Aging Gracefully with *Geriatrix*: A File System Aging Tool

Saurabh Kadekodi, Vaishnavh Nagarajan, Garth A. Gibson

CMU-PDL-17-106

October 2017

Parallel Data Laboratory
Carnegie Mellon University
Pittsburgh, PA 15213-3890

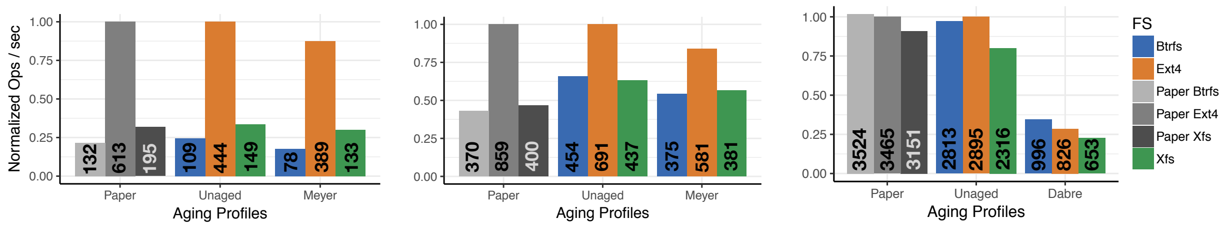
Abstract

File system aging has been advocated for thorough analysis of any design, notably in 1997 [40], but it is cumbersome and often bypassed. Our aging study re-evaluates published file systems after aging using the original paper's benchmarks. We see significant performance degradation on magnetic (HDD) and flash (SSD) disks. With performance of aged file systems on SSDs reduced by as much as 80% relative to the recreated results of prior papers, aging is perhaps more necessary now. More concerning, the rank ordering of aged and tested file systems can change versus published results.

We offer Geriatrix, a simple-to-use aging suite with built-in aging profiles with the goal of making it easier to age, and thereby harder to justify ignoring file system aging in storage research.

Acknowledgements: This research was supported in part by the DOE and Los Alamos National Laboratory, under contract number DEAC52-06NA25396 subcontract 153593-1 (IRHPIT2), the National Science Foundation under awards CNS-1042537 and CNS-1042543 (PRObE, www.nmc-probe.org), and Seagate through the CMU Data Storage Systems Center. We also thank the member companies of the PDL Consortium - Actifio, American Power Conversion, EMC Corporation, Facebook, Google, Hewlett-Packard Labs, Hitachi, Huawei Technologies Co., Intel Corporation, Microsoft Research, NetApp Inc., Oracle Corporation, Samsung Information Systems America, Seagate Technology, Symantec Corporation and Western Digital.

Keywords: file systems, aging, fragmentation, benchmark, tool



(a) Btrfs HDD Varmail (b) Btrfs HDD Webserver (c) Btrfs SSD Fileserver

Figure 1: All three graphs reproduce experiments from the Btrfs ACM TOS publication [36] on aged file system instances. Figures 1a and 1b show aging experiments performed on a HDD using the varmail and webserver profiles respectively. 1a shows modest slowdown (30% in Btrfs, 13% in Ext4 and 9% in Xfs), but does not preserve rank ordering published in the paper. 1b also destroys published rabj order and displays slowdowns of 17% in Btrfs and Ext4 and 12% in Xfs. Figure 1c shows effects of aging on SSD for the fileserver profile. Here rank ordering is preserved, but we observe massive slowdowns of 60% in Btrfs, 75% in Ext4 and 72% in Xfs.

1 Introduction

The performance of a file system usually deteriorates over time. As file systems experience heavy churn, techniques such write-back caching to expedite writes [35, 28, 13], data prefetching to assist reads [7, 32] and self-balancing data structures to contain search times [9] may *pay* for faster normal path performance now with more complex and fragmented on-device images as the system ages. An important factor affecting aged file system performance is poor file system layout [39, 38]. Hence, file system benchmarking practices should consider the effects of aging, an argument raised almost twenty years ago by Smith and Seltzer [40], but one that continues to be ignored.

We set out to understand if advancements in file system design and underlying technologies have eliminated the aging problem by reviewing recent publications and recreating experiments from all we could after applying aging. Table 1 lists 19 papers we reviewed; 12 of 19 (63%) do not mention aging at all. For only 5 of these 19 papers was code readily available, from which BetrFS [17, 48, 11] and Nilfs2 [20] were too immature to sustain aging. As shown in Figure 1, we observe upto 30% performance decline on hard drives (HDD) and a surprising 75% performance decline on solid state drives (SSD) after re-running published benchmarks on aged file system counterparts. This indicates that modern file systems on modern hardware also age poorly, a claim verified by other researchers as well [33].

Aging is a cumbersome but necessary task. We believe that aging is largely avoided because of the impact on results, setup complexity, reproducibility and running time. Any aging tool is expected to exercise the file system heavily, thus making it a long-running activity. In order to address the above concerns, we have built *Geriatrix* - a sophisticated profile driven aging tool with mathematical guarantees that ages a file system according to a reference (old) file system whose characteristics are given as input. We release Geriatrix as open source with seven built-in aging profiles along with a repository of aged images of popular Linux file systems to standardize the practice of age based performance measurements among file system researchers and developers.

2 Related Work

We classify aging tools into three categories - trace replay tools, scripts executing real-world applications and synthetic workload generators.

Trace replay tools are best used with file systems expecting a highly specialized workload. Traces can be captured and replayed at multiple levels - the network level [52], file level [29], file system level [37, 4],

File System	Publication	Aging Avoidable	Aged
yFS [50]	FAST 2003	No	Yes
Nilfs2 [20]	SIGOPS 2006	No	No
TFS [8]	FAST 2007	No	Yes
Data Domain Dedup FS [51]	FAST 2008	No	Yes
Panasas Parallel FS [46]	FAST 2008	No	Yes
CA-NFS [5]	FAST 2009	No	No
HYDRAStor [42]	FAST 2010	No	No
DFS [18]	FAST 2010	No	No
SFS [31]	FAST 2012	No	Yes
BlueSky [43]	FAST 2012	Maybe	No
ZZFS [26]	FAST 2012	Maybe	No
Nested FS in Virt. Env. [21]	FAST 2012	No	No
Btrfs [36]	ACM TOS 2013	No	No
ReconFS [24]	FAST 2014	No	No
F2fs [22]	FAST 2015	No	No
App. Managed Flash [23]	FAST 2016	Maybe	No
NOVA [47]	FAST 2016	No	No
CFFS [49]	FAST 2016	Maybe	Yes
BetrFS [17, 48, 11]	FAST 2015, 2016	No	Yes

Table 1: A subset of major file system publications and whether their paper reports aging experiments. The *Aging Avoidable* column is our understanding of whether aging was avoidable for a given work prior to benchmarking. The *Aged* column refers to whether they performed any aging-like experiment. Two file systems - yFS [50], TFS [8] performed long-running aging experiments; Data Domain FS [51] and Panasas FS [46] had production data, SFS [31] ran a workload twice the size of the disk and CFFS [49] ran a large trace for aging. The remaining 12 papers do not discuss aging or its effects on their file systems.

system call level [45], VFS level [19] and also at the block level [6]. Low level traces are typically file system specific resulting in loss of usefulness for comparing different file systems. Moreover, long traces are not widely available and are hard to capture. Trace replay tools rank high on reproducibility but do not represent all workloads.

The Andrew benchmark [16], Compilebench [25] and the Git-Benchmark [10, 11] are benchmarks which can be loosely classified as aging tools. These tools emulate user behavior by performing typical operations on the file systems like extracting archives, reading files, compiling code, making directories, cloning repositories, etc. Compilebench performs these tasks on Linux kernel sources, while the Git-Benchmark can be run using any git repository. Tools in this category only exercise one workload pattern.

