Open University of the Netherlands
faculty of Management, Science & Technology
Bachelor Computer Science

Synthetic Fragmentation Experiments using WildFragSim

Author:
Robbert Noordzij
851758049

Researcher:
Ir. Vincent van der Meer
Supervisor:
Dr. Ir. Hugo Jonker

July 1, 2019

Open
Universiteit

Abstract

In forensic research, researchers try to answer questions that arise during criminal trials. Question such as what software was used, who has used it, what operating system had access to the disk and how long it has been used for. Based on disks that they receive they try to find answers.

Fragmentation is related to the usage of the computer, the software that was installed, the age of the file system, and the operating system that is running. To research whether it is possible to derive this from the fragmentation, knowledge on the effects of these factors on fragmentation over time needs to be established. Currently researchers have to analyse real disks over time to see the development of fragmentation.

This research finds a method that allows research into fragmentation over time. Using a basic model for text editing, this research demonstrates that the use of synthetic experiments to render fragmentation patterns is possible without privacy concerns.

The first results show that internal fragmentation and data fragmentation only change slowly over time. External fragmentation changes rapidly and suddenly, because of the allocation strategy. Windows Update has a big impact on the file count of the volume. Although the number added by the update is big, their impact on fragmentation is small.

Contents

Chapter 1

Introduction

Forensic researchers try to answer questions that arise during criminal trials. Based on disks that they receive they try to find answers to what software was used, who has used it, what operating system had access to the disk and how long it has been used for.

Fragmentation is the waste of disk space that occurs when using a storage device. The inefficiently used storage reduces storage capacity and affects performance. The development of fragmentation is related to the usage of the computer, the software that was installed, the age of the file system, and the operating system that is running. Fragmentation evolves over time and might be influenced by different factors by different amounts. By analysing the patterns formed by fragmentation, researchers might be able to answer these questions about the disks.

To research whether it is possible to derive this from the fragmentation, knowledge on the effects of these factors on fragmentation over time needs to be established. Currently researchers have to analyse real disks over time to see the development of fragmentation. It would mean that the disk need to be analysed multiple times over its lifespan. This is highly impractical, time consuming, and privacy invading. Since everyone uses a combination of tools and have different behaviours, it does not allow for research into fragmentation factors in isolation. Therefore it is hard to know how fragmentation is affected by software or usage and especially its impact over time.

This research finds a novel method that allows research into fragmentation over time using synthetic experiments. This allows the research to be done without any privacy concerns and enables for the research into specific factors of fragmentation. To simulate a user, a profile is used. During this research, Wild-FragSim is established. This tool allows for the execution of synthetic fragmentation experiments with user profiles, without intruding real users privacy.

The contributions of this research are:

1. a first understanding of the development of fragmentation over time
2. a method for further study of fragmentation over time
3. a method for the study into specific factors of fragmentation
4. a way to generate fragmentation patterns based on different user profiles to answer whether pattern are affected by usage
5. allow for research into the effects of fragmentation on the performance of file systems

This study does not involve creating elaborate profiles and validating the fragmentation patterns created using synthetic experiments to be realistic. However, to investigate the workings of the method, a simple profile will be established. To scope the research, it focuses on the fragmentation on NTFS file systems. NTFS is the default file system for Windows 10, now the most used operating system[1]. The method opens new research possibilities that were to this date impractical, time consuming or even impossible.

[1] `https://netmarketshare.com/operating-system-market-share.aspx`

Chapter 2

Background

2.1 GUID Partition Table and Master Boot Record

Computer storage devices contain information that is used by the operating system to understand the disk layout. Disks are divided into partitions. Partitions can be formatted. When partitions are formatted, they become a volume. A volume has a file system. The type of file system might differ per volume. For the computer to know how to interpret the bits and bytes on a disk correctly, the partition table has meta-data about the partition. Each volume that uses the NTFS file system has a reference to the Master File Table. More about the MFT can be read in Section 2.2.

There are two ways for a computer to store its partition table. There is a MBR (Master Boot Record) and there is a GPT (GUID Partition Table). The Master Boot Record predates the GPT and is seen as outdated. The MBR is limited in the amount of partitions and in the partition size. To resolve these issues GPT was introduced. However, a Master Boot Record is still used and present on modern disks.

The Master Boot Record is always in the first cluster of a disk and is 512 bytes in size. MBR only uses 32 bits to reference sectors, so it is able to handle 2^{32} sectors. When each sector is 512 bytes, this allows a disk to be just more than 2 terabyte. In the GUID partition table addresses are comprised of 64 bits. With a sector size of 512 bytes, this would allow for a disk to be more than 9,000,000,000 terabyte.

The Master Boot Record is still present even when GPT is used for the disk layout. It is there to protect disks from accidental overwrites by an older system that does not understand GPT. It such case the MBR has only one partition that is as big as possible and is marked as read-only. This is called a protective Master Boot Record.

2.2 Master File Table

The MFT (Master File Table) is part of NTFS (New Technology File System) and concludes a list of all the files and folders that are present on the volume with meta-data such as creation and modification time-stamps. The Master File Table marks the physical location of a file on disk.

For each file and folder, there is at exactly one record in the MFT. The Master File Table itself is regular file and it is present in the MFT. Next to the $MFT-record, there are ten other system files in that are specific to the inner workings of NTFS. Also, there are reserved entries that can be used in the future.

