File System Aging

Fabian Hofbeck
Friedrich-Alexander Universität Erlangen-Nürnberg
fabian.hofbeck@fau.de

ABSTRACT

File System Aging may be viewed as a non-issue nowadays, a problem that has been solved long ago. But this assumption does not hold up to the reality: file system aging is still relevant, maybe more so than ever. Aged file systems are ever-present and cause performance losses in many applications. This work provides an overview over past and current approaches to file system aging, as well as give some insight into existing solutions, their effectiveness and new developments.

1 INTRODUCTION

File systems (FSs) tend to show a decrease in performance after some time of use. This decrease in performance is called **aging** and often caused by an increasing fragmentation of the FS. Fragmentation can occur inside a file when its content is scattered, so-called intra-file fragmentation. Logically related files, like the files in one directory, may also be spread across the physical disk, so-called inter-file fragmentation. Aging can affect many operations performed on FSs, including reads, writes and directory traversals.

Smith and Seltzer [15] showed in 1997 that FSs do age under realistic workloads with significant impacts to performance. However, many developers view FS aging as a solved problem [5]. Conway et al. [6] attribute this perception to the extreme aging of the FAT FS. FAT was developed by Microsoft in 1977 for storing data on Floppy disks and later enhanced to support larger disks [16]. Both FAT and FAT32 usually do not employ any strategies to reduce fragmentation. Data is always stored in the next free block, leading to heavy fragmentation [7]. This behavoir, however, is largely dependent on implementation, tests have shown a significantly lower fragmentation with the Linux implementation compared to the Windows implementation [7]. Nowadays FAT has largely been replaced by NTFS on Windows machines [3], and ext on linux machines. While NTFS was introduced in 1993, the newest version of ext, ext4, was indroduced in 2006 and finished development in 2008. The design of ext4 already includes several strategies to reduce fragmentation [12]. Since modern FSs actively try to reduce fragmentation, they display a more subtle aging. This may lead some to regard the aging problem as solved. Another factor to be considered is the extremely rapid aging observed in current research. A user will experience an unaged FS only briefly. Aging is often not a gradual noticeable decline in performance, but the expected state of the FS [5]. This has also led to a lack of aging-related evaluation in recent FS papers [11].

In contrast, recent work by Conway et al. [5] and Kadekodi [11] showed significant slow down due to aging in modern FSs. They found that especially Solid State Drives (SSDs) are severely affected by performance losses, as they are impacted by different aging mechanisms as mechanical Hard Disk Drives (HDDs) are. Since FSs have primarily addressed those aging mechanics heavily affecting HDDs in the past, the performance loss on SSDs is more severe.

A study on fragmentation on mobile devices using Android found that 70% of block accesses are to SQLite databases [10]. Especially these SQLite database files turned out to be among the most heavily fragmented files on mobile devices, measurably affecting performance on these devices.

Conway et al. [6] also examined the assumption that little remaining space will increase fragmentation and thus speed up the aging process. Especially in application benchmarks they found the impact of a FSs fullness to be significantly lower than the performance impact of aging itself.

Section 2 introduces the fundamental processes behind FS aging and how to describe and recreate them scientifically. Additionally, Section 2 also discusses countermeasures to prevent aging, and how modern FSs age.

2 FILE SYSTEM AGING

The following sections detail what causes aging in a FS, how different storage technologies age and how FS age can be quantified. Multiple approaches to artificially age FSs will be compared. Section 2.5 provides an overview over strategies to mitigate aging. How some modern FSs implement these strategies and how effective they are is discussed in Section 2.6.

2.1 The Aging Process

Fragmentation affects performance because read and write operations are spread across the storage device, rather than being in one location. Maximum throughput can only be achieved when the amount of data transferred is large enough.

The Natural Transfer Size (NTS) is the amount of sequential data that must be transferred from or to a storage device to reach a certain percentage of the devices maximum throughput [5]. If a FS splits files or metadata into chunks significantly smaller than the NTS of a device maximum throughput can not be achieved. Conway et al. [5] examined the effective bandwidth for different read sizes on HDDs and SSDs. They found a transfer size of 4MiB to be a reasonable choice for the NTS, on both HDDs and SSDs.

How FSs age is highly dependent on their type, common types being B-Tree based and update-in-place FSs [5]. The design principles of these types that primarily affect their aging are the topic of this section.

