A Measured Evaluation of Artificial Filesystem Aging Tools

by

Zowie van Dillen

Student number:	851876552	
Course code:	IB9906	
Defense date	1 September 2021	
Thesis committee:	prof. dr. Tanja Vos,	Open University
	dr. ir. Hugo Jonker (supervisor),	Open University
	ir. Vincent van der Meer (supervisor),	Zuyd Hogeschool

Abstract

Filesystems accumulate file fragmentation over the course of years of active use. This phenomenon is known as “filesystem aging”. For validating the performance of algorithms related to filesystem aging, it is a huge benefit to have a tool that can help generate filesystem aging in a comparatively short period of time, such as a single day. These tools exist, and are known as “artificial aging tools”.

In this thesis we compare the performance of several artificial aging tools: Geriatrix, Impressions and Compilebench. The comparison is based upon how much they fragment the filesystem, as well as upon how realistic the filesystems they generate are.

We conclude that, by our metrics, Impressions generates filesystem aging most efficiently. We also conclude that all of the aging tools produce allocation patterns that are dissimilar to the allocation patterns of computers that were aged the normal way.

Contents

1. Introduction

When a filesystem is actively being used, it accrues file fragmentation over time. This is known as aging. All filesystem formats currently in use (outside of experimental environments) are prone to aging, although some filesystem formats age more quickly than others.

For research into filesystem aging and for developing algorithms to help combat it, it is useful to be able to measure the effects of an algorithm. Researchers might for example wish to validate that a new strategy to prevent fragmentation does indeed lead to less fragmentation. However, getting the data that is necessary for those measurements is not easy. Under normal circumstances a filesystem ages over the course of years of use. If a researcher wants to measure the impact of a new algorithm, then the authentic way to get an aged filesystem requires using the experimental filesystem algorithms for several years. Clearly, this authentic way of aging filesystems is too slow for practical applications. To get results acceptably quickly, the filesystem has to be aged artificially.

A number of tools have been made for aging filesystems artificially. These tools have different goals and different strengths and weaknesses. Many aging tools only try to achieve a high degree of file fragmentation, whereas other aging tools also try to also get a realistic amount of the lesser known types of fragmentation.

In this thesis, we provide a comparison of a selection of artificial aging tools. Our contributions are:

- We provide a comparison of the performance of Impressions [AAA09], Geriatrix [KNG18] and Compilebench [Mas07], of which two out of three are the most often-cited artificial aging tools that work on modern computers. We both compare them based upon how much fragmentation the various tools yield, as well as how realistic the filesystems they generate are.
- We provide insights into all of the metrics used in this project and how they appear on real computers.
- For a number of metrics, we establish what ranges of values are realistic on NTFS computers that were aged the normal way.
- We introduce a new metric, the normalized average gap size, which helps to gain some additional insight into the distribution of files across a storage volume.
- Based upon our observations, we propose a manner of segregating data for fragmentation-related research.

2. Background

2.1. Storage and filesystems

For long-term storage, operating systems store files on storage devices such as hard drive disks (HDD) or solid state drives (SSD). To determine how and where the files should be placed within the storage device, an organizational scheme is needed to structurize the available space. These schemes are known as filesystem formats.

Generally, each operating system has its own filesystem formats and most operating systems do not offer good support for the filesystem formats of other operating systems, although some filesystem formats are almost universally supported, such as FAT. In the last decades, Windows has had NTFS as its main filesystem format.

A single storage device can also hold multiple filesystems. The storage device is then split into partitions, which each hold an individual filesystem. On modern computers, a storage device is always organized into at least one partition. Partitions are sometimes also referred to as "storage volumes". The term "storage volume" is somewhat more abstract than the term "partition", but they usually refer to the same thing.

Filesystems divide a partition into many small sections known as blocks. Blocks are the smallest unit of allocation of a filesystem, so the data of a file in the filesystem is always spread across a whole number of blocks. At the time of writing, blocks are almost always 4 KiB large. Note that one kibibyte (KiB) is 1024 bytes (2^{10}), as opposed to a kilobyte (KB), which is 1000 bytes.

2.2. Fragmentation

When a new file is created or an existing file is extended, the filesystem needs to allocate blocks to the file. Whenever possible, a file is allocated as a single contiguous sequence of blocks. Any contiguous sequence of blocks is usually called an extent, so in other words, the filesystem tries to allocate a file as a single extent. When there is not enough space available to allocate a file as a single extent, the file is broken into fragments. The file is then said to be fragmented.

Hard drive disks can rapidly read subsequent blocks, but may need to do a mechanical movement known as a disk seek to read a non-subsequent block. This makes random reads very slow compared to subsequent reads, and hence it is preferred to minimize the amount of random reads. Whenever a file is broken into multiple fragments this leads to an increased amount of random reads and therefore makes it take longer to read the entire file. Because of this, it is beneficial to avoid fragmentation as much as possible on a HDD.

SSDs are also affected by fragmentation, although not as much as HDDs. Larger reads and writes generally allow for more efficient usage of the bus that connects the SSD to the RAM. Fragmentation breaks one large file into multiple smaller fragments, which means that what could be one large read or write is now multiple smaller ones. This effect is not nearly as significant as the effects of fragmentation on a HDD, but it exists nonetheless.

It must be mentioned that fragmentation is a word more broad than what we have described thus far, and is commonly used to refer to many phenomena that slow down disk accesses or waste disk space. Fragmentation is usually categorized into internal fragmentation, external fragmentation and data fragmentation [Ran69]. However, we do not believe internal fragmentation and external fragmentation to be relevant for comparing artificial filesystem aging tools, and data fragmentation is simply fragmentation as it has been described in the previous paragraphs. Hence, we will not use this categorization. Conway et al. categorized fragmentation differently, into categories we deemed more relevant for this project [CBJ+17, CKJ+19]:

- *Intra-file fragmentation* is what is most commonly thought of when speaking of fragmentation. This refers to fragmentation within a file, which means that the subsequent blocks of a file are not placed adjacent in physical memory.
- *Inter-file fragmentation* refers to fragmentation between different files that are strongly related to each other, but which are located apart in physical memory. Although it does impact performance, inter-file fragmentation is only rarely considered in research.
- *Free space fragmentation* refers to fragmentation between free sections on a disk. A file system with minimal free space fragmentation will have a single large extent of free space, whereas a file system with a large amount of free space fragmentation will have its free space scattered across the disk, broken into tiny sections. In most situations free-space fragmentation only affects write performance and not read performance, however Kadekodi et al. found that free-space fragmentation also has a strong impact on the aging of some types of SSD [KNG18].

2.3. Filesystem Aging

Over time, as a filesystem keeps being written to, it will almost inevitably accrue fragmentation. This gradual process of accruing fragmentation is known as filesystem aging.

The rate at which filesystems age depends on the filesystem format and the operating system. Generally speaking, filesystem aging is not as much of a concern as it used to be. Although old filesystem formats such as FAT did not take any measures to prevent fragmentation, all modern filesystem formats use some strategy to minimize filesystem aging. Nevertheless it must be said that filesystem aging is not a solved problem. Filesystems do still age, albeit much more slowly than they once did. Some operating systems, such as Windows, automatically defragment the drive when it is found to be fragmented enough to warrant it.

2.4. Artificial Aging Tools

Under normal circumstances, it may take years before a storage volume is significantly aged. However, when someone is researching filesystem aging, having to use a volume for several years before getting any data on the performance of an algorithm makes for rather slow feedback. Artificial filesystem aging tools are computer programs used to automatically age a storage volume. Using one of these programs, one can generally age a filesystem over the course of a day rather than over the course of years.

3. Related work

3.1. Measuring File System Aging and Realism

Layout score The most common technique for measuring the degree of fragmentation is the layout score. The layout score quantifies the amount of fragmentation within a single file and was introduced by Smith and Seltzer to help validate their file system aging tool [SS97]. It is the fraction of blocks in a file that are placed contiguously in memory. When a file has a layout score of 1, there is no fragmentation whatsoever and all of its blocks are located contiguously. A file with a layout score of 0 has no contiguously allocated blocks whatsoever. The same concept can be applied to entire file systems, which is called the *aggregate layout score*. Aggregate layout scores measure the amount of fragmentation in a file system, but they do not directly give any indication of how realistic the aging patterns of a file system are.

Grep test Conway et al. used what they called a "grep test" to measure the performance impact of file system layouts [CBJ+17, CKJ+19]. In a grep test, it is measured how long it takes to complete a recursive grep in the root directory of the file system. This operation searches through all the files in the file system for a random string. The time taken is then divided by the amount of storage space the file system is using to normalize it, so that the results between different file systems are more comparable.

Grep tests measure a combination of intra-file and inter-file fragmentation. They are also heavily affected by the time it takes to traverse file-system-related datastructures. When Kadekodi et al. [KNG18] investigated why a grep test was running over seven times slower on an aged file system, they found that the grep algorithm was spending most of its time on looking up file system datastructures, and not on actually accessing the file contents.

Degree of out-of-orderedness In an investigation into NTFS, Van der Meer et al. [vdMJvdB21] introduced several metrics for the convolutedness of file systems. They had made the observation that later fragments of a fragmented file are often placed before the earlier fragments. They called this out-of-order fragmentation. In their investigation, almost half the fragmented files were placed out-of-order. To measure this phenomenon, they introduced the degree of out-of-orderness. The degree of out-of-orderness is a percentage that indicates how many of the fragments of a file are placed out-of-order.

3.2. Artificially Aging File Systems

Smith and Seltzer's tool The article of Smith and Seltzer [SS97] lays much of the foundation of the field of artificial aging algorithms. In it, they introduce their tool for artificially aging file systems, which uses file system snapshots to try to recreate the aging of an authentically aged file system. A number of snapshots of an authentically aged file system have to be gathered beforehand. Smith and Seltzer's tool will replay these snapshots in order, creating and deleting files to recreate the state of the next snapshot. Each file created and deleted contributes towards aging the file system, so eventually, this process will cause significant aging. Although it does successfully age the file system, it was found that these replays lead to a much less fragmented file system than the original file systems they were based upon.

Compilebench To assess how well Btrfs resists aging , a research team at Oracle developed Compilebench [Mas07]. Compilebench is a file system aging benchmark that ages the file system by repeatedly running git operations on the source code of the Linux kernel and compiling it.

Impressions Agrawal et al. [AAA09] developed a framework for generating file system images that are statistically accurate, which they called the Impressions framework. This framework is not just for aging file systems but for generating them from scratch. It lays an emphasis on generating the file system in a small amount of time and can generate a twelve gigabyte image in half an hour. To attain this level of performance it relies on statistical models to generate the file system. For example, when generating files, it decides the size of a file by sampling a probability distribution. The Impressions framework can be configured to generate fragmented file systems by setting the desired degree of fragmentation. To fragment the file system, it will create and delete temporary files during the normal file system generation process until the requested degree of fragmentation is met. However, this method will always lead to the same degree of fragmentation, so it cannot be used to compare the effectiveness of the anti-fragmentation strategies that different file systems use. Specifically for those situations Impressions contains a second method for aging the file system. In this second method, it will run a specified workload and afterwards report the degree of fragmentation that the file system has in the end. If the same workload is executed on different types of file system, then the resulting fragmentation statistics will be comparable.