Table 2.1 depicts the first few records that are present in every Master File Table. The first record in the Master File Table is the $MFT itself. The second record points to a mirror of the MFT stored at the end of the disk. This $MFTMirr is a duplicate and can be used to recover files when the original Master File Table is corrupt. The $LogFile is a transaction log that can be used to recover from a system failure. The $Volume record of the Master File Table contains information about the volume itself like its label and version. In the $AttrDef the operating system stores a list of attributes that can be used for each record. This set of definitions is different for each operating system. However, for backwards compatibility most attributes are the same for all operating systems. Mostly, only new attributes are added that can be ignored by older systems without any problem. The next system file of the Master File Table is the $, . (dot) or

0	$MFT
1	$MFTMirr
2	$LogFile
3	$Volume
4	$AttrDef
5	.
6	$Bitmap
7	$Boot
8	$BadClus
9	$Secure
10	$Upcase
11	$Extend
12-15	Reserved for furture use
...	Entry for each user file

Table 2.1: The first records of every Master File Table

root entry. It is the root directory of the file system. The $Bitmap file marks for each disk cluster whether it is occupied or not[1]. Each cluster is represented by one bit. The bit is 1 when occupied, otherwise the bit is 0. The file is used for quickly scanning and allocating space for new files. Next to the $Bitmap entry is the $Boot entry that contains the information that allows the system to boot. It is the only file that cannot be relocated[2]. The $BadClus file is a sparse file that is as big as the disk itself. Sparse files are files of which the empty parts are not written to disk as empty but only the meta-data about these empty blocks and thereby saving disk space. Because it is sparse, it does not occupied any physical space. However, when a corrupt cluster is detected, the cluster is added to this file as non-empty. It thereby prevents other files from using these broken clusters. The entry $Secure keeps track of the security attributes of files. For efficiency, the file replaces the $SecurityDescriptor attribute of each record. A new field in $StandardInformation points to the security information in $Secure. The second last reserved file entry is the $Upcase file. This file is a file with only capital letters that is used to compare and sort files. The last entry $Extend is an ordinary directory containing extra meta-data files. In the $Extend directory, there are at least four files for a Windows 10 installation: $ObjId, $Quota, $Reparse and $UsnJrnl. These four files themselves do not have a fixed position in the Master File Table. The records 12 up to and including 15 are reserved for future use and are empty. After the fifteenth record are all the regular entries for each file and directory.

2.2.1 Master File Table-record

A Master File Table-record exists for each file and folder on NTFS. It contains meta-information about its name, creation dates, privileges and location. Each record is build out of a standard information that is present for each record and additional attributes[3].

Each MFT-record is comprised of 1024 bytes. Other sizes are possible, but are not common. The first 56 bytes of a record contain the MFT-record header. In the header, information is kept about the record number, the size of the record so the end can be calculated, and a pointer to the first attribute of the record. Also the header specifies the type of entry, so either file or disk and whether it is in use.

[1]https://whereismydata.wordpress.com/2009/06/01/forensics-what-is-the-bitmap/
[2]https://flatcap.org/linux-ntfs/ntfs/files/boot.html
[3]http://amanda.secured.org/ntfs-mft-record-parsing-parser/

Next to the header, the record consists of attributes. Most important for this research are the $FileName attribute, the $Data attribute and the $StandardInformation attribute.

The $StandardInformation attribute contains information about the creation of the file or directory. The attribute contains for example the creation time, the modification time and the file access time. But it also specifies the owner of the file and the permissions. A special field references the id that corresponds with the id in the $Security file in the Master File Table. The DOS permission field in the attribute is used to flag for example whether the file should be hidden for the user, if the file is sparse, and if the file is write-able[4].

The $FileName attribute logically contains the file name of the file or directory. However, it also includes additional file time-stamps. This attribute also specifies the logical size and the physical size of the file[5]. The physical size is the number of clusters the file occupies on disk. The logical size is the actual size in bytes of the file. The logical size is therefore equal to or smaller than the physical size of a file. It is possible for a file to have more than one attribute of this type. The file name of the attribute is in a specific name-space. Regular file names are in the WIN32 name-space. These file names might be too long for older systems. Therefore other name-spaces are added for Linux systems (POSIX) and DOS. Other $FileName attributes in a different name-space are considered to be an alias[6]. Multiple attributes with the same namespace are hard links. The attribute contains a reference to the parent file or directory. This is used to construct a file tree.

The last attribute considered important for this research is the $Data attribute. The content of this attribute can either be resident or non-resident. Files smaller than approximately 900 bytes are stored in the record itself[7]. These are so called resident files. Most files are more than 900 bytes, so for these files the attribute contains block ranges or data runs that point to the clusters of the file. A file can have more than one $Data attribute. There is an unnamed or default data stream and named data streams. Named data streams are called alternative data stream or ADS in short. ADS are powerful Windows resources that allow for hidden information in files next to the primary data stream [4].

Because each entry in the MFT is the same size, the remaining entry is filled with zeros. This is called padding. In the case more attributes need to be stored than the size of the entry, there is the option to have a non-resident attribute list. It is very unusual for a file to have this attribute[8].

