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

Saurabh Kadekodi, Vaishnavh Nagarajan, Garth A. Gibson

CMU-PDL-16-105
October 2016

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

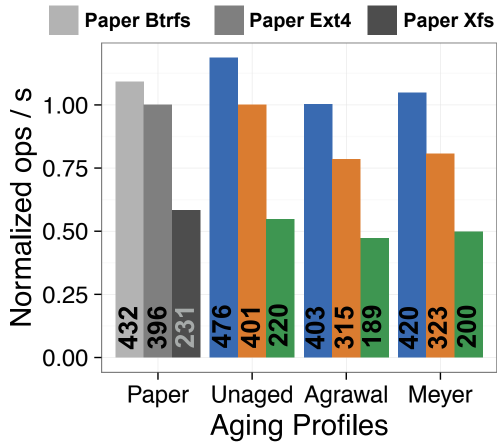
Abstract

File system aging has been advocated for thorough analysis of any design, but it is cumbersome and often bypassed. Our aging study re-evaluates published file systems after aging using the same benchmarks originally used. We see significant performance degradation on HDDs and SSDs. With performance of aged file systems on SSDs dropping by as much as 80% relative to the recreated results of prior papers, aging is even more necessary in the era of SSDs. Still more concerning, the rank ordering of compared file systems can change versus published results.

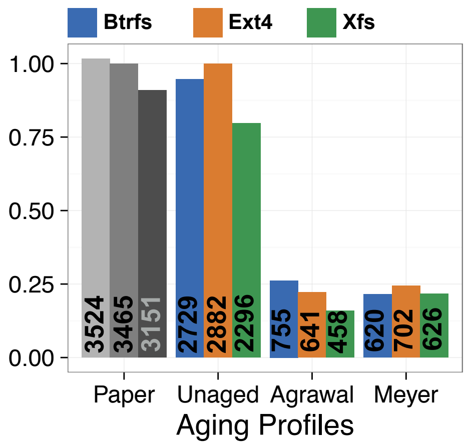
We offer Geriatrix, a simple-to-use aging suite with built-in aging profiles with the goal of making it easier to age, and harder to justify ignoring file system aging in storage research.

Acknowledgements: This research was supported in part by the DOE and Los Alamos National Laboratory, under con- tract number DE-AC52-06NA25396 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 - Broadcom Ltd., Citadel, Dell EMC, Facebook, Google, Hewlett-Packard Labs, Hitachi, Ltd., Intel Corporation, Microsoft Research, MongoDB, NetApp, Inc., Oracle Corporation, Samsung Information Systems America, Seagate Technology, Tintri, Two Sigma, Uber, Veritas and Western Digital.

Keywords: File System, Aging, Disrtribution Fitting, Modeling, Tool



(a)



(b)

Figure 1: Both graphs reproduce Filebench fileserver experiments from the Btrfs ACM TOS publication [31] on aged file system instances on a HDD (a) and a SSD (b). 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.

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 [30, 24, 10], data prefetching to assist reads [6, 28] and self-balancing data structures to contain search times [8] 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 [34, 33]. Hence, file system benchmarking practices should consider the effects of aging, an arugment raised almost twenty years ago by Smith and Seltzer [35], but one that continues to be ignored.

2 Review of Aging in Recent Papers

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 afer applying aging. Table 1 lists 19 papers we reviewed; 13 of 19 (68%) do not mention aging at all. For only 5 of these 19 papers was code readily available, from which NOVA [41] required special hardware, while with BetrFS [13, 42] and Nilfs2 [16] the code was too immature to sustain aging.

We were able to recreate experiments from the Btrfs ACM TOS 2013 [31] and F2fS USENIX FAST 2015 [18] publications. With these recreated experiments, we explored performance on freshly formatted file system images and on aged file system images to test if aging effects could be important. Figure 1 shows the recreated benchmark from the Btrfs paper on a both hard drive (HDD) and a solid state drive (SSD). Our methodology will be discussed in Section 8.

