Forensic Science International: Digital Investigation 33 (2020) 301008

Contents lists available at ScienceDirect

Forensic Science International: Digital Investigation

journal homepage: www.elsevier.com/locate/fsidi

DFRWS 2020 USA — Proceedings of the Twentieth Annual DFRWS USA

An Empirical Study of the NTFS Cluster Allocation Behavior Over Time

Martin Karresand [a, b, *], Geir Olav Dyrkolbotn [a, c], Stefan Axelsson [d]

[a] Norwegian University of Science and Technology (NTNU), Norway
[b] Swedish Defence Research Agency (FOI), Sweden
[c] Norwegian Defence Cyber Academy (NDCA), Norway
[d] Halmstad University, Sweden

ARTICLE INFO

Article history:

Keywords:
Digital forensics
File carving
Cluster allocation pattern
Allocation algorithm
NTFS

ABSTRACT

The amount of data to be handled in digital forensic investigations is continuously increasing, while the tools and processes used are not developed accordingly. This especially affects the digital forensic sub-field of file carving. The use of the structuring of stored data induced by the allocation algorithm to increase the efficiency of the forensic process has been independently suggested by Casey and us. Building on that idea we have set up an experiment to study the allocation algorithm of NTFS and its behavior over time from different points of view. This includes if the allocation algorithm behaves the same regardless of Windows version or size of the hard drive, its adherence to the best fit allocation strategy and the distribution of the allocation activity over the available (logical) storage space. Our results show that space is not a factor, but there are differences in the allocation behavior between Windows 7 and Windows 10. The results also show that the allocation strategy favors filling in holes in the already written area instead of claiming the unused space at the end of a partition and that the area with the highest allocation activity is slowly progressing from approximately 10 GiB into a partition towards the end as the disk is filling up.

Introduction

The amount of data to be handled during digital forensic case work is rapidly increasing and is a major challenge to the digital forensic community (Quick and Choo, 2014b). The problem has been of concern to the digital forensic field for many years (Gladyshev and James, 2017; European Police Office (Europol), 2016; Quick and Choo, 2014a; Breitinger et al., 2013; Roussev, 2012), but the problem has not yet been solved.

Casey (2018) and also Karresand et al. (2019a, b, c) have independently suggested to use the inherent structures in the stored data to improve the digital forensic process. The principle builds on taking advantage of the pattern introduced by the allocation algorithm and in that way improve for example the efficiency when rebuilding files, extracting temporal information (time stamps) from raw data and direct searches to the areas most likely to contain important (user related) data.

The principle is especially valid for the digital forensic sub-field of *file carving*, which is used in digital forensic investigations when there is no file system available in the investigated media. The file carving process is based on using only the properties of the stored data itself (Poisel and Tjoa, 2013; Pal and Memon, 2009). File carving is highly valuable to the digital forensic investigator, but computationally intensive to perform, hence much effort is put into mitigating the increasing amounts of data by different means. In a survey by Quick and Choo (2014b) the following concepts to decrease the amount of work needed to be done are listed; data mining, data reduction and subsets, triage, intelligence analysis and digital intelligence, distributed and parallel processing, visualization, digital forensics as a service (DFaaS) and different artificial intelligence techniques.

The foundation of digital forensics is to use the inherent structures of data, but the idea to use the inherent structures introduced by the storage process in stored data is rather new and not yet fully investigated. Therefore the actual behavior of the allocation algorithm has to be found for any relevant file system. The behavior of file systems from the open source field can be found by studying their code base, but for closed source operating systems (OSs) the behavior is best found by empirical studies of the allocation behavior in experiments and the real world. The currently most

* Corresponding author. Norwegian University of Science and Technology (NTNU), Norway.
E-mail address: martin.karresand@ntnu.no (M. Karresand).

https://doi.org/10.1016/j.fsidi.2020.301008

popular OS is Windows, which has almost 86% of the market share (Net Applications.com, 2019). Windows uses New Technology File System (NTFS) as the default file system and we have therefore chosen to study the allocation behavior of NTFS in recent versions of Windows.

The aim of the project is to gain knowledge on the allocation behavior of NTFS in different modern versions of Windows (versions 7 to 10) and especially if and how the behavior changes over time. This includes the adherence to the *best fit* Carrier (2005) allocation strategy and if the allocation activity is evenly spread over the (logical) addresses of the storage area. We will also test whether the allocation algorithm of Windows and NTFS is version and/or size dependent.

To be able to answer the research questions we have executed an experiment where we used a weighted random distribution to write, expand, shrink and delete files in four different versions of Microsoft Windows (7, 8, 8.1 and 10). To be able to find similarities and differences between the Windows versions we ran eight instances of the same virtual machine for each Windows version. Four of the machines for each Windows version used the same file operation pattern, also referred to as the *standard pattern*, and the other four machines ran unique patterns. The virtual machines used 64 GiB disk, but we also set up three extra virtual machines with 256 GiB disks to detect any differences in allocation behavior due to larger disk sizes. These virtual machines used a modified version of the standard pattern, where each file operation was increased by a factor of 4.6 to compensate for the larger disk size. Every operation was followed by the extraction of the current cluster allocation status taken from the $Bitmap file in the Master File Table (MFT). The allocation status was then used to find the difference in cluster allocation between each operation. The experiment was set to run 10,000 iterations in each virtual machine.

The rest of this paper is organized as follows: The remaining parts of Section Introduction presents related work and our contributions. In Section Experiment we describe the experimental platform and how the experiment was implemented. Section Results presents the result of the study. In Section Discussion we discuss the effects and implications of our result to the research field of digital forensics and especially file carving. Section Conclusion and future work concludes the work and presents ideas of future work to be done.

Background

Silberschatz et al. (2012) describes the theory of file system construction. A file system keeps track of data stored on secondary storage and is organized in different ways. However, all implementations share some common properties; the addressing of the physical storage is abstracted by the file system into logical addresses and the position of the stored data is determined by an allocation algorithm.

Most modern file systems use an index allocation strategy to keep track of the data on disk. The index allocation strategy separates the metadata and the file data and hence the index itself does not suffer from external fragmentation (free holes being to small to be filled with new data), but the data part can if heavily used give rise to fragmented files, requiring regular defragmentation of the file system. There is also a risk of disk space being wasted when using index allocation, especially for small files requiring a full index meta data block to hold just a few index posts.

There are also a number of algorithms used for handling the free space in the data part that is to be populated by new files. The *best fit* algorithm is meant to reduce the risk of file data fragmentation by always utilizing the free space best fitting the file to be written. The idea is to reduce the free space remaining in a block of free clusters due to large differences in what is needed and what is available. However, this strategy requires all the free spaces available to be compared at each file operation before the best fit can be chosen.

NTFS (Microsoft Windows) is using a index allocation strategy (Microsoft, 2018; Hughes, 2009), the index is called the MFT (Microsoft, 2018) and the individual posts are called MFT records. The problem of space being wasted when using index allocation is solved by storing the data of smaller files (up to approximately 700 B[1]) in the MFT records themselves. To allocate space for file data NTFS uses a best fit allocation strategy (Carrier, 2005).

When formatting an NTFS partition 12.5% of the space is reserved for the MFT as default (Microsoft, 2018). The MFT records are 1 KiB in size and usually the size of the smallest allocatable unit (called cluster) in NTFS is 4 KiB. The allocation status of every cluster in the file system is stored in the $Bitmap file, which is record number 6 in the MFT. Each bit in the $Bitmap file represents one cluster in ascending Logical Block Addressing (LBA) order. If a cluster is allocated the corresponding bit in the $Bitmap file is set to 1, hence 0 represents a free cluster.

A file can either be written as a stream or as one large block at once Karresand et al. (2019a). In the first case the OS does not know the final size of the file and therefore cannot optimize the allocation accordingly. This often leads to file fragmentation, but the behavior is partly mitigated by the internal buffering of the OS. Writing a file in one piece gives the OS information on its size and it can therefore optimize the storage by using its standard allocation algorithm. This behavior is probably more common when dealing with smaller files that easily can be held in Random Access Memory (RAM), than for large files. The specific write behavior is software dependent and might incorporate temporary writing of files to protect the data in case of a power loss or hardware failure.

