good benchmark for general file system performance.

We surveyed the past two years' publications in file systems from the USENIX FAST, OSDI, ATC, HotStorage, ACM SOSP, and IEEE MSST conferences. We recorded what benchmarks were used and what each benchmark measures. We reviewed 100 papers, 68 from 2010 and 32 from 2009, eliminating 13 papers, because they had no evaluation component relative to this discussion. For the rest, we counted how many papers used each benchmark. Table 1 shows all the benchmarks that we encountered and reports how many times each was used in each of the past two years. The table also contains similar statistics from our previous study for 1999–2007 years. We were disappointed to see how little consistency there was between papers. Ad-hoc testing—making one's own benchmark—was, by far, the most common choice. While several papers used microbenchmarks for random read/write, sequential read/write and create/delete operations, they were all custom generated. We found this surprising in light of the numerous existing tests that can generate micro-benchmark workloads.

Some of the ad-hoc benchmarks are the result of new functionality: three papers provided ad-hoc deduplication benchmarks, because no standard benchmarks exist. There were two papers on systems designed for streaming, and both of those used custom workloads. However, in other cases, it is completely unclear why researchers are developing custom benchmarks for OLTP or parallel benchmarking. Some communities are particularly enamored with trace-based evaluations (e.g., MSST). However, almost none of those traces are widely available: of the 14 "standard" traces, only 2 (the Harvard traces and the NetApp CIFS traces) are widely available. When researchers go to the effort to make traces, it

Benchmark	Benchmark Type					Used in papers	
	I/O	On-disk	Caching	Meta-data	Scaling	1999-2007	2009-2010
IOmeter	●					2	3
Filebench	●	○	○	○	●	3	5
IOzone		○	○		●	0	4
Bonnie/Bonnie64/Bonnie++		○	○			2	0
Postmark		○	○	○	●	30	17
Linux compile		○	○	○		6	3
Compile (Apache, openssh, etc.)		○	○	○		38	14
DBench		○	○	○		1	1
SPECsfs		○	○	○	●	7	1
Sort		○	○		●	0	5
IOR: I/O Performance Benchmark		○	○		●	0	1
Production workloads	⋆	⋆	⋆	⋆		2	2
Ad-hoc	⋆	⋆	⋆	⋆	⋆	237	67
Trace-based custom	⋆	⋆	⋆	⋆		7	18
Trace-based standard	⋆	⋆	⋆	⋆		14	17
BLAST		○	○			0	2
Flexible FS Benchmark (FFSB)		○	○	○	●	0	1
Flexible I/O tester (fio)	○	○	○		●	0	1
Andrew		○	○	○		15	1

Table 1: Benchmarks Summary. "●" indicates the benchmark can be used for evaluating the corresponding file system dimension; "○" is the same but the benchmark does not isolate a corresponding dimension; "⋆" is used for traces and production workloads

would benefit the community to make them widely available by depositing them with SNIA.

In summary, there is little standardization in benchmark usage. This makes it difficult for future researchers to know what tests to run or to make comparisons between different papers. There must be a better approach.

3 A Case Study

We performed a simple evaluation of Ext2 using Filebench 1.4.8 [10]. We picked Filebench because it seems to be gaining popularity: it was used in 3 papers in FAST 2010 and 4 in OSDI 2010. (Nevertheless, the problems outlined by this paper are common to all other benchmarks we surveyed.) The range of the workloads that Filebench can generate is broad, but we deliberately chose a simple, well-defined workload: one thread randomly reading from a single file. It is remarkable that even such a simple workload can demonstrate the multi-dimensional nature of file system performance. More complex workloads and file systems will exploit even more dimensions and consequently will require more attention during evaluation. Ext2 is a relatively simple file system, compared to, say, Btrfs; more complex file systems should demonstrate more intricate performance curves along performance dimensions.