As Figure 1a shows, aging three file systems with two different aging patterns did not change the coarse grain results on a HDD; individual file systems experienced 10-30% slowdown and the rank ordering of which is fastest in unchanged.

However, using file systems mounted on a SSD, Figure 1b shows drammatic slowdowns after aging;

File System (Publication)	Aging Avoidable	Aged
yFS [44] (FAST ’03)	No	Yes
Nilfs2 [16] (SIGOPS ’06)	No	No
TFS [7] (FAST ’07)	No	Yes
Data Domain Dedup FS [45] (FAST ’08)	No	Yes
Panasas Parallel FS [40] (FAST ’08)	No	Yes
CA-NFS [4] (FAST ’09)	No	No
HYDRAStor [36] (FAST ’10)	No	No
DFS [14] (FAST ’10)	No	No
SFS [27] (FAST ’12)	No	Yes
BlueSky [37] (FAST ’12)	Maybe	No
ZZFS [22] (FAST ’12)	Maybe	No
Nested FS in Virt. Env. [17] (FAST ’12)	No	No
Btrfs [31] (ACM TOS ’13)	No	No
ReconFS [20] (FAST ’14)	No	No
F2fs [18] (FAST ’15)	No	No
App. Managed Flash [19] (FAST ’16)	Maybe	No
NOVA [41] (FAST ’16)	No	No
CFFS [43] (FAST ’16)	Maybe	Yes
BetrFS [13, 42] (FAST ’15, ’16)	No	No

Table 1: Two file systems - yFS [44], TFS [7] performed long-running aging experiments; Data Domain FS [45] and Panasas FS [40] had production data, SFS [27] ran a workload twice the size of the disk and CFFS [43] ran a large trace for aging. The remaining 13 papers do not discuss aging or its effects on their file systems.

as much as 80% of the performance is lost relative to the unaged file system. More concerning still is the changed rank ordering of speeds; Btrfs beats Ext4 when aged and does not unaged.

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 other concerns, we have built Geriatrix - a profile driven aging tool that ages a file system according to a reference (old) file system whose characteristics are given as input.

3 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 [46], file level [25], file system level [32, 3], system call level [39], VFS level [15] and also at the block level [5]. 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.

Then Andrew benchmark [12] and Compilebench [21] are benchmarks which can be used as aging tools. Both tools emulate user behavior by performing typical operations on the file systems like extracting archives, reading files, compiling code, making directories, etc. Compilebench performs these tasks on Linux kernel sources. 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 [35] and Impressions [1]. 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.

4 Why do we need another aging tool?

A 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.

The best aging tools today either replay a trace of file system commands or run scripts of important applications. Smith's aging tool [35] and Impressions [1] come close to what we expect from an aging tool. But, Smith's tool is a twenty year old artefact with dependencies on the Fast File System (FFS) [24]. And Impressions matches an impressive number of aged metrics, but its focus is on generating file system content, not file system layout; in fact Impressions writes data exactly once so there are no mutations or deletions to churn the file system layout state.

To explore the free space fragmentation generated by aging tools, we performed aging on an 80 GB Ext4 partition using the Agrawal aging profile [2] with a 70% fullness target using Impressions and Geriatrix. After aging each with the same target and profile we measured the distribution of the extents of free space using the *e2freefrag* utility. For a baseline comparison, we measure the free space fragments displayed by a freshly formatted Ext4 partition, and, on the other extreme, we also measure the free space extent distribution from a colleague's desktop, a 6 year old 240 GB Ext4 image with 16% utilization. Figure 2 shows that after aging, Impressions still has 86% free space extents of size 1.5 GB on average; only slightly more fragmented than a freshly formatted Ext4 partition. On the other hand, Ext4 aged using Geriatrix and the naturally aged Ext4 image have a variety of free space extent sizes. Informally, if you want to study the on-disk layout fragmentation caused during aging, Impressions is not the right tool, but Geriatrix may be.

5 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.