Related work

Since the research area is new there is not much related work to be found. We have therefore also included work from the file carving area containing some material related to the allocation algorithm of file systems, as well as work related to the placement of data on disk.

We have presented the novel idea of creating a map of the probability of finding user data at different LBA positions of hard disk partitions in three earlier articles (Karresand et al., 2019a, b, c). In the first article (Karresand et al., 2019c) we tried to find any static areas of NTFS partitions. We defined static areas as areas containing the same data at the same logical position in different partitions. During that work we also found that the $MFT started exactly 3 GiB into the NTFS formatted partitions of the over 30 unrelated real world hard drives we looked at. The hard drives had different sizes and contained different versions of Windows (Karresand et al., 2019c). We also used the $Bitmap file to study which part of an NTFS partition had the highest allocation activity (Karresand et al., 2019b). The highest activity was found approximately 10 GiB into a partition. The activity then slowly decreased towards the end of the partition, which was expected. In the third article (Karresand et al., 2019a) we presented new information on the detailed behavior of the allocation algorithm in Windows using NTFS in different file writing situations (writing a file like one block or writing it as a

[1] The maximum size of an internal $Data attribute varies depending on the size of other attributes stored in the MFT record. Most sources give a maximum internal $Data attribute size of 600–700 bytes. Microsoft reports a 900 byte limit (Microsoft, 2018).

stream of data). The results showed that the size of the allocated blocks (i.e. file fragments) decreased during block writing and increased during stream writing. The study was however based on a low number of writing operations, Windows versions and partition sizes. We will therefore validate our results in a larger experiment.

Casey (2018) introduces a new digital forensic research field called digital stratigraphy that draws inspiration from archaeology, which share many common features with (digital) forensics. The idea is to look upon file system activities as layers (strata) that are structured. This structure can be used to complement, improve and expand the information currently retrieved from hard drives and disk images. In the article Casey (2018) shows how the next fit allocation strategy used in File Allocation Table (FAT32) file system (for example FAT16 and FAT32 in Windows) clearly indicates the order of storage of file data. He also touches upon the best fit allocation algorithm used in NTFS and how it behaves, but the bulk of the NTFS part is focused on specific behavior regarding valid data length (VDL) slack that is currently not properly covered by available digital forensic tools. He also covers the effect file tunneling has on the reliability of file system meta data.

Key (2012) has developed an EnScript module to the EnCase software which creates a map of the recoverable sectors of a file found in a file system. It can handle situations where other tools do not work, for example partially damaged files. It is very processor intensive and therefore can only create maps of a few files at a time. Key does not mention to what extent any knowledge on the allocation pattern of the OS and file system is used in the article.

Gladyshev and James (2017) have studied the problem of file carving from a decision-theoretic point of view. They suggest a model where storage media is sampled with a frequency based on different properties of the hard disk and the file type that is to be found. In some specific situations their carving model outperform standard linear carving algorithms, but their solution was not generally applicable at the time of writing. Gladyshev and James (2017) mention using the distribution of data on disk, but do not explain if they take advantage of any structures introduced by the allocation algorithm.

In two articles by van Baar et al. (2014) and van Beek et al. (2015) outlining the DFaaS system Hansken (van Beek et al., 2015) and its predecessor Xiraf (van Baar et al., 2014) the concept of non-linear extraction of data from images is discussed. Both van Baar and van Beek suggest that the MFT records of an NTFS partition are extracted first. The MFT records are then used to find other interesting areas of the file system. Van Baar and van Beek also suggest that the analysis process is used to influence the imaging process by having specified parts being prioritized. As we understand they do not base the priority on the allocation pattern of the analyzed system, but on file name and other higher level meta data found in the MFT records.

Jones et al. (2016) have created a framework to enable studies of (deleted) file persistence in storage media. They use differential forensic analysis to compare snapshots of file systems in use and follow the decay of deleted files over time. This work connects to our experiments, because free areas are meant to be reused by the file system, but depending on the size of the free area the best fit allocation algorithm might not be able to use it for a certain amount of time. The concept of data persistence is of interest to us because the persistence at different areas of storage media indicates that these positions are not reused. This information might correlate with the allocation pattern and its development over time, which is what we are studying.

\def\rm{\tf="Times New Roman (TrueType)"}\def\it{\tf="Times New Roman Italic (TrueType)"}\def\bf{\tf="Times New Roman Bold (TrueType)"}\def\bi{\tf="Times New Roman Bold Italic (TrueType)"}

Fairbanks and Garfinkel (2012) present 12 factors affecting data persistence in storage media. Fairbanks (2015, 2012) also has described the low-level functions of fourth extended filesystem (ext4) and their effect on digital forensics. Although the articles do not describe the inner workings of NTFS the principle is still of interest to us, especially in future extensions of our experiment.

Our main contribution to the digital forensic research field is the result showing that there are differences in the allocation behavior of Windows 10 and older versions of Windows and that the best fit allocation strategy is not fully used. The maximum and median fragment sizes of Windows 7 show interesting linear properties, which are not found in Windows 10. There are also areas within the file system that are rarely used, forming bands of low allocation activity through the file systems. We also show how the allocation of new data is concentrated to an area close to (just before) the middle of a partition, but also how that area is slowly moving towards the middle of the partition, regardless of the size of the partition. The result can be used to determine the sequential order of files, estimate the proper size of file fragments to be carved and where in the create and erase cycle a file system is through the leeward effect found in all Windows versions. The results can also be used to improve the efficiency of the file carving process by helping the digital forensic investigator to prioritize where to start searching for user related data.

Experiment

The experiment was based on iteratively creating, deleting, expanding and shrinking files in unused NTFS formatted partitions in 32 virtual machines running Windows versions 7, 8, 8.1 and 10. The aim was to empirically study how the cluster allocation pattern develops over time and how the allocation frequency varied at different LBA positions. Each file operation iteration contained the following elements:

1. Boot the virtual machine.
2. Based on a precomputed list either create, delete, expand or shrink a file within the virtual machine's NTFS file system.
3. Shut down the machine.
4. Extract the $Bitmap file from the virtual hard disk (using dd from the host)

Since each file operation iteration required the virtual machine to be rebooted a full iteration took several minutes to complete. There were also extra time slots inserted at critical moments to compensate for any variations in execution time during an iteration.

We allowed the experiment to run for 16 days before shutting it down due to time constraints. Most, but not all, of the virtual machines then had completed 10,000 iterations. All virtual machines were run in parallel in a computer cluster to save time, if we had had to run them one-by-one the experiment would have taken up to $35 * 16 = 560$ days to execute, excluding setup time.

The foundation of the experiment was built on having four virtual machines installed with one Windows version each using standard parameters. Then a Python 2.7 execution environment was installed together with the file operation scripts. An autostarted.bat script was placed in the virtual machine to check when the boot sequence was finished. The script was small enough to fit into an MFT record and hence did not require any new cluster allocation outside the MFT. The path setting was modified and the security level of Windows was lowered to allow logging in without a password. The goal was to keep the NTFS file system as pristine as possible to allow us to study the allocation algorithm from the start of the life of the file system. There were however a number of

system processes that also modified the file system in each iteration, which were beyond our control.

We let 16 virtual machines (four machines for each version of Windows, half of the total amount of machines) use exactly the same file operation pattern (the standard pattern) to test if there was any deterministic behavior connected to the allocation (if any similarities of the allocation patterns could be found). Hypothetically it should be, since the virtual machines within each Windows version were exact copies of each other.

To be able to see any deterministic allocation behavior we used scp when distributing the virtual machines to the computer cluster nodes. We did not use the VirtualBox clone function because it make small changes to the virtual machine's settings, which in turn might affect the OS and hence trigger an unintended write operation. Using scp to copy the virtual machines guaranteed them to be identical, which was verified using Secure Hash Algorithm 1 (SHA-1) hash summing. However, we did not have full control of the cluster allocation and deallocation during an iteration of the experiment because of internal OS processes, thus we still had uncontrolled variables affecting the outcome of the standard pattern sub experiment.

Due to instability in the VBoxManage interface and unforeseen popup windows appearing in the virtual machines several of them had to be restarted during the course of the experiment. This might have affected the result of the experiment, but since we used at least four virtual machines for each combination of Windows version and partition size in the experiment the effects of the unplanned reboots were diminished.