Update-in-Place FSs. Update-in-place FSs, like ext4, write new files to a sufficiently large free location and update them in-place. Fragmentation of files is a common issue with this type of FS. For example: a file that should logically be placed between two existing files can not be placed there, resulting in data not being sequential unless files are moved. File deletions and files growing beyond the locally available free space also induce fragmentation and hinder sequential accesses. With increasing fragmentation transfers are

more and more likely to be smaller than the NTS, leading to a drop in performance.

B-Tree based FSs. B-Tree based FSs store logically related items adjacent in the B-Tree but usually do not guarantee that these items are physically adjacent [9]. Insertions will lead to nodes being split and deletions will cause nodes to be merged, scattering data across the disk. This aging of B-Tree based FSs is highly dependent on the leaf size [5, 9]. Leaves much smaller than the NTS will lead to aging due to them being moved around when split and merged. Having NTS-sized leaves would combat this issue but is impractical for other reasons: Large leaves may incur write amplification, meaning a small change to a leaf can require large writes. To reduce write amplification, and thus increase update performance, B-Tree based FSs use small leaf sizes.

2.2 Aging on Different Devices

On HDDs aging is primarily caused by logically related information not being stored physically coherent on disk. A single file split up and stored in multiple locations, or the files in a directory being stored in different locations, are examples of this issue. This discrepancy between logical relation and physical locality leads to repositioning during reads and writes on HDDs.

SSDs on the other hand do not require repositioning, which could lead to the false assumption that they are not affected by aging. Recent work by Kadekodi et al. [11] shows that not only do SSDs age, but the performance drop can be more significant than on HDDs. Figure 1 shows that aged FSs on HDD exhibit little to no significant performance decrease. Neither of the aging tools used, Geriatrix and Impressions, reduced FS performance noticeably on HDDs. The measurements on SSDs, however, display a significant performance hit with either tool.

On SSDs the FS is not the only cause of performance decrease due to aging. SSDs contain a translation layer between the IO interface they expose and the actual flash memory. The Flash Translation Layer (FTL) primarily remaps addresses and performs garbage collection and wear leveling. FTLs are similar to log-structured FSs and are also subject to aging[11]. With age the increasing need for garbage collection and the fragmentation of mapping tables can decrease performance.

Kadekodi et al. attribute the performance drop on SSDs to this aging of effectively two FSs. The additional performance decrease seen only with the Geriatrix tool is attributed to the more realistic free space fragmentation created by the Geriatrix tool.

2.3 Measuring File System Aging

To determine how aged a FS is, metrics are required. These can span from measuring inter-file and intra-file fragmentation to regular measures of FS performance when compared to an unaged version of the FS. Some metrics used in research are explained in the following.

Layout score. Smith and Seltzer introduced the **layout score** to quantify the amount of fragmentation in a file or FS. The layout score for intra-file fragmentation is the fraction of a file that is optimally allocated, that is blocks are sequential on the drive. A file with a layout score of 1 is allocated as one contiguous array

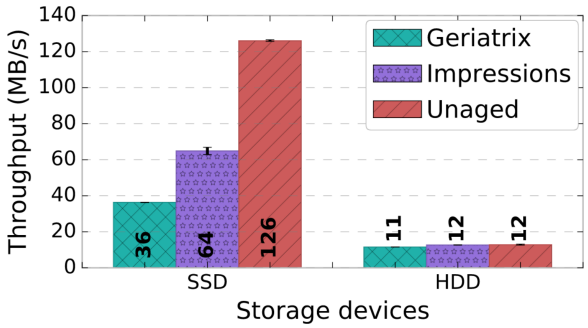


Figure 1: Aging impact on ext4 atop SSD and HDD by Kadekodi et al. [11].
The three bars for each device represent the FS freshly formatted (unaged), aged with Geriatrix, and aged with Impressions [2]. Although relatively small differences are seen with the HDD, aging has a big impact on FS performance on the SSD.

of blocks, while a layout score of 0 means no blocks of the file are adjacent. The layout score of an entire FS is also called **aggregate layout score**. It is the fraction of all blocks of all files that are optimally allocated.

Recent work by Conway et al. expands on the layout score by defining a **dynamic layout score**. To obtain the dynamic layout score the logical block requests made by a FS are captured. The fraction of these requests that was contiguous determines the dynamic layout score. This approach also includes metadata accesses and accesses spanning files into the score [5].