Geriatrix belongs to the category of synthetic workload generators, which also comprises of Smith and Seltzer's aging tool [40] and Impressions [2]. Smith's tool ages by recreating each file in a given reference snapshot and then performing creates and deletes according to the deltas observed in successive reference snapshots. It was one the first tools to point out the degradation of file system performance with age. Impressions on the other hand is a realistic file system image creator that focuses on several file system characteristics including file size and directory depth distributions along with file attributes and contents. These tools take reference from already old file systems in order to perform aging.

3 Why do we need another aging tool?

Aging any software artifact implies understanding how it will stand the test of time. Aging is used to uncover performance deterioration with use, stress the robustness of the software, and identifies fault tolerance and scalability issues, to name a few reasons for using an aging tool. An aging study is a study of as many of these dimensions as possible. We highlight a few aspects a file system aging tool should strive to fulfill.

The best file system aging tools today either replay a trace of file system commands or run scripts of important applications. Smith's aging tool [40] and Impressions [2] come close to what we expect from an aging tool. But, Smith's tool is a twenty year old artifact with dependencies on the Fast File System (FFS) [28]. Impressions matches an impressive number of aged metrics, but is focused on generating realistic file system content, not file system layout; in fact Impressions writes data exactly once so there are no mutations or deletions to fragment the file system layout state.

A Git-benchmark [11, 10] is a recently published aging benchmark that ages the file system by cloning a git repository, repeatedly patching code files (via git pulls) and finally grepping for random strings in the patched repository. We ran the Git-benchmark from [10] on a 20 GB Ext4 partition and observed a $>$7x slowdown when grepping for the same string after 3000 git pulls versus grepping for a string after a single git pull on a fresh Ext4 partition.

In order to understand the dramatic slowdowns this workload experiences, we traced the Ext4 kernel functions to find where in the code most of the time was spent during the arbitrary greps. Function tracing revealed that *ext4_es_lookup_extent* is the function where most of the time is spent during grep, i.e. looking up a file system data structure. Moreover, the capacity utilization at the end of 3000 git pulls was only 4%. On running *e2freefrag*, we observed $>$75% of the free space extents were between 1-2 GB in size suggesting negligible free space fragmentation. Therefore, aging tools like the Git-benchmark are effective at uncovering file system implementation inefficiencies due to churn but do not necessarily produce the required storage fragmentation which is an inevitable consequence of file system aging.

In summary, an effective file system aging tool should:

- run long enough and with enough variation to mutate any size storage,
- touch as many files, directories, directory depths, small files, large files as desired,
- allow reproduction of the same aging workload and / or aged file system image,
- be independent of specific file system implementation,
- offer as realistic as possible an aging workload mimicing at least a few measurements of aged file systems.

4 Aged File System Profiles

Geriatrix allows users to target different profiles for aged file systems. Its profile parameters were inspired by the information easily obtainable from an aged instance of a file system using a metadata tree walk. Geriatrix profiles specify:

- **File System Fullness (bytes, %):** Instance raw size and fraction containing user data.
- **File Size Distribution (bytes, %; bytes, %; ...):** A histogram of file sizes.
- **Directory Depth Distribution (1, %, # subdirs; 2, % #subdirs; ...):** Path depth to individual files and percentage of files at that path depth along with the aggregate number of subdirectories at each depth.

A key feature of Geriatrix is the way it iterates through creation and deletion of files for a long time relative to file system layout tools like Impressions. Geriatrix takes an age distribution in its profile, which can be a histogram of the create timestamps extracted from an existing old file system snapshot. It converts these timestamps into relative values, scaled into unitless relative ages. The files created during a Geriatrix run have a timestamp defined by the operation count issued by Geriatrix. A created file timestamp, taken as a fraction of all files created by Geriatrix is fit to the input age histogram, whose bin has the same fraction of the oldest input histogram bin.

- **Relative Age Distribution (n, %; m %; ...):** A histogram of relative file ages, $n < m < ...$, where younger files, in the first histogram bin makes up the first % of all files in the aged file system image, etc.

As Geriatrix runs, it selects which previously created file to delete so that the resulting age distribution approaches the input distribution. Most of the effort in a Geriatrix run is spent in achieving the relative age distribution because the other distributions are time-independent and hence continuously achieved.

Geriatrix has a repository of seven built-in file system aging profiles from varied sources to assist in holistic aging of file systems. Table 2 provides a description of all the profiles along with the age of oldest file in that profile. We also indicate the amount of wall-clock time it took to age these profiles using in a ramdisk along with the total workload exercised for aging to aid practitioners in understanding the churn produced by each profile when using Geriatrix for aging. Finally, we also show the perfect age distribution convergence achieved on the relative age distribution graphs for each aging profile.

5 Geriatrix Aging Methodology

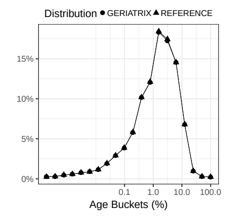
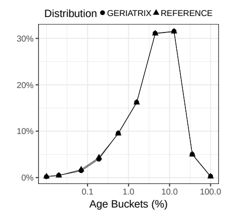
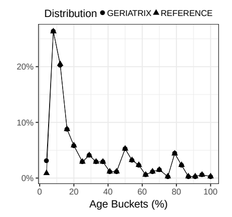
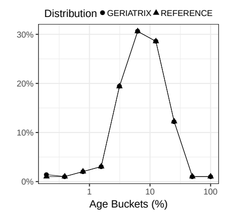
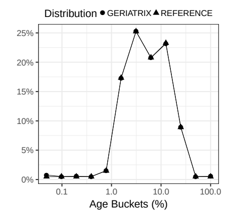
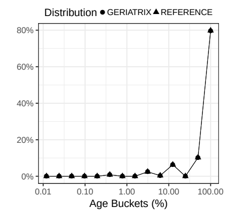
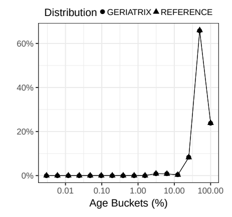
Geriatrix exercises a non-aged file system to achieve the size distribution, directory depth distribution and the relative age distribution, while maintaining the specified file system fullness, by performing a sequence of file create and delete operations. All input distributions are considered independent of each other, so all subsets of files in an aged file system follow all input distributions. Thus, by greedily choosing the size and directory depth of any file being created or deleted (such as to make the largest improvement of the distributions towards their respective target distributions) we trivially end up satisfying both size and directory distributions.

Achieving the relative age distribution is harder. Since files being created, are by definition the youngest files, they always belong to the youngest age bucket. As files become older, they cross bucket boundaries. Using the previously described greedy approach for file deletion, we might choose to delete a file which would have subsequently crossed into older age buckets. Thus, an unintended underflow of files may happen in a completely different bucket than the one we initially chose for deletion, dismissing a trivial proof of convergence of the relative age distribution.

Section 6 provides a formal proof of convergence of the time-dependent relative age distribution. Given an input age distribution, this proof also elucidates a lower bound on the number of operations required for achieving convergence, essentially the runtime estimate of a Geriatrix run. The runtime of a Geriatrix run is dependent on the fullness requirement and the buckets of the input age distribution. Uniform age buckets, with uniform distribution achieve convergence in less runtime. And Geriatrix will run as long as it takes for more non-uniform age distributions.
Geriatrix aging proceeds in two distinct phases.

