Geriatrix: Aging what you see and what you don't see. A file system aging approach for modern storage systems

Saurabh Kadekodi, Vaishnavh Nagarajan, and Gregory R. Ganger, *Carnegie Mellon University;* Garth A. Gibson, *Carnegie Mellon University, Vector Institute*

https://www.usenix.org/conference/atc18/presentation/kadekodi

This paper is included in the Proceedings of the
2018 USENIX Annual Technical Conference (USENIX ATC '18).

July 11–13, 2018 • Boston, MA, USA

ISBN 978-1-939133-02-1

Open access to the Proceedings of the
2018 USENIX Annual Technical Conference
is sponsored by USENIX.

Geriatrix: Aging what you see and what you don't see
A file system aging approach for modern storage systems

Saurabh Kadekodi[1] Vaishnavh Nagarajan[1] Gregory R. Ganger[1] Garth A. Gibson[1,2]
[1]Carnegie Mellon University [2]Vector Institute

Abstract

File system performance on modern primary storage devices (Flash-based SSDs) is greatly affected by aging of the free space, much more so than were mechanical disk drives. We introduce *Geriatrix*, a simple-to-use profile driven file system aging tool that induces target levels of fragmentation in both allocated files (what you see) and remaining free space (what you don't see), unlike previous approaches that focus on just the former. This paper describes and evaluates the effectiveness of Geriatrix, showing that it recreates both fragmentation effects better than previous approaches. Using Geriatrix, we show that measurements presented in many recent file systems papers are higher than should be expected, by up to 30% on mechanical (HDD) and up to 80% on Flash (SSD) disks. Worse, in some cases, the performance rank ordering of file system designs being compared are different from the published results.

Geriatrix will be released as open source software with eight built-in aging profiles, in the hopes that it can address the need created by the increased performance impact of file system aging in modern SSD-based storage.

1 Introduction

The performance of a file system (FS) usually deteriorates over time. As FSs experience heavy churn, techniques such as write-back caching to expedite writes [38, 31, 14], data prefetching to assist reads [8, 35] and self-balancing data structures to contain search times [10] 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 FS performance is poor FS layout [43, 42].

Naturally, therefore, FS benchmarking should consider the effects of aging. But, despite it being an important issue known for over twenty years [44], most research and benchmarks still ignore aging. For example, 65% (13 of 20) recent FS papers we examined (Table 1) neither mention aging nor include it in their evaluations. Unsurprisingly, our experiments confirm that aging continues to be a critical factor for FS performance.

While aging's overall importance has not waned, the particular aspects that have the most impact have

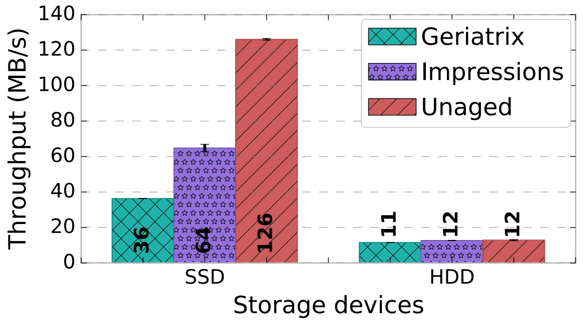


Figure 1: Aging impact on Ext4 atop SSD and HDD. The three bars for each device represent the FS freshly formatted (unaged), aged with Geriatrix, and aged with Impressions [2]. Although relatively small differences are seen with the HDD, aging has a big impact on FS performance on the SSD. Although their file fragmentation levels are similar, the higher free space fragmentation produced by Geriatrix induces larger throughput reductions than for Impressions. The experimental setup is detailed in Section 6.

changed over time, making previous aging approaches... stale. Previous general purpose aging approaches, from the work of Smith and Seltzer [44] to the state-of-the-art *Impressions* [2] tool, focus on achieving representative levels of file fragmentation. Such fragmentation exists when sequential blocks of a file or related metadata / files are scattered among logical block addresses (LBAs) of underlying storage. For FSs atop HDDs, with time-consuming mechanical positioning costs for accessing scattered LBAs, these file fragmentation effects are of most concern. For the Flash-based SSDs that now dominate primary and performance-tier deployments, these effects are less significant, due to LBA remapping and absence of mechanical positioning.

This paper introduces *Geriatrix*, a new FS aging tool for modern storage. In addition to file fragmentation, Geriatrix aggressively induces free space fragmentation and thereby even ages any underlying device remapping structures. As a result, it can recreate the much more significant aging effects seen with SSDs in real systems [36]. As one example, Figure 1 compares three instances of an Ext4 FS: Unaged, aged by Impressions, and aged by Geriatrix. On the HDD, fairly minor performance differences are observed for this particular benchmark, since the workload involves small files and relatively little access locality. On the SSD, however, large differences can be seen, and the Geriatrix-aged FS produces much greater aging effects.

Additional evaluation and experiments, later in the paper, confirm that these greater Geriatrix-induced effects are consistent and correctly representative. In addition, we recreated experiments from recent papers, showing both, that the reported performance fails to represent realistic expectations **and** that the rank ordering of configurations compared is sometimes changed. Figure 2 shows one such re-evaluation, in which we observe both effects.

Geriatrix uses a sophisticated profile-driven approach that ages a FS according to a reference (old) FS. This paper describes how Geriatrix extracts information from a profile and exercises the FS to recreate its fragmentation properties. With both theoretical analysis and experimental comparison to real FS images used as profiles, we show that Geriatrix faithfully reproduces both file and free space fragmentation.

Geriatrix is being released as an open source tool, together with eight built-in aging profiles and a repository of aged images of popular FSs. We hope that its availability will help increase the use of aging in FS benchmarking.

This paper makes three primary contributions. First, it exposes the impact of free space fragmentation and device aging for FSs on SSDs and the failure of existing aging approaches to recreate them. Second, it describes a new aging approach, embodied in Geriatrix, and confirms that it does faithfully recreate these aging effects. Third, it provides extensive evidence, including recreating recently published comparisons and showing that results change, of why aging must be part of benchmarking and offers Geriatrix as an open-source tool for doing so.

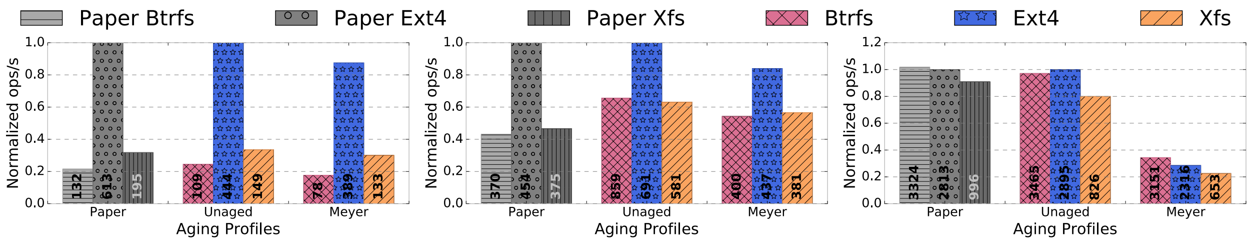
2 Related work

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 FSs expecting a highly specialized workload. Traces can be captured and replayed at multiple levels - the network level [57], file level [32], FS level [40, 4], system call level [50], VFS level [20] and also at the block level [7]. Low level traces are typically FS specific resulting in loss of usefulness for comparing different FSs. 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 [17], Compilebench [27] and the Git-Benchmark [11, 12] are application benchmarks that implicitly involve some aging. These tools emulate user behavior by performing typical activities 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 [44] 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 of the first tools to point out the degradation of FS performance with age. Impressions on the other hand is a realistic FS image creator that focuses on several FS characteristics including file size and directory depth distributions along with file attributes and contents. These tools take reference from already old FSs in order to perform aging.