Geriatrix Kadekodi et al. [KNG18] developed Geriatrix, a framework specifically designed for aging file systems and capable of generating the forms of fragmentation that solid state drives (SSD) greatly suffer from. Traditionally, fragmentation tools focus on the slowdown that results from files being scattered across the disk. Hard disk drives take a lot longer to load data when sequential blocks of logical memory are not placed next to each other on physical memory because the mechanical arm needs to be repositioned to start reading from the new location in physical memory. SSDs do not suffer significant slowdowns from this form of fragmentation. However, SSDs have fragmentation issues of their own, which derive from the hardware implementation.

An SSD is meant to fulfill the same role as an HDD although it makes use of very different hardware, so SSDs contain a layer in the device firmware called the flash translation layer (FTL). This layer helps translate between HDD-centered external interactions and the flash memory based hardware implementation. The FTL performs tasks such as address remapping, garbage collection and wear leveling. To do this, it keeps its own internal datastructures, which are complex and structurally similar to a file system. Just like a file system these datastructures can become fragmented and cause slowdowns. The FTL can age very quickly when both disk space utilization and free space fragmentation are high. Geriatrix continuously ensures both of these conditions, thereby creating exactly the circumstances that greatly speed up FTL aging.

Unlike Impressions, Geriatrix not only generates intra-file fragmentation but also free space fragmentation accurately. In an evaluation it was found that Geriatrix can accurately recreate the impact of nine years of fragmentation in only seven hours.

TraceGen Du et al. [DHSS21] developed TraceGen, a framework that generates realistic file system images that are intended for tests and training in the field of computer forensics. TraceGen works with what are called "stories" in the field of forensics. These are essentially step-by-step plans that build up realistic file systems. The resulting file systems are intended to contain a few clues for forensics to find, but TraceGen is also written to add a lot of realistic clutter to the file system, as to not make it too easy to find the clues. Although it is not the primary goal of TraceGen to generate file system fragmentation, its usage of the computer will inevitably result in fragmentation nonetheless. TraceGen works by playing the instructions in the story back on a virtual machine, so its results are expected to be more realistic than other artificial aging tools, although it is also expected to be slower than the others.

TraceGen is not publicly available at the time of writing, but there are plans to distribute a future version of the framework under an open source license.

4. Methodology

4.1. Metrics

In this study, we compare the filesystems generated by different aging tools to see which performs better. Performance is not a single-dimensional quality in this context. Perhaps one aging tool generates the most intra-file fragmentation, but that does not necessarily mean that it also generates the most realistic filesystems. To show the quality of an artificially aged filesystem, we will provide an overview of a variety of different metrics rather than attempting to capture the full complexity of fragmentation quality in a single metric.

Metrics used in this study fall into two categories. Firstly this study contains metrics for measuring the amount of fragmentation. These are useful for when it is desired to simply generate a lot of fragmentation reasonably quickly. Secondly this study also contains metrics that give somewhat of an indication of how realistic the fragmentation statistics generated by an aging tool were. Realism data is based upon a comparison between the artificially generated filesystems and data from the WildFrag database [vdMJD+19], a non-public database containing file metadata from over 200 personal laptops, almost all belonging to students in higher education.

4.1.1. Amount of fragmentation

Percentage of fragmented files

The percentage of fragmented files is a simple yet still useful metric for indicating how well an aging tool has aged a filesystem. Although it may not be immediately apparent, there is room for discussion in the definition of this metric. As a percentage, the equation for this metric is essentially a fraction with the amount of fragmented files as its numerator. The denominator of this fraction is, however, open to discussion. There are four apparent options for the denominator [vdMJvdB21]:

1. The denominator is the total number of entries in the filesystem tables.
2. The denominator is the number of data files in the filesystem.
3. The denominator is the number of files with blocks assigned.
4. The denominator is the number of files with at least two blocks assigned.

The second option is the most obvious option, but perhaps not the most practical option. Resident files and files with only one block cannot be fragmented but would still contribute towards lowering the percentage when this option is used. The same is true for the first and the third option: Both count files and filesystem entries that cannot possibly be fragmented. The fourth definition is deemed most relevant for measuring fragmentation, since it only counts files that can be fragmented. In this thesis, we use both the first and the fourth option.

Aggregate layout score

The aggregate layout score [SS97] is the most important metric for determining how much an aging tool has aged a filesystem. The aggregate layout score measures the amount of intra-file fragmentation across the entire filesystem. Before we provide a definition for aggregate layout score, we will first provide a definition for ordinary layout scores. These only measure the amount of fragmentation in a single file.

Definition 4.1 (Layout score [SS97]). *The layout score is the amount of locations where a file* ***is*** *fragmented (calculated as* $\text{extents} - 1$*) divided by the amount of locations where a file* ***could be*** *fragmented (calculated as* $\text{blocks} - 1$*). This is then inversed so that unfragmented files have a layout score of 1.0 and completely fragmented files have a layout score of 0.0.*

$$\text{layout score} = 1 - \frac{\text{extents} - 1}{\text{blocks} - 1}$$

This equation only applies for files with more than one block. For files with only one block or no blocks at all, the layout score is defined to be 1.

The aggregate layout score is similar to the layout score, but uses summations of the entire filesystem. Please note that this is not simply an average of all the layout scores of the filesystem.

Definition 4.2 (Aggregate layout score [SS97]). *The aggregate layout score is the amount of locations in a filesystem where a file* ***is*** *fragmented divided by the total amount of locations in the filesystem where a file* ***could be*** *fragmented. This is then inversed so that filesystems with no fragmentation whatsoever have a layout score of 1.0 and filesystems that are completely fragmented have a layout score of 0.0.*

$$\text{aggregate layout score} = 1 - \frac{\sum_{i=1}^{k}(\text{extents}_i - 1)}{\sum_{i=1}^{k}(\text{blocks}_i - 1)}$$

where k denotes the total number of files with more than one block, "blocks_i*" denotes the amount of blocks of the i-th file with more than one block, and "*extents_i*" similarly denotes the amount of extents of file i.*

This equation only applies if a filesystem contains at least one file with more than one block allocated to it. Otherwise, the aggregate layout score is defined to be 1.

As an example: an aggregate layout score of 0.7 means that in 30% of the locations in a filesystem where it is possible for there to be a fragmentation gap, there are in fact gaps, and in 70% of those locations there are no gaps.

Grep test

A Grep test captures a combination of the intra-file fragmentation, inter-file fragmentation and the complexity of the file system [CBJ+17]. In a Grep test, a recursive Grep is performed from the root of the filesystem. This operation will go through all folders and files in the filesystem, looking for all appearances of a given text pattern. The duration of a Grep is a means of representing the impact of the aging of a filesystem in a comparatively realistic usage situation, as opposed to comparatively theoretical metrics like layout scores. The fact that it simultaneously measures all the various factors that affect real-world usage makes it a useful metric for comparing how the various aged filesystems would differ in practice.

Grep tests are also a practical way of measuring inter-file fragmentation. Inter-file fragmentation is fragmentation between files that are strongly related to each other. There are a variety of ways that files can be strongly related to each other. For example, a program might always load two files directly after each other. If the placement of these files is optimized so that they can efficiently be read in sequence, the amount of necessary disk seeks can be minimized, leading to better read performance. We are not aware of any metrics specifically for measuring inter-file fragmentation, but using Grep tests, we can get a glimpse of the impact of inter-file fragmentation as well as the impact of file system complexity, even if we do not have any metrics for measuring these factors in isolation.

Definition 4.3 (Grep test [CBJ+17]). *A Grep test is performed by running the command* `grep -r "$searchtext"` *at the root of a filesystem and measuring the time it takes for the operation to finish.*

A defragmented copy of the filesystem is made and Grep is run at the root of the defragmented filesystem with the same invocation as before. Once more, the time it takes for Grep to finish is measured and the results between the two invocations of Grep can then be compared to derive what impact fragmentation had on the performance of the filesystem.

In these experiments we performed Grep tests using random strings of eight alphanumeric characters as the argument for Grep.

4.1.2. Realism

The following metrics are used to measure the realism of the generated fragmentation. These metrics mostly pertain to the way that fragmented files are placed in memory. However, the fragmentation-pattern-related metrics discussed in this section don not directly measure realism. To determine whether or not the values for the various metrics are realistic or not, we will determine the realistic value ranges from WildFrag and compare the results against these realistic value ranges. If the results of any particular metric lay entirely outside of the realistic value range, it is deemed unrealistic.

Out-of-orderness

Out-of-orderness is a metric that indicates how often fragmentation is out-of-order, which means: How often the fragment after a fragmentation gap appears earlier in physical address space than the fragment before the fragmentation gap. This metric is used to see if the fragmentation generated by artificial aging tools is differently laid out across the storage volume than fragmentation gotten by normal computer usage.

Definition 4.4 (Out of Orderness [vdMJvdB21]). *The out-of-orderness of a filesystem is the ratio of the number of times the next fragment occurs prior to the current vs. the total number of fragmentation gaps:*

$$\text{out of orderness} = \frac{\text{backwards gaps}}{\text{total gaps}}$$

Normalized average gap size

The term "gap size" refers to the distance between two subsequent fragments of a fragmented file. Similarly to out-of-orderness, we use this metric to see if we can find differences between the placement of artificial fragmentation and normal fragmentation.

It is obvious how this should be measured in the case of in-order fragmentation. For out-of-order fragmentation there is some room for discussion. Van der Meer et al. [vdMJvdB21] describe three ways to measure the gap size for gaps between out-of-order fragments:

1. Tail-head distance: The distance between the last block of the first fragment and the first block of the second fragment. In case of out-of-order fragmentation, this means the length of both fragments counts towards the gap size.
2. Carving distance: How far a file carver would have to search for the second fragment when it is has already found the first fragment. In case of out-of-order fragmentation, this is the distance from the beginning of the first fragment to the end

of the first fragment and only counts the length of the second fragment towards the gap size.

3. Shortest distance: The shortest distance between two fragments. In case of out-of-order fragmentation this is the distance from the beginning of the first fragment to the end of the second fragment.

In the context of fragmentation research, the first definition is the most practical. When a process reading a file runs into a fragmentation gap, it essentially jumps from the last block of the first fragment to the first block of the second fragment. The distance of this jump measured by the first definition.

Definition 4.5 (Gap size). *The gap size is the distance between the anterior and the posterior fragment. The distance is measured from the highest block address of the anterior fragment to the lowest block address of the posterior fragment.*

The following function calculates the gap size of the gap between fragment i and fragment $i+1$ of file f:

$$\text{gapsize}(f, i) = |\text{head}(\text{frag}_f(i+1))) - \text{tail}(\text{frag}_f(i))) - 1| \quad \text{for } i < \text{numfragments}(f)$$

Where f denotes the file, i is a number referring a fragment within file f, $\text{frag}_f(i)$ gives the i-th fragment of file f, $\text{head}(x)$ and $\text{tail}(x)$ respectively give the block address of the first and last block of a fragment, and $\text{numfragments}(x)$ gives the amount of fragments in file x.

Note the absolute marks in the definition of gap size: Gap size does not differentiate between in-order and out-of-order fragmentation.