Free space extents. Since layout scores only represent fragmentation of allocated space, a means of measuring fragmentation in the unallocated part of a FS is required. A measure of free space fragmentation used by Kadekodi et al. [11] is the size distribution of free space extents in an aged FS.

A fresh FS instance starts of with large extents of free space. With age the average size of free space extents shrinks, resulting in a distribution of free space extent sizes spanning mainly from very small extents to medium-sized ones.

Operation speed. Operation speed by itself is primarily a measure of FS performance, but it can be used to quantify aging. Running the same workload on aged and unaged instances of the same FS shows the impact aging has on the FS. Operation speed is usually measured in operations per second and is a common measure in work on FSs [11, 13]. This allows direct comparison between results from aged FSs and the corresponding reference material.

Search time. Another approach, also not limited to measuring aged FSs, is measuring the time it takes to search data in a FS. Conway et al. [5, 6] use what they call **grep test**. As the name suggests, this test performs a recursive grep on the root of the filesytem, scanning recursively through both, data and metadata, and recording the duration. As a FS ages, accesses during the scan will become less sequential due to increasing fragmentation. Since the duration of this scan is not only influenced by the FSs age, but

also by the total size of the FS, the measured time is normalized by the FS size.

Write latency. As fragmentation becomes more prevalent in an aged FS, writes incur additional latency. This increasing write latency can be used to measure fragmentation [6]. It is critical, however, to ensure the latency is actually caused by the write operations, and not by, for example, CPU heavy calculations.

2.4 Artificially Aging a File System

In order to compare the aging characteristics or aged performance of different FSs, a reliable and reproducible way of aging a FS is required. This is also a requirement for evaluating the effectiveness of newly implemented countermeasures. The following sections describe different methods of aging a FS for these purposes and discuss their advantages and disadvantages.

Actual use of the FS. One obvious way of aging a FS is actually using it for some time. With normal use this might require months to years and would not be easily reproducible. To achieve reproducibility the *use* would need to be a defined sequence of issued commands or executed applications that is identical for every run.

Some real workloads can be used to age a FS successfully. For example Conway et al. [5] used git, more precisely git *pull* operations, to achieve application level aging. With the exception of BetrFS, all FSs they aged with the git workload showed aging with losses in performance. Their measurements with git in a standard configuration clearly show a feature of git, an automatic garbage collection. This feature is designed to mitigate FS aging on the application level by compressing many smaller files into one larger file called a *pack*. They then repeated their experiments with this feature deactivated and found that, for many FSs, aging was exacerbated.

This shows that using applications to age a FS requires knowledge about the applications, as they may already contain strategies to mitigate or reduce aging. In addition, using applications to age a FS also creates an aged FS instance very specific to the applications used for aging.

Replay. Smith et al. [15] also discussed aging by a realistic workload. This could be achieved by tracing all operations performed on a file system over months or years, while it is in use. Theoretically, this trace could then be used to perform these operations in the same order, but over a much shorter time, on new FSs. However, capturing and more importantly storing these traces is highly impractical due to their size [6, 15]. The solution proposed by Smith et al. is to not collect complete traces, but to collect snapshots of the FS instead. By determining the differences between snapshots it is then possible to generate an artificial workload that recreates those modifications seen between two snapshots.

Previous work [4, 14], however, had shown that most files exist in a FS only for very short amounts of time. Thus, the creation, and subsequent deletion, of these files was not represented in the snapshots taken by Smith et al. They then captured short traces to get realistic representations of how these short lived files are created, modified and deleted. They then added these operations to the snapshot based workload in order to approximate the influence these files have on the aging of a FS.

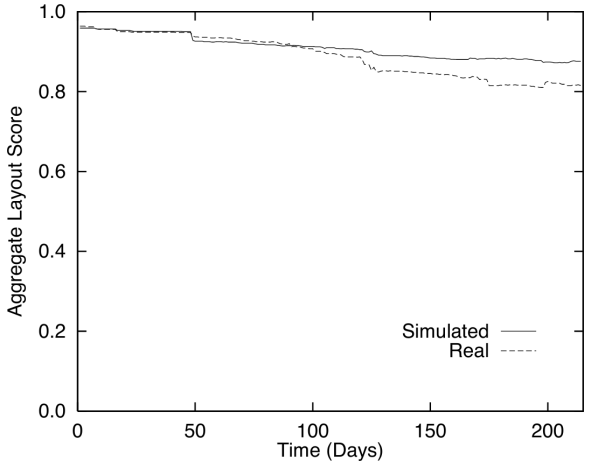


Figure 2: Real vs. simulated file system by Smith et al. [15]. This chart plots the aggregate layout score for each day in the seven month simulation period. The "Simulated" line shows the fragmentation on the artificially aged file system. The "Real" line shows the fragmentation on the original file system from which the aging workload was generated.