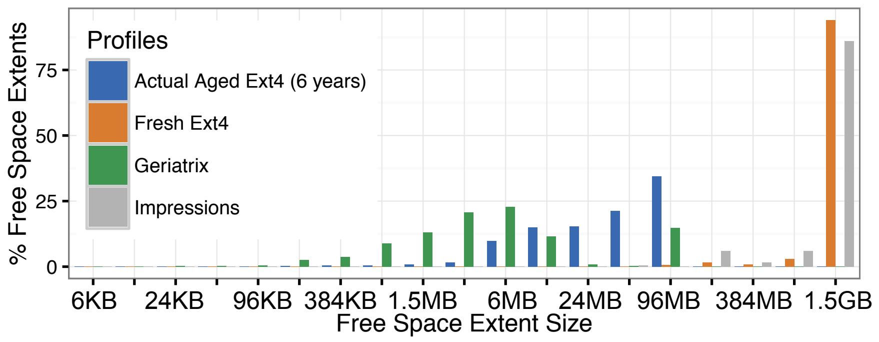


Figure 2: Free space fragmentation comparison of a freshly formatted Ext4 partition, Ext4 aged using Geriatrix and Ext4 aged using Impressions [1], the latter two being aged according to the Agrawal [2] aging profile.

- **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. Files and directories are spread evenly within sibling directories at a given depth.

A key feature of Geriatrix is the way it iterates through creation and deletion of files for a long time relative to simpler aging tools like Impressions. Geriatrix takes an age distribution in its profile, which can be a histogram of the create timestamps of an existing old file system. 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, $n < m < \ldots$, 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.

6 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.

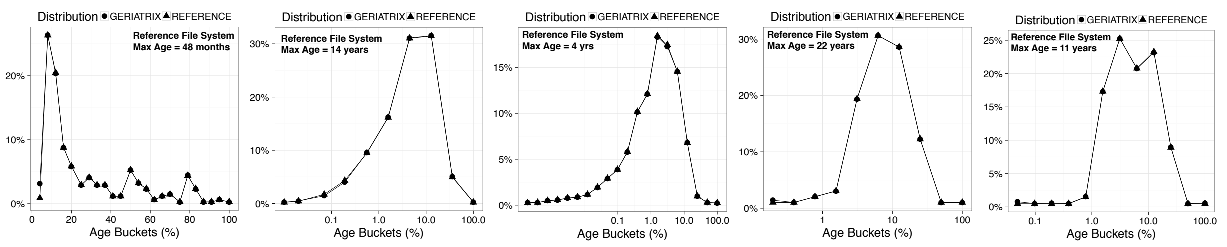


Figure 3: The above graphs show the accuracy of converging to the given age distribution for built-in profiles in our aging tool. Starting from the left, the aging profiles are Meyer [26], Agrawal [2], Douceur [9], Wang-OS and Wang-LANL [38].

Aging Profiles	1 GB Disk Overwrites	RMSE
Agrawal	283	
Meyer	13	
Douceur	22422	$< 0.01\%$
Wang-OS	41	
Wang-LANL	25	

Table 2: Built-in aging profiles.

Appendix A 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.
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.

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 and has five built-in aging profiles which were constructed by referring to long-running file system and metadata publications [2, 9, 38, 26]. Table 2 shows the number of disk overwrites required to age a 1 GB file system image for each of the built-in profiles. We converge to the input age distribution shown in Figure 3 with a root-mean squared error of

Paper	Disk	RAM	CPU (cores)	Linux (Kernel Version)
Btrfs [31]	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 [18]	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)

Table 3: Experimental Configuration.

$< 0.01\%$ for each profile. Since size and directory depth distributions converge trivially, we have not shown them in Figure 3.

- **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 for 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 [5], 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 and (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.