To use this metric in the comparison, the average gap size of a whole filesystem is taken and subsequently normalized so that it can be compared against filesystems of a different size.

Definition 4.6 (Normalized average gap size). *The normalized average gap size of a filesystem fs is calculated by dividing the sum of all gap sizes of all files f by the total amount of gaps across the entire filesytem, and then normalizing the result by dividing it by the size of the filesystem:*

$$\text{norm\_avg\_gap\_size}(fs) = \frac{1}{\text{size}(fs)} \cdot \frac{\sum_{f=1}^{\text{numfiles}(fs)} \sum_{i=1}^{\text{numfragments}(f)-1} \text{gapsize}_{fs}(f, i)}{\sum_{k=1}^{\text{numfiles}(fs)} (\text{numfragments}(k) - 1)}$$

with $\text{gapsize}(f, i)$ *and* $\text{numfragments}(x)$ *defined as before, and where fs denotes the filesystem,* $\text{numfiles}(fs)$ *denotes the amount of files in filesystem fs,* $\text{numfragments}(x)$ *denotes the amount of fragments in file* x *of filesystem fs and* $\text{size}(fs)$ *denotes the size of filesystem fs in blocks.*

When a filesystem has no gaps and therefore the denominator would be zero, the normalized average gap size is defined to be zero. A filesystem is assumed to have storage space, so size_f *is not allowed to be zero.*

Notably, each storage volume in WildFrag is treated as an independent filesystem in this experiment.

4.2. Aging Tool Experiments

A comparison of aging tools requires data. To get this data, we have conducted a series of experiments where each artificial aging tool is used to generate an aged filesystem. The various aging tools were subsequently compared based upon statistics derived from these experiments.

The following procedure has been applied for each individual experiment:

1. A filesystem is prepared by formatting a disk partition with NTFS.
2. The filesystem is aged with one of the aging tools.
3. Information about the aged filesystem and all of its files are gathered in a SQLite database with PriFiwalk [vdMJD+19]
4. The database collected with PriFiwalk is used to derive values for the various metrics.
5. The results are compared to the authentic filesystem data from the WildFrag database to determine whether they are realistic or not.

In this section we will discuss the steps one to four as well as a number of topics that arise upon taking a closer look.

For step 1, we discuss filesystem formats in section 4.2.1, as well as discuss the size of partitions and similar matters. We discuss the execution environment of the experiments in section 4.2.2. Step 2 is discussed in section 4.2.3, and step 4 and 5 are discussed in section 4.2.4 and 4.2.5 respectively.

4.2.1. Preparing Partitions

Before an experiment can be carried out, a filesystem must be prepared to conduct the experiment on.

All experiments were run on a freshly formatted disk. Experiments were run on the same 4 TB external hard drive disk, which was split into 45 partitions of exactly 64 GiB and formatted with NTFS. Note that these partitions were formatted on Windows, in case it makes any difference with the Linux NTFS formatter. By recommendation of a supervisor, a full reformat was performed, which involves overwriting the old data with zeroes, so that no remainders of data could possibly interfere with the results.

The size of 64 GiB was chosen because it is reasonably large, but not too large, because aging tools take substantially longer to age a filesystem the larger the filesystem is. The specific size of 64 GiB was chosen because it is, by programming standards, a nice round number.

All of the disks used in the experiments were formatted with the NTFS filesystem format, which has been the default format for Windows operating systems for several decades. It is worth mentioning that Windows Server operating systems have been moving to using ReFS as their default since 2012, however ReFS is still not included in Home editions of Windows. Although some of the aging tools run on Unix platforms, the fact that all of the partitions use the same filesystem format helps to make them more comparable with each other. Furthermore, the specific choice of the NTFS format helps to make it comparable with the data available from WildFrag, which also consists almost entirely of NTFS volumes.

4.2.2. Operating System

The programs being tested were developed for Unix operating systems and cannot be natively run on Windows. Because the data from WildFrag consists almost entirely of filesystems in the NTFS format, a decision had to be made on how the programs should be run.

NTFS is a filesystem format that is native to Windows. Linux, however, also provides a custom implementation of an NTFS driver for the sake of increasing its ability to interact with Windows. Although the Linux NTFS implementation is fully featured, it is nonetheless not exactly the same as the Windows NTFS driver. We expect that the drivers use slightly different strategies for allocating files and combatting fragmentation, which might taint the results of the experiments. If the Linux NTFS drivers were to be used to generate filesystem images from the Unix programs then the results might not be properly comparable to the authentic filesystem data taken from WildFrag or the filesystem images generated by the Windows NTFS drivers.

Another option is to run all experiments on Windows and use the Windows NTFS drivers for everything. This ensures consistency among the various filesystem images and thereby allows for better comparisons. The trouble is, of course, that Unix programs normally cannot be run on Windows. Fortunately, this becomes possible with the Windows Subsystem for Linux (WSL). WSL is an optional Windows component that provides a Linux shell and allows for command-line interface based Linux programs to be run. The first version of WSL was only a compatibility interface, but the second version contains a full-fledged Linux kernel, which allows for virtually all Linux programs to be run without subtle compatibility problems. The Windows file tree is accessible from the WSL environment by default. This is implemented with a compatibility interface that interacts with the normal Windows NTFS drivers.

4.2.3. Aging Tool Parameters

In principle, it is ideal if all aging tools are configured to do exactly the same amount of work or a very similar amount of work. This helps make the results more comparable. In practice, this is very difficult to achieve, because the various aging tools all behave differently. One benchmark compiles a code repository while the others fill a disk with generated files. Naturally, they also use different types of parameters to determine how much work they should do.

Figure 1 shows the tweakable parameters of each program. All of the programs have more parameters than shown in the table, but the parameters left out of the table are generally filesystem paths and other settings where there is a not a lot of potential for debate over what value should be filled in. Some of the programs take chance distributions as a parameter, but these will be kept to their default values. The parameters in the table are adjusted between different experiments, and with only these parameters the programs have been configured so that all of the tools are doing a similar amount of work.

Compilebench is relatively straight-forward and take the number of operations it should carry out as a parameter. The other aging tools have parameters that are more difficult to compare. Impressions is somewhat unusual in this project because it takes the final layout score as a configuration setting, whereas the layout score is a dependent variable for the other aging tools.

Notably, in the article that introduces Impressions [AAA09], a second mode for fragmenting the file system is briefly mentioned, which does not take the layout score as a parameter but instead runs a certain workload and reports the layout scores in the end. We have not been able to find any further mentions of this mode in the program's documentation nor in its source code. We have reached out to the authors for clarification on how to use this mode, but have not received a response.

A similar amount of work can be defined in multiple ways. It could be similar in the amount of data written, the amount of processor operations or the amount of time taken. In this context we will be using the last definition. Our goal is to let all of the programs run for a similar duration. Since the aging tools are generally single core programs that use as much processing power as they can get for most of their runtime, two aging tools that are allowed to run approximately equally long will have used the processor approximately equally much. Moreover, durations are a very common and relatable type of measurement.

In the first stages of research, we have used the example settings from the documentation of the various aging tools. We measured the amount of time it took for each tool to generate a file system with these settings and used this to estimate approximately how

Aging Tool	Parameter	Description
Impressions	`Layoutscore`	Target layout score. *Suggested: 1.0*
	`FSused`	Size of the filesystem that will be generated *Suggested: 20 MB (for volume of 100 MB)*
	`Randseeds`	Random seed
Geriatrix	`-i`	Number of runs over the file system size (e.g. a value of 100 on a disk of 64 GB would mean 6.4 TB is written) *Suggested: 1000*
	`-u`	Disk fullness as a number between 0 and 1 *Suggested: 0.8*
	`-r`	Random seed *Suggested: 42*
Compilebench	`-r`	The number of iterations *Suggested: 30 or 150*
	`-i`	The amount of directories that the benchmark will start with. *Suggested: 10 or 30*

Table 1: An overview of the tweakable parameters that the various aging tools provide. Random seeds have also been included as an indication of which programs are known to be deterministic. Suggested values have been included for all parameters except Randseeds from Impressions (which was too long).

long the tool will run per unit (per iteration, per GB written or per layout score point). We then estimated which values we should give to the parameters so that the aging tools would all run approximately equally as long as the others.

When estimating how long it will take a tool to run per unit by using the time that it takes to generate one filesystem, we implicitly make the assumption that there is a linear relation between the duration of a run of a tool and the amount of work it was asked to do. This assumption is not quite true, since the tools are, after all, intended to age the filesystem. A tool will be able to finish one operation more quickly on a fresh filesystem than on an aged one, so the cost per operation will slowly increase as the run goes on. Because of this, it may take several estimates before we narrow in on an invocation that causes a tool to run for the desired duration. When any experiment deviates from our expectations, we refine the parameters and rerun the experiment. Using this method, the aging tools will eventually all carry out a similar amount of work.

4.2.4. Data Collection

After an aging tool has completed its run, PriFiwalk [vdMJD+19] is used to create a database which contains information about the filesystem and all of its files. Notably these databases contain no information about the actual contents of the files.

4.2.5. Deriving Statistics

To derive statistics from the database generated by PriFiwalk, we have developed a Python program that takes a PriFiwalk database as its input and generates two CSV files with metrics of each filesystem[1]. The CSV files contain the values of metrics such as the aggregate layout score for each filesystem, as well as simple counts such as the amount of files, the amount of sparse files and the amount of files without blocks. These CSV files were subsequently imported into LibreOffice Calc, where all further data analysis was done.

Moreover, we analyzed the WildFrag database to learn about the various metrics and find insights into what is realistic. The findings of this data analysis can be found in section 5.

For determining what is realistic, we ended up creating a table of 5-number summaries of all the metrics that the aging tools are being compared upon. The statistics from the filesystems generated by the various aging tools were compared against the realistic value ranges found in this table, and any value that was entirely outside of the value range (i.e. below the minimum or above the maximum) was deemed unrealistic.

[1]The source code can be found at https://github.com/Fruitsalad/prifiwalk-measure-tool

4.3. Comparison over time

For large-scale experiments, it is highly beneficial to have a tool that generates fragmentation efficiently, so that more runs can be done within the same amount of time. In situations where an aging tool is used to validate an algorithm intended to reduce aging, it is beneficial for the developers to be able to get feedback within a reasonable time-frame, and as such, it is good to have an aging tool that reaches the expected degree of aging relatively quickly.

The aim of this project is mainly to compare the capabilities of each of the aging tools, but since we are already repeatedly measuring these aging tools, this is also a good opportunity to show which aging tools are faster and which are slower. In this section, we will discuss our methodology for creating a series of graphs that visually compare how well each of the aging tools performs after a certain amount of time has passed.

4.3.1. Acquiring the Necessary Data

To be able to create a graph that shows how well an aging tool performs over time

There are several ways to get the necessary data:

1. It is possible to get the data by configuring all of the aging tools to run for a long time and running each of them once. The aging tools can be paused in the middle of their runs to take measurements and the graph for each aging tool can be based upon different measurements made during the same run.

2. It is possible to get the data by running each of the aging tools multiple times. In this approach, the aging tools fully complete their execution and measurements are only taken afterwards. The graph for each aging tool can be based upon measurements taken after different runs of the same aging tool, with each run taking a different amount of time.

Both methods have their advantages and disadvantages.