This artificial workload created from snapshots and traces is then *replayed* onto a new FS. Even though the artificial workload may have been created from snapshots spanning years of use, the process of applying the workload can be done significantly faster. In many FSs aging is not dependent on the time between operations, but only on the type and most importantly order of operations [15]. This can be used to execute an artificial workload as fast as possible, creating aged FSs almost immediately.

To evaluate how realistic their artificial workloads aged FSs Smith et al. compared artificially aged FSs with ones that had been aged by months of use. The file system not aged by an artificial workload had seen approximately seven months of use. The artificial workload created for comparison took 39 hours to replay onto a fresh FS. They then analyzed how fragmented each of the FSs were. Figure 2 shows that the artificial or simulated workload fails to achieve the same level of fragmentation as the real workload does. Increasingly so as the time used for aging the real/original FS gets longer. Additionally, Figure 3 shows a significant disparity for the file-size range of 2MB to 4MB, where the original FS shows a significantly higher degree of fragmentation.

This shows that, while the *replay* approach by Smith et al. can approximate the gradual increase in fragmentation as time progresses, it becomes less accurate for longer times. The approach also correctly approximates the trend of decreasing fragmentation as files get larger. But for files of specific sizes the artificial workload does not fully reproduce the original FS. The authors attribute those discrepancies to missing information when constructing the artificial workload.

In addition, to those inaccuracies, the *replay* approach also has the issue that it requires extensive and long term collection of FS snapshots to build an artificial workload.

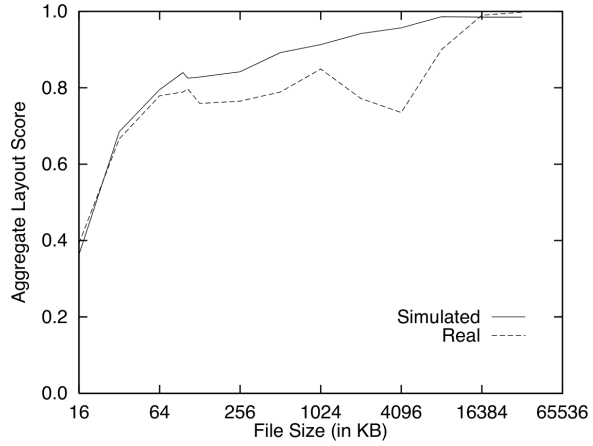


Figure 3: Fragmentation as a function of file size by Smith et al. [15].
File sizes were rounded up to an even number of file blocks, and the aggregate layout score was computed for files of various sizes on the real and simulated file systems. The results are graphed here.

Impressions. Agrawal et al. developed the *Impressions* framework to create representative and statistically accurate filesytem images [2]. The main goal of the Impressions framework is to generate realistic and detailed FS images from given parameters. These parameters may include number of files, directory depth, distribution of file sizes and many more. Another parameter however is more interesting when trying to create aged FSs: Impressions allows the user to specify the desired degree of fragmentation in the generated FS image. This is achieved by creating and then deleting temporary files during regular file creation.

The problem with this approach is that Impressions will repeatedly measure the degree of fragmentation in a FS and then repeat this procedure until the desired fragmentation is achieved. This means that FSs employing strategies to reduce fragmentation will just be subjected to more of these operations. Since this behavior obviously complicates the comparison of FS implementations when it comes to fragmentation, Impressions also offers an alternative. After the FS image has been created Impressions can run a predefined workload on it and report the resulting degree of fragmentation. In this case the result will reflect the FSs ability to reduce fragmentation and thus allows comparisons between FSs and evaluation of mitigation strategies.

In conclusion Impressions has the ability to create aged FSs with some limitations when it comes to comparing different FSs or implementations. These shortcomings can be explained by the original goal behind Impressions, that is creating realistic FS images.