8 Evaluation of Aged File Systems

In order to highlight the impact of aging, we recreated experiments from Btrfs [31] and F2fs [18] publications on unaged and aged file system instances as well as unaged and aged Ext4 and Xfs instances since these two were used for comparison in [31]. All file systems instances were aged on the Ubuntu 14.04 LTS distribution which runs Linux kernel 3.13.0-33. We performed our experiments on the Emulab PRObE cluster [11]. The hardware used for both experiments is described in Table 3. For fair comparison, the memory and number of cores in our benchmark recreations was matched to the setup in the reference publications.

All file systems were aged according to the Agrawal [2] and Meyer [26] aging profiles. We performed aging in memory and stored 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.

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

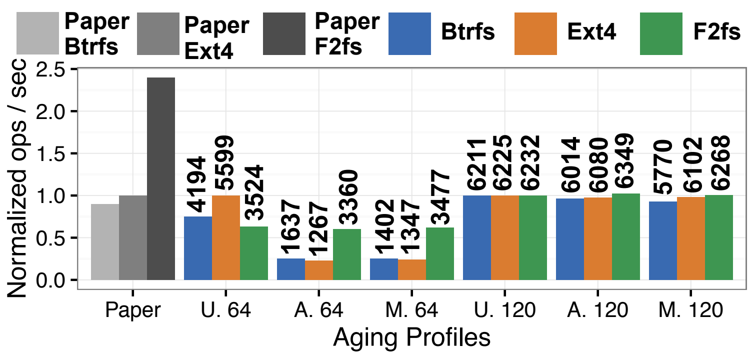


Figure 4: Recreation of the Filebench fileserver benchmark from the F2fs paper [18] on two SSDs - 64 GB (labeled 64) and 120 GB (labeled 120). Performance on 64 drops significantly after aging, especially for Btrfs and Ext4 resulting in a graph that looks similar to published results. Refer Section 8 for detailed explanation. 120 seems unaffected by aging, thus highlighting highly varied performance across different SSDs

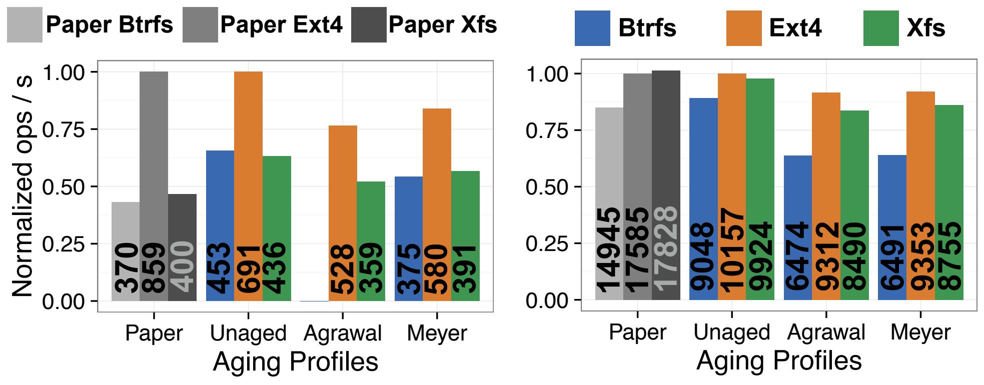
.

The Filebench benchmark [23] was used for all performance measurements with different profiles according to the appropriate reference publication. Each benchmarking run lasted 10 minutes and we performed three runs of each benchmark for capturing variance. We report only the mean since the maximum standard error observed was 1.12%. The primary performance metric reported is *overall operations per second* as reported by Filebench. 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 give 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% fullness target being replayed on a 100 GB partition of a 500 GB HDD.

On comparing unaged file system performance (Figure 1a), 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. It is unsurprising that Xfs was least affected since it is the oldest among all these three file systems. Btrfs fared well considering it is still in active development.

We now compare the same file systems using the webserver profile. The fileserver profile consists of relatively larger file writes and reads while the webserver profile performs thousands of small file operations. Since file systems are usually more sensitive to small file operations, it is understandably harder to reproduce published results and in fact, Figure 5a shows that we get a different rank ordering compared to the paper with Btrfs slightly outperforming Xfs. The performance penalties after aging are between 11-25%. The Btrfs measurement when aged using the Agrawal profile is missing because Btrfs could not complete the execution of the benchmark despite having the required space to do so. This highlights an important use of Geriatrix also a stress testing tool.