When files are removed from the system, neither the entry in the Master File Table and the file from disk are directly deleted. The Master File Table records that the file is no-longer in used and allows the record and allocated blocks to be reused. This allows for deleted files to be recovered when the record has not been reused. In case the record is reused and blocks from the file are untouched, file carvers can recover the file.

2.2.2 Locating the Master File Table

Since Microsoft Windows 7, the format of the disk layout uses the GUID Partition Table format. Older version of Windows are able to interpret the table. However, versions older than Windows 7 have not implemented GPT completely and so do not support all the features[9]. The 64bit Windows XP version is able to see and interpret data disk formatted with GPT. The 32bit version will only see the protective Master Boot Record. Also, Windows 2008, ME and Vista are able to use volumes with a GUID Partition Table. However, they require the 64 bit version to also implement the ability to boot from a GPT disk. The WildFrag database suggest that Windows 10 is the most used operating system [13]. This is also confirmed by research done by Net Applications, who conduct studies in market share of internet technologies such as operating systems[10].

Figure 2.1 show the components that are required to locate the Master File Table on disk.

[4]https://flatcap.org/linux-ntfs/ntfs/attributes/standard_information.html

[5]https://flatcap.org/linux-ntfs/ntfs/attributes/file_name.html

[6]https://www.fireeye.com/blog/threat-research/2012/09/incident-response-ntfs-indx-buffers-part-2-internal.html

[7]https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2003/cc781134(v=ws.10)

[8]https://flatcap.org/linux-ntfs/ntfs/attributes/attribute_list.html

[9]https://docs.microsoft.com/en-us/windows-hardware/manufacture/desktop/windows-and-gpt-faq

[10]https://netmarketshare.com/operating-system-market-share.aspx

First the first 512 bytes from the primary hard disk need to be parsed. These first 512 bytes of the disk are for the Master Boot Record, see also Section 2.1. This Master Boot Record contains information about what partitions exist on the disk in the form of a partition table. For backwards compatibility reasons, the Master Boot Record still exists. This is to protect the disk from accidental overwrites with an older computer.

When the system uses GPT for the disk layout and contains a protective Master Boot Record, the second sector of 512 bytes contains the GUID Partition Table Header. The header contains information about the table such as the number of partitions, size of the entries themselves and the first block and last block of the table. Using the start, the number of entries and the size of the entry, it is possible to collect the full list of GPT entries. Each entry is one partition of the disk.

Not all partitions from the GUID Partition Table have a Master File Table. Only the partitions that are formatted become a volume. Volumes that are identified with UUID - `EBD0A0A2-B9E5-4433-87C0-68B6B72699C7` - are NTFS formatted and contain a MFT. The partition entry points to the first and last cluster of the partition. On this partition, the Volume Boot Record can be found in the first cluster of the partition. This boot record is comparably to the Master Boot Record, except it is not limited by size so it can accommodate for bigger volumes.

The VBR contains a reference to the first and last location of the Master File Table. The first record of the Master File Table contains the Master File Table itself. The record also contains the size which is most likely 1024 bits. Parsing the first record of the master file table reveals the data runs of the master file table itself. Reading and combining all the clusters that were referenced in this first entry result in the complete Master File Table of the partition.

When the Master Boot Record is not protective, it might contain partitions that are of type NTFS - `7`. These partitions contain a VBR directly on their first sector. This means here is no GUID Partition Table on the disk.

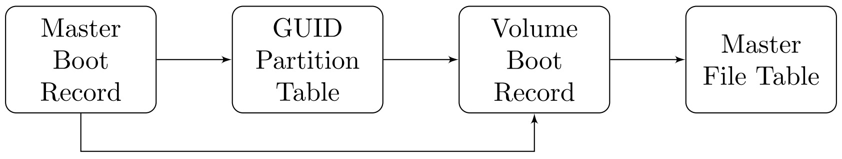


Figure 2.1: Locating the Master File Table on a disk

2.3 Fragmentation

Volumes are divided into evenly sized clusters. Files are assigned to one or more of these clusters. Depending on the allocation strategy, the file will most likely be placed in consecutive blocks. When files are assigned to clusters on a storage device, the waste that occurs is called fragmentation.

There are three types of fragmentation that can occur, either together or on their own. External and data fragmentation can be mitigated by means of defragmentation tools. Internal fragmentation can be reduces using block sub-allocation schemes. This is supported by for example FreeBSD UFS2[12]. It is not supported in NTFS.

2.3.1 Internal Fragmentation

Internal Fragmentation is the waste that occurs when a file does not completely fill up a cluster. This happens when a file or its remaining is just smaller of the size of a block[9]. This remaining part of a cluster can not be used by any other file, since there is no way to point to the middle of a cluster in the MFT.

From the MFT the internal fragmentation can be calculated by taking the difference between the physical and logical size of a file.

Also the padding inside a MFT entry can be seen as internal fragmentation, since more space is allocated than used. However, this is generally not considered as internal fragmentation in the context of a disk.

Internal fragmentation is also referred to as slack space. Internal fragmentation is the more general term that also applies to memory. Slack space is more specific to file systems.

2.3.2 External Fragmentation

External Fragmentation occurs when a smaller file is put in the old location of a bigger file. There is now a small space left. Effectively, these small groups of clusters are no longer useful since files are usually bigger and are seen as waste. Using these remaining clusters for file storage results in the last form of fragmentation: data fragmentation.