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

Figure 2: All three graphs reproduce experiments from the Btrfs ACM TOS publication [39] on aged FS instances. The *Paper* bars are normalized to the paper's Ext4 FS measurements, while the experiments we recreated are normalized to the unaged Ext4 performance on our hardware. Figures 2a and 2b show aging experiments performed on a HDD using the varmail and webserver profiles respectively. 2a shows modest slowdown (30% in Btrfs, 13% in Ext4 and 9% in Xfs), but preserves the rank ordering published in the paper. 2b disrupts published rank ordering and displays slowdowns of 17% in Btrfs and Ext4 and 12% in Xfs. Figure 2c shows effects of aging on SSD for the fileserver profile. Here too the published rank ordering is not preserved, but we observe massive slowdowns of 65% in Btrfs and 72% in Ext4, Xfs.

File System	Publication	Needed Aging?	Performed Aging?
yFS [55]	FAST 2003	Yes	Yes
Nilfs2 [21]	SIGOPS 2006	Yes	No
TFS [9]	FAST 2007	Yes	Yes
Data Domain Dedup FS [56]	FAST 2008	Yes	Yes
Panasas Parallel FS [51]	FAST 2008	Yes	Yes
CA-NFS [5]	FAST 2009	Yes	No
HYDRAStor [46]	FAST 2010	Yes	No
DFS [19]	FAST 2010	Yes	No
SFS [34]	FAST 2012	Yes	Yes
BlueSky [47]	FAST 2012	Maybe	No
ZZFS [29]	FAST 2012	Maybe	No
Nested FS in Virt. Env. [23]	FAST 2012	Yes	No
Btrfs [39]	ACM TOS 2013	Yes	No
ReconFS [26]	FAST 2014	Yes	No
F2fs [24]	FAST 2015	Yes	No
App. Managed Flash [25]	FAST 2016	Maybe	No
NOVA [52]	FAST 2016	Yes	No
CFFS [54]	FAST 2016	Maybe	Yes
BetrFS [11, 18, 53]	FAST 2015, 2016	Yes	Yes in [11]
Strata [22]	SOSP 2017	Yes	No

Table 1: A subset of major FS publications and whether their paper reports aging experiments. The *Needed Aging?* column is our understanding of whether aging could have affected published results, i.e. was aging (or commentary about it) necessary as a part of benchmarking. The *Performed Aging?* column refers to whether any aging-like experiment was performed. Our analysis reveals two FSs - yFS [55], TFS [9] performed long-running aging experiments; Data Domain FS [56] and Panasas FS [51] had production data (and therefore had seen aging in the field), SFS [34] ran a workload twice the size of the disk and CFFS [54] ran a large trace for aging. BetrFS [11] aged using the Git-benchmark (see § 3), which highlighted file fragmentation caused by age, but induced limited free space fragmentation. The remaining 13 papers do not discuss aging or its effects on their FSs.

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 for several reasons, such as uncovering performance deterioration with use, stressing the robustness of the software, and identifying fault tolerance and scalability issues. This section discusses four aspects of aging that current approaches insufficiently satisfy.

Free space fragmentation. State-of-the-art FS aging tools either replay a trace of FS commands or run scripts of important applications. Smith's aging tool [44] 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 FS (FFS) [31]. Impressions matches an impressive number of aged metrics, but is focused on generating realistic FS content, not FS layout. These tools only target file fragmentation using a metric called *layout score*, that measures the disk contiguity of files after aging.

Git-benchmark [11, 12] is a recently published aging benchmark that ages the FS by cloning a git repository, repeatedly patching code files (via git pulls) and finally grepping for random strings in the patched repository. In this study as well, the authors only measure file fragmentation by extending the layout score (which they call the *dynamic layout score*) which additionally accounts for the contiguity in the FS access pattern for a file.

We ran the Git-benchmark from [12] on a 20GB Ext4 partition and observed a $> 7\times$ 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 FS data structure. Since the capacity utilization at the end of 3000 git pulls was only 4%, we measured the free space fragmentation and observed that $> 75\%$ of the free space extents were between 1-2GB. Thus, although the Git-benchmark successfully caused some file fragmentation, it caused little of the free space fragmentation prevalent in aged FSs.

To the best of our knowledge, the above mentioned tools are the only general purpose FS aging tools; and none of them produce the required free space fragmentation which is an inevitable consequence of aging. Geriatrix's fills this void by inducing adequate free space fragmentation, proof of which is shown in § 5 and whose effect in aging SSDs has been exemplified in Figure 1.

Device aging. SSDs contain a complex translation layer in the device firmware called a flash translation layer (FTL). FTLs are the reason that an SSD can act as a drop-in replacement for an HDD despite having entirely different hardware. FTLs primarily perform the tasks of address remapping, garbage collection and wear leveling. SSD device characteristics force the FTL to operate very similarly to a complex log-structured FS [41]. With time, the increased garbage collection work (interfering with the foreground work) combined with fragmented FTL mapping tables can hurt performance. Thus, in the case of an SSD, two systems are aging simultaneously, the FTL and the FS that the SSD has been formatted with. Since FTLs are proprietary, users typically have no insight into how well (or poorly) an FTL has aged.

Shingled magnetic recording (SMR) is a new hard drive architecture wherein adjacent tracks on a HDD are partially overlapped to increase the number of tracks on the disk, thus increasing the disk capacity. This technology was an answer to the traditional HDDs having surpassed the superparamagnetic effect [45], which disallows increasing sectors-per-track in order to achieve larger disk capacities. Commercially available SMR HDDs have a firmware different from, but as complicated as the FTL. Aghayev et al. [1] showed that the interference of the firmware while performing foreground tasks caused high performance fluctuations. One of the key aspects of a firmware driven SMR disk is the existence of a persistent cache at the center of the drive. Suppose we are benchmarking a fresh FS on a SMR drive, we might conceivably never hit the persistent cache limit (which is impossible as the drive actually ages), thus circumventing the large read-modify-write cycles that the firmware would have performed leading to low throughput.

The churn that the FS and the device endure while Geriatrix attempts to age according to an aging profile forces device aging as well, providing a more realistic aging effect.

Aging write-optimized FSs. Write-optimized FSs focus on expediting writes at the cost of possibly slower reads. Log-structured FSs [41] are a classic example of a write-optimized FS. Moreover, most modern storage devices operate as log-structured FSs internally, as mentioned in Section 3. In this architecture, every file rewrite causes file fragmentation because in-place updates are disallowed. Therefore, the true impact of aging a write-optimized FS is usually noticed when garbage collection starts interfering with foreground activity, a well studied problem also known as segment cleaning [41, 6, 28, 48]. These FSs are hard to age since they need to be forced into frequent garbage collection which is only possible at high space utilization and with significant free space fragmentation. Geriatrix fulfills these two requirements allowing for effective aging of write-optimized FSs.