The experiment was run on eleven nodes in a large computer cluster. The cluster is managed by Swedish Defence Research Agency (FOI) and we therefore were not allowed to make any changes to the cluster nodes' OS or configuration, which forced us to use alternative tools to extract the data. This did however not affect the experimental results.

Each cluster node ran four virtual machines, one for each version of Windows in our test (see Table 1). The $Bitmap file from the MFT of NTFS was used to check which clusters were affected by each file operation. To enable us to extract the $Bitmap file after each operation the virtual machines were configured to use fixed size disks, which can be directly handled by common Linux tools. We limited the size of the fixed virtual disks to 64 GiB to be able to use four virtual machines in each node and still have space for the $Bitmap file copies. Each copy was 2 MiB large and there would be 40,000 $Bitmap copies (over 78 GiB) in each cluster node when the experiment was finished. If we had used larger virtual disks we would have had to decrease the number of virtual machines, which in turn would have affected the reliability of the results. The hard drive size of 64 GiB was therefore found to be a reasonable trade-off between reliability and a realistic hard drive size.

The virtual machines used four internal Python scripts for the experiment, one for each type of file operation. The scripts and the resulting $Bitmap files were placed in a external folder shared with the host, one for each machine, to isolate the machines from each other. This also meant that we avoided cluttering the virtual disk with data and therefore minimized the risk of unspecified behavior due to several machines accessing the same file at the same time. The file operations were executed as the local user of the virtual machines to simulate the activity of a real user.

Table 1
The four versions of Windows used in our experiment.

Name	Version
Windows 7 Professional SP 1	7601
Windows 8 Enterprise	9200
Windows 8.1 Enterprise	9600
Windows 10 Enterprise	1703

The execution of the experiment was controlled by a Python script on the host node. The script selected one of four actions; *create*, *delete*, *increase* and *decrease* based on a configuration file containing a precomputed weighted random selection. The selection was biased towards file creation and extension, where $\frac{1}{4}$ of the operations were set to create, $\frac{9}{40}$ to erase, $\frac{11}{40}$ to increase and $\frac{1}{4}$ to decrease. The process was set to create files until the disk was 30% full and then switch back to either erase- or create-only mode if the usage of the disk reached above 95% or below 5%. The communication between the host script and the virtual machine scripts was done using the VBoxManage interface.

The experiment emulated a file sharing or multimedia consuming user that alternated his or her file operations between small and large files. The size of the small files varied between 4 KiB and 4 MiB and the size of the large files between 1 MiB and 1 GiB. The size of the large files might not seem very large, but since the virtual disks were only 64 GiB in size a 1 GiB file corresponds to a 32 GiB file on a 2 TiB disk.

All write operations were stream writing operations, i. e. the OS of the virtual machine did not know the size of the file to be written in advance, which was meant to represent for example a file being downloaded from the internet. However, stream writing operations might give a more fragmented allocation result than block writing operations (Karresand et al., 2019a).

The script responsible for managing the write operations in the virtual machines on a host node checked if the currently active virtual machine was started before it sent the file operation command. There was also a check of the completion of a file operation, as well as a check of the exit status of the virtual machine when it was being shut down. The exit status of each file operation was also checked and if it indicated an error the transaction counter was decremented and the same operation was retried. Every transaction was logged in a file indicating the sequence number, the action performed, the name of the affected file and its current size.

The three Python scripts that executed write operations on the virtual machines were set to write the iteration sequence number and a individual sequence number into every 512 byte sector of the file to be written. This enabled us to see the raw write pattern in the virtual disk file if ever needed. The iteration sequence number was also given as file name to further increase the traceability. The *create* and *decrease* file scripts both wrote new files (using the wb flag in the Python open command). The *increase* script appended new data at the end to an existing file, using the ab flag. Therefore increased files could contain multiple number sequences.

To avoid unnecessarily burdening the virtual machines the four file operation scripts run by the virtual machines were kept as simple as possible and most of the control functions (for example the status check of the virtual machine and file operation) were executed by the main script on the host node. The randomization of the file operations was done beforehand and held in a configuration file used to control the script on the host. In that way the same file operations could be executed on several machines in parallel. This also avoided the problem of having to individually seed several random functions, now the seeding was centralized.

Since we were not allowed to install any software on the host nodes in the computer cluster we chose to use the dd tool to extract the $Bitmap file in each iteration. That required us to know the exact location of the $Bitmap file in advance, which we solved by using fixed size virtual disks. The size and location of the $Bitmap file would not change since the size of the disk was static. A better solution might be to use the icat tool from the Sleuth Kit by Carrier

(2014). That will be fixed in the next version of our experimental platform.

When the experiment was finished we extracted the LBA position of the affected clusters in each file operation from the $Bitmap copies. The extraction process gave us information on all clusters that had been allocated or freed during each file operation. Since we could not control the behavior of the OS any allocation changes induced by the OS were also included.

To test if there were any differences in allocation behavior connected to the size of the hard drive we extended the experiment with three virtual machines having 256 GiB hard drives. The machines were installed with Windows 7, 8.1 and 10 in the same way as the 64 GiB machines. The standard pattern was used to enable comparison to the 64 GiB machines using the same pattern. Due to the 256 GiB machines being started later than the 64 GiB machines the Windows 10 machine only executed 8331 iterations before being stopped. We therefore have limited the result to the first 8331 iterations in all machines.

Result

To increase the readability of the paper we have chosen to only show graphs for Windows 7 and 10. The differences between the graphs for Windows 7, 8 and 8.1 are often small and we therefore let Windows 7 represent all three Windows versions below Windows 10. We can of course use data from all three versions in the same graph, but that will decrease the visibility of the specific features we want to show, because the data are not equal, only similar.

Please observe that the figures are showing the statistical properties of the allocation patterns, not the actual allocations for each file operation. Showing the actual allocations would require us to plot up to hundreds of thousands of data points for each file operation, which obviously is not feasible in this publication format. We also use different scales (log and linear) on the Y-axis of the plots to increase the visibility. The maximum allocatable position of a 64 GiB partition is almost 17,000,000 clusters and approximately 67,000,000 clusters for 256 GiB partitions. Since some of the figures use different units the maximum value of their Y-axes might differ.

In many graphs there seems to be a disturbance visible as an area of low activity centered around file operation 5350. This effect comes from the rapid decrease in file system utilization that can be seen in for example Figs. 1 and 2, where the thick black curve at the top of each graph shows the degree of utilization. Please observe that the utilization curve has been moved upwards with approximately 2,500,000 clusters to increase visibility.

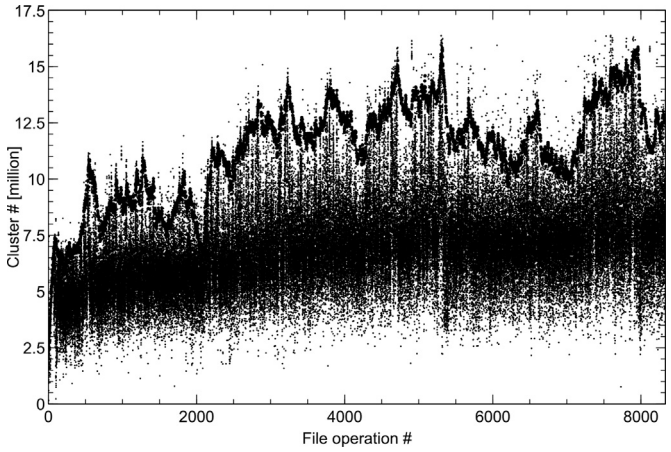

Fig. 1. The mean allocation position for all included Windows versions in 64 GiB hard drives using the standard file operation pattern. We have also included a plot of the file system utilization, which corresponds to the black line at the top of the graph. The file system utilization plot is raised by 2,526,780 clusters to increase the visibility.