An important advantage of the first approach is that it makes it possible to get many measurements in a short time span. There only needs to be one run of each tool, and within that run, the aging tool can be paused to take measurements at a very high frequency if it is found to be desirable. The second approach does not allow for this. Measurements can only be taken after a run has been completed, so it will take n runs to take n measurements. This means that it may take five times as long to take five measurements using the second approach as it would take to take fifty measurements using the first approach. Clearly, the first approach is preferable for generating a lot of data.

However, the first approach also has an important disadvantage: It is difficult to implement. While the second approach requires no special technology, the first approach requires a pausing mechanism. Both Windows and Linux provide a mechanism for pausing a running process: Linux provides the signal `SIGSTOP` and the Windows API provides the undocumented function `_NtSuspendProcess`. Unfortunately, a simple application of these mechanisms is not suitable for this situation. The problem is open file pointers: Pausing a program does not cause it to relinquish its open files. Because all of the aging tools are constantly writing to files, it is very likely that they will still have a file pointer open if they are paused at an arbitrary point in time. The issue with open files is that the disk must be unmounted from Windows to be able to run the program that takes the measurements. When files on the volume are still open, it is only possible to forcibly unmount a volume, which will close all the open files from that disk and most likely crash the aging tool that had a file open.

Another option is to implement a pause functionality into each of the aging tools. This allows for timing the interruption better, as the aging tools can be paused specifically when they do not have any files open. All of the aging tools would have to be individually modified to implement this change. Fortunately, that is possible, since the source code of each of the aging tools is publicly available.

There is another potential issue with the first approach. The assumption is implicitly made that each of the aging tools is continuously aging the filesystem. This is not necessarily true. Measurements taken in the middle of a run of an aging tool do not have to be representative of the final results that this aging tool would ever provide. This is because an aging tool might use a strategy that divides the process into various stages where a representative result is only obtained after the final stage has finished. Fortunately, although all of the aging tools included in this project do work in various stages, this only appears in the form of short preparatory stages before the actual aging process starts. Once the aging itself starts, all of the aging tools work continuously, so we do not believe this concern applies for the specific tools that have been included in this project.

Altogether, we consider the first option to be the better option, mainly because it is very difficult to configure the aging tools to run for a similar amount of time. This is significantly worsened by the fact that the only way to see if the aging tools are properly configured to run for a similar amount of time is to let them finish their execution and see how long it took. Aging a 64 GiB partition takes quite a while, so this is a rather time-intensive process. Because of this, the choice was made to use the first option.

To collect the data necessary for the graphs, we use the same metrics and techniques as discussed in section 4.2.5.

5. Analysis of authentically aged filesystems

To learn more about how each of the metrics appear in real data, we conducted an investigation into each of them using the WildFrag database. The realistic value ranges of each of the metrics was determined, which can be found in the conclusions. In this section we analyze each metric in more depth and spend some time looking for correlations between a few of the metrics. Moreover, we define groupings for filesystem sizes and degrees of fullness and use these to find more correlations in the WildFrag data.

5.1. Data culling

Based upon first inspections of the data in WildFrag, we filtered out large amounts of filesystems which we believed to be irrelevant for determining the performance of artificial filesystem aging tools. An overview of the constraints we selected and the amount of filesystems that were filtered out can be seen in table 2.

Constraint	Culled
Filesystem must be NTFS	4
Partition size must be at least 25 GB	362

Table 2: An overview of the requirements for partitions included in this data analysis

The second constraint was mainly put in place to remove boot partitions from the dataset. It also culls out smaller partitions, which often serve a special purpose, such as storing photographs or a disk image of a virtual machine. The focus here is to research fragmentation in general purpose partitions, since that is the type of partition which artificial aging tools will most likely be imitating. Hence culling smaller partitions is desirable.

After culling, 368 volumes remain, of which 198 volumes are on SSDs and 170 on HDDs.

5.2. NTFS partition size

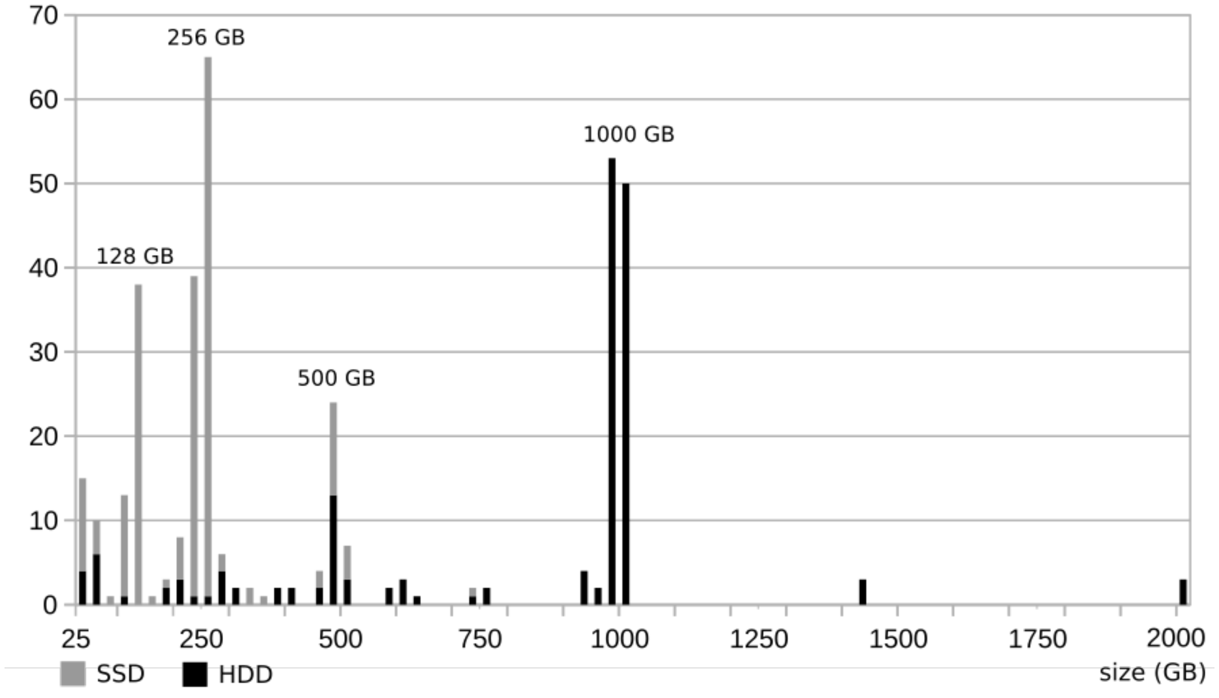


Figure 1: NTFS partition sizes in WildFrag
Histogram

Partitions of less than 25 GB are by far the most common in WildFrag. These are almost all special-purpose partitions, such as boot partitions and Windows recovery partitions. Above 25 GB, the most common filesystem sizes for student laptops are slightly below 128 GB, 256 GB, 500 GB and 1000 GB, as can be seen in figure 1. This should be no surprise for readers familiar with the current storage device market as these are the most common sizes currently available for SSDs and HDDs. Partition sizes were usually somewhat smaller than the size of the storage device because most devices were divided into more than one partition, as can be seen in figure 2.

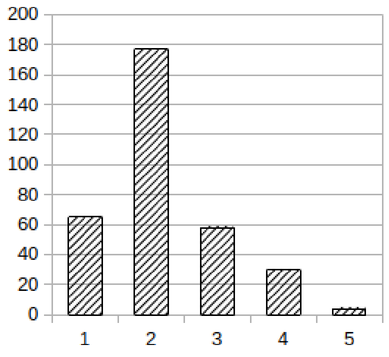


Figure 2: The amount of partitions per storage device
Histogram

The largest SSD partition in the dataset is 750 GB and the largest HDD partition is 2 TB. The average size of all SSD partitions (larger than 25 GB) was 225 GB and for HDD partitions 814 GB.

Taking the data from figure 1 into consideration, we divide the partitions into different groupings based upon their size. We will use these groupings to report on size correlations in the rest of this section, and later in the conclusions we will use them again to compare artificially aged filesystems against similarly sized filesystems from WildFrag.

Lower bound	Upper bound	Count	Percentage
25 GB	100 GB	26	7%
100 GB	150 GB	55	15%
200 GB	300 GB	118	32%
450 GB	550 GB	35	9%
950 GB	1050 GB	105	29%
	remainder	29	8%

Table 3: Partition size groupings.
Lower bound is inclusive, upper bound is exclusive.

Table 3 shows the various partition size categories as well as the amount of partitions that were not placed in any of the categories. Categories were placed around the peaks in figure 1 so that a handful of categories, each less than 100 GB large, still contain 92% of the datapoints. The remaining 8% of partitions were of unusual sizes, often in clusters of two or three partitions, but never enough to justify a new category.

5.3. Total storage size

Twice in this paper we group entire computers by their storage size instead of individual partitions. These groupings can be seen in table 4 and were derived in a similar manner as the partition size categories.

Lower bound	Upper bound	Count	Percentage
200 GB	300 GB	40	19%
450 GB	550 GB	32	15%
950 GB	1050 GB	20	9%
1100 GB	1150 GB	38	18%
1200 GB	1300 GB	56	26%

Table 4: Total storage size groupings.
Lower bound is inclusive, upper bound is exclusive.

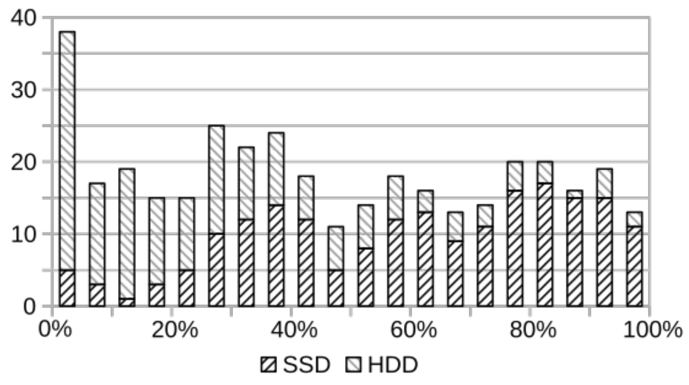


Figure 3: Fullness per partition (in WildFrag)
Histogram

5.4. Fullness

Figure 3 shows how common each degree of fullness is across all partitions in WildFrag (that were not culled). The strongest pattern that can be seen in this graph is that SSD partitions tend to be more full and HDDs tend to be less full. In fact, many HDD partitions are near-empty. This occurs due to two factors:

1. HDD partitions are larger, as seen in figure 1
2. 76% of computers with a HDD also had an SSD

Concerning the first factor, it should be no surprise that disk fullness correlates with the amount of available storage space. The second factor implies that hard drive disks were usually used as secondary storage. SSDs are preferred for their faster access speeds, and hence computer users presumably tend to put as much data as possible on the SSD, leaving the secondary HDD largely empty.