Aging as a stress tester. An effective use of an aging exercise could be in the form of a stress tester. The high churn expected to be exercised by an aging tool can expose design flaws like overflows / underflows, data structure inefficiencies, concurrency and consistency issues among others. Geriatrix produces orders of magnitude more churn than the state of the art aging tools present today, rewriting data amounting to several times the specified FS image size. Thus, Geriatrix is as much a FS stress tester as it is an aging tool.

4 Geriatrix design and implementation

Geriatrix exercises a non-aged FS to match an *aging profile* which is provided as input, by performing a sequence of file create and delete operations. The profile contents are inspired by a combination of the usual parameters that affect a file's on-disk layout and which are easily obtainable from an aged instance of a FS using a single metadata tree walk. A Geriatrix aging profile comprises of:

- **FS fullness (bytes, %):** Partition 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.
- **Relative age distribution (n, %; m %; ...):** A histogram of relative file ages, $n < m < ...$, where younger files, in the first histogram bin make up the first % of all files in the aged FS image, and so on. More specifically, we extract create timestamps of files from an existing old FS snapshot following which we sort, enumerate and bin files into relative age buckets. These buckets approximate the age of each file relative to the other files, decoupling them from the absolute time of their creation (thus making their age unitless).

At a high level, Geriatrix aging proceeds in two distinct phases.

1. A ***rapid aging*** phase in which files are only created (one file creation per time instant or *tick*), to rapidly achieve the fullness target. Since this phase does not perform any deletions, there is no fragmentation induced in rapid aging. It merely achieves the required fullness while ensuring that the file size and directory depth distributions are met.
2. A ***stable aging*** phase in which each operation (one operation per tick) is either a file creation or a deletion based on a fair coin toss. This phase is designed to fit the relative age distribution while maintaining all other parameters. The roughly equal number of creations and deletions are necessary to maintain the fullness target, and in some sense mimic the steady-state operation of a FS operated at a certain fullness.

We now discuss how we ensure that the distributions of the file system being aged match the target distributions at the end of a Geriatrix run. We assume that all input distributions are mutually independent. Thus, we can easily achieve the size and directory depth distributions by drawing a size and directory depth value from their respective distributions and creating or deleting a file corresponding to those values. Note that rapid and stable aging phases continuously strive to maintain both, size and directory depth distributions.

However, achieving the target age distribution is harder. Note that the relative age of a file at the end of a Geriatrix run is the ratio of the number of ticks (or operations) that have passed since the file was created to the total number of ticks (say T) in that run. Thus, without knowing T, it is impossible to determine the final relative age bucket of any file. However, to direct the algorithm towards the target age distribution, it is necessary to know the final relative age bucket and thus, T. We overcome this by theoretically estimating a sufficiently large T within which it is possible to perform create / delete operations that achieve the target age distribution. Then, during the run, we use our estimate of T to compute the index of the final relative age bucket of any file; based on this, we perform clever deletions to ensure that the target age distribution is achieved. Appendix A provides this estimate of T and shows that when we stop the algorithm at T, it has indeed converged to the target age distribution.

Geriatrix has a repository of eight built-in file system aging profiles, most of which are from long-running file system and metadata publications [3, 13, 49, 33] to assist practitioners in conducting file system aging experiments. Table 3 provides a description of all the profiles along with the age of oldest file in every profile. We also indicate the wall-clock time it took to age these profiles in a ramdisk along with the total workload size generated during aging. Finally, we also show the empirical proof of our relative age convergence theorem via the perfect convergence (a root mean-squared error of $<$0.01%) achieved on the relative age distribution graphs for each aging profile (complete overlap in the graphs in Table 3).

The aging tool is a C++ program (built using the Boost library) designed to run on UNIX platforms. It has the ability to age both POSIX and non-POSIX FSs.

- **Reducing setup complexity:** Geriatrix is profile driven with eight built-in aging profiles. We also provide a repository of popular Linux FSs aged using the built-in profiles for standardized comparison.
- **Parallel aging:** Geriatrix has a configurable thread pool that exploits multi-threading in file systems to expedite aging substantially.
- **Reproducibility:** A user-defined seed governs all the randomness in Geriatrix, thus allowing every single-threaded Geriatrix execution to be exactly reproducible. For multi-threaded executions, the operating system scheduler may interleave threads differently resulting in different execution patterns across runs.
- **Rollback utility:** Aging experiments can take a prohibitively long time. Once a FS image has been aged, taking a snapshot of the image to be able to restore the same image for multiple tests is usually faster than re-aging. This requires a whole disk overwrite, which on today's multi-TB 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 [7], we monitor the blocks that were modified during benchmark execution and effectively "undo" the perturbation caused by benchmarking by overwriting the dirtied blocks from the static snapshot 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 FS 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. the max number of disk overwrites for aging

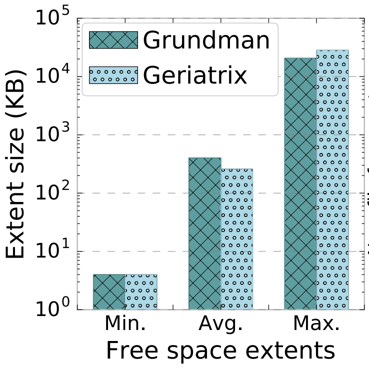
  Once any stopping condition is met, Geriatrix stops and displays the values of all three stopping conditions. The user can choose to accept the aging performed, or revise the condition(s) and resume aging.

5 Evaluation of Geriatrix as an aging tool

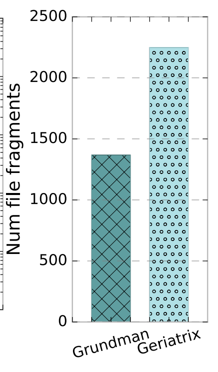
We evaluate the fidelity of Geriatrix's aging by comparing the file and free space fragmentation it induces on a fresh Ext4 partition to that of the source file system for the selected built-in profile. We do this comparison for two Geriatrix profiles: Grundman (extracted from a nine year old 90GB FS with approximately 90% fullness) and Dabre (extracted from a one year old 20GB FS with approximately 80% fullness). Despite being designed to only match externally visible measures of a FS, Geriatrix induces appropriate free space and file fragmentation by exercising the FS extensively.

We measure the distribution of free space extents using the *e2freefrag* utility. Figure 3 shows results for five file systems: the original Grundman FS image (aged naturally over 9 years), a fresh FS with no aging, a fresh FS aged by Geriatrix using the Grundman profile, a fresh FS

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



(a) Extents



(b) File Fragmentation

Figure 4: 4a 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 Grundman aging profile. 4b shows the number of fragments allocated as a result of a 2GB file being copied to both FS images.

aged by Impressions using the same file size distribution, and a fresh FS with the Grundman image files copied to it.

The primary takeaway is that the Geriatrix-aged FS matches the original Grundman FS closely, which can be seen by comparing the colorful bars, while the other three (gray bars) do not. As expected, the freshly formatted Ext4 has very large free space extents, mostly between 1-2GB. Grundman and the Geriatrix-aged FS have much smaller extents and more spread out free space extent distributions ranging from 4KB to 32MB. The "Copied" and Impressions-aged FSs are similar to the freshly-formatted file system, with low free space fragmentation. The comparison using the Dabre profile (graph omitted due to space constraints) shows very similar results.