The external fragmentation F_{ext} is generally defined as the following formula:

$$F_{ext} = 1 - \frac{C_{free}}{C_{disk}}$$

in which C_{free} is biggest block of consecutive free clusters and C_{disk} the number of clusters in total. Since the $Bitmap file of NTFS is especially designed for quickly finding free file clusters, the file is also useful for calculating the external fragmentation. By counting the longest streak of consecutive zeros in $Bitmap, the biggest block of free space can be found. The whole size of the $Bitmap files in bits is the size of the disk, since each cluster is represented in the file by either a one or a zero.

2.3.3 Data Fragmentation

When a file does no longer fit in consecutive clusters, it will be split and the blocks will be assigned to available clusters. When a file is broken apart it is called Data Fragmentation.

Data fragmentation is proven to affect the performance of the system [18]. It causes for the operation system to seek for the file fragments. However, this effect is not as apparent on a Solid State Disk than on a more traditional HDD where a physical head moves across the disk.

In the Master File Table, each record has one or more $Data attributes attached. These $Data attributes specifies whether a file is resident or non-resident. Only non-resident files can have data fragmentation, since resident files fit within the MFT entry it self and are therefore not split across multiple clusters. The number of data runs in the attribute is the number of fragments the file is in.

2.4 Virtual Machine

A virtual machine is a computer program that represents a computer that can run on shared hardware. This allows for isolated environments which are consistent across hosts. The virtual machine knows nothing about its host. This improves portability of software. Also it enables the use of a different operating system than the host OS.

VirtualBox[11] - acquired by Oracle - is one of the best known providers of virtualisation software. Their virtualisation product is free to use and can run on Windows, Mac and Linux. This allows for Windows Virtual Machines to be ran on Mac and Linux. Thereby, VirtualBox allows for automation through its API and provides bindings for multiple programming languages such as JAVA and Python.

Virtual machines are often used by cloud services to share physical hardware between multiple customers. This relies on the same concept, however it uses different techniques.

2.5 Keyboard Protocol

Keyboards communicate via the PS/2 protocol. A key press is built up out of a make code and a break code[5]. When the user uses multiple keys, then the make and break codes are combined. For example, the `a`-key has a make and break code of `0x1E` and `0x9E` respectively. The shift has the codes `0x2A` and `0xAA` for making and breaking. When the user wants a capital A, he will use the `a`-key in combination of with the shift and so the keyboard sends the make code of the shift, the make code of the `a`-key, the break key of the

[11] `https://www.virtualbox.org/`

a-key and finally the break key of the shift. So in the case of a lowercase a the keyboard sends `0x1E` and `0x9E` with a short pause in between the codes. In the case of a capital A, the keyboard sends `0x2A`, `0x1E`, `0x9E`, `0xAA`.

Chapter 3

Related Work

Fragmentation Development

Fragmentation of computer hard drives is a widely studied subject within forensics. However, most research focuses on file carving like Garfinkel's in 2007[7]. It shares valuable observations on fragmentation and file reconstructing. The research mentions that most files will not be fragmented. Garfinkel does not plot his data over time. In 2009, Pal has researched the Factors of fragmentation[15], but he also does not compares his results over time. No research was found that focuses on synthetic experiment to discover how data fragmentation evolves on an NTFS file system.

Research into the fragmentation development has not been conducted for computers. There has been research into development of fragmentation on mobile devices, done by Cheng Ji et al[8]. His method is to run user actions over and over again and collect the data multiple times during the experiments. Their research was focused on the impacts fragmentation has on the performance of the mobile device. They have set up limited scenarios they deemed common for mobile users. Also, the research repetitively does the same task and does not try to mimic a real user. Compared to mobile phones, personal computers and laptops are used differently, so it is expected that the results will differ.

Automation of User Behaviour

Cheng Ji controlled the mobile phone from a computer without the need to install something on the device. In the study, Cheng Ji used something called the UIAutomator framework. Similar frameworks exists for Windows. TESTAR uses the Accessibility API of the operating system to control the applications[2]. This needs to be supported by the application under test and requires low level access to the operating system, applications can only be controlled when the tool runs on the same system. Also, TESTAR does not do any measurements during the test and does not allow for a control and experiment.

Installing extra software to accommodate for the experiment is expected to influence the results and is therefore unwanted. Others suggest to use the visible elements to locate user actions. The use of screen captures to locate controls on the screen has been performed by a tool called Sikuli in 2009[21]. The tool was built to automate visual interactions, such as finding a bus on a map and minimising all open windows. It eliminates the need for exact coordinates on the screen or the use of Windows' Accessibility APIs. Sikuli is installed on the operating system itself and so it would still influence the experiment. However, the method could be used to control experiments without the need to install the software like other researchers have done before.

Researchers in Sweden have developed a tool that simulates full systems[11]. This tool is developed to run simulations on virtual machines with unchanged binaries. This reduces contamination by the testing infrastructure. This research was conducted by Simics. Simics focuses on embedded systems. The architecture of Simics to control the simulations and collect measurements from outside the virtual machine is useful for investigating fragmentation influenced by user profiles.