1. **Rapid Aging:** At the beginning of a Geriatrix run, the aging tool performs only creations to rapidly achieve the fullness target. At the end of this phase, the fullness, size distribution and directory depth distributions are all met. The rest of Geriatrix aging is designed to fit the relative age distribution.

Table 2: List of built-in aging profiles in Geriatrix with their descriptions, the age of the oldest file in each profile, the duration to age a 50 GB Xfs partition in memory with Geriatrix for every profile, and the number of disk overwrites (workload) performed.

Profile	*Description*	*Age (yrs)*	*Duration (min)*	*Overwrites (50 GB)*	*Age Distribution*
Douceur[*]	Dating back to 1998, this is the oldest file system aging profile referenced from a study of file system contents by Douceur et al. [12]. It captures an aggregate analysis of over 10000 commercial PCs running Microsoft Windows.	4	NA	22422 (1 GB)	
Agrawal	Referenced from [3], this aging profile is built using results from a metadata study of Windows file systems running in 2004. It captures desktop workloads from Microsoft employees' computers with most of them running NTFS (80%) along with FAT32 (15%) and FAT (5%).	14	466	253	
Meyer	This profile is referenced from [30], a deduplication study conducted on 857 Windows desktop computers at Microsoft. The snapshots of the file systems were taken in 2009.	2	78	159	
Wang-OS	Aimed at studying high-performance computing file system environments [44], this aging profile captures characteristics from NetApp's WAFL [15] installations at CMU's Parallel Data Lab; an educational cluster setup for systems' research. The file system snapshot is from 2011.	22	231	34	
Wang-LANL	From the same study as above, this aging profile captures the file system environment in Los Alamos National Lab (LANL) running Panasas File System [46]. This study also dates to 2011.	11	146	28	
Dabre	An aging profile captured from in 2017 from the root partition of a colleague's laptop running Ext4.	1	91	4042	
Pramod	Another aging profile captured from in 2017 from the root partition of a colleague's laptop running Ext4.	3.75	27	17	

[*]Douceur 1 GB image required a 22.4 TB workload to converge, thus taking too long to converge for 50 GB.

2. **Stable Aging:** The stable aging phase randomly chooses to either create or delete a file based on a fair coin toss and greedily selects the file size, depth and, for delete, the age bucket that makes the largest progress towards the target distribution.

6 Formal Proof Of Convergence of the Age Distribution

We setup the following notations to show proof of convergence. Assume we need to age a file system with K files on average after the rapid aging phase. Let the age distribution be arbitrarily bucketed into B buckets with the oldest bucket indexed as $b = B$ and the youngest bucket as $b = 1$. Let T be the total number of operations performed in one Geriatrix run and K be the average number of live files existing in the file system at the end of the entire aging process. Let s_b be the length of time for relative age bucket b. Let g_b be the relative number of files required in bucket b at the end of aging. Then we can predict the number of operations required to achieve convergence as follows:

The age distribution after T operations such that:

$$T \geq max\{\ 2Kg_b s_b, \forall b < B \frac{K(2(g_B-1))}{s_B} \frac{K}{s_B}$$

would have converged to the reference distribution i.e. the relative number of files in age bucket b would be g_b.

We will begin by listing a set of necessary conditions and show that they are also sufficient for convergence of the relative age distribution.

Recall that we fit the relative age distribution only in the stable aging phase. Therefore, we first assume that the rapid aging phase ends in the oldest bucket itself. In order for this to happen, we should have performed at least K (creation) operations in the oldest bucket during a run. Since we perform exactly Ts_b operations for every bucket b, this translates to requiring that $Ts_B \geq K$. This is our first necessary condition.

Next, let the total number of stable aging operations we make in any bucket b be O_b. For the oldest bucket, $O_B = Ts_B - K$ and in the other buckets $O_b = Ts_b$. Let C_b and D_b respectively be the number of stable aging creation and deletion operations performed in any bucket b. For every bucket b, we need Kg_b files at the end of aging. For bucket B, if $K + C_B < Ks_B$, we will not be able to achieve this. Hence, a necessary condition here is that $K + C_B \geq Ks_B$. For the other buckets, we will similarly require that $C_b \geq Ks_b$.

We will now assume that $C_b \approx \frac{O_b}{2}$ and $D_b \approx \frac{O_b}{2}$ (since we perform creations and deletions in the stable aging phase using a fair coin toss) and argue that as long as the above inequalities are satisfied, our operations will converge to the right distribution i.e., these conditions are also sufficient. Less formally, we only need to show that we have sufficiently many delete operations to delete excess files in each bucket. We will exemplify using the youngest bucket. In particular, the youngest bucket can afford (around) $\frac{O_1}{2}$ delete operations which can be partly used to delete the excess of $\frac{O_1}{2} - Ks_1$ files, hence achieving the required number of files in bucket 1. The remaining Ks_1 deletes can be spent on older buckets. Therefore, when we consider the bucket 2, the next youngest bucket, we not only have $\frac{O_2}{2}$ delete operations from bucket 2 itself, but an additional Ks_1 borrowed delete operations. However, we will only need to use $\frac{O_2}{2} - Ks_2$ deletes in bucket 2, thus saving ourselves $K(s_1 + s_2)$ delete operations for buckets $b > 2$. By induction, when we reach the oldest bucket, having ensured that all younger buckets have the right amount of files, we are left with $K\sum_{b=2}^{B} s_b$ delete operations borrowed from the younger buckets. We will also have $\frac{O_B}{2}$ delete operations belonging to bucket B. Meanwhile, we need to delete around $K + C_b - Ks_b = K + \frac{O_B}{2} - Ks_B$ excess files in this bucket. Fortunately, it turns out that $K + \frac{O_B}{2} - Ks_b$ is in fact equal to the total number of delete operations available for this bucket $K\sum_{b=2}^{B} s_b + \frac{O_B}{2}$, hence achieving the right number of files in this bucket too (since $\sum_{b=1}^{B} s_b = 1$).

Now to provide a lower bound for T, recall that we assumed $C_b \approx \frac{O_b}{2}$ and we required $K + C_B \geq Ks_B$ along with $C_b \geq Ks_b$ for all $b < B$. Thus, we get $T \geq \frac{K(2(g_B-1))}{s_B}$ and $T \geq \frac{2Kg_b}{s_b}, \forall b < B$.

7 The Aging Suite

The aging tool is a C++ program (built using the Boost library) designed to run on unix platforms. It has the ability to age any POSIX compliant file system.

- **Reducing Setup Complexity:** Geriatrix is profile driven. It has seven built-in aging profiles to match long-running file system and metadata publications [3, 12, 44, 30] described in Section 4. We converge to the input age distribution shown in the age distribution column in Table 2 with a root-mean squared error of $< 0.01\%$ for each profile. Since size and directory depth distributions converge trivially, we have not shown them pictorially.

- **Parallel Aging:** Geriatrix has a configurable thread pool that exploits multi-threading in file systems to expedite aging substantially.

- **Reproducibility:** Every Geriatrix run is seeded to ensure exact reproducibility of operations compared to non-multi-threaded Geriatrix runs. The suite contains a hardcoded seed which can be overridden via a custom seed parameter.

- **Rollback Utility:** Aging experiments can take a prohibitively long time. Once a file system image has been aged, taking a snapshot of the image to be able to restore the same image for multiple tests usually takes less time than re-aging. This does require a whole disk overwrite, which on today's large disks can take several hours, so we have developed a rollback utility to undo the effects of a short benchmark run on an aged image without having to replay the entire aged image again. Using the *blktrace* utility [6], we monitor the blocks that were modified during benchmark execution and replace them from a copy of the aged image. *blktrace* adds overhead when running a benchmark, but is often negligible and can be mitigated further by writing the *blktrace* output to an in-memory file system or sending it across the network.