Figure 4a compares the minimum, average and maximum free space extent sizes for Grundman and the Geriatrix-aged FSs. Both have the same smallest free space extent of 4KB. The average free space extent size of naturally-aged Grundman is only 144KB larger than its Geriatrix counterpart, while the largest free space extent is only 7.2MB smaller. These numbers are very close considering the total partition size of 90GB.

Figure 4b measures the fragmentation of a new 3GB file copied to each of the naturally-aged Grundman image and the Geriatrix-aged FS, using the *filefrag* utility. The image aged by Geriatrix splits the file into 2250 fragments while the naturally-aged FS only splits it into 1368 fragments. Despite Geriatrix over-splitting the file, its aging is two orders of magnitude higher than the number of fragments created writing 3GB to a freshly formatted Ext4.

Since Geriatrix refrains from taking shortcuts, and performs millions of operations before declaring a FS *aged*, it closely approximates the FS state caused by natural aging. The Geriatrix aging experiment reported in Figures 3 and 4 took approximately 420 minutes. Recreating the fragmentation naturally occurring in nine years with only 420 minutes of aging is an acceleration of $>11000\times$.

6 How Geriatrix changes conclusions

To highlight the impact of aging, we recreated experiments from Btrfs [39], F2fs [24] and NOVA [52] publications on unaged and aged FS instances. We also produced aged instances of Ext4 (used for comparison across all three papers) and Xfs (used for comparison in the Btrfs and NOVA papers).

Experimental setup. We performed all our experiments on an Emulab PRObE cluster [15]. The hardware used and the setups for experiments is described in Table 2. For fairer comparison, we matched the memory and the number of cores in our benchmark when recreating expts. from the Btrfs [39] and F2fs [24] publications.

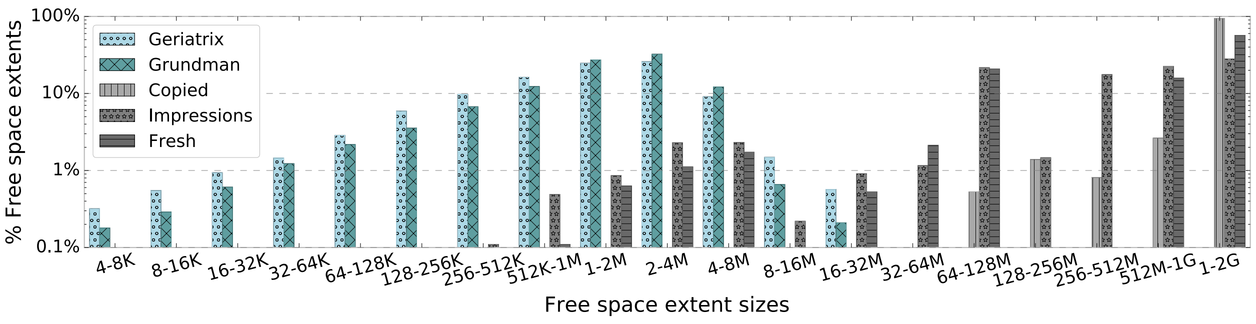


Figure 3: Free space fragmentation comparison of an actual old Ext4 FS image (Grundman) with Geriatrix driven by the Grundman profile, Impressions driven by the Grundman profile's file size distribution, a partition with the contents of the original Grundman image copied over and a freshly formatted Ext4 partition. Other than the freshly formatted image, all other FS images are approximately 90% full. Geriatrix induces free space fragmentation very similar to the naturally aged Grundman image with no large free space extents, hence causing appropriate free space fragmentation.

Paper	Disk	RAM	CPU (cores)	Linux (Kernel Version)
Btrfs [39]	500GB HDD (WDC WD5000YS-01MPB0)	2GB	Intel Xeon E7 (8)	Ubuntu 14.04 LTS (3.13.0-33)
	64GB SSD (Crucial M4-CT064M4SSD2)	2GB	AMD Opteron (8)	Ubuntu 14.04 LTS (3.13.0-33, 4.4.0-31)
F2fs [24]	64GB SSD (Crucial M4-CT064M4SSD2)	4GB	Intel Core i7 (4)	Ubuntu 14.04 LTS (4.4.0-31)
	120GB SSD (ADATA SSD S510)	4GB	Intel Core i7 (4)	Ubuntu 14.04 LTS (4.4.0-31)
NOVA [52]	64GB NVM (Emulated in DRAM)	8GB	AMD Opteron (8)	Ubuntu 14.04 LTS (4.13)

Table 2: Experimental Configuration.

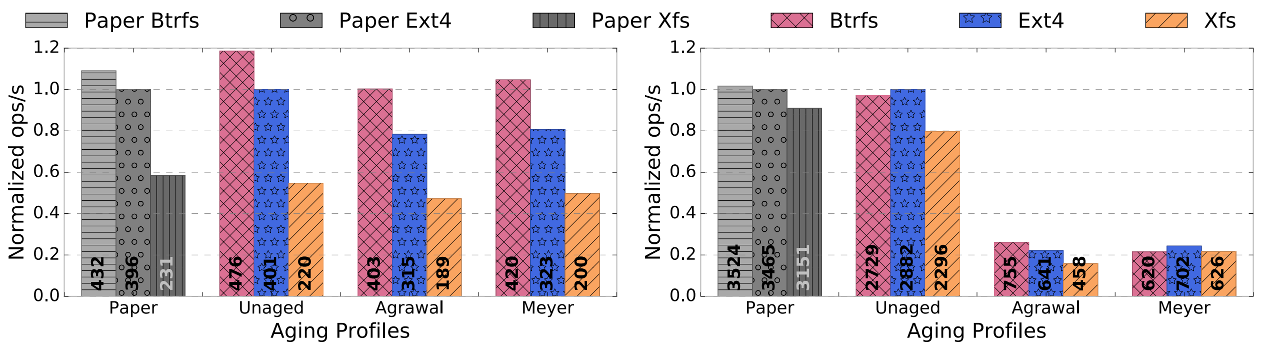
We used four of Geriatrix's built-in profiles in our experiments: Agrawal [3], Meyer [33], Dabre and Pramod. We performed aging in memory and captured the resulting aged images. We consciously decided to not age the FSs on the device as we wanted to prevent device aging from affecting the FS aging. Prior to each benchmark run we copied the corresponding aged image onto a disk (using *dd* to the raw device) and mounted the FS on the aged image. All FSs were mounted using default mount options.

The Filebench benchmark [30] 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 to capture 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 code potentially newer than the one used for publication, exact reproduction of paper results even for unaged instances of FSs 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 FS performance numbers are normalized to unaged Ext4 performance on the same hardware. We chose Ext4 because it is the default FS rolled out with most Linux distributions today. All HDD experiments were conducted using 100GB aged images with a 80% capacity utilization target being replayed on a 100GB partition of a 500GB HDD.

HDD experiments. Figure 2a in Section 1 compares the performance of Btrfs, Ext4 and Xfs on an HDD for the Filebench varmail workload, representing a mail server workload of tiny (≈16KB) file operations. After aging using the Meyer profile, we see 9-30% slowdowns, with Btrfs being most affected by aging, followed by Ext4 and then Xfs.