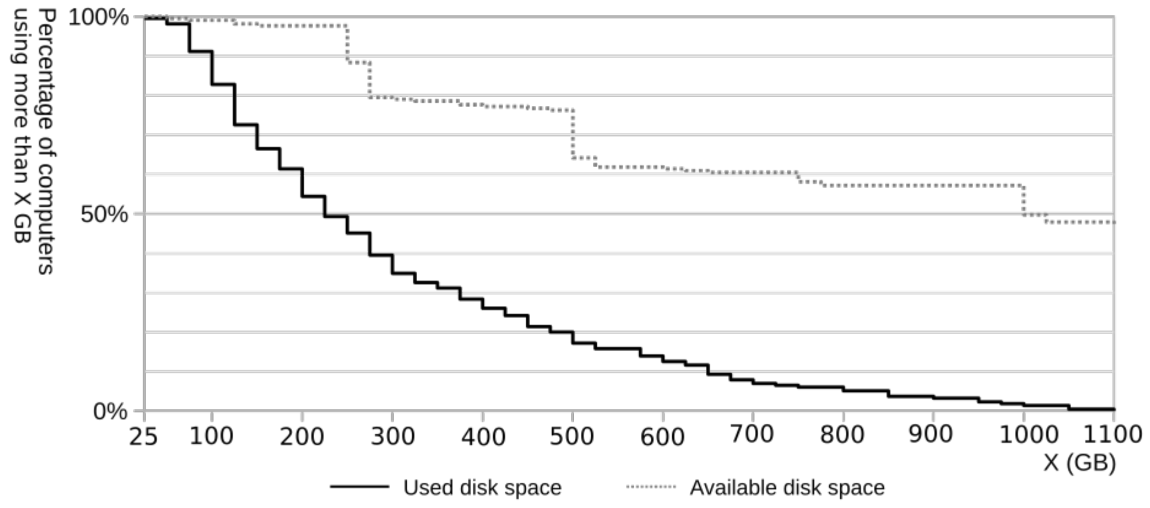


Figure 4: Disk space usage across computers in WildFrag.
Please note that these numbers are per-computer and not per-volume, unlike earlier graphs.
Cumulative histogram with upside-down y-axis

Figure 4 shows how much disk space most computer users actually use. As can roughly be seen in the graph, the median used disk space is 220 GB. This graph is also a source of practical consumer advice: Although 50% of computers had more than a terabyte of disk space, 99% of students did not actually need that much space. Not a single student used more than 1.1 TB of space.

Based upon the statistic that the median used disk space is 220 GB, one might expect that computers with less than 300 GB of disk space will have a very high fullness. Figure 6 shows that this is not entirely true: In the group of computers with approximately 256 GB of storage space, only 20% of computers had a fullness above 80%, and only 10% had a fullness above 90%. On average the fullness of computers in the 256 GB group is still much higher than for computers with more storage space, but it is clear that the median of used space in the 256 GB group is lower than the median among all computers. It turns out, even if it almost goes without saying, that people with less storage space on their computer are more conservative with their storage space than others.

Figure 5 displays a clear correlation between storage space and fullness: For computers with more storage space the average fullness is lower than for computers with less storage space. This correlation is greatly amplified by the two small bubbles for computers with very large amounts of storage space, but even looking at only the big bubbles, the correlation is still apparent.

Just like with partition sizes, it proved useful to group partitions by their fullness. Because figure 3 does not show any strong patterns in the frequency of different degrees of fullness, the decision was made to divide the full range of filesystem fullness into five equally sized groupings of 20%.

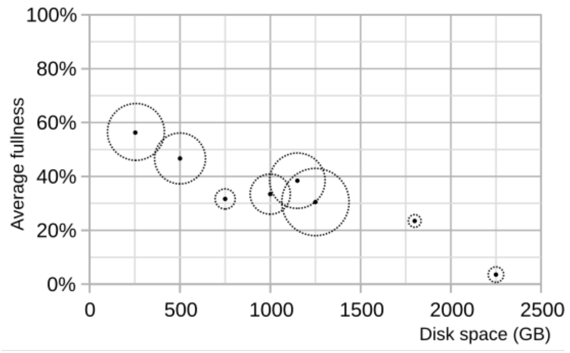


Figure 5: The average degrees of fullness across various groupings of total storage space.
The size of a bubble indicates the amount of computers in a group.

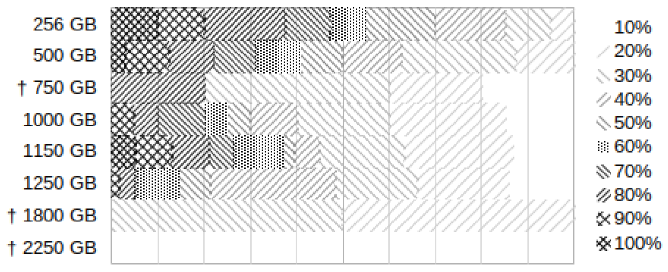


Figure 6: Fullness distributions across various groupings of total storage space.
Darker coloring means the fullness is higher.
† Small group (less than twenty computers)

5.5. Aggregate layout score

On modern computer systems, aggregate layout scores are almost always very high. Only 2% of computers had an aggregate layout score lower than 0.99, as can roughly be seen in figure 7. The lowest aggregate layout score in all of WildFrag is held by an SSD partition of approximately 100 GB, at 0.926. In contrast, in Smith and Seltzer's 1997 paper [SS97] they measured a filesystem degrade from an aggregate layout score of 0.99 to a score of 0.81 in only seven months. It must be mentioned, however, that their computer used Fast Filesystem and not NTFS.

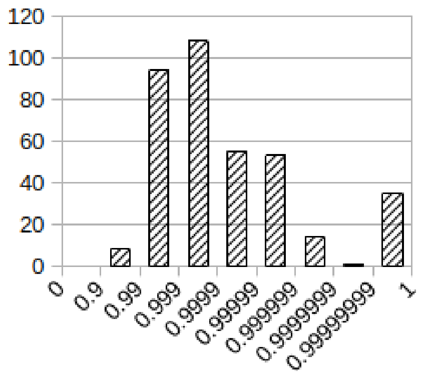


Figure 7: Aggregate layout scores across all volumes in WildFrag.
Histogram

A possible factor in the high aggregate layout scores measured across all computers in WildFrag is the fact that Windows automatically defragments filesystems nowadays. Notably, this is not only true for HDDs, Windows also defragments SSDs under certain conditions [Han].

Figure 8 shows how aggregate layout scores are distributed in all of the different partition size groupings. The group of 25 GB to 100 GB is somewhat anomalous, but amongst the other groups there is a very clear shift towards 1.0 as the partitions get bigger. In fact, the few 2 TB partitions in WildFrag all had a layout score of exactly 1.0, meaning that there was no fragmentation whatsoever. These partitions are not depicted in figure 8.

The unusually high amount of filesystems with an aggregate layout score of 1.0 in the smallest grouping seems to be because those partitions are used for a special purpose. All but one of them had a size of either exactly 32 GiB or exactly 64 GiB. We have not been able to find out what the exact purpose of these partitions is.

Above 64 GiB, the only partitions with an aggregate layout score of exactly 1.0 were HDD partitions with a low fullness. Those partitions do not seem to be heavily used, since almost all of them were near-empty, with a fullness below 0.5%. It is apparently very unusual for a filesystem that is in day-to-day use to have no fragmentation at all, even with automatically scheduled defragmentation.

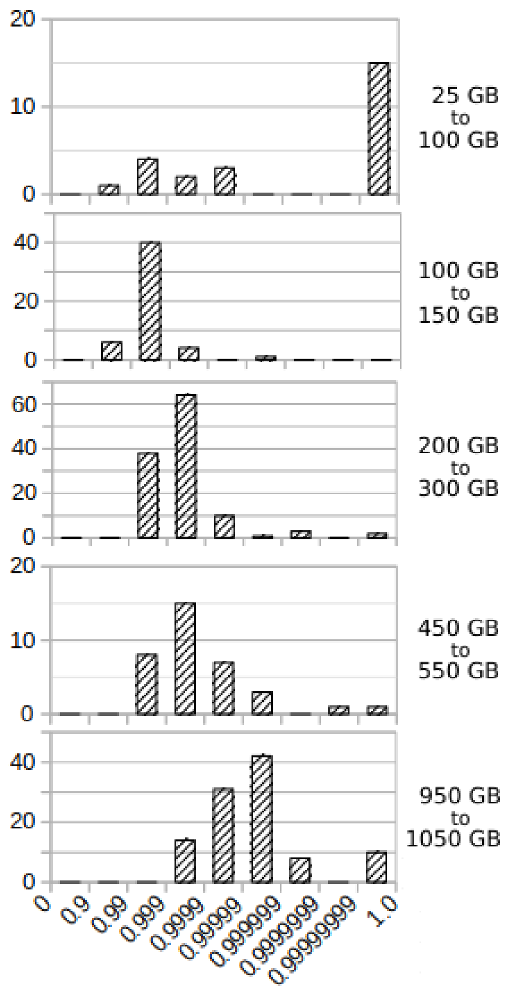


Figure 8: Aggregate layout scores in different partition size groupings. *Histograms*

5.6. Out-of-Orderness

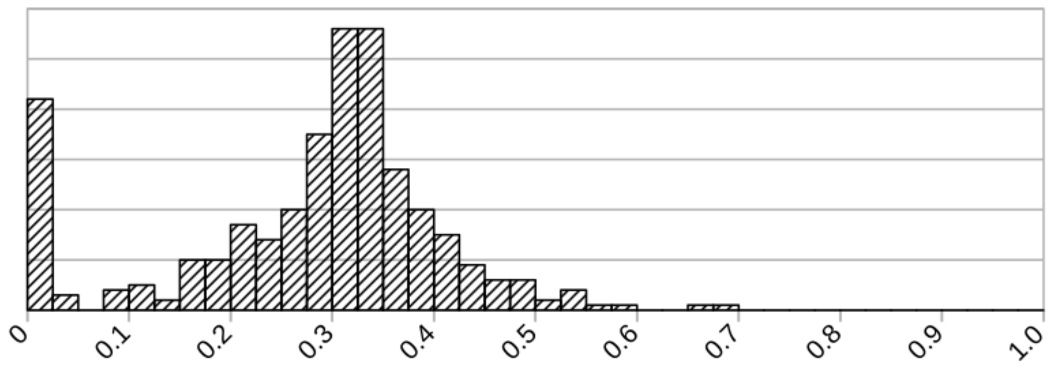


Figure 9: Out-of-orderness across partitions in WildFrag.
Histogram

.

The out-of-orderness of a partition is the fraction of fragmentation gaps where the fragment after the gap is placed earlier in physical memory than the fragment before the gap. Figure 9 shows that out-of-orderness is generally neatly distributed around 0.32, meaning that, on average, there is a 32% chance that a randomly chosen gap is out-of-order. The peak at 0.0 consists entirely of partitions with no fragmentation or very little fragmentation.

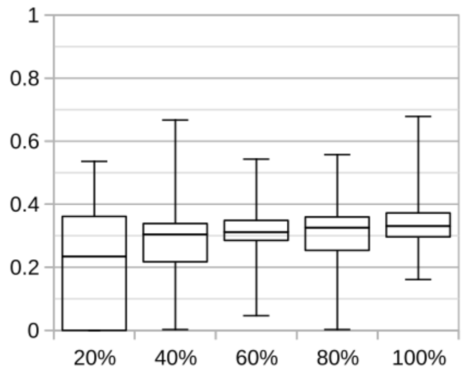


Figure 10: Out-of-orderness distribution per fullness category
Box plot

In figure 10 some irregularities can be seen in the lower fullness categories, but above 40% fullness the distributions start to close in on 0.32 more consistently. In the lowest fullness category, the low first and second quartile marks are due to partitions with no fragmentation.

