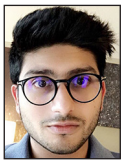
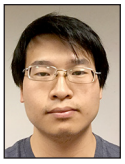
How to Fragment Your File System

ALEX CONWAY, AINESH BAKSHI, YIZHENG JIAO, YANG ZHAN, MICHAEL A. BENDER, WILLIAM JANNEN, ROB JOHNSON, BRADLEY C. KUSZMAUL, DONALD E. PORTER, JUN YUAN, AND MARTIN FARACH-COLTON

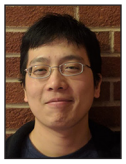
Alex Conway is a PhD student at Rutgers University, New Brunswick, New Jersey. His research interests include the theory and application of external memory algorithms and data structures, caching algorithms, and file systems. alexander.conway@rutgers.edu



Ainesh Bakshi is looking forward to starting a PhD in computer science at Carnegie Mellon University in 2017. He graduated from Rutgers University, New Brunswick, New Jersey. His research interests broadly include algorithms and theoretical machine learning, with a focus on algorithms that look to bridge the gap between theory and practice. aineshbakshi@gmail.com



Yizheng Jiao is a PhD student at Stony Brook University. His research interests focus on storage system design and implementation. Currently, he is working on a write-optimized file system for high-speed storage devices as well as efficient memory system design of big data applications. He also is interested in hacking the Linux kernel storage stack and database index engine. yizheng@cs.unc.edu



Yang Zhan is a PhD student at the University of North Carolina at Chapel Hill and is advised by Donald Porter. His research mainly focuses on write-optimized data structures. He also works on concurrent data structures. yzhan@cs.unc.edu

File systems attempt to avoid aging, or fragmentation over time, by strategically allocating space for files. System implementers and users alike treat aging as a solved problem. Here, we present a realistic workload, based on Git, that can cause these best-guess file-block-placement heuristics to fail, inducing large performance declines due to aging. This performance decline cannot be prevented with more caching or larger disks, and SSDs reduce but do not eliminate the aging effects. Our Git-based aging scheme can simulate a year of aging in under an hour. To make it easy for practitioners to incorporate aging into benchmarks, we have open-sourced our aging scripts at betrfs.org.

File-system *fragmentation* occurs when a file system stores a file or directory's contents in discontiguous ranges of disk blocks. As a file system becomes more fragmented, performance can drop significantly, since reading the file requires issuing multiple I/Os to disk. The performance drop can be particularly severe on rotating disks, where each I/O may require a disk seek. Maintaining locality in a file system as files grow, shrink, and are renamed can be challenging.

For many years, file systems did not include effective measures for avoiding fragmentation. The seminal work of Smith and Seltzer [7] showed that FFS file systems age under realistic workloads, and this aging affects performance.

Users mitigated fragmentation in early file systems by running special tools to defragment their file systems. Defragmenters reorganize file contents so that each file is stored in a contiguous range of disk blocks.

Modern file systems, on the other hand, strive to avoid fragmentation by applying best effort heuristics at allocation time. For example, file systems try to place related files close together on disk, while also leaving empty space for future files [1, 4, 5, 8]. These and other heuristics attempt to stay ahead of fragmentation wrought by normal file-system usage.

Fragmentation is thus widely viewed as a solved problem. For example, the Linux System Administrator's Guide [9] says:

> Modern Linux file systems keep fragmentation at a minimum by keeping all blocks in a file close together, even if they can't be stored in consecutive sectors. Some file systems, like ext3, effectively allocate the free block that is nearest to other blocks in a file. Therefore it is not necessary to worry about fragmentation in a Linux system.

As a result, few users run defragmentation tools. Furthermore, few file-system benchmarks attempt to age the file system before measuring its performance.

In this article, we demonstrate that modern file systems can still suffer from fragmentation under representative workloads, and we describe a simple method for quickly inducing aging. Our results suggest that fragmentation can be a first-order performance concern—some file systems slow down by over 20x over the course of our experiments. We show that fragmentation causes performance declines on both hard drives and SSDs, when there is plentiful cache available, and even on large disks with ample free space.