Geriatrix. To address some ot the shortcomings of the previous approaches Kadekodi et al. [11] developed their tool *Geriatrix*. Most previous approaches to artificial aging have focused on file fragmentation, due to the heavy impact repositioning had on HDD performance. As these costs are not a primary concern in modern flash based SSD memory, creating aged FSs for SSDs may require a different approach.

Similar to the way Impressions created aged file systems, Geriatrix also performs operations on an unaged FS until it matches a previously specified aging-profile. The parameters for this profile can be obtained from an already aged FS, similar to, but significantly less extensive than the data required for the replay approach.

Geriatrix then ages a FS in two stages, rapid aging and stable aging. Rapid aging only creates files to reach the fullness specified in the aging profile as fast as possible. Stable aging then randomly creates and deletes files to match the file-age distribution also specified in the profile. Other parameters of the profile like directory depth distribution and file size distribution are maintained during both stages of aging.

To provide reproducibility the random choices during the stable aging stage are controlled by a user defined seed, thus assuring identical results when using the same seed.

Geriatrix will terminate stable aging once a predefined condition is met. That condition may be an execution time, confidence in the age distribution or number of disk overwrites. Unlike Impressions where the target was a specified degree of fragmentation, the target here will usually be the age distribution of the profile. This allows easier comparisons between FSs and FS implementations.

Figure 4 shows that Geriatrix replicates free space fragmentation of an actually aged ext4 FS very closely, while Impressions does not. Impressions seems to display free space fragmentation closer to an unaged instance of ext4.

The aging impact of Impressions and Geriatrix on ext4 running on SSD and HDD is shown in Figure 1. Geriatrix is able, especially on SSDs, to achieve a significantly greater aging induced performance loss than Impressions. The authors attribute this performance difference on SSDs to the greater free space fragmentation achieved by Geriatrix.

2.5 Countermeasures

A FS implementation can employ several strategies to counteract or mitigate aging effects. These strategies may not be effective in all situations or on all types of storage media. The following sections describe known strategies and their limitations.

Grouping files. The common problem of files that are logically related but physically spread out on a disk can be addressed by physically grouping related files together. Since related files may be added over time it is necessary to reserve space for a group larger than all the files may initially need. The method by which files are considered related may differ between FSs. One possibility is to group files in one directory, another might be to group files that have been created roughly at the same time.

As files grow and more files are added to a directory this approach will reach its limitations. Files or additional content will eventually be placed outside the group, again breaking locality.

Extents. By allocating space not per block, but groups of sequential blocks, so-called **extents** [5], FSs can reduce bookkeeping overhead. An additional benefit of extents, especially when choosing larger ones, is reduced intra-file fragmentation.

When allocating a file, the first extent that can fit the file completely can be chosen. This ensures smaller files are never split up and larger files are split into larger pieces. Like with grouping

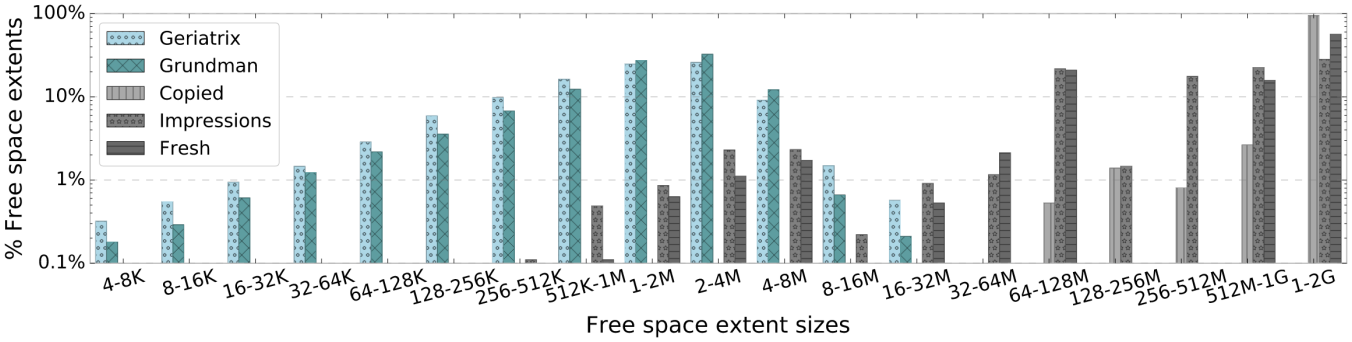


Figure 4: Free space fragmentation comparison by Kadekodi et al. [11].
Free space fragmentation comparison of an actual old ext4 FS image (Grundman) with Geriatrix driven by the Grundman profile, Impressions driven by the Grundman profile's file size distribution, a partition with the contents of the 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.

files, locality will be disturbed once a file grows beyond its original extent.