Creation of User Profiles

In 2016, Kaklanis researched the standardisation of user-profiles[10]. These profiles are for building simulators and cognitive research. The research done by Kaklanis is useful for formalising user profiles for fragmentation research. Kaklanis incorporates a lot of detail in the user profile. For a computer user such granularity is important. For example, pauses while typing allow Microsoft Word Office to write work to disk[1]. To not impact the performance while typing, this is done while the user is not actively interacting with the document.

To create user profiles that mimic computer users, their typing behaviour is valuable. Galinsky of U.S. Department of Health and Human Services surveyed data-entry operators and monitored their *keystrokes per hour*[6]. While it is not clear whether data-entry operators behave the same as other typists, the measurements from Galinsky his research are a starting point. Alves studied the pauses between keystrokes and words in 2007[1]. The values found in both researches can be used to set up the a profile that behaves like a real user, including type-speed, corrections and pauses and coffee breaks. Research done by Vizer can be used to alternate the profile for stress factors[19]. The article by Vizer also states the ratio's of keys such as the delete and backspace keys. Also Rosenqvist researched the pauses between words and between keystrokes in 2015[17]. The results from Rosenqvist match the ones from Alves. However, based on the deviation found Rosenqvist suggests that there is not a single value for the pauses. These values will be used to model a profile of a basic office worker.

Validation of Fragmentation Patterns

Fragmentation analyses on more than 200 computers has been conducted by Vincent van der Meer. His data set, WildFrag database, contains the fragmentation information of the files of more than 200 computers [13]. This data set is a snapshot of the fragmentation. It could be used to detect a correlation between the creation date of a file and its fragmentation. However, it does not show how fragmentation evolves over time for a system as a whole. However, this data set can be used to validate the generated patterns by comparing them with this data set. This makes it possible to evaluate the accuracy of the user profiles used for the experiments.

[1]`https://support.office.com/en-us/article/recover-your-office-files-dc901de2-acae-47f2-9175-fb5a91e9b3c8`

Chapter 4

Requirements

R.1	The fragmentation should be created by user behaviour.
R.2	The experiment should not influence the fragmentation of the system under test.
R.3	During the experiment multiple data points should be collected.
R.4	The reports of the experiment should be user-friendly and usable for further automation.
R.5	Same user interaction should be able to run on multiple different computer systems to isolate single factors.
R.6	The data from this experiment should be able to be shared for further research.
R.7	The experiment should be able to run for a long period of time unattended.

Table 4.1: Requirements for analysing fragmentation development over time

Table 4.1 shows an overview for the requirements that are established for researching the development of fragmentation over time.

R.1 Mimic User Behaviour

The first requirement for the method is that it should try to be as close to real user behaviour. For the fragmentation patterns to be realistic, the actions that generate them should be realistic. Users use their computer for a wide variety of tasks, like text editing, video editing, browsing and e-mailing.

Also, the fragmentation is affected by the actions. Within Windows there are multiple ways to delete a file, each result in a different fragmentation pattern. When a user deletes his file using the recycle bin, Windows creates shadow files that will affect the fragmentation[20]. When a file is removed through the command line, it is directly marked as deleted without extra files. It is assumed that most users will delete files through the user interface and therefore use the recycle bin. Since this results in different patterns, it is important that actions are as close to the user behaviour as possible.

The method needs to accommodate for this diversity in software to have the ability to run analysis. Also, the supported software should be able to be extensible. This will allow for the experiment to use new software versions.

R.2 Not affected by Experiment

There multiple ways to execute user actions, but some of these need to be controlled by an application that is installed on the system under test. Installing software might influence the fragmentation patterns in a way that is not realistic since most users will not install such a tool. The impact of such instalment is unknown and therefore not wanted.

For the experiments to be realistic, it is preferred that no extra software needs to be installed on the system under test for the method to work. This requirement ensures that the fragmentation data that is generated is not contaminated by tools. To rule out the effects of the experiment, the experiment should be controlled from outside of the user space. This is true for the simulation, but also for the extraction of the measurements.

R.3 Metrics

To analyse the fragmentation data generated, it is required to gather data points over time. Data points such as the number of fragmented files and free disk space should be collected on an interval. Also, the amount of internal fragmentation, external fragmentation, and data fragmentation are interesting metrics that should be gathered by the tool. Also, these measurements are important to be gathered externally to prevent any skew in the data. The tool should be able to support more and other data points in the future.

These initial metrics allow for a first case study into the development of fragmentation over time with a basic user profile. Allowing for more metrics, will enable feature research into other effects.

R.4 User-friendly Reports

The metrics gathered during the experiments should be reported back to the researcher for further analysis. Preferably, the data should be reported in a format that is usable in other tools for further analysis. For example, files with comma-separated values (CSV) can be easily processed further by other languages and are human readable. Thereby, CSV files can be opened in Excel and plotted. This reduces the complexity of the tool, there it does not have to accommodate for a plotting system itself.

R.5 Multiple Machines

To be able to research the effects of a single factor on the fragmentation of a system, it is needed to run the same user interactions on multiple machines. This can be used to see the impact on fragmentation when tools as Dropbox are used by running an experiment with Dropbox installed and a control without Dropbox installed. By running the same interactions on separate machines, the results can be compared.

R.6 Privacy

Privacy is a very important topic today. There are laws that prohibit the distribution of personal data without consent. This also makes it harder for forensic researchers to share the data of disks which they researched. Validation of results is therefore problematic. Also the WildFrag database has been constructed with privacy in mind, by removing file names and unknown file extensions. For further research, it is important that the raw data and results can be shared freely without any privacy concerns.