Michael A. Bender is a Professor of Computer Science at Stony Brook University. He was Founder and Chief Scientist at Tokutek, Inc., an enterprise database company, which was acquired by Percona in 2015. Bender's research interests span the areas of data structures and algorithms, I/O-efficient computing, scheduling, and parallel computing. Bender received his BA in applied mathematics from Harvard University in 1992 and obtained a DEA in computer science from the Ecole Normale Superieure de Lyon, France, in 1993. He completed a PhD on scheduling algorithms from Harvard University in 1998. bender@cs.stonybrook.edu

William Jannen teaches at Williams College, where he attempts to design systems that accommodate the physical characteristics of their underlying media. He is also an artist and a player of games. wjannen@cs.stonybrook.edu

Rob Johnson is a Senior Researcher at VMware and Research Assistant Professor at Stony Brook University. He developed BetrFS, invented the quotient filter, founded cache-adaptive analysis, broke the High-bandwidth Digital Content Protection (HDCP) crypto-system, and co-authored CQual, a static analysis tool that has found dozens of bugs in the Linux kernel. rob@cs.stonybrook.edu

Until July 2016, Bradley C. Kuszmaul was a Research Scientist in the Computer Science and Artificial Intelligence Laboratory at the Massachusetts Institute of Technology (MIT CSAIL), where his research focused on performance engineering of multicore software as well as data structures and algorithms that optimize cache and disk I/O. He has now joined the Bare Metal Cloud group at Oracle. bradley@mit.edu

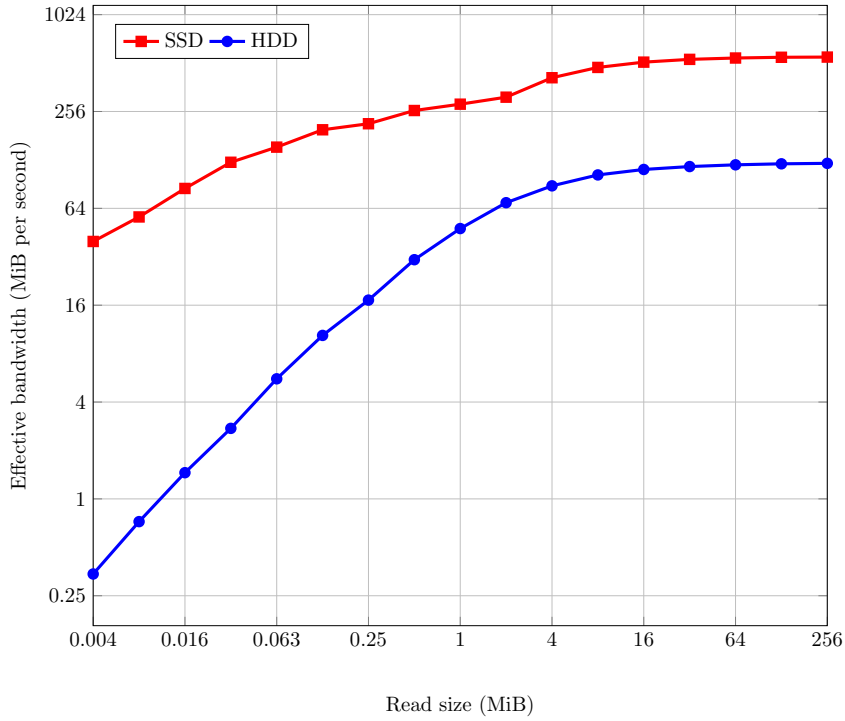


Figure 1: Effective bandwidth vs. read size (higher is better). Even on SSDs, large I/Os can yield an order of magnitude more bandwidth than small I/Os.

Fragmentation remains important because there is a large gap between sequential and random I/O performance of storage devices [2]. On rotating disks, even a few seeks can have an outsized effect on performance. For example, if a file system places a 100 MiB file in 200 disjoint pieces (i.e., 200 seeks) on a disk with 100 MiB/s bandwidth and 5 ms seek time, reading the data will take twice as long as reading it in an ideal layout.

Even on SSDs, which do not perform mechanical seeks, a decline in locality can harm performance [6]. Figure 1 shows that both HDDs and SSDs achieve substantially higher throughput when reading large blocks. On both types of hardware, we found that a surprisingly large read block of 4 MiB is necessary to achieve 75% of device bandwidth (see [2] for the specifics of our experimental setup).

Our technique for causing fragmentation makes it easy for file-system implementers and benchmarkers to incorporate aging into their evaluations. Our technique can cause years' worth of file-system aging in just a few hours and can take regular measurements as the file system ages. File systems begin aging almost immediately in our experiments, meaning that implementers and benchmarkers can use our tools to induce significant aging in under an hour.