- **Multiple Stopping Conditions:** For many users, waiting for $< 0.01\%$ root mean square convergence of a Geriatrix run might be overkill. Thus, we have introduced multiple stopping conditions:

  1. the amount of time the ager is allowed to run
  2. the confidence [1] of the age distribution fit
  3. a maximum number of disk overwrites during aging

  Once any stopping condition is met, Geriatrix stops and displays the values of all three stopping conditions. The user can choose to revise the conditions and resume aging, measured as the total amount of data written over the total writable space on the disk.

8 Evaluation of Geriatrix as an Aging Tool

We measure the power of Geriatrix aging by comparing the file and free space fragmentation induced on a fresh Ext4 partition when it is aged according to the measures of a one year old 20 GB Ext4 partition

[1] Confidence of the convergence of distributions is calculated using the chi-squared goodness-of-fit statistic.

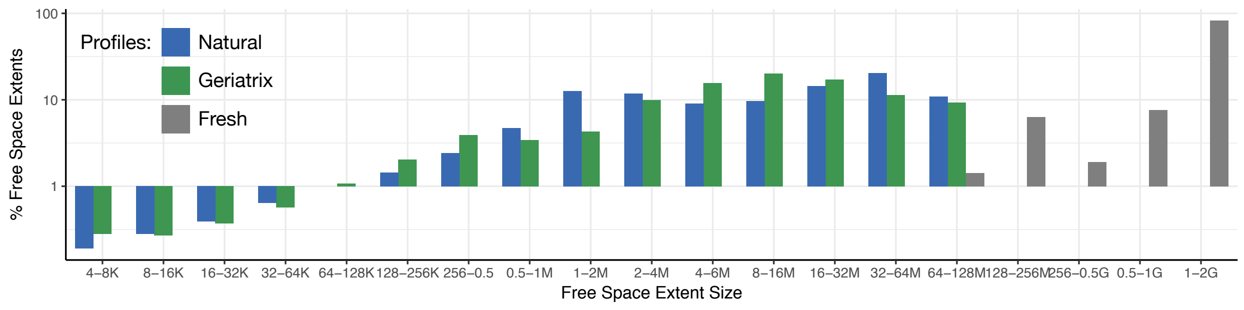
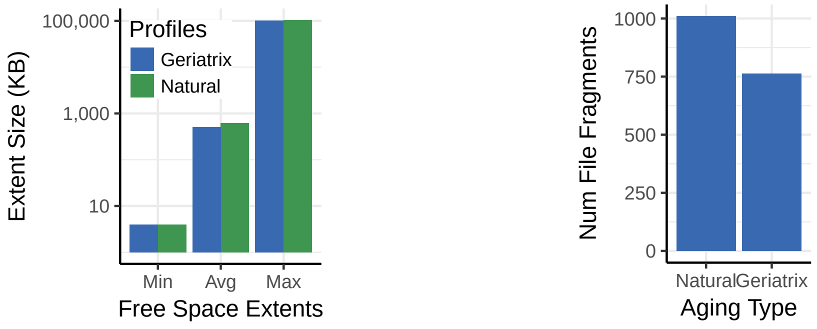


Figure 2: Free space fragmentation comparison of a freshly formatted Ext4 partition, an actual old Ext4 file system image (Dabre), and Ext4 aged with Geriatrix driven by the Dabre profile.



(a) Extents

(b) File Fragmentation

Figure 3: 3a compares the the minimum, average and maximum size of free space extents for a naturally aged Ext4 image and an Ext4 image aged using Geriatrix for the Dabre aging profile. Figure 3b shows the number of fragments allocated as a result of a 2 GB file being copied to both file system images.

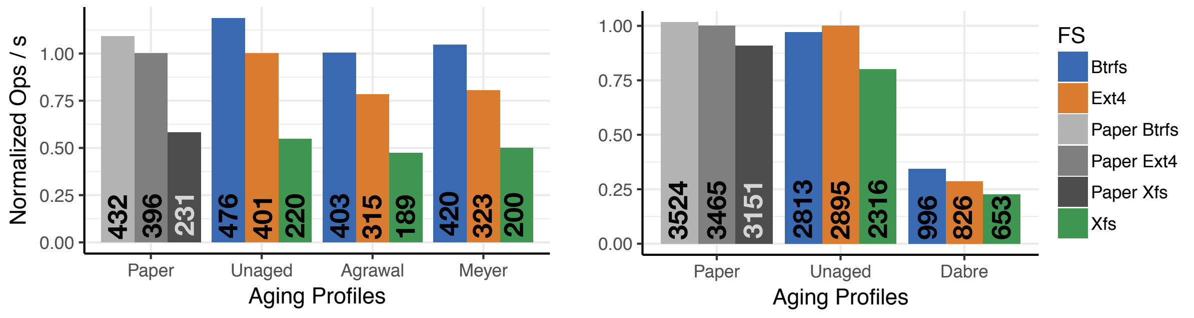
with approximately 80% capacity utilization[2]. Note that Geriatrix is designed to match externally visible measures of a file system, not storage allocation fragmentation. But, because Geriatrix exercises the file system extensively to achieve the externally visible goals, it also induces storage fragmentation. We measured the distribution of the extents of free space using the *e2freefrag* utility. For a baseline comparison, we also compare with the free space fragments present in a freshly formatted Ext4 partition of the same size.

The free space fragmentation comparison is shown in Figure 2. Freshly formatted Ext4 has very large free space extents, mostly between 1-2 GB. The naturally aged Ext4 has a more spread out free space extent distribution ranging from 4 KB to 100 MB. The file system image aged using Geriatrix has an extent distribution very similar to the naturally old image. Moreover, both aged images do not have any GB-sized extents.

[2]obtained from a colleague's laptop (also added as a built-in aging profile called Dabre - refer Table 2)

Paper	Disk	RAM	CPU (cores)	Linux (Kernel Version)
Btrfs [36]	500 GB HDD (WDC WD5000YS-01MPB0)	2 GB	Intel Xeon E7 (8)	Ubuntu 14.04 LTS (3.13.0-33)
	64 GB SSD (Crucial M4-CT064M4SSD2)	2 GB	AMD Opteron (8)	Ubuntu 14.04 LTS (4.4.0-31)
F2fs [22]	64 GB SSD (Crucial M4-CT064M4SSD2)	4 GB	Intel Core i7 (4)	Ubuntu 14.04 LTS (4.4.0-31)
	120 GB SSD (ADATA SSD S510)	4 GB	Intel Core i7 (4)	Ubuntu 14.04 LTS (4.4.0-31)
NOVA [47]	64 GB NVM (Emulated in DRAM)	8 GB	AMD Opteron (8)	Ubuntu 14.04 LTS (4.13)

Table 3: Experimental Configuration.



(a) Btrfs HDD Fileserver

(b) Btrfs SSD Fileserver

Figure 4: Both graphs reproduce Filebench fileserver experiments from the Btrfs ACM TOS publication [36] on aged file system instances on a HDD (Figure 4a) and a SSD (Figure 4b). Aged Btrfs and Ext4 performed at most 22% slower on the HDD but supported the prior paper's published rank ordering whereas aged Btrfs and Ext4 on a SSD degraded benchmark performance by as much as 80%, and changed the rank ordering of compared file systems.