Figure 2b compares the same FSs using the webserver workload. The webserver workload is highly multi-threaded and again, operates on tiny files, although it avoids issuing expensive fsync operations and mimics webservers that have to perform more whole-file reads and log appends. Since FSs are usually more sensitive to small file operations, it is perhaps understandably harder to reproduce published results; indeed, 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 ranking, 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 exemplifies the role of Geriatrix as a stress testing tool.

Figure 5a compares the FSs on the fileserver workload,



(a) Btrfs HDD Fileserver

(b) Btrfs SSD Fileserver

Figure 5: Both graphs reproduce Filebench fileserver experiments from the Btrfs ACM TOS publication [39] on aged FS instances on a HDD (Figure 5a) and a SSD (Figure 5b). 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 FSs. SSD experiments in 5b were performed using the Linux kernel version 3.13.0-33

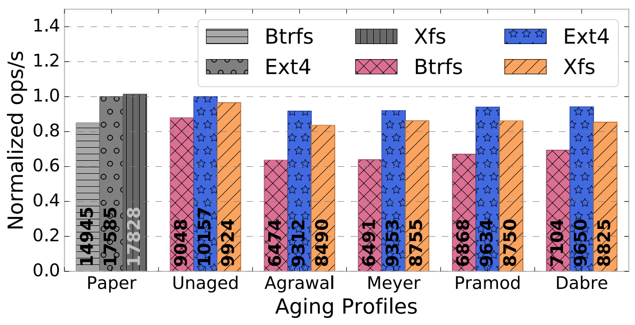


Figure 6: Filbench webserver recreations from the Btrfs paper [33] on SSD. Btrfs is most affected by aging with its performance dropping by 30%. Ext4 and Xfs performance drops by a maximum of 10% and 15% respectively.

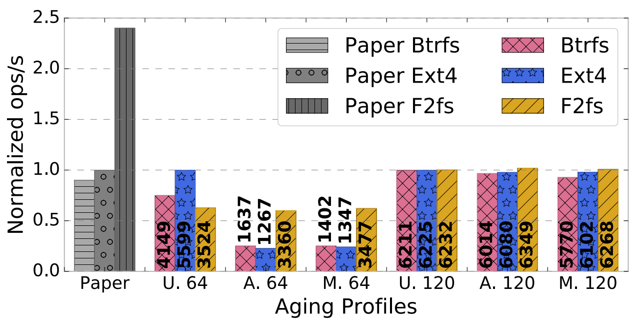


Figure 7: Recreation of the Filebench fileserver benchmark from the F2fs paper [24] on two SSDs - 64GB (labeled 64) and 120GB (labeled 120). Performance on 64 drops significantly after aging, especially for Btrfs and Ext4 resulting in a graph that looks similar to published results. Refer Section 6 for detailed explanation. 120 seems unaffected by aging, thus highlighting highly varied performance across different SSDs

which consists of relatively larger file writes and reads (≈ 128KB) compared to thousands of tiny file operations in webserver and varmail. We observe that unaged Btrfs is 10% faster in our setup, unaged Ext4 is marginally better while unaged Xfs is about 5% slower, keeping performance similar to published results with slightly increased performance gaps between the FSs. After aging using the Agrawal profile, we observe a 10-22% performance drop with Ext4 being the most affected FS.

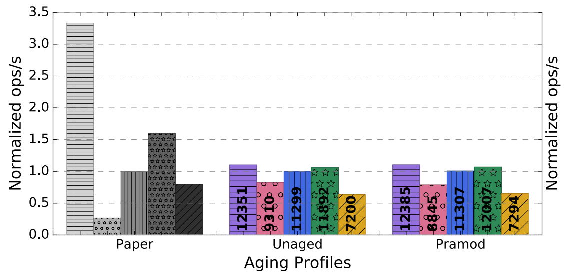
SSD experiments. The SSD experiments were conducted on a 64GB SSD with a 59GB aged FS image with a 70% fullness target. The reason for choosing 70% was to allow the benchmarking workload to fit after aging. SSDs are available in a variety of product price-point classes and have highly variable performance making reproduction of SSD results on different hardware unlikely. This is evident from figures 2c and 5b which show a 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 ranged from being the best in aged reproduction using the Dabre (Figure 2c) and Agrawal profiles to being the worst in the aged reproduction using the Meyer profile. Apart from the rank ordering, we observed massive post-aging slowdowns on SSDs from 65-80%. As explained in Sections 1 and 3, we attribute this performance drop to SSD device aging along with FS aging, with both effects being exaggerated by the free space fragmentation caused by Geriatrix.

The webserver results shown in Figure 6 also show significant slowdowns, but are not so dramatic. Btrfs appears to be the most affected by aging, showing up to a 30% performance drop, while Xfs and Ext4 degrade by $\leq$15% and $\leq$10%, respectively.

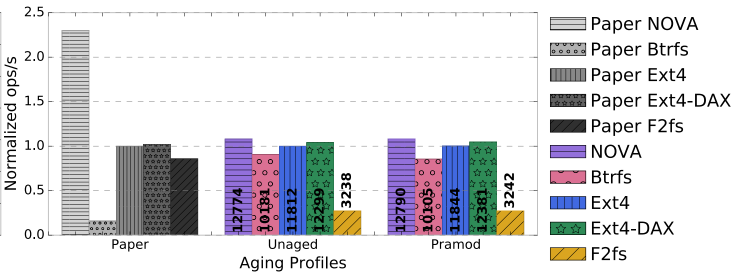
It was typical of SSDs from a few years ago to not be able to sustain more than 2 minutes of continuous writing before performing inline cleaning [37]. Our benchmarking technique involves writing an aged image on almost the entire surface of the SSD, performing a 10 minute benchmark run followed by unmounting the FS 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 widely across devices.

Figure 7 is the recreation of F2fs [24] 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 64GB Crucial SSD with 59GB aged FS images (bars labeled 64) and a 120GB ADATA SSD with 100GB aged FS images (bars labeled 120). Ext4 is the winning FS when comparing unaged FS instances on 64GB SSD, and Xfs is marginally better on the 120GB SSD, while published results report F2fs performance was 2.4× that of Ext4. Aging on 64GB SSD shows interesting behavior as the performance of all three FSs drops (61-67% for Btrfs, 76-78% for Ext4 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 FSs are aged. In contrast, the 100GB FSs on the 120GB SSD age much more gracefully with only Btrfs showing as much as 7% performance penalty after aging. This again suggests that SSDs themselves age in different (and non-trivial) ways along with the FSs running on them.

Emulated NVM experiments. We also perform aging experiments on NOVA [52] – a log-structured FS 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. A 64GB partition similar to the SSD experiments was aged with 70% utilization. The NOVA paper performed experiments with 64GB persis-

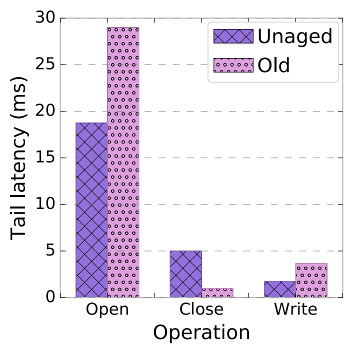


(a) NOVA Fileserver

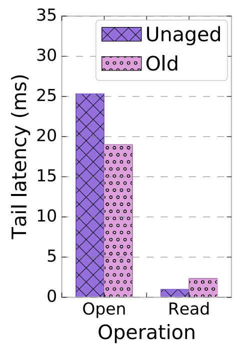


(b) NOVA Varmail

Figure 8: Both graphs reproduce experiments from the NOVA FAST 2016 publication [52] on aged FS instances using emulated non-volatile memory. All FSs age gracefully with Btrfs seeing the largest performance hit of 5% on the fileserver workload (Figure 8a) and 6% on the varmail workload (Figure 8b). Graphs from the paper are for reference and in this case cannot be compared to our reproductions because of different hardware and configuration.