Our experiments measure the throughput and latency of the random read operation. We used an Intel Xeon 2.8GHz machine with one SATA Maxtor 7L250S0 disk drive. We artificially decreased the RAM to 512MB to facilitate our experiments. Section 3.1 describes our observations related to the throughput, and Section 3.2 highlights the latency results.

3.1 Throughput

In our first experiment we increased the file size from 64MB to 1024MB in steps of 64MB. For each file size we ran the benchmark 10 times. The duration of the run was 20 minutes, but to ensure steady-state results we report only the last minute. Figure 1 shows the throughput and its relative standard deviation for this experiment. The sudden drop in performance between 384MB and 448MB is readily apparent. The OS consumes some of the 512 MB of RAM and the drop in performance corresponds to the point when the file size exceeds the amount of memory available for the page cache.

So, what should a careful researcher report for the random read performance of Ext2? For file sizes less than 384MB, we mostly exercise the memory subsystem; for file sizes greater than 448MB, we exercise the disk system. This suggests that researchers should either publish results that span a wide range or make explicit both the memory- and I/O-bound performance.

It was surprising, at first, that such a sudden performance drop happens within a narrow range of only 64MB. We zoomed into the region between 384MB and 448MB and observed that performance drops within an even narrower region—less than 6MB in size. This happens because even a single rare read operation that induces I/O lasts longer than thousands of in-memory operations—a worsening problem in recent years as the gap between I/O and memory/CPU speeds widens. More modern file systems rely on multiple cache levels (using Flash memory or network). In this case the performance curve will have multiple distinctive steps.

Figure 1 also shows the relative standard deviation

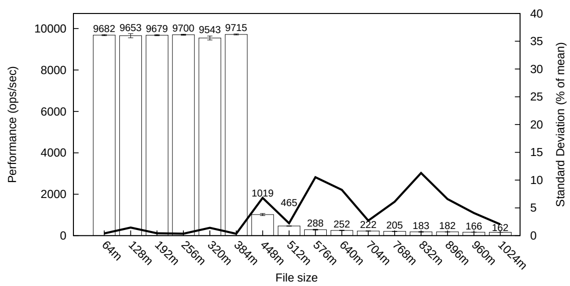


Figure 1: Ext2 throughput and its relative standard deviation under random read workload for various file sizes

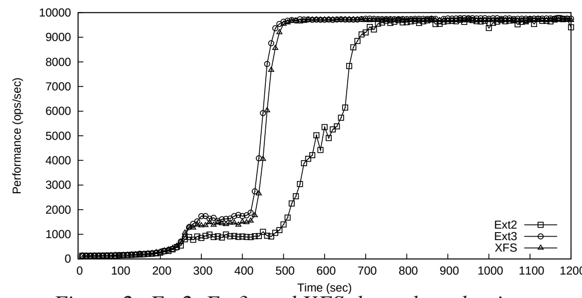


Figure 2: Ext2, Ext3, and XFS throughput by time

for the throughput. The standard deviation is not constant across the file sizes. In the I/O-bound range, the standard deviation is up to 5 times greater than it is in the memory-bound range. This is unsurprising given the variability of disk access times compared to the relative stability of memory performance. We observed that in the transition region, where we move from being memory-bound to being disk-bound, the relative standard deviation skyrockets by up to 35% (not visible on the figure because it only depicts data points with a 64MB step). Just a few megabytes more (or less) available in the cache affect the throughput dramatically in this boundary region. It is difficult to control the availability of just a few megabytes from one benchmark run to another. As a result, benchmarks are very fragile: just a tiny variation in the amount of available cache space can produce a large variation in performance.