Figure 3a compares the minimum, average and maximum free space extent sizes between naturally aged Ext4 and Ext4 aged using Geriatrix. Both have the same smallest free space extent of 4 KB. The average free space extent size of naturally aged Ext4 is only 112 KB smaller than its Geriatrix counterpart, while the largest free space extent in the naturally aged Ext4 is only 2 MB smaller than the Geriatrix image. For a 20 GB image, the free space fragmentation distribution after aging using Geriatrix is almost identical to the naturally aged file system.

Figure 3b measures the fragmentation of a 2 GB file copied to both, the naturally aged Ext4 and Ext4 aged with Geriatrix, using the *filefrag* utility. The image aged by Geriatrix splits the file into 763 fragments while the naturally aged file system splits it into 1010 fragments. Despite a roughly 25% difference in the number of file extents created in both cases, Geriatrix aging is orders of magnitude more natural than the number of fragments created writing 2 GB to a freshly formatted Ext4.

Since Geriatrix refrains from taking shortcuts, and performs millions of operations before declaring a file system *aged*, it approximates the file system state caused by natural aging to a close degree. This is surprising because there is no fragmentation goal or monitoring during a run of Geriatrix. The Geriatrix aging experiment reported in Figure 2 and 3 took approximately 90 minutes. Recreating the fragmentation naturally occurring in one year with only 90 minutes of aging is an acceleration of 5800x! However, the cost of aging depends on the aging profile and the file system capacity, only 20 GB here. Practical optimizations explained in section 10 can help limit aging cost.

9 Evaluation of Aged File Systems

In order to highlight the impact of aging, we recreated experiments from Btrfs [36], F2fs [22] and NOVA [47] publications on unaged and aged file system instances. We also produced aged instances of Ext4 (used for comparison across all three papers) and Xfs (used for comparison with the Btrfs and NOVA papers). All file systems were aged on the Ubuntu 14.04 LTS distribution. Except NOVA all our aging experiments were conducted with the stock Linux kernel 3.13.0-33 installed in Ubuntu; NOVA required special kernel packages to support non-volatile memory emulation. We performed all our experiments on an Emulab PRObE cluster [14]. The hardware used for experiments is described in Table 3. For fair comparison, we matched the memory and the number of cores in our benchmark when recreating experiments from the Btrfs [36] and F2fs [22] publications.

All file systems were aged according to the Agrawal [3], Meyer [30], Dabre and Pramod aging profiles.

We performed aging in a large memory and captured the resulting aged images. Prior to each benchmark run we copied the corresponding aged image onto a disk (using dd to the raw device) and mounted the file system on the aged image. All file systems were mounted using default mount options.

The Filebench benchmark [27] was used for all performance measurements with different profiles according to the appropriate reference publication. The primary performance metric reported is *overall operations per second* as reported by Filebench. Each benchmark run lasted about 10 minutes and we performed three runs of each benchmark and captured variance. We report only the mean since the maximum standard deviation observed was below 2. Since our hardware is not identical to what was used in the papers and since we are testing with newer code, exact reproduction of paper results even for unaged instances of file systems is unlikely. With SSDs, the performance variability across devices is especially high. For ease of comparison, we include raw data on the bar graphs, but normalize bar heights. The published results (leftmost gray bars) are normalized to the published Ext4 results and the aged file system performance numbers are normalized to unaged Ext4 performance on the same hardware. We chose Ext4 because it is the default file system rolled out with most Linux distributions today. All HDD experiments were conducted using 100 GB aged images with a 80% capacity utilization target being replayed on a 100 GB partition of a 500 GB HDD.

Figure 1a in Section 1 compares the performance between Btrfs, Ext4 and Xfs on an HDD using the Filebench varmail application - a mail server workload comprised of tiny file operations. After aging using the Meyer profile, we see 9-30% slowdowns, with Btrfs being most affected after aging, followed by Ext4 and the least affected was Xfs. This is in line with the age of the file system (in terms of their development) and hence their stability.

We now compare the same file systems using the webserver worklaod in Figure 1b. The webserver workload is highly multithreaded and operates on tiny files, although it avoids issuing expensive fsync operations and mimics webservers which have to perform more whole-file reads and log appends. Since file systems are usually more sensitive to small file operations, it is perhaps understandably harder to reproduce published results, and in fact, unaged Btrfs performs 22% faster on our hardware than reported in the paper, while unaged Xfs also performs 10% faster than the paper. We also see a minor inversion of rank, with unaged Btrfs outperforming unaged Xfs. The performance penalties after aging are between 12-17%. We show only the Meyer aging profile in webserver and varmail because Btrfs could not sustain aging for the Pramod profile, and although Btrfs successfully completed aging for Dabre and Agrawal profiles, it did not complete execution of the benchmark despite having the required space to do so. This highlights an important use of Geriatrix also a stress testing tool.

Our last HDD comparison is shown in Figure 4a on the fileserver workload, which consists of relatively larger file writes and reads compared to thousands of small file operations in webserver and varmail. We observe that unaged Btrfs has improved by 10%, unaged Ext4 is marginally better while unaged Xfs is about 5% slower on our hardware, keeping performance largely similar to published results with slightly increased performance gaps between the file systems. After aging, we observe a 10-22% performance drop after aging with Ext4 being the most affected after aging using the Agrawal profile.

The SSD experiments were conducted on a 64 GB SSD with a 59 GB aged file system image with a 70% fullness target. The reason for choosing 70% was to allow the benchmarking workload to fit after aging. SSDs are available in a variety of product price-point classes and have highly variable performance making reproduction of SSD results on different hardware unlikely. Figure 4b shows the completely different rank ordering of unaged Btrfs, Ext4 and Xfs compared to published results on the fileserver workload. While Btrfs had the best performance in the paper, it was the best only in one aged reproduction using the Dabre profile 1c, although after having suffered a 60% slowdown. Ext4 appears to be the winner in our reproductions after aging using the Agrawal and Meyer profiles. Aging appears to degrade file systems performance much more on SSDs compared to HDDs with aged performance dropping by 73-80%. We attribute this performance drop to the flash translation layer (FTL) in the SSD performing continuous garbage collection. It was typical

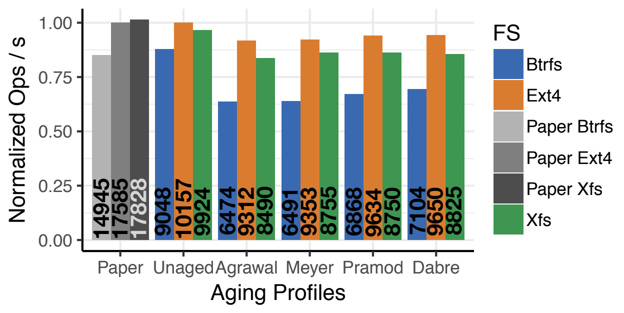


Figure 5: Filbench webserver recreations from the Btrfs paper [33] on SSD. Btrfs is most affected by aging with its performance dropping by 30%. Ext4 and Xfs performance drops by a maximum of 9% and 15% respectively.

of SSDs from a few years ago to not be able to sustain more than 2 minutes of continuous writing before performing inline cleaning [34]. Our benchmarking technique involves writing an aged image on almost the entire surface of the SSD, performing a 10 minute benchmark run followed by unmounting the file system and repeating the process with 100% device utilization. An entire surface rewrite should be equivalent to a giant trim obviating the need to perform any internal garbage collection in the FTL, but this is dependent on firmware implementation which varies substantially across devices. Most seriously, the rank ordering of aged file systems is different compared to published results making aging a much more important exercise in the context of SSDs. The webserver results shown in Figure 5 also show significant slowdowns, but are not so dramatic. Btrfs appears to be the most affected by aging showing a 50% performance drop but Ext4 and Xfs do not degrade much showing a maximum of performance penalty of 35% and 18% respectively.