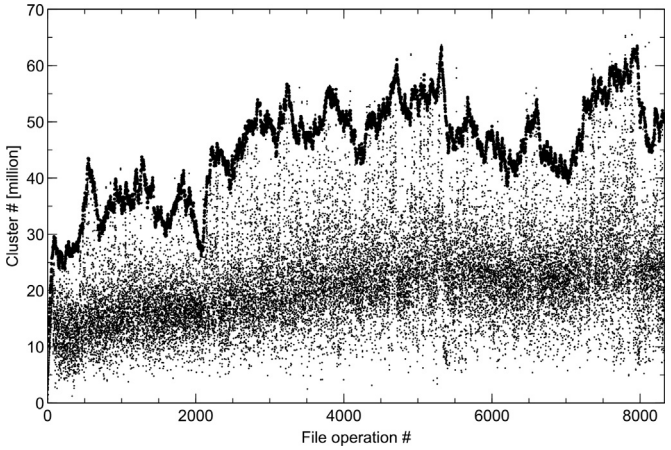

Fig. 2. The mean allocation position of the Windows 7, 8.1 and 10 having 256 GiB hard drives. We have also included a plot of the file system utilization (multiplied by 4 to fit the larger disk size) to enable comparison with the corresponding 64 GiB hard drives. Please observe that the maximum allocatable position in a 256 GiB partition is approximately 67,000,000 clusters and that the scale of the Y-axis therefore differs from the corresponding plots of the 64 GiB partitions.

As can be seen in Fig. 1 the mean position of the newly allocated clusters for each file operation in the 64 GiB virtual main partitions correlates with the amount of allocated clusters in the file system, i.e. the degree of utilization of the file system. The mean allocation position patterns are similar for all four Windows versions, but not equal. Each partition in the experiment adds a few unique outliers to the graph. The file system utilization plot, derived from the standard pattern file operations configuration file, added on top of the mean position graph has been raised by 2,526,780 clusters (approximately 9.6 GiB) to increase its visibility. The value represents the difference between the maximum value of the standard pattern file operations configuration file and the maximum allocated cluster position of one of the Windows 7 machines using that file.

The correlation between the mean position and the utilization of the file system shown in Fig. 1 also appear in the main partition of the 256 GiB virtual disks, regardless of the installed OS. This is shown in Fig. 2. The included plot of the file system utilization is multiplied with 4 to compensate for the larger hard disk size and also to increase the visibility. As can be seen the bulk of the mean allocation positions in the 256 GiB disks, as well as the highest mean allocation values, correlate well with the mean allocation positions in the 64 GiB disks.

The maximum allocation position is an indication of how the OS utilizes the free area at the end of the file system. This is shown in Fig. 3. As can be seen the highest allocated position for each file operation also increases as the number of operations increases. The increase is divided into steps, which are correlated to increases in the utilization of the file system. There are however no corresponding rapid decreases in the maximum positions when the utilization decreases. Instead the current level is only slowly decreasing until the utilization gets a new maximum value. In the plots the effect looks like the formation of clouds on the leeward of a mountain range. Although the effect is strictly visual and has nothing to do with how physical clouds are formed, we will be referring to the effect as the "leeward effect" in the rest of the article.

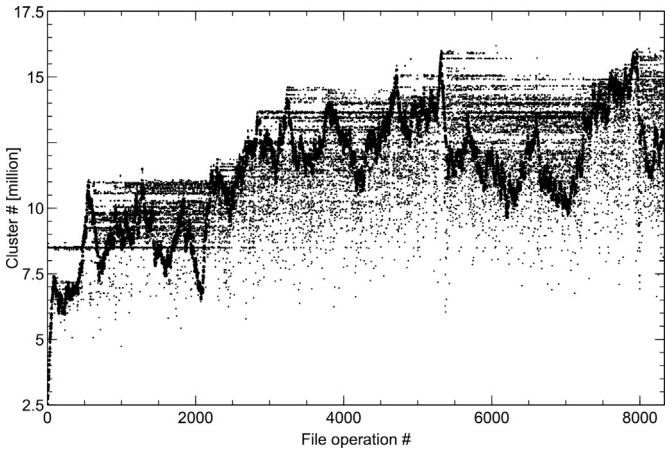

Fig. 3. The maximum allocation position for Windows 7 in 64 GiB hard drives using the standard file operation pattern. We have added the file system utilization curve (the black line at the top) to the graph to increase the visibility of the leeward effect of the allocations.

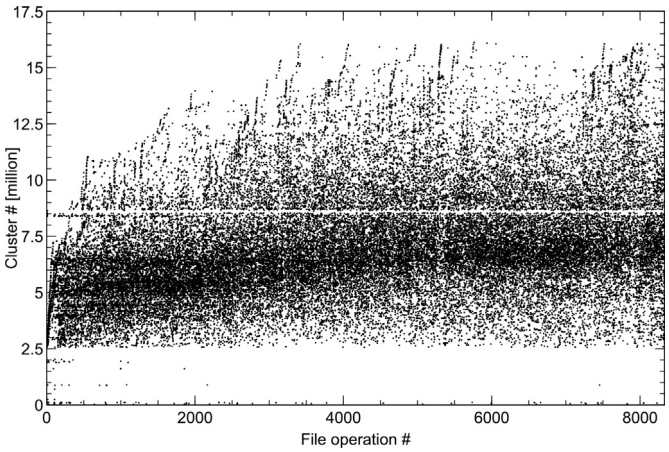

Fig. 5. The median allocation position for Windows 7 in 64 GiB hard drives using the standard file operation pattern. Please observe the horizontal sparse part in the middle, which is missing in Windows 10.

In Fig. 4 the maximum allocation position for Windows 10 is shown. The leeward effect is less distinct here, instead the maximum allocation positions remain at the same level until the next increase in the file system utilization, making the plot look more like heavy fog than leeward clouds. The larger amount of allocations at high cluster addresses is also manifested by the lower amount of allocations at positions below the file system utilization curve.

The median allocation position graph of Windows 10 lacks a feature that the graph of the older Windows versions show (see Fig. 5). Windows 7, 8 and 8.1 all have an approximately 100,000 clusters wide unused area in the middle of their partitions centered around cluster 8,600,000 for Windows 7 and 8,400,000 for Windows 8 and 8.1. The area is more or less visible for all three Windows versions, but in Windows 7 it is visible from the start of the file operations (see Figs. 3 and 5) from a significant lower bound of the unused area, which is not the case for Windows 8 and 8.1.

The graph (see Fig. 6) of the statistical mode, here defined as the middle position of the largest consecutive group of clusters allocated in a file operation, of Windows 7 also shows an unused area in the middle of the partition, which can also be seen in Fig. 5. The allocated positions in the mode graph are however almost evenly distributed in the allocated area and also show a sharp border to the sparsely allocated area between cluster position 125,000 and 2,550,000. This border is less sharp in Fig. 5, but that might be an effect of the median being a calculated value in difference to the mode being a factual value. Hence the mode value is closer to the actual behavior of the file allocation algorithm. As for the median allocation position graph in Fig. 5 the unused area in the middle of the Windows 8.1 partitions start to vanish around file operation 9000 and is not present in Windows 10.

The Windows 10 mode graph in Fig. 7 is similar, but not equal, to the Windows 7 graph in Fig. 6. Both graphs show data from the virtual machines using the standard file operation pattern. The unused area close to the middle of the partition is lacking in Fig. 7 and there is more allocation activity at the first part of the partition. The Windows 10 graph also shows how the highest allocated positions are reused after their initial allocation to a higher degree than in Windows 7 (see Fig. 6). This is manifested by the higher amount of leeward effect in Fig. 7 (the peaks are not as visible in the Windows 10 plot as in the Windows 7 plot). Since the figures show the statistical mode of the allocation for a file operation each data point corresponds to the middle of the

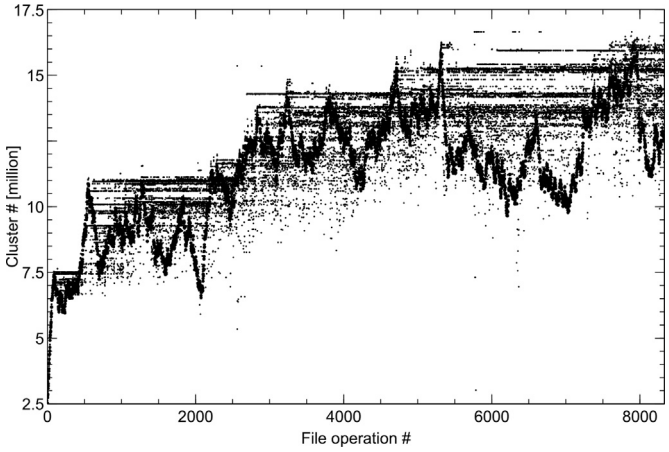

Fig. 4. The maximum allocation position for Windows 10 in 64 GiB hard drives using the standard file operation pattern. We have added the file system utilization curve (the black line at the top) to the graph to increase the visibility of the leeward effect of the allocations.