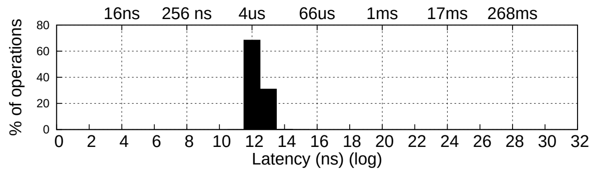
We reported only the steady-state performance in the above discussion; is it correct to do so? We think not. In the next experiment we recorded the throughput of Ext2, Ext3, and XFS every 10 seconds. We used a 410MB file, because it is the largest file that fits in the page cache. Figure 2 depicts the results of this experiment. In the beginning of the experiment no file blocks are cached in memory. As a result all read operations go to the disk, directly limiting the throughput of all the systems to that of the disk. At the end of the experiment, the file is completely in the page cache and all the systems run at memory speed. However, the performance of these file systems differs significantly between 4 and 13 minutes. What should the careful researcher do? It is clear that the interesting region is in the transition from disk-bound to memory-bound. Reporting results at either extreme will lead to the conclusion that the systems behave identically. Depending on where in the transition range a researcher records performance, the results can show differences ranging anywhere from a few percentage points to nearly an order of magnitude! Only the *entire* graph provides a fair and accurate characterization of the file system performance across this (time) dimension. Such graphs span both memory-bound to I/O bound dimensions, as well as a cache warm-up period. Self-scaling benchmarks [3] can collect data for such graphs.

3.2 Latency

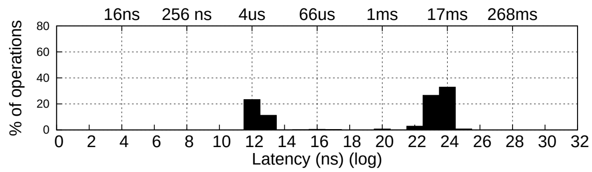
File system benchmarks, including Filebench, often report an average latency for I/O operations. However, *average* latency is not a good metric to evaluate user satisfaction when a latency-sensitive application is in question. We modified Filebench to collect latency histograms [6] for the operations it performs. We ran the same workload as described in the previous section for four different file sizes spanning a wide range: 64MB, 1024MB, and 25GB. Figure 3 presents the corresponding histograms. Notice that the X axes are logarithmic and that the units are in nanoseconds (above) and log_2 bucket number (below). The Y axis units are the percentage of the total number of operations performed.

For a 64MB file (Figure 3(a)) we see a distinctive peak around 4 *micro*seconds. The file fits completely in memory, so only in-memory operations contribute to the latency. When the file size is 1024MB we observe two peaks on the histogram (Figure 3(b)). The second peak on the histogram corresponds to the read calls that miss in the cache and go to disk. The peaks are almost equal in height because 1024MB is twice the size of RAM and, consequently, half of the random reads hit in the cache (left peak), while the other half go to disk (right peak). Finally, for a file that is significantly larger than RAM—25G in our experiments—the left peak becomes invisibly small because the vast majority of the reads end up as I/O requests to the disk ((Figure 3(c)). Clearly, the working set size impacts reported latency significantly, spanning over 3 orders of magnitude.

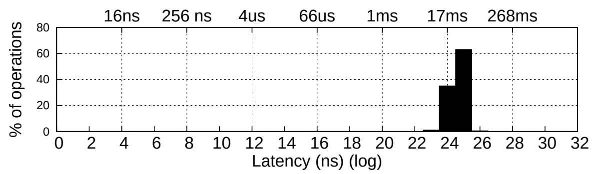
In another experiment, we collected latency histograms periodically over the course of the benchmark. In this case we used a 256MB file that was located on Ext2. Figure 4 contains a 3-D representation of the results. As the benchmark progresses, the peak corresponding to disk reads (located near the 2^{23} ns) fades away and is replaced by the peak corresponding to reads from the page cache (around 2^{11}ns). Again, depending on exactly when measurements are taken, even a careful researcher might draw any of a number of conclusions about Ext2's performance—anywhere from concluding that Ext2 is very good, to Ext2 being very bad, and everywhere in between. Worse, during most of the bench-



(a) 64MB file



(b) 1024MB file



(c) 25GB file

Figure 3: Ext2 read latency histograms for various file sizes

mark's run, it is bi-modal: trying to achieve stable results with small standard deviations is nearly impossible.