To research fragmentation patterns and allow for the distribution of the patterns and data produced, they need to be the result of a privacy aware experiment. The privacy of users should not be invaded but still be realistic (R.1). However, it is important that no traceable user data is used to protect the privacy of users.

R.7 Run unattended

The last requirement states that the method should be able to run for longer periods of time unattended. This is because some fragmentation patterns are most likely to appear when the disk utilisation is high. This especially the case for data fragmentation where the data itself gets stored across the disk and is no longer in sequential order. This will only happen when the system has been used for a while.

Also, computers are used for extensive periods of time sometimes up to a couple of years. This means that the experiments should run for a long time as well. It is inconvenient for a researcher to control the experiment for its full duration to correct certain problems, therefore it is necessary that these experiment are fault and can recover.

Chapter 5

Methodology

The are multiple ways to conduct research into the development of fragmentation over time. One would be to observe the fragmentation on a wide range of computers over time. In the WildFrag database, the files of more than 200 computers have been analysed [13]. By running this snapshot multiple times the evolution of fragmentation can be followed for this set of computers. Secondly, it might be possible to derive the fragmentation over time by looking at the data set in WildFrag database. By looking at the file's creation time and its fragmentation, one might be able to conclude whether newer files have a higher chance of being fragmented. Thirdly, it is possible to run synthetic experiments. By simulating user interactions and analysing the development of fragmentation on an interval.

Method 1: Long running Observations

The first method would give a realistic overview since the data is coming from real computer users (R.1). However, the process will be taking a lot of time to gather data. Computers are being used for multiple years and over these years the fragmentation needs to be measured. Also, the data collection is time consuming. For example, it has taken Vincent van der Meer over a year to gather the information for 200 computers. Thereby, users use their computer for banking and emailing and with that there are serious privacy concerns (R.6). When using this method, it is likely that the number of computers would shrink due to malfunctioning machines, people that withdraw their consent or other reasons because of which the fragmentation data can no longer be extracted. Users also might change their behaviour over time, since they change jobs or start using different software. Because of that, it will not be clear what affects the fragmentation on a machine. Also, it will be hard to tie the fragmentation pattern to a specific profile, since this profile changes over time along with the user. This method will also not allow for research into a single factor by mirroring the user interactions (R.5). These tests will not be unattended completely. Test subjects have to come back to get their disks analysed multiple times (R.3) or a tool should be developed that submits this data over time from the computer itself to a server. This client tool would however influence the system (R.2). The metrics are thereby inflexible since they cannot be changed over the course of the research.

Method 2: Analyse Snapshot Information

Method two is based on the already gathered data by Vincent van der Meer. The files of 200 computers are stored with information about the (generally known) extension, block information and time stamps. File names and the unknown extensions are left out because of privacy issues (R.6). Most of the computers in the data set are computers used by students. This would mean that all the fragmentation patterns are more or less generated by the same behaviour. Because there is no other information stored with this data, it is not possible to derive the user profile or used software. Therefore this method can not be used to isolate one single fragmentation factor (R.5). Also, this method does not show how fragmentation evolves over time of the whole system (R.1). Lastly, it is not possible to gather new metrics for these 200 computers (R.3).

Method 3: Run Synthetic Experiments

Thirdly, the fragmentation data can be generated by means of a synthetic experiment. By running user interactions that mimic real users on a system, it is possible to create fragmentation patterns (R.1). Synthetic experiments can be conducted by having a virtual machine set up and initiating user actions like creating, modifying and deleting files. These fragmentation patterns are without any of the privacy concerns since this tool simulates users and does not have any user information (R.6). By using Virtual Machines, it is possible to set up an experiment focusing on one single fragmentation factor such as file sharing tools. By setting up two virtual machines, one with the factor and another as control, and then running the same user interactions on both machines. This will enable researchers to see the impact of a specific factor on fragmentation (R.5). This method allows for the researcher to rerun the experiment and extract different metrics (R.3). The disks created during simulations can also be used in further research into fragmentation.

For the experiment to be realistic and privacy aware, the interactions need to be modelled into user profile. The profiles should be simple. It is undo-able to incorporate each and all interactions for the whole lifespan of a system. The model needs to be simplified though realistic. Executing interactions on a system as fast as possible will not result in a realistic pattern. Windows utilises the user his pauses to do house keeping on the system such as downloading updates and removing temporarily created files. Research on type speeds, user behaviour, and coffee- and lunch breaks will be incorporated to improve the realism of the profiles.

Method Selection

The second method does not allow for an analysis of fragmentation on a systems level, since it uses a snapshot (R.3). Also it does not allow to look into factors that effect the fragmentation because the data does not contain user profiles or software installed (R.5). Since the third option has no privacy concerns and is able to test fragmentation in a limited time, this method is most suitable. Compared to the first option, the data gathering is more convenient and consistent, and does not intrude privacy (R.6). Also, since user profiles are created, tests can be rerun with different factors, to focus on a single factor or for other metrics (R.5, R.3). This would not be possible in method one, since all the metrics a researcher would like to collect need to be clear upfront. By using virtualisation that has support for a SDK, the interactions and metrics can be ran from outside of the machine. The tool would than not affect the fragmentation data on the machine (R.2). The automated interaction with the virtual machine, also allows for a consistent data extraction that can be user-friendly and further automated (R.4). Virtual machines are able to run for longer periods of time without any interaction of a researcher (R.7).