We have no explanation for why out-of-orderness is distributed around the number 0.32.

5.7. Fragmentation gap size

"Normalized Average fragmentation Gap Size" (NAGS) is the rather lengthy name of a metric introduced in section 4.1. It is used to measure how large fragmentation gaps are on average. The average is calculated per partition and normalized by dividing it by the size of the partition.

Figure 12 shows the NAGS distribution across all partitions, grouped by their fullness category. As shown in the graph, gap sizes are on average generally between 5% and 10% of the size of the partition. On partitions that were less than 40% full, gap sizes were usually smaller than that.

Fullness	Mean NAGS
0% to 20%	0.020
20% to 40%	0.060
40% to 60%	0.086
60% to 80%	0.093
80% to 100%	0.105

Figure 11: Average NAGS per fullness category

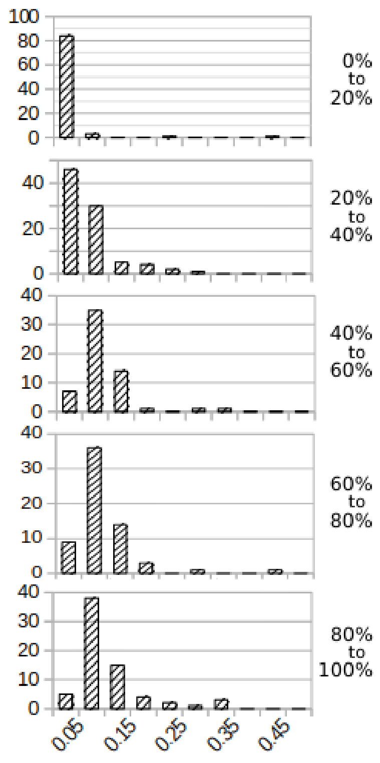


Figure 12: NAGS across different fullness categories.
Please note that only the range 0.00 to 0.50 is shown. No partition had a NAGS above 0.50.

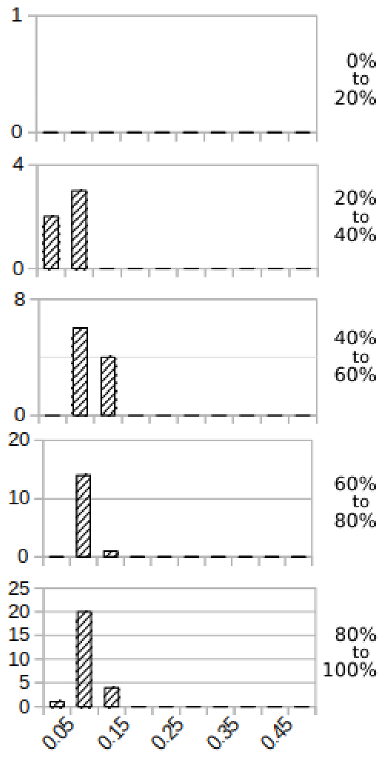


Figure 13: NAGS across different fullness categories, showing only partitions with a size between 100 GB and 200 GB.

6. Experiment Results

6.1. Realistic value ranges

Metric	Realistic value range				
	MIN	25%	MEDIAN	75%	MAX
size	101 GB	125 GB	127 GB	127 GB	127 GB
fullness	81%	88%	90%	95%	99%
Files					
number of files	154,288	265,415	320,626	405,727	746,481
average file size	137 kB	199 kB	228 kB	273 kB	3,471 kB
% of files ≥ 2 blocks	35%	41%	45%	49%	74%
% of files empty	0.36%	0.79%	2.14%	2.85%	9.76%
Fragmentation (amount)					
aggregate layout score	0.98099	0.99117	0.99192	0.99414	0.99977
% of files fragmented	0.29%	6.18%	7.11%	8.03%	10.12%
% of files ≥ 2 blocks fragmented	0.58%	13.65%	15.75%	17.74%	25.23%
Fragmentation (miscellaneous)					
out of orderness	16%	31%	34%	39%	49%
norm. avg. gap size	4.3%	7.3%	8.6%	9.5%	11.5%
NTFS features					
% of files sparse	0.14%	0.56%	0.76%	0.93%	1.51%
% of files compressed	0.01%	0.26%	0.43%	1.65%	31.67%
% of files hardlinks	2.75%	4.53%	5.28%	7.80%	18.39%

Figure 14: Realistic value ranges based upon partitions from 100 GB to 200 GB with fullness above 80%

Table 14 shows a five-number summary for a selection of fragmentation and filesystem-related statistics. These statistics were derived from WildFrag partitions between 100 GB and 200 GB with a fullness of at least 80%.

The size and fullness constraints were chosen based upon findings from section 5. For comparison with the artificially aged filesystems, which are all 64 GiB, it would have been ideal to be able to derive realistic value ranges from 64 GiB partitions. Unfortunately, as found in section 5, most filesystems of less than 100 GB were anomalous and not statistically similar to general-purpose filesystems. The decision was made to use filesystems between 100 GB and 200 GB to derive the realism statistics because this is the nearest range with a significant amount of non-anomalous filesystems.

Similar tables of five-number summaries given other constraints on the selection of filesystems can be found in appendix A.

6.2. Aging tool results

	Impressions		Geriatrix		Compilebench	
size	68.7 GB[2]		68.7 GB[2]		68.7 GB[2]	
fullness	88%		91%		22%	
running time	11h 30min		12h		12h	
Files						
number of files	216,223		539,800		1,074,778	HIGH
avg. file size	277 kB		115 kB	LOW	13 kB	LOW
% of files $\geq$ 2 blocks	61%		50%		50%	
empty files	18 (0.0%)	LOW	0 (0.0%)	LOW	410 (0.0%)	LOW
Fragmentation (amount)						
aggregate layout score	0.973	LOW	0.987		0.920	LOW
% of files fragmented	33%	HIGH	20%	HIGH	18%	HIGH
% of files $\geq$ 2 blocks f.	54%	HIGH	40%	HIGH	36%	HIGH
recursive grep duration	4.13 min/GiB		4.32 min/GiB		2.89 min/GiB	
defrag. grep duration	1.23 min/GiB		1.77 min/GiB		4.04 min/GiB	
Fragmentation (miscellaneous)						
out of orderness	39.6%		34.6%	GOOD	10.8%	LOW
norm. avg. gap size	3.6%	LOW	22.4%	HIGH	2.0%	LOW
NTFS features						
sparse files	0	LOW	0	LOW	0	LOW
compressed files	0	LOW	0	LOW	0	LOW
hardlinks	0	LOW	0	LOW	0	LOW

LOW: This statistic is lower for the artificially aged filesystem than the lowest real value.
HIGH: This statistic is higher for the artificially aged filesystem than the highest real value.
"GOOD": This statistic is in the second or third quartile of the realistic value range (i.e. somewhat close to the median).

Table 5: Statistics of the filesystems generated by the various aging tools, and an indication of how this compares against the realistic value ranges for partitions between 100 GB and 200 GB with over 80% fullness.

[2]68.7 GB is 64 GiB

Table 5 shows an overview of the statistics of partitions generated by the various aging tools. These were compared against the statistics in table 14 and values that were found to be outside of the realistic value range were marked in red. Outliers in the category "Fragmentation (amount)" were not marked in red because the aging tools were intended to age the filesystems a lot and it would be unfitting to make it seem like filesystems with unrealistically large amounts of fragmentation are a bad result.

Impressions was instructed to achieve the aggregate layout score of 0.83, but did not come close to reaching its target despite finishing its execution after eleven hours and thirty minutes. The other aging tools were not instructed to achieve any particular degree of fragmentation and are assumed to have generated a filesystem that was as aged as possible. Even if it is to be expected, it is still to be noted that aging tools tend to produce filesystems with an unrealistically high degree of fragmentation.

None of the aging tools generate filesystems that make use of NTFS features, which was quite as expected since all of the tools are written to be independent of filesystem format.

6.3. Performance over Time

In this section, we show how each of the aging tools performs over time, as discussed previously in section 4.3. The thought behind this was to create a graph where the "sweet spot" of running times of each tool would be clearly visible, as well as a comparison that would allow us to definitively recommend one aging tool for aging a filesystem in less than a certain amount of time, and another aging tool for situations where more time is available. Regrettably the results did not show anything of the kind. Nevertheless, we will share the results we found. It still provides some enlightenment about the workings of each aging tool.

6.3.1. Filesystem fullness

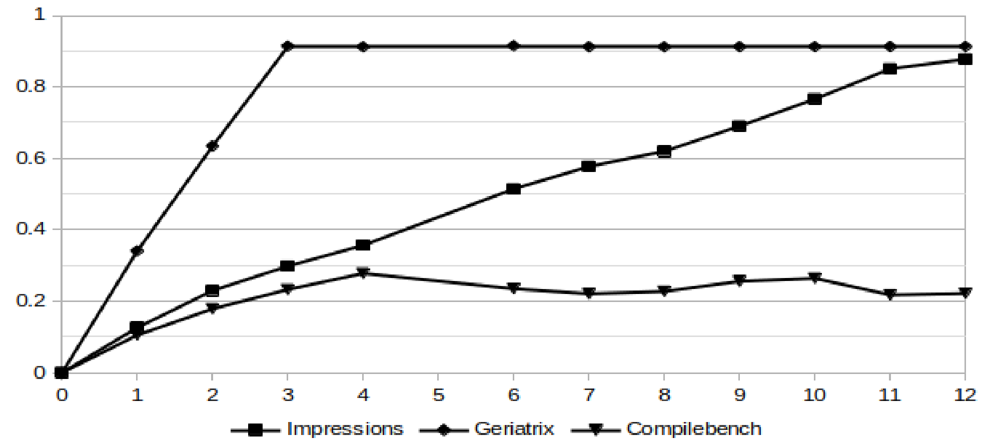


Figure 15: Filesystem fullness of each aging tool, measured throughout a twelve hour run of each aging tool.

The various artificial aging tools filled up the filesystems at wildly different rates, as can clearly be seen in figure 15. Geriatrix was the fastest to fill up the filesystem, reaching the intended fullness after only three hours. Afterwards, Geriatrix keeps the fullness stable and busies itself with fragmentation-related work.

Compilebench similarly started by filling up the filesystem, however it did this at a much slower pace than Geriatrix, finishing after approximately four hours. When this is done, it starts to carry out random operations such as compiling cloning, or deleting a code repository in an attempt to age the filesystem.

Unlike the other two aging tools, Impressions does not work in stages. It gradually fills up the filesystem and finishes its execution when the targeted fullness is reached.