Delayed allocation. Delaying allocation can be a good addition to using extents. As a file is created or grows the required space is not immediately allocated, instead changes are buffered. This means that when the buffers are written to disk and the space finally needs to be allocated the extents chosen will be larger. The extents can now not only fit the original file as it was created but also any data appended before the buffers were flushed.

Since the FS does not represent the actual state of the file during delayed allocation, the delay will usually be limited to a very short amount of time [5]. As with grouping files by creation time this approach can only improve locality for files that are created practically simultaneously.

If files or data are added or deleted over longer time periods, delayed allocation is not able to prevent fragmentation. A solution can be to involve the application itself in extent selection. By providing developers with the means to specify how large a file is likely going to be, a larger extent can be chosen at the time of file creation. However, this will not reduce inter-file fragmentation.

Packing. Directories containing many small files can benefit from packing all files in as few blocks or extents as possible. Especially combined with grouping these files, sequential accesses to multiple small files in a directory can be sped up in this manner. In addition, to the files themselves, metadata can also be packed into those blocks or extents.

2.6 Modern File Systems

Modern FSs usually employ some strategies to mitigate aging, but most still show significant performance losses due to aging. This section discusses ext4 exemplary for update-in-place FSs and BtrFS as a B-Tree based FS. Additionally, Section 2.6 introduces the B^{ε}-Tree based BetrFS. All of them age differently and their approaches to reduce the impact of aging are also different.

ext4. As mentioned above ext4 is of the update-in-place type, thus its aging pathology is explained in Section 2.1. ext4 implements all of the mitigation strategies detailed in Section 2.5 in some form [1, 12].

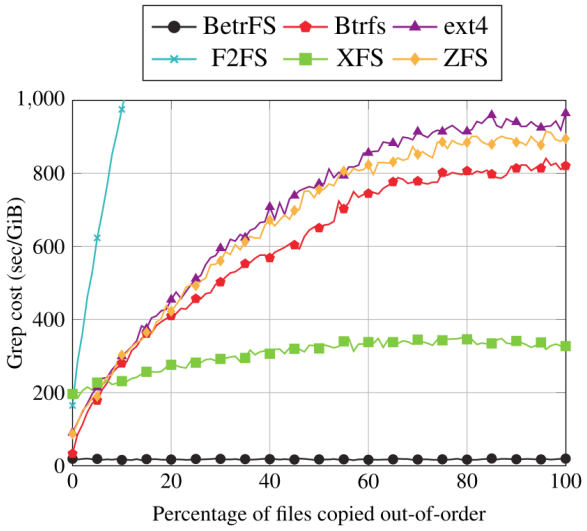
Extents in ext4 can be up to 128MiB big, allowing files with sizes up to the extent size to be stored completely inside one single extent. As studies have shown [3, 8], most files found in FSs are between 8KiB and 8MiB in size. Thus 128MiB is sufficiently large to fit most files. Combined with delayed allocation, which is also implemented in ext4, this reduces the likelyhood of files being split on creation. However, these studies also show that the majority of stored bytes is found in increasingly larger files. As these large files, often being database or blob files, grow and shrink over time they may cause substantial fragmentation.

ext4 also implements a form of packing and grouping, it will try to spread out files from different directories initially, while also trying to place directories close to their parent. This leaves space for additional files in the directory, as well as file file growth [1, 12]. Metadata will usually be stored close to a files actual data or, in the case of directories, close to the files contained within it [1, 12].