(a) NOVA Fileserver Latency



(b) NOVA Varmail Latency

Figure 9: 9a shows the latency of the slowest operations for the fileserver workload. Aged NOVA slows down by upto 60% on file opens and upto 2× on file writes. In 9b we see slowdowns of upto 2.3× on reads of the slowest file in the varmail workload. 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.

tent memory devoted to NOVA and 32GB DRAM for the rest of the system. The experiment dataset size was made slightly larger than DRAM (more than 32GB) forcing NVM device interaction during an experiment run. However, that meant that the authors used more than half the 64GB NVM device for their benchmark run. Generally, a <50% utilized FS would be considered too underutilized for running a realistic post-aging benchmark. Hence, we kept utilization at 70%, reduced the DRAM size to 8GB and exercised a workload of more than 8GB to still ensure that the benchmarking workload was larger than the system memory. Furthermore, the authors performed their experiments on special Intel NVM hardware. Thus, we discourage the direct comparison of performance in Figures 8a and 8b with the NOVA paper results.

Figure 8a compares NOVA's performance on the fileserver workload before and after aging with Btrfs, 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 FSs before and after aging. The same is true in the case of the varmail workload shown in Figure 8b where Btrfs is the most affected FS with a slowdown of approximately 6%.

Although aging does not affect throughput, its effect is seen in the tail latencies. Figure 9a shows the highest latency encountered by Filebench categorized by operation. The slowest file open in the fileserver workload was 60% slower after aging. Writing the slowest file also took twice as long compared to a freshly formatted NOVA image. Surprisingly, closing the slowest file was 5× faster after aging. In the varmail workload, the slowest aged read was 2.3× slower than the slowest unaged read. In contrast, the opening of the slowest aged file was approximately 25% faster. Both these observations can be attributed to the log-structured design of NOVA, as log-structured FSs 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 FSs do not show significant reduction in throughput after aging. An aging exercise for these FSs is mainly about exposing inefficiencies in FS implementations that usually get hidden behind massive device latencies.

7 Conclusion

The Geriatrix aging tool creates representative fragmentation of both files and free space. Unlike with HDDs, file system performance on SSDs is greatly impacted by free space fragmentation that has been largely absent from prior aging approaches. Being released as open-source, Geriatrix will enable file system benchmarking to include aging relevant to modern storage.

8 Acknowledgements

We thank Keith Smith, Rob Johnson and the anonymous reviewers for their feedback and guidance. We also thank the members and companies of the PDL Consortium (Alibaba, Broadcom, Dell EMC, Facebook, Google, Hewlett-Packard Enterprise, Hitachi, Intel, IBM, Micron, Microsoft Research, MongoDB, NetApp, Oracle, Salesforce, Samsung, Seagate, Toshiba, Two Sigma, Veritas and Western Digital) for their interest, insights, feedback, and support.

A Proof of convergence of the age distr.

Assume we need to age a FS that has K files at the end of the rapid aging phase, and continues to have approximately K files throughout aging (i.e. one Geriatrix run). Let T be the total number of operations (each corresponding to a tick) performed in one Geriatrix run. Let the target relative age distribution be specified by binning the T ticks into B buckets, with the oldest bucket indexed as $b = B$ and the youngest bucket as $b = 1$ (i.e., the files created at the end of the run will fall in $b = 1$). Let s_b be the relative size of the age bucket b, i.e., files created in the first Ts_B operations will fall in relative age bucket B, and those in the next Ts_{B-1} operations in bucket $B-1$ and so on). Let g_b be the relative number of files in bucket b according to the target distribution i.e., there must be approximately Kg_b files in bucket b at the end of a run. Then we can predict the number of operations required to achieve convergence as follows:

Theorem A.1. *After*

$$T = max \begin{cases} \frac{2Kg_b}{s_b}, \forall b < B \\ \\ \frac{K}{s_B} \end{cases} .$$

the Geriatrix run would have converged to the target age distribution i.e., the number of files in age bucket b would equal Kg_b (approximately).

Proof. First, let us examine how the rapid aging phase, which consists of the first K file creations, affects the age distribution of the system. Since the oldest bucket corresponds to the first Ts_B operations, and $Ts_B \geq K$, all the files created in the rapid aging phase end up falling in bucket B.

Next, we will examine how the stable aging phase can achieve the target age distribution. Let O_b be the total number of stable aging operations that correspond to the ticks corresponding to bucket b. For the oldest bucket, $O_B = Ts_B - K$ and in the other buckets $O_b = Ts_b$. Further, let C_b and D_b respectively be the number of stable aging creation and deletion operations performed corresponding to bucket b. 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.

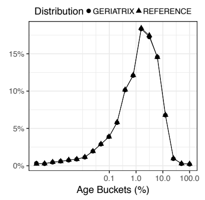
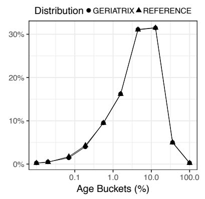
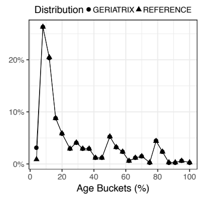
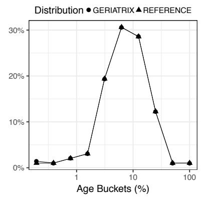
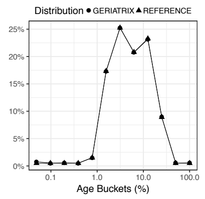
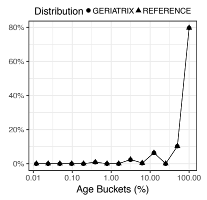
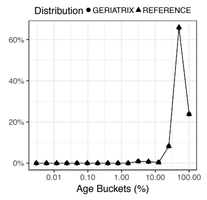
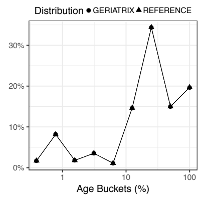
Now note that the C_b creations corresponding to bucket b will create files which will all fall in bucket b. On the other hand, the D_b deletions corresponding to bucket b may be performed either on files in bucket b or on any pre-existing file in an older bucket b' such that $b' > b$. We will show that we can distribute these deletions in such a manner that the target age distribution of Kg_b files in bucket b can be achieved for every b.

First, observe that in the youngest bucket $b = 1$ we create exactly $C_1 = Ts_1/2$ files. Now, since $T \geq 2Kg_1/s_1$, $C_1 \geq Kg_1$. Thus, there would be an excess of $C_1 - Kg_1 = Ts_1/2 - Kg_1$ file creations in this bucket which need to be deleted. Since we have $D_1 = Ts_1/2$ deletion operations available for this bucket, we can use $Ts_1/2 - Kg_1$ of them to remove these excess files and achieve the target for this bucket, and use the excess Kg_1 delete operations for the older buckets.