Single number benchmarks rarely tell the whole story. We need to get away from the marketing-driven single-number mindset to a multi-dimensional continuum mindset.

4 Conclusions and Future Work

A file system is a complex piece of software with layers below and above it, all affecting its performance. Benchmarking such systems is far more complex than any single tool, technique, or number can represent. Yes, it makes our lives more difficult, but will greatly enhance the utility of our work. Let's begin by defining precisely what dimension(s) of file system behavior we are evaluating. We believe that a file system benchmark should be a suite of nano-benchmarks where each individual test measures a particular aspect of file system performance and measures it well. Next, let's get away from single-number reporting. File system performance is extremely sensitive to minute changes in the environment. In the interest of full disclosure, let's report a range of values that span multiple dimensions (e.g., timeline, working-set size, etc.). We propose that at a minimum, an encompassing benchmark should include in-memory, disk layout, cache warm-up/eviction, and meta-data operations performance evaluation components.

Our community needs to buy in to doing a better job. We need to reach agreement on what dimensions to measure, how to measure them, and how to report the results of those measurements. Until we do so, our papers are destined to provide incomparable point answers to subtle and complex questions.

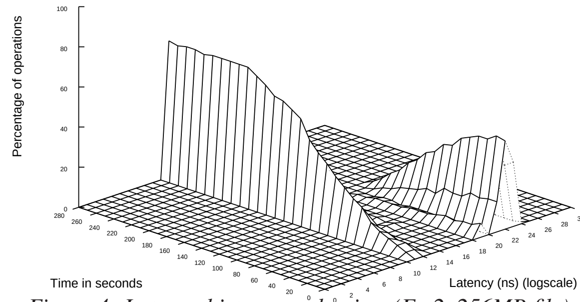


Figure 4: Latency histograms by time (Ext2, 256MB file)

References

[1] T. Bray. The Bonnie home page. `www.textuality.com/bonnie`, 1996.

[2] D. Capps. IOzone Filesystem Benchmark. `www.iozone.org/`, July 2008.

[3] P. M. Chen and D. A. Patterson. A new approach to I/O performance evaluation - self-scaling I/O benchmarks, predicted I/O performance. In *Proceedings of the 1993 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 1–12, Seattle, WA, May 1993. ACM SIGOPS.

[4] R. Coker. The Bonnie++ home page. `www.coker.com.au/bonnie++`, 2001.

[5] J. H. Howard. An Overview of the Andrew File System. In *Proceedings of the Winter USENIX Technical Conference*, February 1988.

[6] N. Joukov, A. Traeger, R. Iyer, C. P. Wright, and E. Zadok. Operating System Profiling via Latency Analysis. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI 2006)*, pages 89–102, Seattle, WA, November 2006. ACM SIGOPS.

[7] J. Katcher. PostMark: A New Filesystem Benchmark. Technical Report TR3022, Network Appliance, 1997. `www.netapp.com/tech_library/3022.html`.

[8] Omar Al-Ubaydli Neal S. Young, John P. A. Ionnadis. Why current publication practices may distort science. *PLoS Med*, 5, October 2008. `http://www.plosmedicine.org/article/info:doi/10.1371/journal.pmed.0050201`.

[9] OSDL. Iometer project. `www.iometer.org/`, August 2004.

[10] FileBench, July 2008. `www.solarisinternals.com/wiki/index.php/FileBench`.

[11] D. Tang and M. Seltzer. Lies, damned lies, and file system benchmarks. Technical Report TR-34-94, Harvard University, December 1994. In VINO: The 1994 Fall Harvest.

[12] A. Traeger, N. Joukov, C. P. Wright, and E. Zadok. A Nine Year Study of File System and Storage Benchmarking. *ACM Transactions on Storage (TOS)*, 4(2):25–80, May 2008.

[13] E. Zadok, S. Shepler, and R. McDougall. File System Benchmarking. `www.usenix.org/events/fast05/bofs.html#bench`, December 2005.