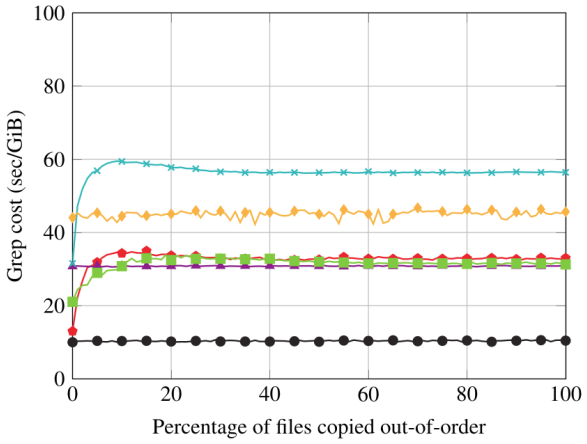
In intra-file aging benchmarks ext4 shows a slowly decreasing dynamic layout score but performs better than BtrFS [5]. Figure 5c shows that ext4 immediately exhibits a very low dynamic layout score in inter-file aging benchmarks.

BtrFS. BtrFS is B-Tree based (see section 2.1) and employs extents and delayed allocation in the same manner as ext4. Due to the way metadata is stored in BtrFS, packing is implemented differently. Usually only file and directory metadata is stored inside BtrFSs B-Trees. Howeever, smaller files can be stored directly in the tree nodes, though files may be split for this procedure since BtrFS uses small nodes [5].

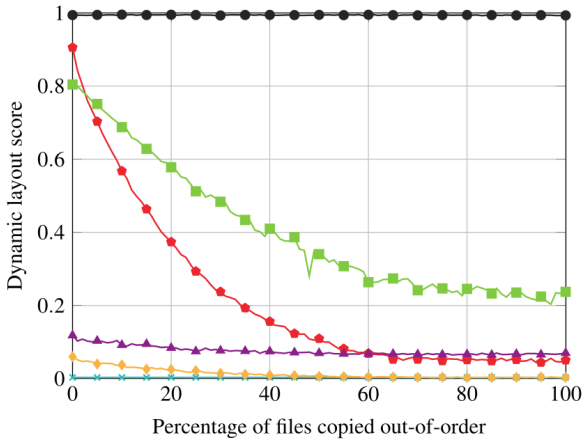
There are other theoretical ways of combating aging in BtrFS, like keeping leaves arranged in logical order, but they would most likely incur unacceptable performance overheads, even in unaged instances.



(a) Recursive grep cost: HDD (Lower is better).



(b) Recursive grep cost: SSD (Lower is better).



(c) Dynamic layout score (higher is better).

Figure 5: Interfile benchmark by Conway et al. [5]. The TensorFlow github repository with all files replaced by 4KiB random data and copied in varying degrees of order.

Section 2.1 discussed the importance of leaf size when it comes to aging B-Tree based FSs. Conway et al. [5] found that, as expected, the aging of Btrfs is inversely related to the leaf size.

Figures 5a and b show that BtrFS is closely matched to ext4 when it comes to inter-file aging benchmarks, when measuring grep cost. Only on HDDs a slightly better aged performance is noticeable in BtrFS. Only the dynamic layout score seen in Figure 5c starts off at a significantly higher value than ext4, but then continues to fall below ext4.

By analyzing the layout after their intra-file benchmark Conway et al. also found that BtrFS had completely interleaved the data after repeatedly appending to a few small files. This explains why BtrFS performed poorly in the intra-file benchmark.

BetrFS. BetrFS is a B^{ε}-Tree based FSs developed by Jannen et al. [9] with the goal of benefiting from larger, NTS sized, leaves. By attaching a buffer to each node, BetrFS avoids the high cost of small updates associated with larger leaves. Small updates can be buffered and consequently batched together, once significant changes have accrued. As a result, searches may be slower since the buffers also need to be searched. Overall, however, the query cost is identical to B-Tree based FSs, while the insertion cost is lower than on traditional B-Tree FSs [9].

The node size chosen for BetrFS is 4MiB, which is equal to the NTS discussed in section 2.1. Another benefit of this large node size is that all data can be stored in a second B^{ε}-Tree. By ensuring that the trees are always lexicographically sorted BetrFS can almost always place logically adjacent file physically adjacent on disk [9].

Conway et al. [5] show that, in their experiments, the intra-file, inter-file and git benchmarks, BetrFS does not display significant aging and has stable performance throughout their tests (see Figure 5). They conclude that even though BetrFS will require significantly more writes to disk than for example ext4, it can still perform as well as an unaged ext4 FS. With both FSs aging it can drastically outperform ext4 and BtrFS FSs.

However, research by Kadekodi et al. [11] shows that the git benchmark does not create free space fragmentation as seen in naturally aged FSs. Thus BetrFS may exhibit free space fragmentation, as it would not show in the experiments performed by Conway et al.