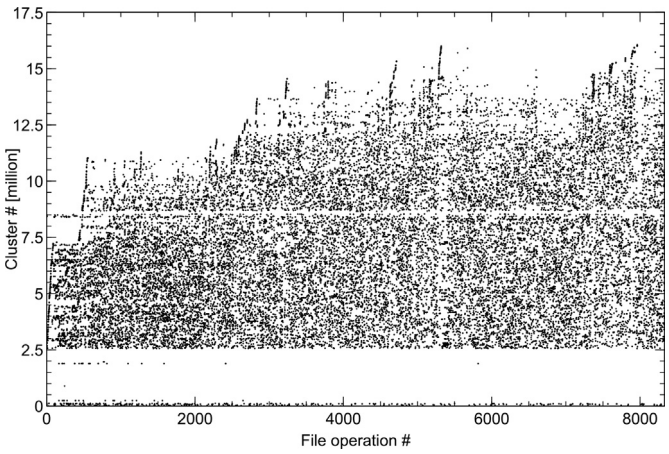

Fig. 6. The mode allocation position for Windows 7 in 64 GiB hard drives using the standard file operation pattern. Please observe the thin horizontal sparse area in the middle of the plot, the same sparse area can be seen in Fig. 5.

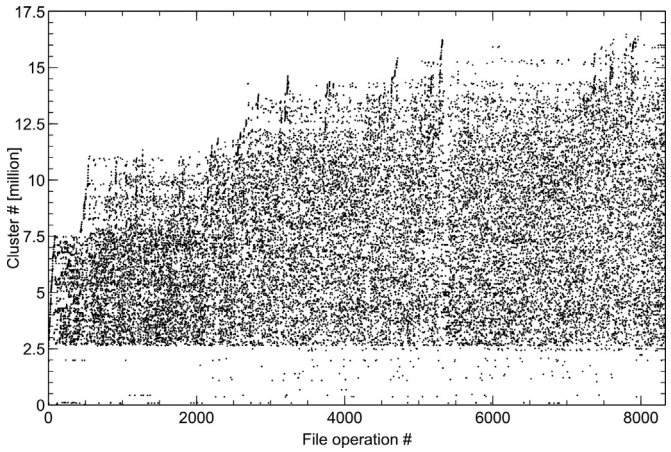

Fig. 7. The mode allocation position for Windows 10 in 64 GiB hard drives using the standard file operation pattern.

largest consecutively allocated area of that file operation, thus they show real allocations.

The sparsely allocated area below cluster 2,550,000 (see Fig. 6) has a denser allocation pattern for Windows 8 and 8.1, than for Windows 7. The order of usage from sparse to dense for that area is Windows 7, 10, 8 and 8.1. The size of the sparse area is the same in the 256 GiB hard disks, hence it does not represent the 12.5% of the volume size set aside for the MFT. However, all areas contain the MFT, which starts exactly 3 GiB into the main partition in all hard disks, regardless of size and version of Windows (Karresand et al., 2019c). When checking the allocation of every 50,000 cluster in the sparse area we found that almost all of the files are OS related files and no more than 5% of the files are created by the scripts.

The standard deviation value of the allocated positions after each file operation is high, between 2,000,000 and 3,000,000 clusters, and is rapidly increasing at the beginning (up to approximately 500 file operations), where it levels out. This can be seen in Fig. 8. The rapid increase at the beginning of the graph is due to the large contiguous area of free space when the disk is newly formatted (when the best fitting area available for allocation is

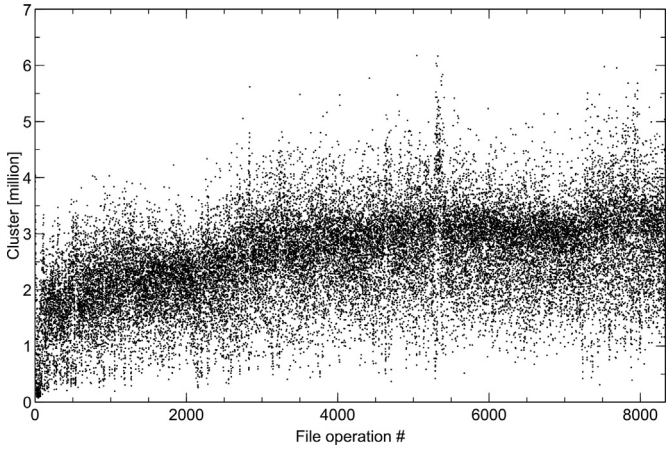

Fig. 8. The standard deviation of the allocation position for Windows 7. The four 64 GiB partitions from the experiment using the standard file operation pattern are included. Since the standard deviation is measured in clusters, not cluster number (position), the Y-axis does not show the full size of a 64 GiB partition.

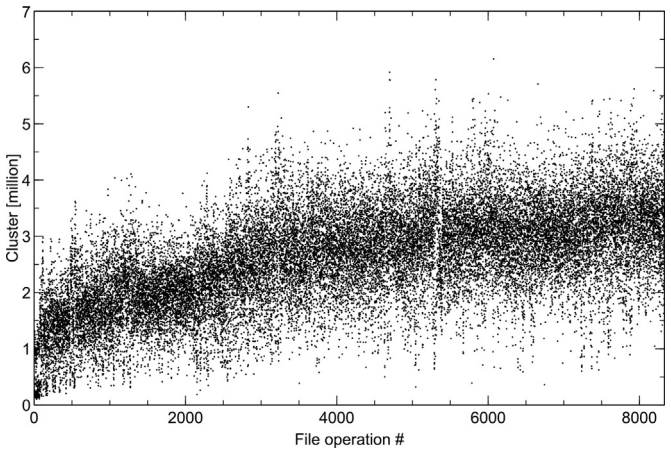

Fig. 9. The standard deviation of the allocation position for Windows 10. The four 64 GiB partitions from the experiment using the standard file operation pattern are included. Since the standard deviation is measured in clusters, not cluster number (position), the Y-axis does not show the full size of a 64 GiB partition.

much larger than the required space). Please observe that standard deviation is measured in clusters, not cluster number (position) and therefore the Y-axes of the standard deviation graphs do not show the full size of a 64 GiB partition.

Worth noticing is that the Windows 10 standard deviation, which can be seen in Fig. 9, is more dense and less varied than for Windows 7. On the other hand it does not level out to the same degree as for Windows 7. Neither the standard deviation graph in Fig. 8 nor the graph in Fig. 9 change much if we include allocation data from all file operation patterns.

We also collected statistics on the file fragments (groups of allocated clusters) during the experiment. Three metrics are worth noticing; the number of fragments, as well as the maximum and median size of the fragments. Please observe the log scale of the Y-axis in the file fragment graphs (Figs. 10–15).

The number of fragments is an indicator of the allocation algorithm's priority regarding filling holes versus keeping file data contiguous. Fig. 10 shows that in Windows 7 most of the file operations (using the standard file operation pattern) generate approximately 20 fragments. As can be seen the number of

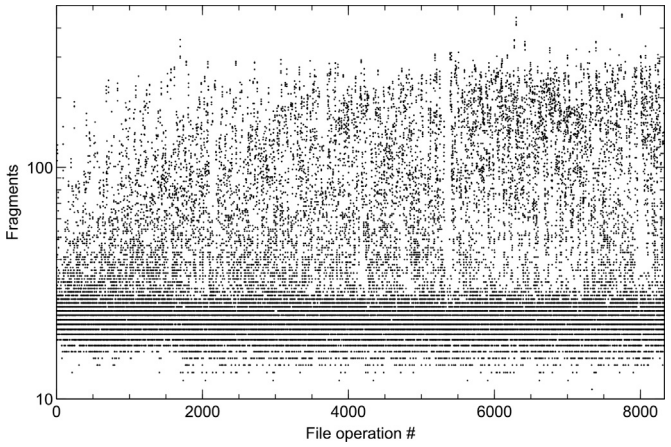

Fig. 10. The number of fragments allocated in each file operation for Windows 7 in 64 GiB hard drives using the standard file operation pattern. Please observe the log scale of the graph.