Figures 6 is the recreation of F2fs [22] results on SSDs comparing Btrfs, Ext4 and F2fs using the Filebench fileserver profile. To capture variability of performance across devices, we chose SSDs of different makes and sizes - a 64 GB Crucial SSD with 59 GB aged file system images (bars labeled 64) and a 120 GB ADATA SSD with 100 GB aged file system images (bars labeled 120). Ext4 is the winning file system when comparing unaged file system instances on 64 GB drives, and Btrfs is marginally better on the 120 GB drives, while published results report F2fs performance was 2.4x that of Ext4. Aging on 64 GB drives shows interesting behavior as the performance of all three file systems drops (61-67% for Ext4, 76-78% for Btrfs and 2-5% for F2fs) and the outcome looks similar to results that the earlier paper reported. The authors most likely aged the SSD firmware by performing repeated benchmark runs resulting in behavior similar to what is seen when file systems are aged. In contrast, the 100 GB file systems on the 120 GB drive age much more gracefully with only Btrfs showing as much as 7% performance penalty after aging. This suggests that SSDs themselves age in non-trivial ways along with the file systems running on them.

We also perform aging experiments on NOVA [47] - a log-structured file system intended for non-volatile memory (NVM). We used a modified Linux kernel version 4.13 to age NOVA since it required special kernel libraries for enabling persistent memory emulation, absent in Linux kernel 3.13.0-33. A 64 GB partition similar to the SSD experiments was aged with 70% utilization. The NOVA paper performed experiments with 64 GB persistent memory devoted to NOVA and 32 GB DRAM for the rest of the system. The experiment dataset size was made slightly larger than DRAM (more than 32 GB) forcing NVM device interaction during an experiment run. However, it appears that more than half the 64 GB NVM device was used in the published benchmark run. A <50% utilized file system seemed too less for conducting a realistic post-aging experiment. Hence, we reduced the DRAM size to 8 GB and exercised a workload of more than 8 GB to still ensure that the benchmarking workload was larger than the system memory. Moreover, the authors performed their experiments on special Intel NVM hardware. Thus, the performance in Figures 7a and 7b, should not be directly compared with the NOVA paper results.

Figure 7a compares NOVA's performance on the fileserver workload before and after aging with Btrfs,

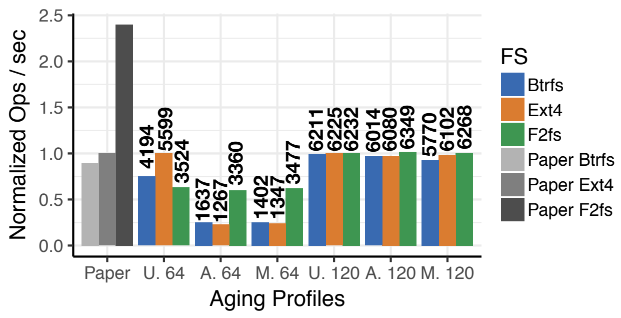
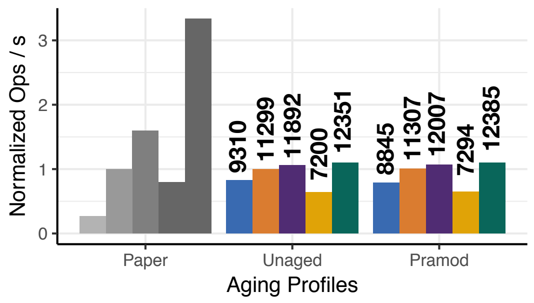
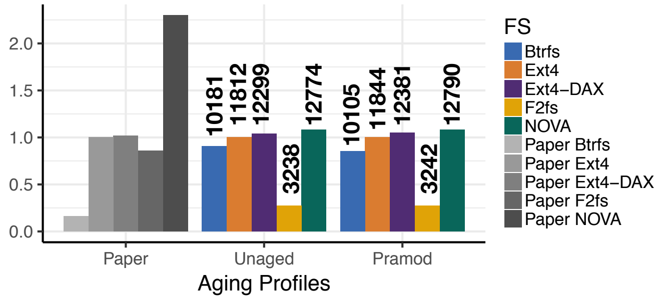


Figure 6: Recreation of the Filebench fileserver benchmark from the F2fs paper [22] on two SSDs - 64 GB (labeled 64) and 120 GB (labeled 120). Performance on 64 drops significantly after aging, especially for Btrfs and Ext4 resulting in a graph that looks similar to published results. Refer Section 9 for detailed explanation. 120 seems unaffected by aging, thus highlighting highly varied performance across different SSDs

.



(a) NOVA Fileserver



(b) NOVA Varmail

Figure 7: Both graphs reproduce experiments from the NOVA FAST 2016 publication [47] on aged file system instances using emulated non-volatile memory. All file systems age gracefully with Btrfs seeing the largest performance hit of 5% on the fileserver workload (Figure 7a) and 6% on the varmail workload (Figure 7b). Graphs from the paper are for reference and in this case cannot be compared to our reproductions because of different hardware and configuration.

Ext4, Ext4-DAX and F2fs. Ext4-DAX is Ext4 mounted with the *-o dax* option to enable direct-access to the emulated NVM device bypassing the buffer cache, thus avoiding duplicate caching of data. With the largest difference of 5% observed in Btrfs, we see virtually no difference in all file systems before and after aging. The same is true in the case of the varmail workload shown in Figure 7b where Btrfs is the most affected file system with a slowdown of approximately 6%.

Although throughput is unaffected, the tail latencies highlight the effects of aging. Figure 8 shows the latency encountered by Filebench for a particular operation. The slowest file open in the fileserver workload was 60% slower after aging. Writing the slowest file was also took twice as long compared to a freshly formatted NOVA image. Surprisingly, closing the slowest file was 5x faster after aging. In the varmail workload, the slowest aged read was 2.3x slower than the slowest unaged read. In contrast, the opening of the slowest aged file was 25% faster. Both these observations can be attributed to the log-structured design of NOVA because log-structured file systems are not read-optimized. We speculate that reorganization of files after cleaning might have led to the open call being executed faster.

As expected, with no mechanical parts and DRAM-like latency, NVM or in-memory file systems age gracefully. An aging exercise for these file systems is mainly about exposing inefficiencies in file system implementations that usually get hidden behind massive device latencies.

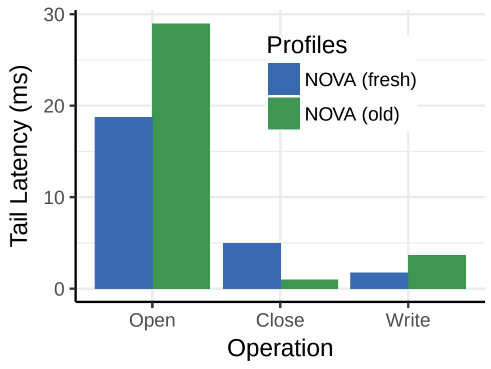


Figure 8: This graph shows the latency of the slowest operations for the fileserver workload. Aged NOVA slows down by upto 60% on file opens and upto 2x on file writes.

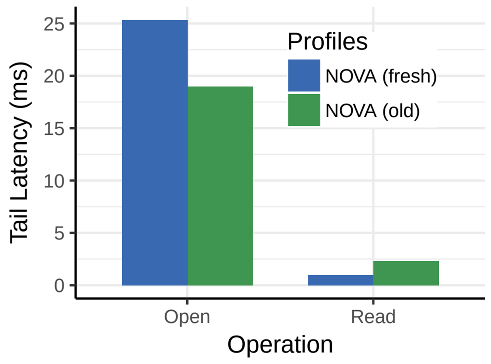


Figure 9: We see slowdowns of upto 2.3x on reads of the slowest file in the varmail tail latencies shown above. The aged opens are contrastingly faster for the varmail workload compared to the fileserver workload. Slowest file open during the varmail workload run is 25% faster after aging.