We will now use induction to show that we can similarly achieve the target number of files for the older buckets (except the oldest which we will handle separately). Specifically, assume that for operating on the files in bucket b where $b \neq B$, we have $K\sum_{b'=1}^{b-1} g_{b'}$ excess delete operations available from the operations corresponding to the younger buckets (this is equal to zero for $b = 1$). Next, recall that have $C_b = Ts_b/2$ file creations in this bucket. But since $T \geq 2Kg_b/s_b$, $C_b \geq Kg_b$. Thus, there would be an excess of $Ts_b/2 - Kg_b$ files in this bucket that need to be deleted. However, we have $Ts_b/2$ deletion operations available from the ticks corresponding to this bucket and a further $K\sum_{b'=1}^{b-1} g_{b'}$ excess deletion operations available from younger buckets. Hence, we can delete these excess $Ts_b/2 - Kg_b$ files to achieve the target of Kg_b files in this bucket. Then, we will have the remaining $Kg_b + K\sum_{b'=1}^{b-1} g_{b'} = K\sum_{b'=1}^{b} g_{b'}$ deletion operations as excess which can be used for the older buckets, thus satisfying the induction hypothesis.

Now, for the oldest bucket we will have $K\sum_{b'=1}^{B-1} g_{b'} = K(1 - g_B)$ excess deletion operations available (since $\sum_{b'=1}^{B} g'_b = 1$) from the younger buckets, besides $Ts_B/2$ deletions corresponding to this bucket. At the same time, we have created K files in this bucket in the rapid aging phase and $Ts_B/2$ files in the stable aging face i.e., $K + Ts_B/2$ files in total. Hence, we will have an excess of $K + Ts_B/2 - Kg_B = K(1 - g_B) + Ts_B/2$ files, which happens to be equal to the total number of deletions available. Hence, we can achieve the target file number in this bucket *while using up all the available operations*. Thus, we use exactly T operations to achieve the target age distribution.

□

Profile	*Description*	*Age (yrs)*	*Duration (min)*	*Overwrites (50 GB)*	*Age Distribution*
Douceur[*]	Referenced from a study of FS contents by Douceur et al. [13] in 1998. It captures an aggregate analysis of over 10000 commercial PCs running Microsoft Windows.	4	NA	22422 (1 GB)	
Agrawal	Referenced from a metadata study of Windows desktop FSs from Microsoft in 2004 [3]. Most computers ran NTFS (80%) along with FAT32 (15%) and FAT (5%).	14	466	253	
Meyer	Referenced from a deduplication study conducted on 857 Windows desktop computers at Microsoft [33]. Snapshots of the FSs were taken in 2009.	2	78	159	
Wang-OS	Referenced from an HPC FS environment study [49] performed on NetApp's WAFL [16] installations at CMU's Parallel Data Lab (an educational cluster setup for systems' research) in 2011.	22	231	34	
Wang-LANL	Referenced from the same study as Wang-OS [49] from Panasas FS [51] installations at Los Alamos National Lab (LANL).	11	146	28	
Dabre	Captured in 2017 from the root partition of a colleague's laptop running Ext4.	1	91	4042	
Pramod	Captured in 2017 from the root partition of a colleague's laptop running Ext4.	3.75	27	17	
Grundman	Captured in 2018 from the home partition of a colleague's laptop running Ext4.	8.75	142	2388	

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

Table 3: 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 50GB Xfs partition in memory with Geriatrix for every profile, and the number of disk overwrites (workload) performed.

References

[1] Aghayev, A., Shafaei, M., and Desnoyers, P. Skylighta window on shingled disk operation. *ACM Transactions on Storage (TOS)* (2015).

[2] Agrawal, N., Arpaci-Dusseau, A. C., and Arpaci-Dusseau, R. H. Generating realistic impressions for file-system benchmarking. *ACM Transactions on Computer Systems (TOCS)* (2009).

[3] Agrawal, N., Bolosky, W. J., Douceur, J. R., and Lorch, J. R. A five-year study of file-system metadata. *ACM Transactions on Storage (TOS)* (2007).

[4] Aranya, A., Wright, C. P., and Zadok, E. Tracefs: A file system to trace them all. In *USENIX File and Storage Technologies (FAST)* (2004).

[5] Batsakis, A., Burns, R., Kanevsky, A., Lentini, J., and Talpey, T. Ca-nfs: A congestion-aware network file system. *ACM Transactions on Storage (TOS)* (2009).

[6] Blackwell, T., Harris, J., and Seltzer, M. I. Heuristic cleaning algorithms in log-structured file systems. In *USENIX Annual Technical Conference (ATC)* (1995).

[7] Brunelle, A. D. Block i/o layer tracing: blktrace. *HP, Gelato-Cupertino, CA, USA* (2006).

[8] Cao, P., Felten, E. W., Karlin, A. R., and Li, K. A study of integrated prefetching and caching strategies. *ACM SIGMETRICS Performance Evaluation Review* (1995).

[9] Cipar, J., Corner, M. D., and Berger, E. D. Tfs: A transparent file system for contributory storage. In *USENIX File and Storage Technologies (FAST)* (2007).

[10] Comer, D. Ubiquitous b-tree. *ACM Computing Surveys (CSUR)* (1979).

[11] Conway, A., Bakshi, A., Jiao, Y., Jannen, W., Zhan, Y., Yuan, J., Bender, M. A., Johnson, R., Kuszmaul, B. C., Porter, D. E., et al. File systems fated for senescence? nonsense, says science! In *USENIX File and Storage Technologies (FAST)* (2017).

[12] Conway, A., Bakshi, A., Jiao, Y., Zhan, Y., Bender, M. A., Jannen, W., Johnson, R., Kuszmaul, B. C., Porter, D. E., Yuan, J., et al. How to fragment your file system. *USENIX ;login:* (2017).

[13] Douceur, J. R., and Bolosky, W. J. A large-scale study of file-system contents. *ACM SIGMETRICS Performance Evaluation Review* (1999).

[14] Feiertag, R. J., and Organick, E. I. The multics input/output system. In *ACM Symposium on Operating Systems Principles (SOSP)* (1971).

[15] Gibson, G., Grider, G., Jacobson, A., and Lloyd, W. Probe: A thousand-node experimental cluster for computer systems research. *USENIX ;login:* (2013).

[16] Hitz, D., Lau, J., and Malcolm, M. A. File system design for an nfs file server appliance. In *USENIX Winter Conference* (1994).

[17] Howard, J. H., Kazar, M. L., Menees, S. G., Nichols, D. A., Satyanarayanan, M., Sidebotham, R. N., and West, M. J. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems (TOCS)* (1988).

[18] Jannen, W., Yuan, J., Zhan, Y., Akshintala, A., Esmet, J., Jiao, Y., Mittal, A., Pandey, P., Reddy, P., Walsh, L., et al. Betrfs: A right-optimized write-optimized file system. In *USENIX File and Storage Technologies (FAST)* (2015).

[19] Josephson, W. K., Bongo, L. A., Li, K., and Flynn, D. Dfs: A file system for virtualized flash storage. *ACM Transactions on Storage (TOS)* (2010).

[20] Joukov, N., Wong, T., and Zadok, E. Accurate and efficient replaying of file system traces. In *USENIX File and Storage Technologies (FAST)* (2005).