The gold standard for realistically aging a file system is to replay a trace of file-system operations from a real system. Unfortunately, such traces are almost impossible to find. Smith and Seltzer proposed to approximate such traces by interpolating changes between successive file-system snapshots collected during a multi-year experiment [7]. Unfortunately, years-long collections of file-system snapshots have also been hard to come by.

Donald E. Porter is an Assistant Professor of Computer Science at the University of North Carolina at Chapel Hill and, by courtesy, at Stony Brook University. His research aims to improve computer system efficiency and security. In addition to recent work on write-optimization in file systems, recent projects have developed lightweight guest operating systems for virtual environments, system security abstractions, and efficient data structures for caching. porter@cs.unc.edu

Jun Yuan is an Assistant Professor of Computer Systems at Farmingdale State College of SUNY, New York. Her research interests primarily lie in systems and data structures. In addition to write-optimized file systems, she has researched programming-language-based security and access control on the Android OS. yuanj@farmingdale.edu

Martin Farach-Colton is a Professor of Computer Science at Rutgers University, New Brunswick, New Jersey. His research focuses on both the theory and practice of external memory and storage systems. He was a pioneer in the theory of cache-oblivious analysis. His current research focuses on the use of write optimization to improve performance in both read- and write-intensive big data systems. He has also worked on the algorithmics of strings and metric spaces, with applications to bioinformatics. In addition to his academic work, Professor Farach-Colton has extensive industrial experience. He was CTO and co-founder of Tokutek, a database company that was founded to commercialize his research. During 2000–2002, he was a Senior Research Scientist at Google. farach@cs.rutgers.edu

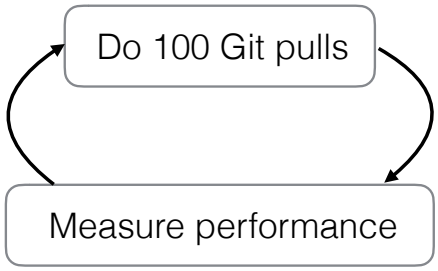


Figure 2: The Git workload

The key idea behind our aging technique is that we can view open-source Git (or any other version control system) repositories as collections of snapshots of the developers' file systems. Furthermore, replaying a repository's revision history will replay a significant portion of the developers' actual file system activity, since many developers pull changes from their collaborators multiple times per day. Thus replaying the revision history should induce fragmentation similar to that experienced by the developers when they were working on the project.

The large number of open-source projects—many of them with over a decade of history—means that we can now easily induce representative aging in file systems. Our scripts, available at betrfs.org, make it straightforward for developers and benchmarkers to integrate aging into their performance measurements.

How to Age Your File System

In the experiments in this article, we replay commits to the Linux kernel Git repository hosted on github.com. We start from the first commit and proceed in chronological order. After every 100 Git pulls, we unmount and remount the file system, clear all caches, and measure read performance (Figure 2).

We measure performance by the wall-clock time required to perform a recursive `grep` starting from the root directory of the file system. This operation descends through the directory structure, reading the content of each file. This `grep` reads a sequence of file and metadata blocks, which we call the logical order of the file-system blocks. Fragmentation occurs when two logically successive blocks are not stored in adjacent logical block addresses on the storage device. Greater fragmentation means that the average I/O size is smaller. As shown in Figure 1, this reduces the effective bandwidth, causing the `grep` to take longer.

We divide fragmentation into three categories:

- *Intrafile* is fragmentation involving blocks from the same file.
- *Interfile* is fragmentation involving blocks from two different files.
- *Metadata* is fragmentation involving at least one metadata block.

A recursive `grep` measures the impact of all these types of fragmentation on overall file-system performance.

When we run our Git aging workload, various statistics of the file system will naturally change over time as files and directories are created, modified, and deleted. For example, as a project progresses, it might include more small files, or subdirectories may include more source files. In order to make direct comparisons, we need to normalize for such changes. First, we normalize for file-system size by reporting the `grep` time in seconds per GiB. We obtain the file-system size from the output of `du`.