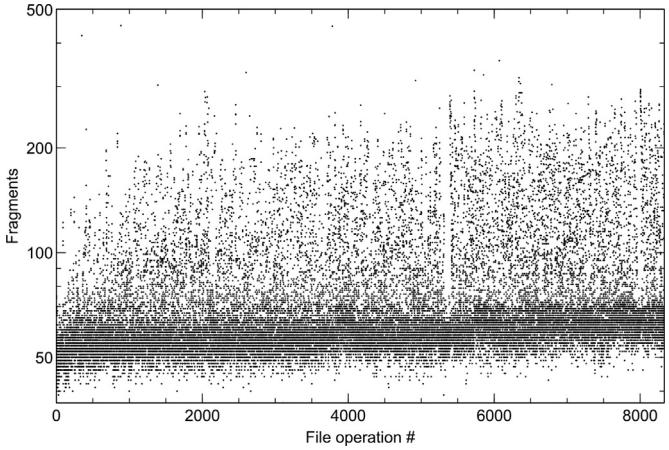

Fig. 11. The number of fragments allocated in each file operation for Windows 10 in 64 GiB hard drives using the standard file operation pattern. Please observe the log scale of the graph and the truncated maximum value of the Y-axis, which hides an outlier of 2043 fragments at file operation 5765.

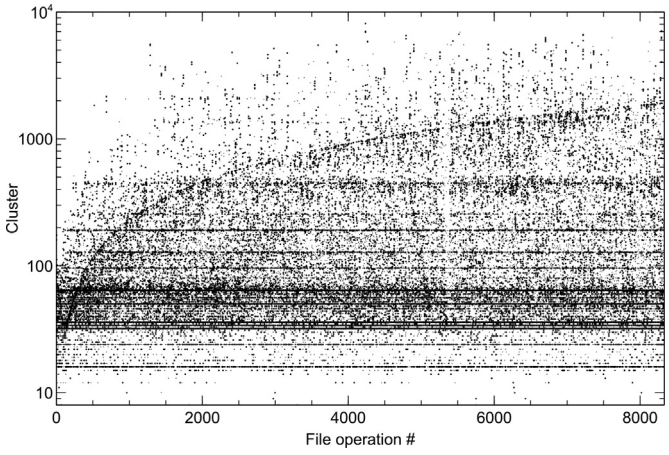

Fig. 12. The median size of the allocated fragments for Windows 7 in 64 GiB hard drives. There is a linearly increasing trend from 0 to approximately 2000 clusters and also a number of horizontal lines at exponentially increasing distances. Please observe the log scale of the graph.

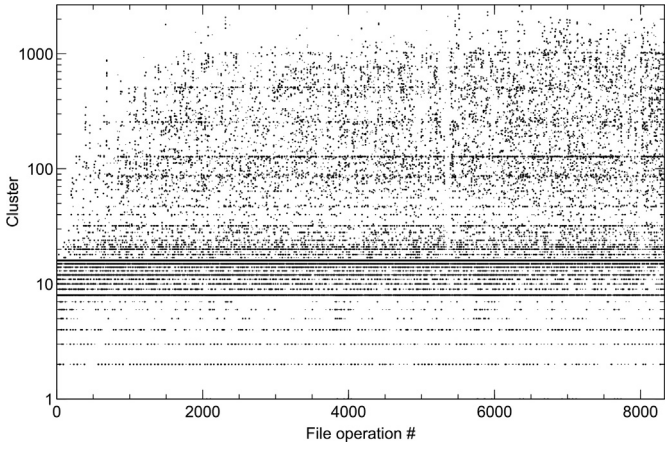

Fig. 13. The median size of the allocated fragments for Windows 10 in 64 GiB hard drives. There are clearly visible lines at approximately 100, 250, 500, 750 and 1000 clusters. Please observe the log scale of the graph.

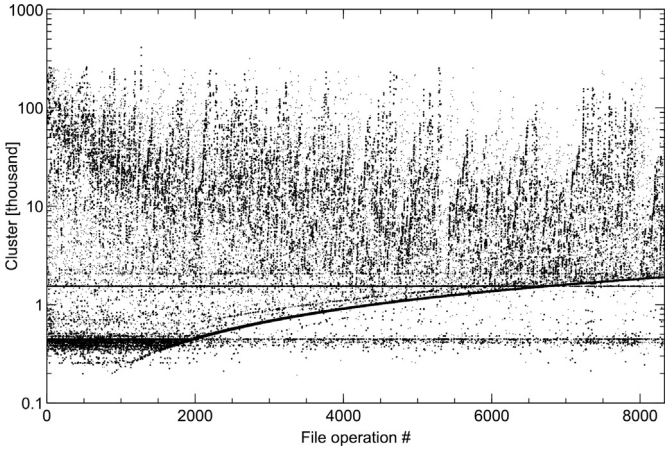

Fig. 14. The size of the largest sequence of allocation positions (file fragments) for Windows 7 in 64 GiB hard drives. Please observe the log scale of the Y-axis.

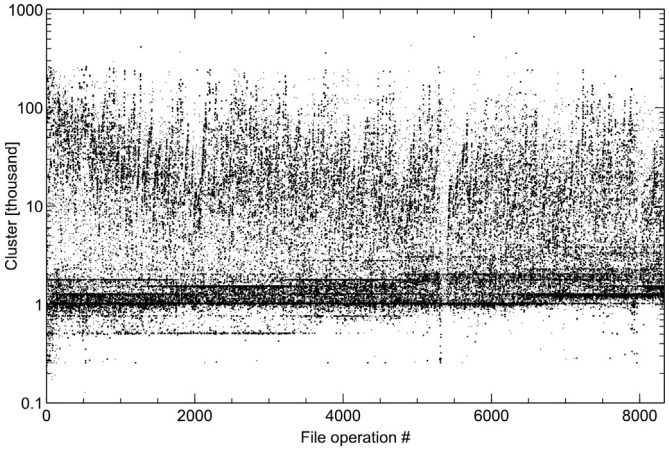

Fig. 15. The size of the largest sequence of allocation positions (file fragments) for Windows 10 in 64 GiB hard drives. Please observe the log scale of the Y-axis.

fragments never reaches above 500 for Windows 7. The number of fragments also slowly increases over time, but with a solid foundation around 20 fragments per file operation.

The number of fragments per file operation for Windows 10, as shown in Fig. 11, is approximately three times larger than for Windows 7, giving a general size of 60 fragments per file operation. However, all but one file operation give well below 500 fragments, with an outlier of 2043 fragments at file operation 5765. As in Fig. 10 the trend is a slow increase of the number of fragments as the number of file operations increases.

The median size of the fragments for Windows 7 using the standard file operation pattern is shown in Fig. 12. Large fragment size values indicate that the allocation algorithm tries to keep the fragmentation down and as can be seen there are a number of median fragment sizes above 7000 clusters in the Windows 7 virtual hard drives. As can also be seen the median fragment size has a line that increases linearly from 0 to approximately 2000 clusters (the logarithmic scale transforms the line to a curve). There are sharp horizontal lines at approximately the same distances in the graph, which means that due to the logarithmic scale of the Y-axis they are placed at exponentially increasing distances. There is much

activity in a band centered around fragments of 64 clusters. The amount of very large median fragment sizes slowly increases towards the end of the graph.

The graph showing the median fragment size for Windows 10 in Fig. 13 lacks the linearly increasing trend found in Fig. 12. However the horizontal lines in the Windows 7 graph are present in Windows 10 too, although starting at a lower level. The standard fragment size in Windows 10 is 16 clusters according to our results. Windows 10 has smaller median fragment sizes than Windows 7 in general, but still has the same slowly increasing amount of large median fragment sizes as Windows 7.

Fig. 14 shows how the size of the largest file fragment for each file operation is decreasing as the number of file operations is increasing. There is also a large amount of fragments of approximately 500 clusters created at the beginning of the experiment. After approximately 1500 file operations the size of the biggest fragments level out at approximately 20,000 contiguous clusters and the band at 500 cluster is thinner. There is also a significant amount of outliers, many of them as large as 200,000 clusters, a few times even higher. The data in Fig. 14 includes both the standard file operation pattern and unique patterns, still there is a clearly visible linearly increasing trend (remember the log scale of the Y-axis) starting at fragments of approximately 500 clusters and reaching to 2000 clusters at the end. The same line can be seen in Fig. 12. Finally there is a thin horizontal line of fragments of approximately 1500 clusters in size, which gets weaker at approximately 7000 file operations, when the linearly increasing trend reaches it.