[21] Konishi, R., Amagai, Y., Sato, K., Hifumi, H., Kihara, S., and Moriai, S. The linux implementation of a log-structured file system. *ACM Operating Systems Review (SIGOPS)* (2006).

[22] Kwon, Y., Fingler, H., Hunt, T., Peter, S., Witchel, E., and Anderson, T. Strata: A cross media file system. In *ACM Symposium on Operating Systems Principles (SOSP)* (2017).

[23] Le, D., Huang, H., and Wang, H. Understanding performance implications of nested file systems in a virtualized environment. In *USENIX File and Storage Technologies (FAST)* (2012).

[24] Lee, C., Sim, D., Hwang, J., and Cho, S. F2fs: A new file system for flash storage. In *USENIX File and Storage Technologies (FAST)* (2015).

[25] Lee, S., Liu, M., Jun, S., Xu, S., Kim, J., et al. Application-managed flash. In *USENIX File and Storage Technologies (FAST)* (2016).

[26] Lu, Y., Shu, J., and Wang, W. Reconfs: A reconstructable file system on flash storage. In *USENIX File and Storage Technologies (FAST)* (2014).

[27] Mason, C. Compilebench. https://oss.oracle.com/ mason/compilebench.

[28] Matthews, J. N., Roselli, D., Costello, A. M., Wang, R. Y., and Anderson, T. E. Improving the performance of log-structured file systems with adaptive methods. In *ACM Symposium on Operating Systems Principles (SOSP)* (1997).

[29] Mazurek, M. L., Thereska, E., Gunawardena, D., Harper, R. H., and Scott, J. Zzfs: a hybrid device and cloud file system for spontaneous users. In *USENIX File and Storage Technologies (FAST)* (2012).

[30] McDougall, R., and Mauro, J. Filebench. http://www.nfsv4bat.org/Documents/nasconf/2004/filebench.pdf, 2005.

[31] McKusick, M. K., Joy, W. N., Leffler, S. J., and Fabry, R. S. A fast file system for unix. *ACM Transactions on Computer Systems (TOCS)* (1984).

[32] Mesnier, M. P., Wachs, M., Simbasivan, R. R., Lopez, J., Hendricks, J., Ganger, G. R., and O'hallaron, D. R. //trace: Parallel trace replay with approximate causal events. In *USENIX File and Storage Technologies (FAST)* (2007).

[33] Meyer, D. T., and Bolosky, W. J. A study of practical deduplication. *ACM Transactions on Storage (TOS)* (2012).

[34] Min, C., Kim, K., Cho, H., Lee, S.-W., and Eom, Y. I. Sfs: random write considered harmful in solid state drives. In *USENIX File and Storage Technologies (FAST)* (2012).

[35] Patterson, R. H., Gibson, G. A., Ginting, E., Stodolsky, D., and Zelenka, J. Informed prefetching and caching. In *ACM Symposium on Operating Systems Principles (SOSP)* (1995).

[36] Petrovic, S. The effects of age on file system performance. Master's thesis, Masaryk University, 5 2017.

[37] Polte, M., Simsa, J., and Gibson, G. Enabling enterprise solid state disks performance. *Workshop on Integrating Solid-state Memory into the Storage Hierarchy* (2009).

[38] Ritchie, O., and Thompson, K. The unix time-sharing system. *The Bell System Technical Journal* (1978).

[39] Rodeh, O., Bacik, J., and Mason, C. Btrfs: The linux b-tree filesystem. *ACM Transactions on Storage (TOS)* (2013).

[40] ROSELLI, D. S., LORCH, J. R., ANDERSON, T. E., ET AL. A comparison of file system workloads. In *USENIX Annual Technical Conference (ATC)* (2000).

[41] ROSENBLUM, M., AND OUSTERHOUT, J. K. The design and implementation of a log-structured file system. *ACM Transactions on Computer Systems (TOCS)* (1992).

[42] SELTZER, M., SMITH, K. A., BALAKRISHNAN, H., CHANG, J., MCMAINS, S., AND PADMANABHAN, V. File system logging versus clustering: A performance comparison. In *USENIX Technical Conference Proceedings (TCON)* (1995).

[43] SMITH, K., AND SELTZER, M. I. File layout and file system performance. *Tech. Rep. TR-35-94, Harvard University* (1994).

[44] SMITH, K. A., AND SELTZER, M. I. File system aging increasing the relevance of file system benchmarks. In *ACM SIGMETRICS Performance Evaluation Review* (1997).

[45] THOMPSON, D. A., AND BEST, J. S. The future of magnetic data storage techology. *IBM Journal of Research and Development* (2000).

[46] UNGUREANU, C., ATKIN, B., ARANYA, A., GOKHALE, S., RAGO, S., CALKOWSKI, G., DUBNICKI, C., AND BOHRA, A. Hydrafs: A high-throughput file system for the hydrastor content-addressable storage system. In *USENIX File and Storage Technologies (FAST)* (2010).

[47] VRABLE, M., SAVAGE, S., AND VOELKER, G. M. Bluesky: a cloud-backed file system for the enterprise. In *USENIX File and Storage Technologies (FAST)* (2012).

[48] WANG, J., AND HU, Y. Wolf-a novel reordering write buffer to boost the performance of log-structured file systems. In *USENIX File and Storage Technologies (FAST)* (2002).

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

[50] WEISS, Z., HARTER, T., ARPACI-DUSSEAU, A. C., AND ARPACI-DUSSEAU, R. H. Root: Replaying multithreaded traces with resource-oriented ordering. In *ACM Symposium on Operating Systems Principles (SOSP)* (2013).

[51] WELCH, B., UNANGST, M., ABBASI, Z., GIBSON, G. A., MUELLER, B., SMALL, J., ZELENKA, J., AND ZHOU, B. Scalable performance of the panasas parallel file system. In *USENIX File and Storage Technologies (FAST)* (2008).

[52] XU, J., AND SWANSON, S. Nova: a log-structured file system for hybrid volatile/non-volatile main memories. In *USENIX File and Storage Technologies (FAST)* (2016).

[53] YUAN, J., ZHAN, Y., JANNEN, W., PANDEY, P., AKSHINTALA, A., CHANDNANI, K., DEO, P., KASHEFF, Z., WALSH, L., BENDER, M., ET AL. Optimizing every operation in a write-optimized file system. In *USENIX File and Storage Technologies (FAST)* (2016).

[54] ZHANG, S., CATANESE, H., AND WANG, A. A.-I. The composite-file file system: decoupling the one-to-one mapping of files and metadata for better performance. In *USENIX File and Storage Technologies (FAST)* (2016).

[55] ZHANG, Z., AND GHOSE, K. yfs: A journaling file system design for handling large data sets with reduced seeking. In *USENIX File and Storage Technologies (FAST)* (2003), vol. 3, p. 2nd.

[56] ZHU, B., LI, K., AND PATTERSON, R. H. Avoiding the disk bottleneck in the data domain deduplication file system. In *USENIX File and Storage Technologies (FAST)* (2008).

[57] ZHU, N., CHEN, J., CHIUEH, T.-C., AND ELLARD, D. Tbbt: scalable and accurate trace replay for file server evaluation. In *ACM SIGMETRICS Performance Evaluation Review* (2005).