In order to measure potential aging, after each measurement, we copy the entire file system to a freshly formatted file system on another partition and repeat the performance

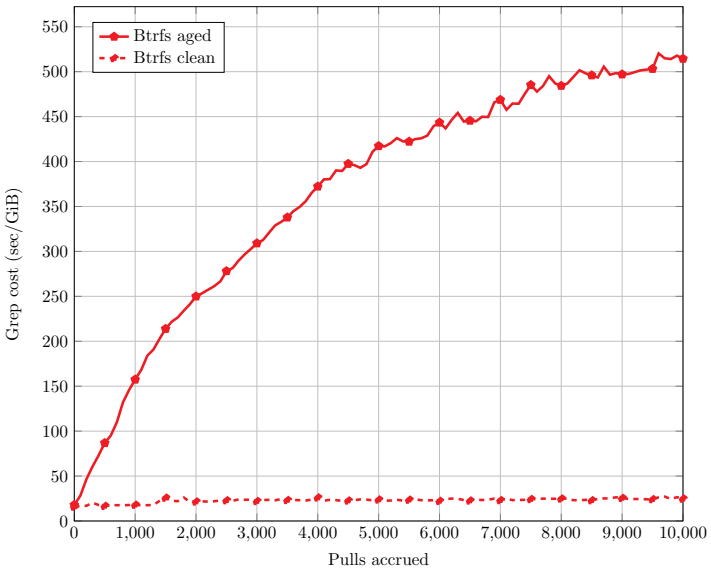


Figure 3: Git aging workload on btrfs on HDD. The overall slowdown is 20.6x. Lower is better.

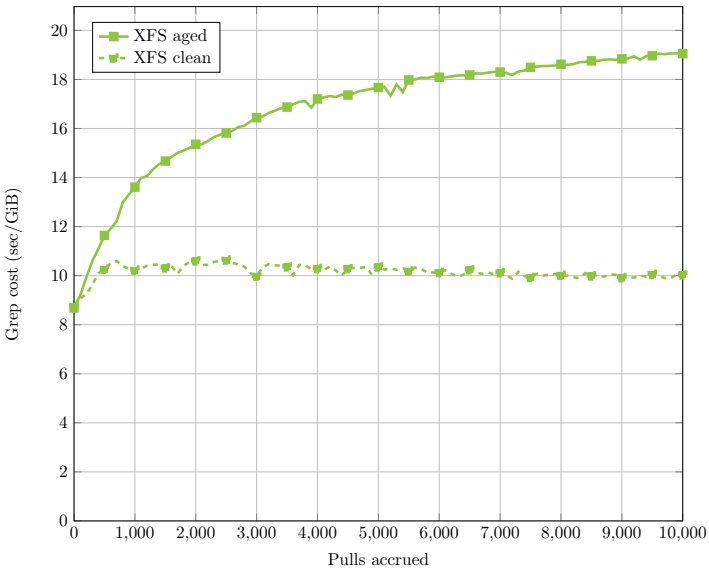


Figure 4: Git aging workload on XFS on SSD. The overall slowdown is 1.9x. Lower is better.

measurement there. We call this copy of the file system the *clean* instance, since the file system does not undergo any changes after the files are copied to it. The logical states of both file systems are the same; any performance difference between the aged and clean instances of a file system are due to the history of preceding operations.

Do modern file systems age? Figure 3 shows the results of aging btrfs with Git on a hard drive. The `grep` performance drops by a factor of 20 after 10,000 pulls. This drop in performance happens quickly; it only takes 100 pulls for a 2x slowdown and 1100 pulls for a 10x slowdown. Moreover, the `grep` ends up being very slow in absolute terms; by the end of the test it takes more than eight minutes to `grep` through 1 GiB.

In this article, we present only one file system in each experiment. Our USENIX FAST paper evaluates five popular Linux file systems under all of these experimental conditions and finds similar results [2].

Do SSDs fix aging? When we run the same workload on an SSD, we would expect to see less aging as a result of the superior random-read performance. Figure 4 shows the results of aging XFS with Git on an SSD. Although the slowdown due to aging is smaller, it is still significant. After 10,000 pulls, `greps` in the aged file-system instance are 1.9x slower than in the clean instance. After 800 pulls, the slowdown is 25%, and after 2,500 pulls, the slowdown is 50%.