6.3.2. Aggregate Layout Score

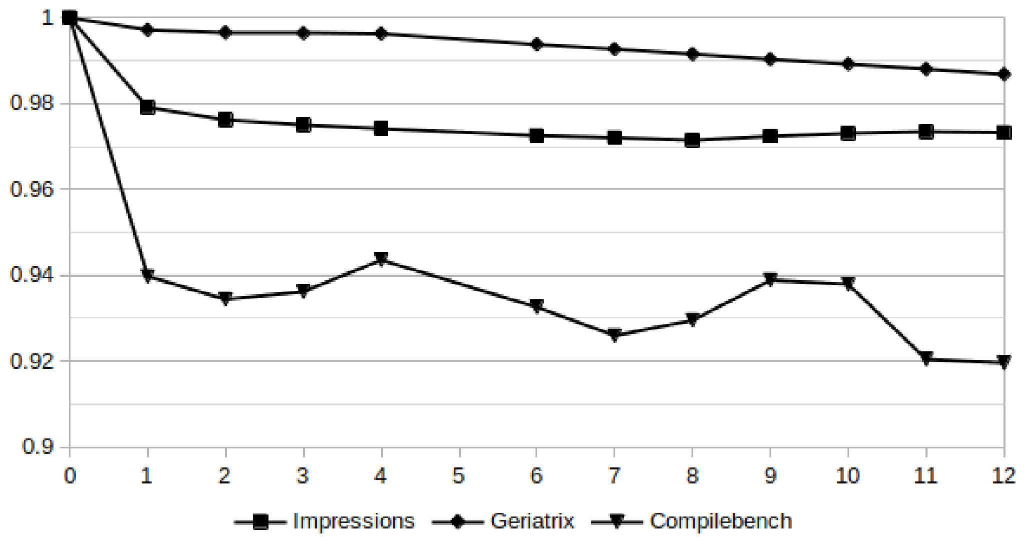


Figure 16: Aggregate layout scores of each aging tool, measured throughout a twelve hour run of each aging tool.

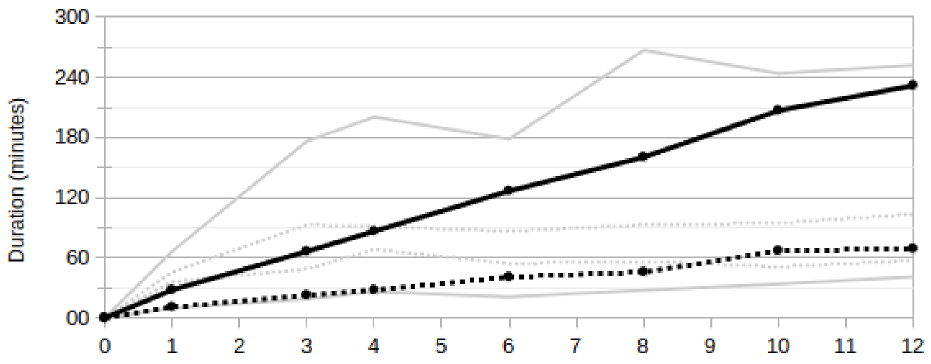
Aggregate layout scores stayed surprisingly high for most aging tools. A high aggregate layout score indicates a low degree of fragmentation, so this is not a good thing. The lowest aggregate layout score in all of WildFrag was 0.926. Most of the artificial aging tools did not get nearly this low.

Compilebench reached the lowest aggregate layout scores despite never reaching a high filesystem fullness. Its aggregate layout score drops down to 0.94 in the first hour and then fluctuates around 0.93.

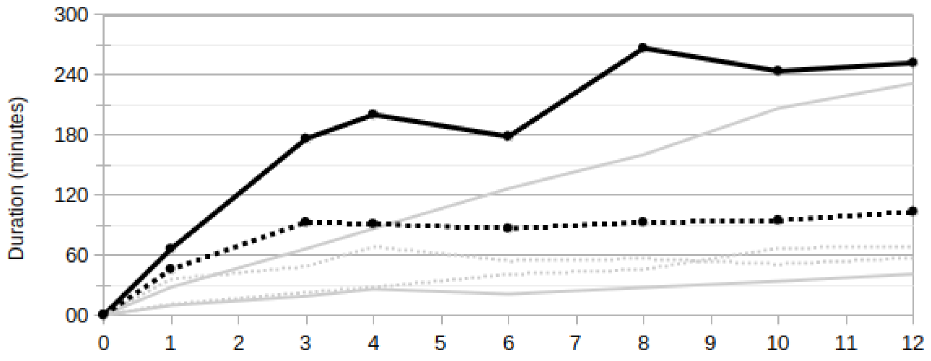
Even though Impressions was instructed to generate a filesystem with a layout score of 0.815, and even though it finished its run after eleven hours and thirty minutes, it did not actually come close to reaching the targeted aggregate layout score. Instead, its aggregate layout score stayed at 0.97.

Lastly, Geriatrix had the highest aggregate layout scores, but had a very slow but consistent downwards movement that indicates that it might eventually reach a low aggregate layout score.

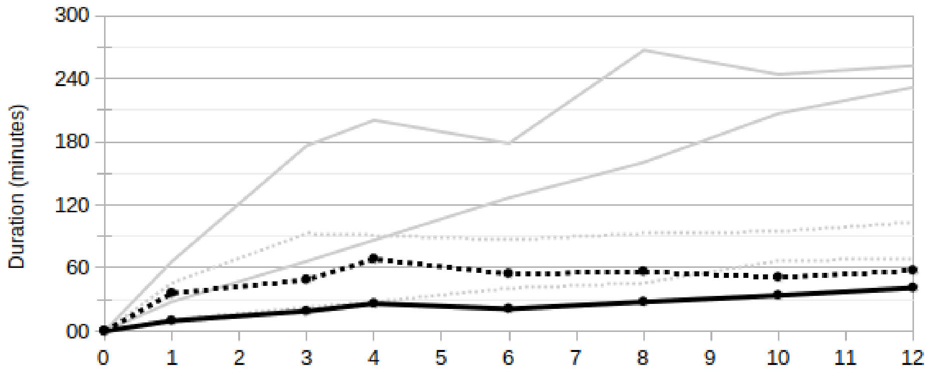
6.3.3. Grep Tests



(a) Impressions



(b) Geriatrix



(c) Compilebench

Figure 17: The duration of Grep tests, generally measured every two hours throughout a twelve hour run of each aging tool. *Solid lines indicate the duration on the artificially aged partition and dotted lines indicate the duration on a defragmented copy of that partition.*

For each aging tool, the time it took to go through the artificially aged filesystem with Grep was measured after generally every two hours of aging. This was also done on a defragmented copy of the filesystem so that the impact of the artificial aging can be assessed by comparing the performance on the aged filesystem with the performance on the defragmented filesystem. The results can be seen in figure 17.

Grep tests were done only every two hours because defragmenting a 64 GiB partition and going through the entire filesystem with Grep both in the aged and the defragmented version could take up to twelve hours for one aging tool. These durations were surprisingly dissimilar from the durations in the paper that proposed Grep tests [CBJ+17]. Their Grep tests almost always took less than a minute per gibibyte.

The results in figure 17 correlate very strongly with filesystem fullness. It might be obvious, but the fuller the filesystem, the longer it takes Grep to go through it. To reduce the impact of fullness and more clearly show the impact of fragmentation, we have also included figure 18, which shows how many times longer the Grep test took for the original filesystem than for the defragmented copy.

Unexpectedly, it consistently took longer to Grep through the defragmented copy of the Compilebench partition than the aged original. We have gone through some efforts to confirm that this was not simply a data analysis mistake or due to wrongly conducted defragmentation. Although the aggregate layout score of the original is consistently around 0.93 and the aggregate layout score of the defragmented copy never went below 0.99997, the defragmented copy is still slower in Grep tests. Grep tests are affected by so many factors that it makes it rather difficult to find the cause of this unusual phenomenon, so we have not been able to find the reason for it.

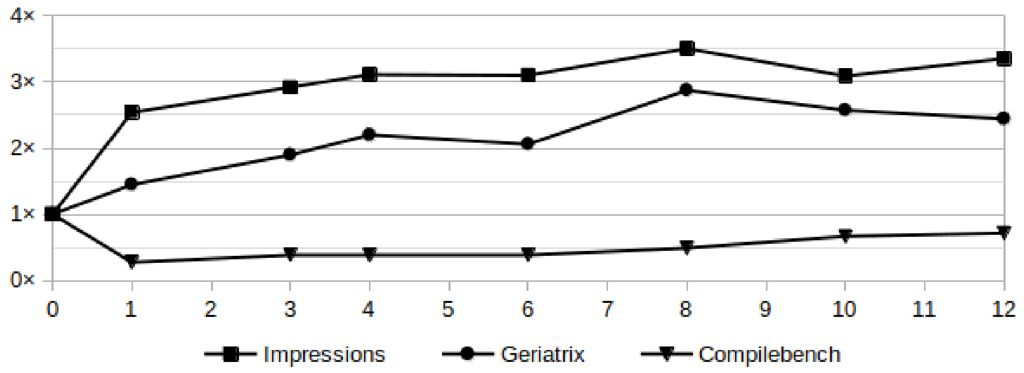


Figure 18: For each artificial aging tool, how many times longer a Grep test takes for the original partition than for the defragmented copy.

Impressions shows a steady growth, where grepping through the fragmented partition consistently takes thrice as long as the defragmented partition. The Grep duration of Geriatrix fluctuates somewhat more but is always the longest of the three. Nonetheless it is only 2.5× as long as the duration on the defragmented copy in the end. The duration on the defragmented copy of the Geriatrix partition very clearly correlates with the fullness of the partition, and stops growing once Geriatrix is done filling up the filesystem.

7. Conclusions

7.1. Artificial filesystem aging tools

Out of the three artificial aging tools being measured, we have found that Impressions appears to generate fragmentation the most efficiently. This is apparent from its aggregate layout score and percentage of files fragmented in table 5, as well as the large difference between its aged and defragmented Grep test scores in figure 18. Compilebench got the lowest aggregate layout score, but strangely enough performs the worst in almost all of the other metrics.

Geriatrix is said in its article to produce large amounts of free space fragmentation. Regrettably, free space fragmentation was not within the scope of this comparison due to time constraints. What we did measure is that Geriatrix produced an immensely high normalized average gap size, so its approach to aging certainly does not produce the most realistic allocation patterns.

Not only Geriatrix was unrealistic. All of the artificial aging tools produced statistical outliers in either out-of-orderness or normalized average gap size. This indicates that the allocation patterns of all the measured artificial aging tools produce are not particularly realistic. In many situations it is not necessarily important how realistic the allocation patterns are. For anyone who simply wishes to generate a filesystem with a high degree of fragmentation, both Impressions and Geriatrix will be satisfactory.

However, because measuring performance on a filesystem with unrealistic allocation patterns and potentially other unrealistic qualities could lead to results that deviate from the performance on authentically aged filesystems, it might be desirable to test at least once with a more realistically aged filesystem. Moreover, none of the aging tools compared in this thesis are appropriate for applications within contexts such as digital forensics research, where realism is crucial. For situations in which a highly realistic artificially aged filesystem is desirable, the artificial aging tools that were included in this comparison will not be satisfactory. We believe that a slower simulation-based approach, such as TraceGen [DHSS21] and the approach suggested by W. Hueting [Hue21], may lead to more realistic filesystem images. Unfortunately we cannot confirm this at the time of writing due to TraceGen not being publicly released as of yet and because Hueting's implementation of his suggested approach is not sufficiently stable.