10 Practical Tips for Aging

- **Using *fallocate*:** *fallocate* or *posix_fallocate* is the system call that forces the file system to allocate metadata without allocating any data. As a result, the file system data structures are correctly manipulated reserving the necessary disk blocks for the file contents, but the contents are never flushed to disk resulting in much faster aging. Geriatrix tries to use *fallocate* where implemented.
- **Aging in Memory:** Age on an in-memory file system and take a snapshot of the aged in-memory file system image [3]. This results in orders of magnitude improvement in aging speeds.

11 Device Idiosyncrasies

Primary storage no longer means HDDs. A range of devices have found their place in the storage hierarchy and their characteristics dictate interesting implications for aging.

[3]In 2016, AWS EC2 has 2 TB memory instances for rent

11.1 Aging SSD based file systems

SSDs contain a complex translation layer in the device firmware called a flash translation layer (FTL). FTLs are the reason than an SSD can act as a drop-in replacement for an HDD despite having a radically different architecture. FTLs primarily perform the tasks of address mapping, garbage collection and wear leveling. SSD device characteristics force the FTL to operate very similarly to a complex log structured file system. Moreover, the FTL mapping tables can get fragmented over time resulting in a significant performance hit. Thus, in the case of an SSD, two systems are aging with time, the FTL and the file system running on the SSD. Since the FTL is proprietary, users typically have no insight into how well (or poorly) the FTL has aged. Responsible aging of SSD based file systems should take this factor into account, and hence perform both operations, namely the aging process and the post-aging performance benchmark on the same device.

11.2 Aging on Large HDDs

In the era of multi-terabyte HDDs, it might take prohibitively long to age a file system image spanning an entire disk surface. For practical reasons, if you perform aging on an image much smaller than the capacity of the disk then seeks tend to be artificially localized. One way of artifically lengthening seeks is by constructing on the large disk a logical volume (whose size matches the size of an aged image) by stitching physical partitions spread over the entire disk surface. This is an imperfect scheme since gaps between physical partitions belong to logically contiguous extents.

11.3 Aging on Shingled Disks

Shingled magnetic recording (SMR) is a new disk architecture wherein adjacent tracks on an HDD are partially overlapping to increase the number of tracks on the disk, thus increasing the disk capacity. This technology came about to address traditional HDDs having surpassed the superparamagnetic effect [41], which essentially disables further increasing sectors-per-track in order to achieve larger disk capacities. Commercially available SMR drives have a firmware different than, but as complicated as an FTL. Aghayev et al. [1] showed the massive interference of the firmware while performing foreground tasks resulting in a high variance in performance. One of the key aspects of a firmware driven shingled disk is the existence of a persistent cache at the center of the drive. Suppose we are benchmarking a fresh file system on an SMR drive. We might conceivably never hit the persistent cache limit, hitting which would cause the firmware to perform large read-modify-write cycles that reduces performance significantly. As this particular issue is hidden in the HDD firmware, the right way to benchmark a file system intended for an SMR drive, is to age the file system on the drive. This would bring the performance of the firmware also into account resulting in much more holistic measurements.

12 Conclusion

File systems continue to suffer from fragmentation and data structure complexity during extended use: an effect called aging. Although the aging problem has been known for a long time, recent research on file systems usually ignores it. More importantly, aging effects are sometimes more dramatic in solid state disks than on hard drives. To assist file system developers and researchers, we offer Geriatrix, a synthetic workload generator guided by an aged file system profile. With Geriatrix, aging file systems before benchmarking will be mechanical and reproducible, if still expensive.

References

[1] Abutalib Aghayev, Mansour Shafaei, and Peter Desnoyers. Skylighta window on shingled disk operation. *ACM Transactions on Storage (TOS)*, 11(4):16, 2015.

[2] Nitin Agrawal, Andrea C Arpaci-Dusseau, and Remzi H Arpaci-Dusseau. Generating realistic impressions for file-system benchmarking. *ACM Transactions on Computer Systems (TOCS)*, 5(4):16, 2009.

[3] Nitin Agrawal, William J Bolosky, John R Douceur, and Jacob R Lorch. A five-year study of file-system metadata. *ACM Transactions on Storage (TOS)*, 3(3):9, 2007.

[4] Akshat Aranya, Charles P Wright, and Erez Zadok. Tracefs: A file system to trace them all. In *USENIX File and Stroage Technologies (FAST)*, pages 129–145, 2004.

[5] Alexandros Batsakis, Randal Burns, Arkady Kanevsky, James Lentini, and Thomas Talpey. Ca-nfs: A congestion-aware network file system. *ACM Transactions on Storage (TOS)*, 5(4):15, 2009.

[6] Alan D Brunelle. Block i/o layer tracing: blktrace. *HP, Gelato-Cupertino, CA, USA*, 2006.

[7] Pei Cao, Edward W Felten, Anna R Karlin, and Kai Li. A study of integrated prefetching and caching strategies. *ACM SIGMETRICS Performance Evaluation Review*, 23(1):188–197, 1995.

[8] James Cipar, Mark D Corner, and Emery D Berger. Tfs: A transparent file system for contributory storage. In *USENIX File and Stroage Technologies (FAST)*, volume 7, pages 215–229, 2007.

[9] Douglas Comer. Ubiquitous b-tree. *ACM Computing Surveys (CSUR)*, 11(2):121–137, 1979.

[10] Alex Conway, Ainesh Bakshi, Yizheng Jiao, Yang Zhan, Michael A Bender, William Jannen, Rob Johnson, Bradley C Kuszmaul, Donald E Porter, Jun Yuan, et al. How to fragment your file system.

[11] Alexander Conway, Ainesh Bakshi, Yizheng Jiao, William Jannen, Yang Zhan, Jun Yuan, Michael A Bender, Rob Johnson, Bradley C Kuszmaul, Donald E Porter, et al. File systems fated for senescence? nonsense, says science! In *USENIX File and Stroage Technologies (FAST)*, pages 45–58, 2017.

[12] John R Douceur and William J Bolosky. A large-scale study of file-system contents. *ACM SIGMETRICS Performance Evaluation Review*, 27(1):59–70, 1999.

[13] Richard J Feiertag and Elliott I Organick. The multics input/output system. In *ACM Symposium on Operating Systems Principles (SOSP)*, pages 35–41. ACM, 1971.

[14] Garth Gibson, Gary Grider, Andree Jacobson, and Wyatt Lloyd. Probe: A thousand-node experimental cluster for computer systems research. *USENIX ;login:*, 2013.

[15] Dave Hitz, James Lau, and Michael A Malcolm. File system design for an nfs file server appliance. In *USENIX winter*, volume 94, 1994.

[16] John H Howard, Michael L Kazar, Sherri G Menees, David A Nichols, Mahadev Satyanarayanan, Robert N Sidebotham, and Michael J West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems (TOCS)*, 6(1):51–81, 1988.

[17] William Jannen, Jun Yuan, Yang Zhan, Amogh Akshintala, John Esmet, Yizheng Jiao, Ankur Mittal, Prashant Pandey, Phaneendra Reddy, Leif Walsh, et al. Betrfs: A right-optimized write-optimized file system. In *USENIX File and Stroage Technologies (FAST)*, pages 301–315, 2015.