Does caching fix aging? If most or all of our file system fits in cache, then the on-disk layout will not affect `grep` performance, since reads will be served from cache. We evaluated the sensitivity of the Git workloads to varying amounts of system RAM and, therefore, varying amounts of available disk cache. We use the same Git aging procedure, except that we do not flush any caches or remount the hard drive between iterations. The size of the data on disk is initially about 280 MiB and grows throughout the test to approximately 1.2 GiB.

The results for ext4 on a hard drive are summarized in Figure 5. When there is sufficient memory to keep all the data in cache, the `grep` is very fast. As soon as the size of the file system grows above a threshold, however, the warm-cache performance of `grep` quickly approaches the cold-cache performance. Furthermore, once the file system is no longer cached, the warm-cache performance is in all cases worse than the cold-cache performance of a clean copy of the file system. Unless all data fits into cache, therefore, fragmentation is a major driver of file-system performance.

Do big disks fix aging? The results shown in Figures 3 and 4 were performed on a 20 GiB partition in which the file system size never exceeded 1.2 GiB; therefore, the partition is never more than 6% full. If we run the Git workload on partitions of different sizes, as shown in Figure 6, we see that having a larger partition does not eliminate (or even mitigate) aging.

In fact, as the partition gets larger, the clean performance of ext4 gets worse. This is because ext4 spreads data across the partition in order to leave room for future files. Thus, the larger partition size actually results in longer seeks.

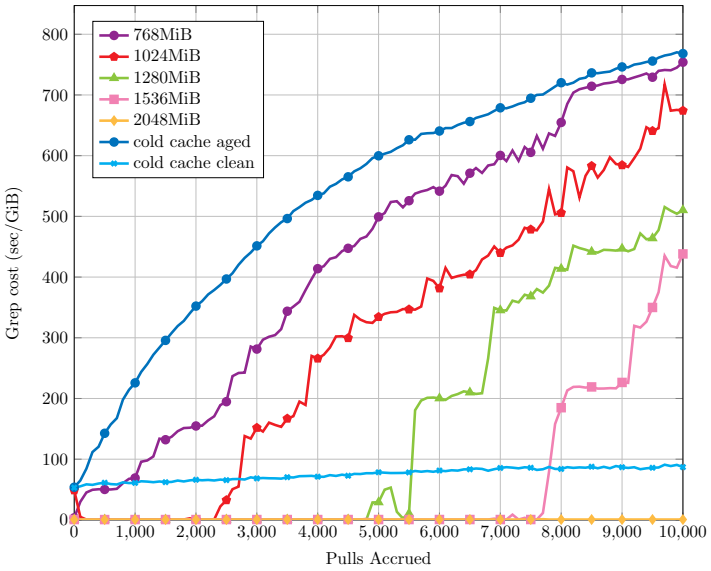


Figure 5: grep costs as a function of Git pulls with warm cache and varying system RAM on ext4 (top). Lower is better.

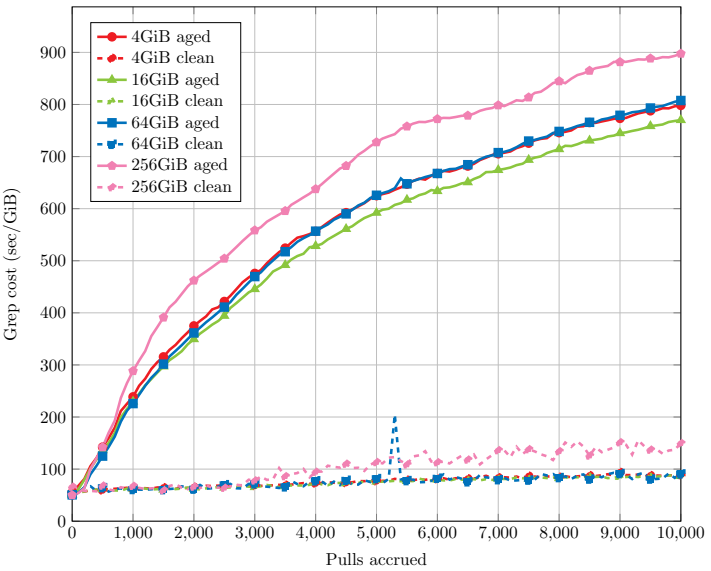


Figure 6: grep costs as a function of Git pulls with varying partition size on ext4. Lower is better.

Conclusion

The experiments above show that modern file systems can still age substantially under workloads representative of a typical software developer's file-system usage. They also show that SSDs, caching, and large disks do not prevent aging in today's file systems, though SSDs can help.