The graph showing the largest fragment for each file operation in Windows 10 (see Fig. 15) lacks the linearly increasing trend (please do not forget the log scale) found in Windows 7. Instead there is a band of maximum fragment sizes centered around 1500 clusters. Apart from that the graph shows the same decreasing maximum fragment sizes at the beginning and the same leveling out at approximately 20,000 clusters large fragments as for Windows 7 (see Fig. 14).

As mentioned in the beginning of Section Result we omitted showing graphs of the results for Windows 8 and 8.1 for readability reasons. Most of the these results were close to the Windows 7 results, but with a few exceptions. Most notably the fragment statistics of Windows 8 and 8.1 were closer to the Windows 10 results, than the Windows 7. However, the differences were small in all cases.

Discussion

The $Bitmap files extracted during the experiment contain not only traces of the file operations executed by our scripts, but also any operations executed by the OS during each iteration. Especially the start and stop phases of an iteration will induce changes to the MFT and its records. Since we only want to see where (which LBAs) the OS allocates clusters when writing data the deallocation operations are irrelevant. We therefore have filtered out operations where allocated clusters have been freed. The remaining data will include clusters allocated by the system too, but that is a minor problem because the system activities often affects already allocated clusters (appending information to existing log files for example). An MFT record is 1 KiB in size and the smallest allocatable unit in a 64 GiB NTFS partition is 4 KiB, hence every fourth file creation will possibly give rise to a new cluster being allocated in the MFT (not until the preallocated MFT space is used up). When for example log and system files grow and require a new cluster to be allocated the cluster position will most probably be allocated to the same areas as ordinary user files. The inclusion of system file operations will therefore have a low impact on the statistical metrics used.

The main conclusion to draw from the result is that the allocation behavior differs in Windows 10 compared to the older versions of Windows, an important fact to remember during digital forensics case work involving, for example, suspicion of manipulation of file system time stamps. Another important conclusion to draw is that the allocation activity is highest in the lower middle cluster positions and only slowly moving towards the end of the partition as the file system ages. Hence any file carving searches for user data should preferably start there and not at the beginning of a hard drive.

We can also conclude that similar file operations executed in differently sized hard drives still generate similar, but not equal, results (compare Figs. 1 and 2). The similarity might actually be even higher in reality, because the instability of the experimental platform caused unique system states for the individual virtual machines, causing system files to be written at different occasions in each machine. Those activities therefore might have allocated free areas that were allocated to files written by the scripts in other machines.

The file system utilization plots included in Figs. 1–3 have been raised by 2,526,780 clusters. That corresponds to the area in the file system where the OS files are written during installation. The same area is clearly visible in Fig. 6 showing the statistical mode of the allocation pattern. We found that Windows 7 and 10 are less likely to allocate files in that area than Windows 8 and 8.1 and that the sparse area has the same size regardless of size of the hard drive. Of course the size of the area will differ depending on the size of the installed OS, but the required size of a Windows installation is the same for at least Windows 7 to 10 (Microsoft, 2017a, b, c).

Since we do not differentiate between allocations originating from the file operation scripts and system file allocations we cannot be sure what type of file has been allocated to the sparse area containing the OS files (we can only see what is currently allocated there). Furthermore the statistical metrics only show parts of the reality, hence the allocation activity in the sparse area might be high, but only for small files. Nevertheless the results show that the allocation activity differs between areas in the partitions and between the versions of Windows included in the experiment, which is important to know in for example file carving investigations.

We have not yet found any theoretically or scientifically sound explanation of the unused area found in the middle of the Windows 7 partitions (see the median and mode graphs in Figs. 5 and 6). There are no system files allocated there in the virtual machines from the experiment, neither the $MFTMirr file as suggested by Carrier (2005), nor the pagefile.sys as suggested by colleagues. This is also true for the six unrelated home and office computers running different versions of Windows (from 7 and up) we checked to see if the hypothesis holds for real world computers. Hence the system file hypothesis is falsified.

When checking the unused area in the middle of the Windows 7 partitions the area is allocated to files written during the experiment, although we can only see file system information for the last few hundred file operations due to (possibly) earlier deletions. The files found in the unused area have all been written after file operation 8,331, which is the upper bound used for the graphs due to a few virtual machines having to be stopped prematurely. When checking the data for the virtual machines that executed all 10,000 file operations the unused area is present for all operations for Windows 7, but for Windows 8.1 it is vanishing in the last 1000 file operations. The $Bitmap files of the Windows 8 virtual machines all got out of sync for different reasons during the last 1000 operations and hence we did not get any reliable data from them after file operation 9000. The most probable reason is a breakdown of the VBoxManage service, which caused the script to download the $Bitmap file at the wrong occasion. The Windows 8 machines had

become slower and slower over time and finally the restart timeout of 10 min was exceeded. The failure had a severe impact on the results after file operation 9000. However, up to file operation 9000 the results are reliable and the unused area is clearly visible when we plot the Windows 8 data. We can also conclude that the area is not present in Windows 10, thus Microsoft seems to have updated the allocation algorithm in Windows 10.

The unused area in the middle of the partitions might also be an artifact of the experimental setup. Since we tried to keep the virtual machines identical an error during the installation of the OS might have had a large impact on the results for the machines using the standard pattern. However, the fact that half of the included virtual machines ran unique sequences of file operations and still retained the same unused area contradicts the installation error hypothesis.

The mode allocation position graphs (Figs. 6 and 7) for both Windows 7 and 10 are more or less evenly distributed over the used area of the storage media. This might simply be an effect of the random deletion of files during the experiment. However, if it is not the effect is that the area where there might be interesting material in a partition is increasing as the file system is utilized, hence an old hard disk requires a larger area to be searched. This is however contradicted by the slowly increasing mean and median allocation position seen in Figs. 1 and 5.

The phenomenon of the maximum allocation position, described as the leeward effect on clouds of a mountain range, is interesting. There is a clear difference between Windows 7 and 10, where the latter is biased towards continuing using any high allocation addresses reached. This means that Windows 10 is using the storage area more evenly than Windows 7. All virtual machines used the default settings in the storage section of VirtualBox, which therefore emulated a mechanical hard disk. Hence all virtual machines should behave the same based on the hardware setup. Consequently there is a difference in the behavior of the allocation algorithms between Windows 7 and Windows 10, which needs to be studied further.

The decrease of the maximum file fragment size as the file system grows, which can be seen in Fig. 14 is natural, since when the free areas fill up and files are deleted the groups of contiguous free cluster areas will be smaller. The spikes in the graph at higher file operation numbers originate from the still unused areas at the end of the partition. If we had been able to run the experiment for an even longer period the maximum fragment size would probably have decreased even more.

The large standard deviation of the allocated positions for each file operation clearly indicates significant file fragmentation and consequently the allocation algorithm's focus on filling holes in the already used area of a partition before allocating files to the yet unused part at the end of a partition. We do not know the exact reason for this behavior, but we think it might be introduced by the fact that all file write operations use stream writing. When the OS does not know the size of the file in advance the strategy is to assume it is small and hence use it to fill in any holes in the already used area of a partition. If the file then turns out to be larger, the size of the allocated areas will automatically grow, since all small holes are occupied. On the other hand, if there is a large free area available, it is better to use that first to at least postpone file fragmentation to a situation when the partition is more heavily used. Hence the chosen strategy depends on the focus of the allocation algorithm; filling in holes or avoiding file fragmentation.

The best fit allocation strategy that NTFS uses is meant to decrease the amount of file fragmentation by optimizing the used area with regard to lost space at the ends of the free area that is being allocated. The behavior we can see from the result is however not fully adhering to that strategy, but that might be questioned from a philosophical point of view. If the focus lies on minimizing the lost (remaining) free area after each allocation the behavior of not using the free space at the end of the partition first and then start using the free areas left from file deletions can be understood. Fitting an allocation into an hole left by a file deletion actually leaves less remaining space around the allocated area, than if the large unused area at the end of the partition was used. If we take the large number of file fragments created by that strategy into consideration this type of behavior becomes less understandable, especially since the OSs saw the disks as mechanical hard drives, which are negatively affected by fragmentation.