The SSD experiments were conducted on a 64 GB SSD with a 59 GB aged file system image with a 70% fullness target. 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 1b shows the com-



(a) (b)

Figure 5: Filbench webserver recreations from the Btrfs paper [31]. HDD results (shown in (a)) maintains overall rank ordering with Ext4, Btrfs and Xfs slowing down by 25%, 18% and 11% respectively. Btrfs aged with the Agrawal profile could not complete the benchmark and hence is missing. SSD performance (shown in (b)) indicates Btrfs being most affected by aging with its performance dropping by 29%. Ext4 and Xfs performance drops by a maximum of 9% and 15% respectively.

pletely different rank ordering of unaged Btrfs, Ext4 and Xfs compared to published results on the fileserver profile. While Btrfs was the most performant in the paper, Ext4 appears to be the winner in our reproduction. 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 of SSDs from a few years ago to not be able to sustain more than 2 minutes of continuous writing before performing inline cleaning [29]. 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 5b are not so dramatic. Btrfs appears to be the most affected by aging showing a 29% performance drop but Ext4 and Xfs do not degrade much showing a maximum of performance penalty of 9% and 15% respectively.

Figures 4 is the recreation of F2fs [18] 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.

9 Practical Tips for Aging

9.1 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.

9.2 Aging in Memory

Age on an in-memory file system and take a snapshot of the aged in-memory file system image [2]. This results in orders of magnitude improvement in aging speeds.

9.3 Aging for Large Drives

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.

10 Conclusion

File systems continue to suffer from fragmentation and data structure complexity during extended use: an effect called aging. Although the aging problemhas been known for a long time, recent research on file systems usually ignores it. More importantly, aging effects are sometimes more drammatic 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.

11 Acknowledgements

We would particularly like to thank George Amvrosiadis, Purvasha Chakravarti, Siddharth Singh, Harshad Shirwadkar and Shweta Jain for their valuable inputs during the development of this project.

This research was supported in part by the DOE and Los Alamos National Laboratory, under con- tract number DE-AC52-06NA25396 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 - Broadcom Ltd., Citadel, Dell EMC, Facebook, Google, Hewlett-Packard Labs, Hitachi, Ltd., Intel Corporation, Microsoft Research, MongoDB, NetApp, Inc., Oracle Corporation, Samsung Information Systems America, Seagate Technology, Tintri, Two Sigma, Uber, Veritas and Western Digital.

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

A Formal Proof Of Convergence of the Age Distribution

Theorem A.1. *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 index as $b = B$ and the youngest bucket as $b = 1$. Let the relative size of bucket b be s_b. 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 g_b relative number of files in bucket b. We claim that our age distrubution converges at:*

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

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

First, we assume that the rapid aging phase ends in the oldest bucket itself. In order for this to happen, we should perform at least K operations in the oldest bucket. Since we perform Ts_b operations in bucket b, this translates to $Ts_B \geq K$. 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. Now note that in any bucket b, we need Ks_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 and 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 $\sum_{b=1}^{B} s_b = 1$). □

References

[1] Nitin Agrawal, Andrea C Arpaci-Dusseau, and Remzi H Arpaci-Dusseau. Generating realistic impressions for file-system benchmarking. *ACM Transactions on Storage (TOS)*, 5(4), 2009.

[2] 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), 2007.

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

[4] 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), 2009.

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

[6] 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), 1995.

[7] James Cipar, Mark D Corner, and Emery D Berger. Tfs: A transparent file system for contributory storage. In *File and Storage Technologies (FAST)*. USENIX, 2007.

[8] Douglas Comer. Ubiquitous b-tree. *ACM Computing Surveys (CSUR)*, 11(2), 1979.