Virtualisation

For running virtual computer environments, the are numerous options such as VirtualBox, VMWare or cloud services as Azure in which one is able to create machines with the Windows Operating System. Both VMWare and VirtualBox supply a SDK with which the virtual machine can be automated. VMWare has a licensing fee associated[1] opposed to VirtualBox which is free to use on different platforms. Running virtual machines in a cloud environment will become costly since they are charged by the hour and experiments for fragmentation are long running. Since VirtualBox is free and Microsoft delivers free Windows images for VirtualBox, VirtualBox is most suitable for running synthetic fragmentation experiments. VirtualBox provides multiple implementations of their API[2]. This allows for running the virtual machines in the cloud where the API is able to communicate remotely through SOAP. However, because of the overhead accompanied with SOAP and TCP communication, the local XCOM binding is more suitable when the virtual machine is ran on the same host machine as the simulation.

Fragmentation Measurements

To gather the metrics about fragmentation, the files on disk need to be analysed. This can be done by scanning the complete disk. Since the scope is limited to NTFS. There are the options to use the

[1]`https://store.vmware.com/store?Action=home&Locale=en_IE&SiteID=vmwde`
[2]`https://download.virtualbox.org/virtualbox/SDKRef.pdf`

$Bitmap file that shows whether file clusters of the disk are used or free, and the $MFT can be analysed, which contains the meta -data for all the files on the system. The $Bitmap is enough for analysing the occupation rate of the disk. However, it is not possible to analyse all forms of fragmentation, since it does not contain information about files. Also, for the $Bitmap file to locate, the $MFT is needed. To analyse the fragmentation of the system, in stead of analysing all the files of the machine it is sufficient to extract the Master File Table. This Master File Table contains an index of all the files on the machine. By doing so, it is not necessary to scan the complete disk for fragmentation and therefore saves time. The records of the MFT store the block ranges that the files occupy on the disk. With these block ranges, the fragmentation can be calculated. Using the Master File Table allows for analysing the time-stamps in relation to the fragmentation in the future.

WildFragSim

To enable research into fragmentation development over time without any privacy implications, WildFragSim is developed. WildFragSim is the tool that allows for synthetic experiments on one or more machines using user profiles.

While developing WildFragSim, some challenges were encountered. These challenges are explained in Appendix A. The full documentation is with the source code of the project and explained in Appendix B. A simplified class diagram is appended as Appendix C. The interactions between classes is pictured in the Appendix D.

Extensibility of WildFragSim

WildFragSim is built with extensibility in mind. Not only for the software that is supported for experiments but also for WildFragSim itself.

Actions implement a common interface and use the visitor pattern. The visitor pattern allows new operations to be added to an object without changing the implementation of the object itself [16]. This means that actions can be implemented in isolation and not on the main virtual machine abstraction. The actions are exposed to WildFragSim in the `ActionFactory`. Using reflection, the actions are instantiated using the action definition from the profile. This was inspired by the same way JDBC instantiates drivers[3]. This allows for adding new actions without having to alter the implementation of WildFragSim. This is especially useful when WildFragSim is used as a library in an other tool. Because all actions are initialised on start-up, this reduces the time spent on creating objects while running the experiments. The visitor pattern allows the actions to be reused multiple times during the course of an experiment, but also on multiple machines at once (R.5). Measurements and Conditions follow the same strategy.

Model of the Master File Table

The Master File Table is the key component that WildFragSim uses to inspect for fragmentation and finding files on the disk of the virtual machine, instead of using the user interface of Windows or command line tools available in Windows (R.2). However, the Master File Table is stored in bytes on the machine. In Section 2.2.2 is explained how to locate the Master File Table. In the software, all entities such as the Master Boot Record, Volume Boot Record and Master File Table entry has been modelled as separate objects. Compared to having one object for this, having multiple allows for a good separation of concerns. Also, each object is responsible for its validation. For example, the Master Boot Record checks the signature of the bytes and its length. This allows the model to evolve in the future. The model has no binding to its origin on a virtual machine disk. This allows the model to be used separate from the experiments as a library.

Conclusion

To conclude, for researching fragmentation patterns over time this, WildFragSim is developed that simulates user interactions by running them on a virtual machine. On an interval the tool will extract the Master

[3]`https://docs.oracle.com/javase/tutorial/jdbc/basics/connecting.html`

File Table and analyse it for fragmentation patterns. By allowing the tool to run remotely, the tool will not contaminate the data of the machine.

For the fragmentation patterns to be realistic, the user profiles used for the experiments need to be realistic. The tool needs to be versatile enough to accommodate for a wide variety of profiles. Using the research done by Alves[1] and Galinsky[6], a basic profile is established to simulate an office worker. More about establishing the user profile can be read in Chapter 6.

Chapter 6

Simulating User using Profile

User profiles are there to describe the user behaviour of a real user so that WildFragSim can mimic the behaviour. To describe a real user in the form of such profile, further study is needed. Since it is complex to generalise the user profiles, WildFragSim is supplied with the ability to extend the interactions with the software.