Furthermore, these results demonstrate that many modern file system design features, such as delayed allocation, cylinder or block groups, and extents, do not prevent aging. The file systems in these benchmarks included some or all of these features, but they aged nonetheless.

Our USENIX FAST paper delves into other file-system design tradeoffs related to aging and confirms that our research prototype file system, BetrFS [3, 10], exhibits almost no aging [2].

Our Git-based method for inducing aging makes it easy to incorporate aging into file-system benchmarks. Our scripts are available at betrfs.org.

Acknowledgments

Part of this work was done while Jiao, Porter, Yuan, and Zhan were at Stony Brook University. This research was supported in part by NSF grants CNS-1409238, CNS-1408782, CNS-1408695, CNS-1405641, CNS-1161541, IIS-1247750, CCF-1314547, a NetApp Faculty Fellowship, and VMware.

References

[1] R. Card, T. Ts'o, and S. Tweedie, "Design and Implementation of the Second Extended Filesystem," in *Proceedings of the First Dutch International Symposium on Linux*, pp. 1–6: http://e2fsprogs.sourceforge.net/ext2intro.html.

[2] A. Conway, A. Bakshi, Y. Jiao, Y. Zhan, R. Johnson, B. C. Kuszmaul, and M. Farach-Colton, "File Systems Fated for Senescence? Nonsense, Says Science!" in *Proceedings of the 15th USENIX Conference on Fi-le and Storage Technologies (FAST '17),* pp. 45–58: https://www.usenix.org/system/files/conference/fast17/fast17-conway.pdf.

[3] W. Jannen, J. Yuan, Y. Zhan, A. Akshintala, J. Esmet, Y. Jiao, A. Mittal, P. Pandey, P. Reddy, L. Walsh, M. Bender, M. Farach-Colton, R. Johnson, B. C. Kuszmaul, and D. E. Porter, "BetrFS: A Right-Optimized Write-Optimized File System," in *Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST '15),* pp. 301–315: https://www.usenix.org/system/files/conference/fast15/fast15-paper-jannen_william.pdf.

[4] A. Mathur, M. Cao, S. Bhattacharya, A. Dilger, A. Tomas, and L. Vivier, "The New ext4 Filesystem: Current Status and Future Plans," in *Proceedings of the Ottawa Linux Symposium (OLS),* vol. 2 (2007), pp. 21–34: https://www.kernel.org/doc/ols/2007/ols2007v2-pages-21-34.pdf.

[5] M. K. McKusick, W. N. Joy, S. J. Leffler, and R. S. Fabry, "A Fast File System for UNIX," *ACM Transactions on Computer Systems (TOCS),* vol. 2, no. 3 (August 1984), pp. 181–197: https://cs162.eecs.berkeley.edu/static/readings/FFS84.pdf.

[6] C. Min, K. Kim, H. Cho, S. Lee, and Y. I. Eom, "SFS: Random Write Considered Harmful in Solid State Drives," in *Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST '12),* pp. 139–154: https://www.usenix.org/system/files/conference/fast12/min.pdf.

[7] K. A. Smith and M. Seltzer, "File System Aging—Increasing the Relevance of File System Benchmarks," in *Proceedings of the 1997 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS),* pp. 203–213: https://www.eecs.harvard.edu/margo/papers/sigmetrics97-fs/paper.pdf.

[8] S. Tweedie, "EXT3, Journaling Filesystem," *Proceedings of the Ottawa Linux Symposium (OLS),* (July 20, 2000), pp. 24–29: http://olstrans.sourceforge.net/release/OLS2000-ext3/OLS2000-ext3.pdf.

[9] L. Wirzenius, J. Oja, S. Stafford, and A. Weeks, *Linux System Administrator's Guide*, The Linux Documentation Project, Version 0.9, 2004: http://www.tldp.org/LDP/sag/sag.pdf.

[10] J. Yuan, Y. Zhan, W. Jannen, P. Pandey, A. Akshintala, K. Chandnani, P. Deo, Z. Kasheff, L. Walsh, M. Bender, M. Farach-Colton, R. Johnson, B. C. Kuszmaul, and D. E. Porter, "Optimizing Every Operation in a Write-Optimized File System," in *Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST '16)*, pp. 1–14: https://www.usenix.org/system/files/conference/fast16/fast16-papers-yuan.pdf.