Conclusion and future work

We can conclude that there actually are differences in the allocation behavior of different Windows version using NTFS, that the size of the storage media is not affecting the allocation behavior and that the behavior changes over time as the file system grows. Likewise the adherence to the best fit allocation strategy can be questioned. The allocation activity is not evenly spread over the storage area, instead it is concentrated to the already used areas. A strictly best fit allocation strategy would not fragment files if there where free space available to fit the file in one block. All Windows versions used in the experiment differentiate between mechanical hard drives and solid-state drives (SSDs), but since all virtual drives were set to emulate mechanical hard drives such differentiation cannot be the reason behind the behavior.

The results from the experiment are directly applicable to the digital forensic case work by showing that it is more probable to find older data closer to the beginning of the partition and newer data closer to the end of the used area. In the same way we have shown that the priority of the allocation algorithm is to get rid of holes left by file deletions, not to use the whole disk to decrease the risk of file fragmentation. The knowledge gained from the experiment is especially important in file carving where the goal is to reconnect fragments of files into the original files again. By decreasing the area to be search for file fragments the process will be more efficient and hence faster.

The results can also be used to improve the creation of time lines (work as another source of time stamp information) by the fact that the size of file fragments decreases as the file system grows. A file having large (and few) fragments has a higher probability of being older than a file with many small fragments, although the effect is small.

As future work we will stabilize the experimental platform and expand the scope of the experiment to also include other file systems, hard drive sizes and OSs, as well as both stream writing and block writing file operations. We will also isolate our file operations from the OS related operations and use tools from the Sleuth Kit to increase the resolution and reliability of the results. Together these improvements will enable us to determine if it is possible to use the allocation pattern as a means to improve the reliability of time stamps and possibly even work as a sequential time stamp, showing the writing order of files. The results will also be used to find out more about the standard fragment size, number of fragments, their probable placement on disk (logical position) etcetera, which will be of great help in file carving situations. The information on differences between stream and block writing operations can also be used to improve file carving processes by giving a first indication of the type of file of a fragment and also when finding the correct ordering of the found fragments.

Acknowledgements

The research leading to these results has received funding from the Research Council of Norway programme IKTPLUSS, under the R&D project Ars Forensica grant agreement 248094/O70. We would also like to thank the Swedish Defence Research Agency (FOI) for their support by letting us use their Cyber Range And Training Environment (CRATE) computer cluster.

References

van Baar, R., van Beek, H., van Eijk, E., 2014. Digital forensics as a service: a game changer. Digit. Invest. 11, S54–S62. https://doi.org/10.1016/j.diin.2014.03.007. proceedings of the First Annual DFRWS Europe.

van Beek, H., van Eijk, E., van Baar, R., Ugen, M., Bodde, J., Siemelink, A., 2015. Digital forensics as a service: game on. Digit. Invest. 15, 20–38. https://doi.org/10.1016/j.diin.2015.07.004. special Issue: Big Data and Intelligent Data Analysis.

Breitinger, F., Stivaktakis, G., Baier, H., 2013. Frash: a framework to test algorithms of similarity hashing. Digit. Invest. 10, S50–S58. https://doi.org/10.1016/j.diin.2013.06.006 (the Proceedings of the Thirteenth Annual DFRWS Conference).

Carrier, B., 2005. File System Forensic Analysis. Addison-Wesley Professional.

Carrier, B., 2014. Tsk tool overview. http://wiki.sleuthkit.org/index.php?title=TSK_Tool_Overview.

Casey, E., 2018. Digital stratigraphy: contextual analysis of file system traces in forensic science. J. Forensic Sci. 63, 1383–1391. https://doi.org/10.1111/1556-4029.13722. https://onlinelibrary.wiley.com/doi/abs/10.1111/1556-4029.13722.

European Police Office (Europol), 2016. Internet Organised Crime Threat Assessment (IOCTA) 2016. Technical Report. European Cybercrime Centre (EC3).

Fairbanks, K., 2012. An analysis of ext4 for digital forensics. Digit. Invest. 9, S118–S130. https://doi.org/10.1016/j.diin.2012.05.010 (the Proceedings of the Twelfth Annual DFRWS Conference).

Fairbanks, K., 2015. A technique for measuring data persistence using the ext4 file system journal. In: 2015 IEEE 39th Annual Computer Software and Applications Conference, pp. 18–23. https://doi.org/10.1109/COMPSAC.2015.164.

Fairbanks, K., Garfinkel, S., 2012. Column: factors affecting data decay. J. Digit. Forensic Secur. Law 7 (2).

Gladyshev, P., James, J., 2017. Decision-theoretic file carving. Digit. Invest. 22, 46–61. https://doi.org/10.1016/j.diin.2017.08.001.

Hughes, J., 2009. The four stages of ntfs file growth. https://blogs.technet.microsoft.com/askcore/2009/10/16/the-four-stages-of-ntfs-file-growth/ accessed 24-10-2018.

Jones, J., Khan, T., Laskey, K., Nelson, A., Laamanen, M., White, D., 2016. Inferring previously uninstalled applications from residual partial artifacts. In: Annual ADFSL Conference on Digital Forensics, Security and Law, pp. 113–130.

Karresand, M., Axelsson, S., Dyrkolbotn, G., 2019a. Disk cluster allocation behavior in windows and ntfs. Mobile Network. Appl. https://doi.org/10.1007/s11036-019-01441-1.

Karresand, M., Axelsson, S., Dyrkolbotn, G., 2019b. Using ntfs cluster allocation behavior to find the location of user data. Digit. Invest. 29, S51–S60. https://doi.org/10.1016/j.diin.2019.04.018.

Karresand, M., Warnqvist, Å., Lindahl, D., Axelsson, S., Dyrkolbotn, G., 2019c. Creating a Map of User Data in NTFS to Improve File Carving. Springer International Publishing, Cham, pp. 133–158 (chapter 8).

Key, S., 2012. File Block Hash Map Analysis. https://www.guidancesoftware.com/app/File-Block-Hash-Map-Analysis. Accessed 28-04-2018.

Microsoft, 2017a. System requirements. https://support.microsoft.com/en-gb/help/12660/windows-8-system-requirements accessed 30-04-2018.

Microsoft, 2017b. Windows 10 system requirements. https://support.microsoft.com/en-us/help/4028142/windows-windows-10-system-requirements accessed 30-04-2018.

Microsoft, 2017c. Windows 7 system requirements. https://support.microsoft.com/en-us/help/10737/windows-7-system-requirements accessed 30-04-2018.

Microsoft, 2018. How ntfs works. https://technet.microsoft.com/pt-pt/library/cc781134(v=ws.10).aspx accessed 30-09-2018.

Net Applicationscom, 2019. Desktop operating system market share. https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=10&qpcustomd=0.

Pal, A., Memon, N., 2009. The evolution of file carving. IEEE Signal Process. Mag. 26, 59–71. https://doi.org/10.1109/MSP.2008.931081.

Poisel, R., Tjoa, S., 2013. A comprehensive literature review of file carving. In: 2013 International Conference on Availability. Reliability and Security, pp. 475–484. https://doi.org/10.1109/ARES.2013.62.

Quick, D., Choo, K., 2014a. Data reduction and data mining framework for digital forensic evidence: storage, intelligence, review and archive. Trends Issues Crime Crim. Justice 1–11.

Quick, D., Choo, K.K.R., 2014b. Impacts of increasing volume of digital forensic data: a survey and future research challenges. Digit. Invest. 11, 273–294. https://doi.org/10.1016/j.diin.2014.09.002.

Roussev, V., 2012. Managing terabyte-scale investigations with similarity digests. In: Peterson, G., Shenoi, S. (Eds.), Advances in Digital Forensics VIII: 8th IFIP WG 11.9 International Conference on Digital Forensics, Pretoria, South Africa, January 3-5, 2012, Revised Selected Papers. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 19–34.

Silberschatz, A., Galvin, P., Gagne, G., 2012. Operating System Concepts, 9 ed. Wiley.