Furthermore, the following observations were made in regards to the performance of artificial aging tools:

- Most artificial aging tools did not manage to get the aggregate layout score lower than 0.97, which means that they did not fragment the filesystem as much as they were expected to. The tool Impressions specifically failed to age the filesystem as much as it said it would, reaching an aggregate layout score of 0.973 instead of the targeted 0.83.

- All artificial aging tools saw the biggest drop in their aggregate layout score in the first hour of their run. After the first hour the aggregate layout score either stagnated or decreased only slowly.
- Despite the aggregate layout scores being higher than expected, they were still lower than those of most authentic NTFS filesystems, and the percentage of files fragmented was also always higher than on real computers.
- For filesystems generated by artificial aging tools, the distribution of allocations across the storage partition was very different from authentic filesystems, as is apparent from the unusual values for the normalized average gap size.
- Artificial aging tools generate very few empty files (i.e. files with no data) when compared to real filesystems, or even none at all.
- None of the aging tools being tested made any use of features specific to the filesystem format, which somewhat reduces their comparability to real filesystems.

Concerning fragmentation, we have observed that artificial aging tools manage to generate a lot of fragmentation especially in the form of the percentage of files fragmented (section 6.2) and their performance in Grep tests (section 6.3.3). Aggregate layout scores were unexpectedly high but still lower than most real systems.

Concerning realism, we have observed that there is sufficient room to improve upon the realism of artificial filesystem aging tools. We find the observation concerning normalized average gap sizes especially important, since it indicates that the allocation patterns of artificial filesystem aging tools are dissimilar from the allocation patterns that result from years of authentic use of a computer system.

7.2. Data segregation proposal

We found that there is an important distinction between filesystems which are regularly used for general purposes on one hand and filesystems that are only rarely used as well as special-purpose filesystems on the other hand. We suggest that researchers take care to separate these types of filesystems so that more meaningful statistics can be derived for computers that see a significant amount of use. For example, in the analysis of aggregate layout scores in section 5.5 and the analysis of out-of-orderness in section 5.6, it is clearly visible that there is a pattern that applies in general, and a very different second pattern that only applies for filesystems that are not heavily used. Taking the average of these peaks in the data lead to a number that is simply less meaningful than the averages for each type of filesystem individually.

Filesystem fullness, the appearance frequency of NTFS features and the aggregate layout score are good indicators for telling which category an NTFS filesystem belongs to. Unused filesystems will have a very low fullness and special purpose partitions tend to not use special NTFS filetypes, such as sparse files or hardlinks.

8. Future Work

8.1. Improvement of the comparison

The comparison in this thesis could be improved upon in several ways.

As was already mentioned in the conclusions, we did not measure free space fragmentation even though one of the aging tools is said to produce substantially better free space fragmentation in the article it was introduced in. Because of this, adding free space fragmentation as a metric is a clear option for improvement.

Secondly, for determining the quality of an output in regards to its realism, it could be helpful to have more metrics that are specifically intended for making the filesystems generated by artificial aging tools more comparable against authentically aged filesystems. Examples include the number of directories, metric that provide insights into the realism of file contents, statistics of each filetype or more metrics that provide insights into the distribution of files across the storage volume.

8.2. Validating and analyzing unusual observations

Several observations were made that could use a deeper analysis.

In section 5.6 it was found that out-of-orderness is relatively neatly distributed around 0.32 in NTFS. However, we did not find the reason for this phenomenon. A deeper analysis could help unveil the reason why this metric gravitates around this specific number.

In section 6.3.3, it was found that the aged filesystem being generated by Compilebench consistently performed better than a defragmented copy of that filesystem, which ran counter to our expectations. A deeper analysis could uncover the reason for this.

Appendices

A. Additional realistic value range tables

Metric	Realistic value range *based upon partitions from 100 GB to 200 GB*				
	MIN	25%	MEDIAN	75%	MAX
size	101 GB	125 GB	127 GB	127 GB	199 GB
fullness	24%	59%	76%	90%	99%
Files					
number of files	12	187,130	240,927	319,569	746,481
average file size	62 kB	200 kB	228 kB	303 kB	16 MB
% of files ≥ 2 blocks	35%	42%	46%	50%	79%
% of files empty	0%	0.47%	0.78%	2.51%	9.76%
Fragmentation (amount)					
aggregate layout score	0.9357	0.9919	0.9944	0.9982	1.0000
% of files fragmented	0.3%	4%	6%	8%	33%
% of files ≥ 2 blocks frag.	0.6%	9%	14%	16%	57%
Fragmentation (miscellaneous)					
out of orderness	16%	30%	34%	37%	49%
norm. avg. gap size	3.1%	6.6%	7.9%	9.3%	12.9%
NTFS features					
% of files sparse	0%	0.56%	0.80%	0.97%	1.51%
% of files compressed	0%	0.25%	0.42%	0.78%	31.67%
% of files hardlinks	0%	5.09%	7.99%	13.67%	24.91%

Metric	Realistic value range *based upon all partitions above 25 GB*				
	MIN	25%	MEDIAN	75%	MAX
size	27 GB	228 GB	255 GB	983 GB	2000 GB
fullness	0%	21%	43%	74%	100%
Files					
number of files	3	47,278	203,887	318,495	1,548,772
average file size	2 kB	232 kB	381 kB	1,570 kB	135,279 kB
% of files ≥ 2 blocks	11%	46%	51%	65%	97%
% of files empty	0%	0.15%	0.53%	1.34%	19.13%
Fragmentation (amount)					
aggregate layout score	0.92571	0.99891	0.99976	0.99999	1.00000
% of files fragmented	0%	0.09%	0.72%	2.50%	33.33%
% of files ≥ 2 blocks fragmented	0%	0.17%	1.41%	4.66%	57.14%
Fragmentation (miscellaneous)					
out of orderness	0%	28%	33%	39%	100%
norm. avg. gap size	0%	2.4%	6.1%	9.2%	43.1%
NTFS features					
% of files sparse	0%	0.01%	0.12%	0.37%	7.41%
% of files compressed	0%	0%	0.10%	0.33%	31.67%
% of files hardlinks	0%	0%	5.85%	13.95%	28.78%

B. Reflection

This project generally did not go that well. Most of the work I have done has regrettably not been included in the final thesis. This includes approximately thirteen other aging tool experiments, a few aging tools that were not included in the end, as well as a number of avenues of research that turned out to be dead ends. In general, I would say that I had the poor luck during this project of repeatedly betting on avenues of research that turned out to be unfruitful. For instance, I had originally intended to run the experiments on a cloud server, but it was decided that experiments should be run on Windows, and I was not able to rent a Windows cloud instance affordably. Running aging software on my own hardware was not such an appealing prospect, so I eventually ended up buying an external HDD to run the experiments on.

Another example of an unfruitful avenue is the way in which the experiments over time were performed. I had previously spent a lot of time trying to make each of the aging tools run a similar amount of time as the other aging tools. This turned out to be much more difficult than I had expected it to be, and I abandoned that avenue in favor of pausing the aging tools each hour.

I believe I was already a capable programmer prior to starting this project, but because this project mainly revolved around running the aging tools and getting things to work, I feel that I have greatly strengthened other IT skills in which I was never quite as strong. Over the course of this project I have spent a great deal of time with the commandline, shellscripts, Powershell and SSH. Previously I had never had a need to use them quite so thoroughly, but now I feel like I have a decent grasp on them.

Although the initial goal of this project was to compare other aging tools against Hueting's simulator, I ended up not including Hueting's simulator in the final comparison mostly due to time constraints. In its current state, Hueting's simulator is very unstable (crashing after about 15 minutes) and hardly manages to age the filesystem before it crashes. Improving upon Hueting's simulator had been one of the goals of this project, but in the end I did not have sufficient time for it. If it were more stable, Hueting's simulator is assumed to perform somewhat similarly to Compilebench, although somewhat more realistic but significantly slower. Because it tries to simulate only programming, it will very likely create an abnormally large amount of very small files, which is not necessarily realistic.

References

[AAA09] Nitin Agrawal, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. Generating realistic *impressions* for file-system benchmarking. *ACM Trans. Storage*, 5(4):16:1–16:30, 2009. 4, 9, 18

[CBJ+17] Alexander 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. File systems fated for senescence? nonsense, says science! In Geoff Kuenning and Carl A. Waldspurger, editors, *15th USENIX Conference on File and Storage Technologies, FAST 2017, Santa Clara, CA, USA, February 27 - March 2, 2017*, pages 45–58. USENIX Association, 2017. 6, 8, 13, 39

[CKJ+19] Alex Conway, Eric Knorr, Yizheng Jiao, Michael A. Bender, William Jannen, Rob Johnson, Donald E. Porter, and Martin Farach-Colton. Filesystem aging: It's more usage than fullness. In Daniel Peek and Gala Yadgar, editors, *11th USENIX Workshop on Hot Topics in Storage and File Systems, HotStorage 2019, Renton, WA, USA, July 8-9, 2019*. USENIX Association, 2019. 6, 8

[DHSS21] Xiaoyu Du, Chris Hargreaves, John Sheppard, and Mark Scanlon. Tracegen: User activity emulation for digital forensic test image generation. *Digital Investigation*, 2021. 10, 41

[Han] Scott Hanselman. The Real and Complete Story — Does Windows Defragment your SSD? https://www.hanselman.com/blog/the-real-and-complete-story-does-windows-defragment-your-ssd
Note: The author of this article is a Microsoft employee. 29

[Hue21] Wouter Hueting. Feasibility of simulating the java programming process. 2021. 41

[KNG18] Saurabh Kadekodi, Vaishnavh Nagarajan, and Gregory R. Ganger. Geriatrix: Aging what you see and what you don't see. A file system aging approach for modern storage systems. In Haryadi S. Gunawi and Benjamin Reed, editors, *2018 USENIX Annual Technical Conference, USENIX*

ATC 2018, Boston, MA, USA, July 11-13, 2018, pages 691–704. USENIX Association, 2018. 4, 6, 8, 9

[Mas07] Chris Mason. Compilebench, 2007. https://oss.oracle.com/~mason/compilebench/. 4, 9

[Ran69] Brian Randell. A note on storage fragmentation and program segmentation. *Communications of the ACM*, 12(7):365–ff, 1969. 6

[SS97] Keith A. Smith and Margo I. Seltzer. File system aging - increasing the relevance of file system benchmarks. In John Zahorjan, Albert G. Greenberg, and Scott T. Leutenegger, editors, *Proceedings of the 1997 ACM SIGMETRICS international conference on Measurement and modeling of computer systems, Seattle, Washington, USA, June 15-18, 1997*, pages 203–213. ACM, 1997. 8, 9, 12, 28

[vdMJD$^+$19] Vincent van der Meer, Hugo Jonker, Guy Dols, Harm M. A. van Beek, Jeroen van den Bos, and Marko C. J. D. van Eekelen. File fragmentation in the wild: a privacy-friendly approach. In *IEEE International Workshop on Information Forensics and Security, WIFS 2019, Delft, The Netherlands, December 9-12, 2019*, pages 1–6. IEEE, 2019. 11, 16, 20

[vdMJvdB21] Vincent van der Meer, Hugo Jonker, and Jeroen van den Bos. A contemporary investigation of NTFS file fragmentation. *Digital Investigation*, 2021. 8, 11, 14