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

[10] Richard J Feiertag and Elliott I Organick. The multics input/output system. In *Proceedings of the third ACM symposium on Operating systems principles*. ACM, 1971.

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

[12] 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), 1984.

[13] 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 *File and Storage Technologies (FAST)*. USENIX, 2015.

[14] 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), 2010.

[15] Nikolai Joukov, Timothy Wong, and Erez Zadok. Accurate and efficient replaying of file system traces. In *File and Storage Technologies (FAST)*. USENIX, 2005.

[16] Ryusuke Konishi, Yoshiji Amagai, Koji Sato, Hisashi Hifumi, Seiji Kihara, and Satoshi Moriai. The linux implementation of a log-structured file system. *ACM SIGOPS Operating Systems Review*, 40(3), 2006.

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

[18] Changman Lee, Dongho Sim, Jooyoung Hwang, and Sangyeun Cho. F2fs: A new file system for flash storage. In *File and Storage Technologies (FAST)*. USENIX, 2015.

[19] Sungjin Lee, Ming Liu, Sangwoo Jun, Shuotao Xu, Jihong Kim, et al. Application-managed flash. In *File and Storage Technologies (FAST)*. USENIX, 2016.

[20] Youyou Lu, Jiwu Shu, and Wei Wang. Reconfs: A reconstructable file system on flash storage. In *File and Storage Technologies (FAST)*. USENIX, 2014.

[21] Chris Mason. Compilebench. http://oss.oracle.com/mason/compilebench/, 2008.

[22] 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 *File and Storage Technologies (FAST)*. USENIX, 2012.

[23] Richard McDougall and Jim Mauro. Filebench. www.solarisinternals.com/si/tools/filebench/, 2005.

[24] 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), 1984.

[25] 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 *File and Storage Technologies (FAST)*. USENIX, 2007.

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

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

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

[29] Milo Polte, Jiri Simsa, and Garth Gibson. Enabling enterprise solid state disks performance. *Workshop on Integrating Solid-state Memory into the Storage Hierarchy*, 2009.

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

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

[32] Drew S Roselli, Jacob R Lorch, Thomas E Anderson, et al. A comparison of file system workloads. In *USENIX annual technical conference, general track*, 2000.

[33] Margo Seltzer, Keith A Smith, Hari Balakrishnan, Jacqueline Chang, Sara McMains, and Venkata Padmanabhan. File system logging versus clustering: A performance comparison. In *Proceedings of the USENIX 1995 Technical Conference Proceedings*. USENIX Association, 1995.

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

[35] Keith A Smith and Margo I Seltzer. File system agingincreasing the relevance of file system benchmarks. In *ACM SIGMETRICS Performance Evaluation Review*, volume 25. ACM, 1997.

[36] 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 *File and Storage Technologies (FAST)*. USENIX, 2010.

[37] Michael Vrable, Stefan Savage, and Geoffrey M Voelker. Bluesky: a cloud-backed file system for the enterprise. In *Proceedings of the 10th USENIX conference on File and Storage Technologies*. USENIX Association, 2012.

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

[39] Zev Weiss, Tyler Harter, Andrea C Arpaci-Dusseau, and Remzi H Arpaci-Dusseau. Root: Replaying multithreaded traces with resource-oriented ordering. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. ACM, 2013.

[40] 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 *File and Storage Technologies (FAST)*. USENIX, 2008.

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

[42] 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 *File and Storage Technologies (FAST)*. USENIX, 2016.

[43] 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 *File and Storage Technologies (FAST)*. USENIX, 2016.

[44] Zhihui Zhang and Kanad Ghose. yfs: A journaling file system design for handling large data sets with reduced seeking. In *File and Storage Technologies (FAST)*. USENIX, 2003.

[45] Benjamin Zhu, Kai Li, and R Hugo Patterson. Avoiding the disk bottleneck in the data domain deduplication file system. In *File and Storage Technologies (FAST)*. USENIX, 2008.

[46] 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. ACM, 2005.