[18] William K Josephson, Lars A Bongo, Kai Li, and David Flynn. Dfs: A file system for virtualized flash storage. *ACM Transactions on Storage (TOS)*, 6(3):14, 2010.

[19] Nikolai Joukov, Timothy Wong, and Erez Zadok. Accurate and efficient replaying of file system traces. In *USENIX File and Stroage Technologies (FAST)*, volume 5, pages 25–25, 2005.

[20] Ryusuke Konishi, Yoshiji Amagai, Koji Sato, Hisashi Hifumi, Seiji Kihara, and Satoshi Moriai. The linux implementation of a log-structured file system. *ACM Operating Systems Review (SIGOPS)*, (3):102–107, 2006.

[21] Duy Le, Hai Huang, and Haining Wang. Understanding performance implications of nested file systems in a virtualized environment. In *USENIX File and Stroage Technologies (FAST)*, page 8, 2012.

[22] Changman Lee, Dongho Sim, Jooyoung Hwang, and Sangyeun Cho. F2fs: A new file system for flash storage. In *USENIX File and Stroage Technologies (FAST)*, pages 273–286, 2015.

[23] Sungjin Lee, Ming Liu, Sangwoo Jun, Shuotao Xu, Jihong Kim, et al. Application-managed flash. In *USENIX File and Stroage Technologies (FAST)*, pages 339–353, 2016.

[24] Youyou Lu, Jiwu Shu, and Wei Wang. Reconfs: A reconstructable file system on flash storage. In *USENIX File and Stroage Technologies (FAST)*, pages 75–88, 2014.

[25] Chris Mason. Compilebench. http://oss.oracle.com/mason/compilebench.

[26] Michelle L Mazurek, Eno Thereska, Dinan Gunawardena, Richard HR Harper, and James Scott. Zzfs: a hybrid device and cloud file system for spontaneous users. In *USENIX File and Stroage Technologies (FAST)*, page 16, 2012.

[27] Richard McDougall and Jim Mauro. Filebench. http://www.nfsv4bat.org/Documents/nasconf/2004/filebench.pdf, 2005.

[28] Marshall K McKusick, William N Joy, Samuel J Leffler, and Robert S Fabry. A fast file system for unix. *ACM Transactions on Computer Systems (TOCS)*, 2(3):181–197, 1984.

[29] Michael P Mesnier, Matthew Wachs, Raja R Simbasivan, Julio Lopez, James Hendricks, Gregory R Ganger, and David R O'hallaron. //trace: Parallel trace replay with approximate causal events. In *USENIX File and Stroage Technologies (FAST)*, 2007.

[30] Dutch T Meyer and William J Bolosky. A study of practical deduplication. *ACM Transactions on Storage (TOS)*, 7(4):14, 2012.

[31] Changwoo Min, Kangnyeon Kim, Hyunjin Cho, Sang-Won Lee, and Young Ik Eom. Sfs: random write considered harmful in solid state drives. In *USENIX File and Stroage Technologies (FAST)*, page 12, 2012.

[32] R Hugo Patterson, Garth A Gibson, Eka Ginting, Daniel Stodolsky, and Jim Zelenka. *Informed prefetching and caching*, volume 29. ACM, 1995.

[33] Samuel Petrovic. The effects of age on file system performance. Master's thesis, Masaryk University, 5 2017. An optional note.

[34] Milo Polte, Jiri Simsa, and Garth Gibson. Enabling enterprise solid state disks performance. 2009.

[35] OM Ritchie and Ken Thompson. The unix time-sharing system. *The Bell System Technical Journal*, 57(6):1905–1929, 1978.

[36] Ohad Rodeh, Josef Bacik, and Chris Mason. Btrfs: The linux b-tree filesystem. *ACM Transactions on Storage (TOS)*, 9(3):9, 2013.

[37] Drew S Roselli, Jacob R Lorch, Thomas E Anderson, et al. A comparison of file system workloads. In *USENIX Annual Technical Conference (ATC)*, pages 41–54, 2000.

[38] Margo Seltzer, Keith A Smith, Hari Balakrishnan, Jacqueline Chang, Sara McMains, and Venkata Padmanabhan. File system logging versus clustering: A performance comparison. In *USENIX 1995 Technical Conference Proceedings (TCON)*, pages 21–21. USENIX Association, 1995.

[39] Keith Smith and Margo I Seltzer. File layout and file system performance. *Tech. Rep. TR-35-94, Harvard University*, 1994.

[40] Keith A Smith and Margo I Seltzer. File system aging increasing the relevance of file system benchmarks. In *ACM SIGMETRICS Performance Evaluation Review*, volume 25, pages 203–213. ACM, 1997.

[41] David A Thompson and John S Best. The future of magnetic data storage techology. *IBM Journal of Research and Development*, 44(3):311–322, 2000.

[42] Cristian Ungureanu, Benjamin Atkin, Akshat Aranya, Salil Gokhale, Stephen Rago, Grzegorz Calkowski, Cezary Dubnicki, and Aniruddha Bohra. Hydrafs: A high-throughput file system for the hydrastor content-addressable storage system. In *USENIX File and Stroage Technologies (FAST)*, volume 10, pages 225–239, 2010.

[43] Michael Vrable, Stefan Savage, and Geoffrey M Voelker. Bluesky: a cloud-backed file system for the enterprise. In *USENIX File and Stroage Technologies (FAST)*, pages 19–19. USENIX Association, 2012.

[44] Yifan Wang. A statistical study for file system meta data on high performance computing sites. Master's thesis, Southeast University, 2012.

[45] Zev Weiss, Tyler Harter, Andrea C Arpaci-Dusseau, and Remzi H Arpaci-Dusseau. Root: Replaying multithreaded traces with resource-oriented ordering. In *ACM Symposium on Operating Systems Principles (SOSP)*, pages 373–387. ACM, 2013.

[46] Brent Welch, Marc Unangst, Zainul Abbasi, Garth A Gibson, Brian Mueller, Jason Small, Jim Zelenka, and Bin Zhou. Scalable performance of the panasas parallel file system. In *USENIX File and Stroage Technologies (FAST)*, volume 8, pages 1–17, 2008.

[47] Jian Xu and Steven Swanson. Nova: a log-structured file system for hybrid volatile/non-volatile main memories. In *USENIX File and Stroage Technologies (FAST)*, pages 323–338, 2016.

[48] Jun Yuan, Yang Zhan, William Jannen, Prashant Pandey, Amogh Akshintala, Kanchan Chandnani, Pooja Deo, Zardosht Kasheff, Leif Walsh, Michael Bender, et al. Optimizing every operation in a write-optimized file system. In *USENIX File and Stroage Technologies (FAST)*, pages 1–14, 2016.

[49] Shuanglong Zhang, Helen Catanese, and Andy An-I Wang. The composite-file file system: decoupling the one-to-one mapping of files and metadata for better performance. In *USENIX File and Stroage Technologies (FAST)*, pages 15–22, 2016.

[50] Zhihui Zhang and Kanad Ghose. yfs: A journaling file system design for handling large data sets with reduced seeking. In *USENIX File and Stroage Technologies (FAST)*, volume 3, page 2nd, 2003.

[51] Benjamin Zhu, Kai Li, and R Hugo Patterson. Avoiding the disk bottleneck in the data domain deduplication file system. In *USENIX File and Stroage Technologies (FAST)*, volume 8, pages 1–14, 2008.

[52] Ningning Zhu, Jiawu Chen, Tzi-Cker Chiueh, and Daniel Ellard. Tbbt: scalable and accurate trace replay for file server evaluation. In *ACM SIGMETRICS Performance Evaluation Review*, volume 33, pages 392–393. ACM, 2005.