3 CONCLUSION

This work introduces current and past research into file sytem aging and discusses the relevance of further research. While there have been improvements over the years, many problems are still unsolved. Most modern file systems still show significant aging with performance loss, especially on newer storage technologies like SSDs. If SSDs or even faster devices are to be used to their full potential, additional solutions are required. Even when using several strategies to limit inter-file and intra-file fragmentation, current file systems often fail to achieve the physical locality required to retain performance in the long term. Even though these problems have been ignored in the past, there have been some promising current developments. At first glance BetrFS seems to be largely immune to aging and thus merits further investigation.

REFERENCES

[1] [n.d.]. The Linux Kernel documentation: ext4 Data Structures and Algorithms. https://www.kernel.org/doc/html/latest/filesystems/ext4/index.html. Accessed: 10.01.2020.

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

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

[4] Mary G Baker, John H Hartman, Michael D Kupfer, Ken W Shirriff, and John K Ousterhout. 1991. Measurements of a distributed file system. In *ACM SIGOPS Operating Systems Review*, Vol. 25. ACM, 198–212.

[5] Alex Conway, Ainesh Bakshi, Yizheng Jiao, William Jannen, Yang Zhan, Jun Yuan, Michael A. Bender, Rob Johnson, Bradley C. Kuszmaul, Donald E. Porter, and Martin Farach-Colton. 2017. File Systems Fated for Senescence? Nonsense, Says Science!. In *15th USENIX Conference on File and Storage Technologies (FAST 17)*. USENIX Association, Santa Clara, CA, 45–58. https://www.usenix.org/conference/fast17/technical-sessions/presentation/conway

[6] Alex Conway, Eric Knorr, Yizheng Jiao, Michael A. Bender, William Jannen, Rob Johnson, Donald Porter, and Martin Farach-Colton. 2019. Filesystem Aging: It's more Usage than Fullness. In *11th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 19)*. USENIX Association, Renton, WA. https://www.usenix.org/conference/hotstorage19/presentation/conway

[7] Giel de Nijs, Ard Biesheuvel, Ad Denissen, and Niek Lambert. 2006. The effects of filesystem fragmentation. In *Proc. 2006 Linux Symposium*, Vol. 1. Citeseer, 193–208.

[8] John JD Douceur and Bill Bolosky. 1999. A large-scale study of file-system contents. (1999).

[9] William Jannen, Jun Yuan, Yang Zhan, Amogh Akshintala, John Esmet, Yizheng Jiao, Ankur Mittal, Prashant Pandey, Phaneendra Reddy, Leif Walsh, et al. 2015. BetrFS: A right-optimized write-optimized file system. In *13th {USENIX} Conference on File and Storage Technologies ({FAST} 15)*. 301–315.

[10] Cheng Ji, Li-Pin Chang, Liang Shi, Chao Wu, Qiao Li, and Chun Jason Xue. 2016. An empirical study of file-system fragmentation in mobile storage systems. In *8th {USENIX} Workshop on Hot Topics in Storage and File Systems (HotStorage 16)*.

[11] Saurabh Kadekodi, Vaishnavh Nagarajan, and Gregory R Ganger. 2018. Geriatrix: Aging what you see and what you don't see. A file system aging approach for modern storage systems. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*. 691–704.

[12] Aneesh Kumar KV, Mingming Cao, Jose R Santos, and Andreas Dilger. 2008. Ext4 block and inode allocator improvements. In *Linux Symposium*, Vol. 1.

[13] Richard McDougall and Jim Mauro. 2005. FileBench. *URL: http://www.nfsv4bat.org/Documents/nasconf/2004/filebench.pdf* (2005).

[14] John K Ousterhout, Herve Da Costa, David Harrison, John A Kunze, Mike Kupfer, and James G Thompson. 1985. *A trace-driven analysis of the UNIX 4.2 BSD file system.* Technical Report. CALIFORNIA UNIV BERKELEY DEPT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCES.

[15] Keith Smith and Margo Seltzer. 1997. File System Aging - Increasing the Relevance of File System Benchmarks. *ACM SIGMETRICS Performance Evaluation Review* 25 (07 1997). https://doi.org/10.1145/258612.258689

[16] Matti Vanninen and James Z Wang. 2009. On Benchmarking Popular File Systems. *Clemson University Study* (2009).