Profiles are defined in YAML-format. YAML is a human-readable serialisation format that is supported cross-language[3]. This allows for understandable profiles that are configured easily in any text-editor and can be parsed by WildFragSim. The format is chosen because of its readability and because it is less redundant than other options, such as XML.

Scenarios

Profiles are separated into scenarios. A single scenario is built up out of one or more actions. A scenario is an atomic group of actions that will not be interrupted. For example, clicking on the Word icon is an action and writing a text file is a scenario. Writing a text exists out of clicking the Windows start icon, finding Word, opening a new blank text document and finally actually writing the text. Grouping actions into scenarios allows for reusable building blocks to be created.

Scenarios are given a weight. This weight is being used to select a scenario for execution. This allows for a seemingly random behaviour, but with the possibility to make some scenarios more recurring during the simulation. For example, it more likely to write a document than to delete one.

Actions

Each action is either one single click, a series of keystrokes, or a composition of multiple actions to make one new action. The actions validate the state of the virtual machine and do basic error handling. For example, it will check whether a a window is already open or not and it locates a file in the Master File Table that can be edited or deleted.

Actions need to be as close to what the user does as possible. For example, it is important for the fragmentation patterns to move files to recycle bin first. According to the analyses done by Yang et al., files that are moved to the recycle bin are renamed and an extra file is created to accompany the original file[20]. When the recycle bin is emptied both files are marked as deleted in the Master File Table. When someone deletes a file directly by bypassing the recycle bin, the file is directly marked for deletion in the Master File Table without any renames or extra files being created.

Conditions

Conditions can be used in profiles as a guard for a scenario. For example, it is not possible to delete a file that has not been created. For these scenarios a condition can be added, that the file should be available before it executes that scenario. By default, the `Always` condition is used and so by default each scenario is run.

File state

WildFragSim uses the model of the Master File Table to locate files. By creating the files with a common prefix, it is possible to relocate the file in the Master File Table in another scenario. Since the file name is part of the $FileName-attribute in the Master File Table, the actual name has no effect on the fragmentation.

By using the Master File Table, WildFragSim does not have to incorporate a database with files it has already created and can re-use.

Typist Model

No user will write a full Word document at once. Therefore WildFragSim incorporates the ability to define a typist model. The breaks and pauses between keystrokes allow the operating system to do its house keeping, such as deleting temporary files and auto-saving Word documents.

The first basic user profile is configured based on research done by Alves, Rosenvinq and Galinsky. The group studied by Alves typed four words and then paused for five seconds, called burst length and pause length. The pause between keystrokes was 320 ms on average. Since office workers will take breaks, the break schedule from Galinsky is used. Galinsky stated that a conventional break schedule comprises of a 15 minute break during the first shift of four hours, a lunch break of 30 minutes, and a 15 minutes break during the second shift[6]. For this basic profile, it is expected that the system is terminated outside of office hours. For WildFragSim to continue running for a longer period of time without having to wait for a full night while the machine does nothing, the profile is configured to work for two hours consecutively, pause for 15 minutes to simulate a coffee break, work for two more hours, and then pause for 30 minutes.

The pause in between keystrokes is not the same for each user. It is also affected by stress and mood as discovered by Vizer[19]. Rosenqvist states that since the standard deviation is high, there is no single pause value[17]. Alves distinguishes nine separate pauses between keystrokes. It is not to be expected that the differences in these pauses will influence the fragmentation on disk. Therefore all the pauses between keystroke are configured as one. Next to the keystroke pause, the profile accepts a word pause that is the pause between each word in a sentence. Also, the burst length and burst pause are incorporated in the profile. Burst length is the number of words a user can type and the pause between each burst is called burst pause. The pauses and their location are visualised in Figure 6.1.

$$\{\text{The } P_w \text{ quick } P_w \text{ brown } P_w \text{ fox}\}\ (B_l = 4)\ P_b\ \{\text{jumps } P_w \text{ over } P_w \text{ the } P_w \text{ lazy } P_w \text{ dog}\}\ (B_l = 5)$$

Figure 6.1: Location of pauses within a text, keystroke pause is omitted in the figure (P_w = word pause, B_l = burst length, P_b = burst pause)

Using the mean and standard distribution found by researchers, it is possible to approximate the same behaviour. A Gaussian pseudo random number generator generates values that are normally distributed[1]. Since there is no limit on the value returned, it is important to constraint the value. A pause can for example never be negative.

Table 6.1 displays the differences between the researched values and the values that were simulated using the Gaussian random number generator. For the purpose of this research, these values are close enough. These pauses give Word the chance to activate its auto recovery, in which it will write a copy to disk.

Keystroke Pause	**Researched by Alves**	**Simulated in WildfragSim** (n = 1000)
Mean	320 ms	323 ms
Standard deviation	140 ms	139 ms

Table 6.1: Comparison between researched and simulated keystroke pause times using Gausian method

[1]https://docs.oracle.com/javase/8/docs/api/java/util/Random.html#nextGaussian--

Limitations

The model is not able to adopt to situations that it has never seen before. For example, exceptions or failures in the software or Windows will not be handled. It is not able to confirm or decline pop-up questions when it is within a scenario.

The typist model approximates the typing speed of a real user, including its pauses and even a break schedule. However there are some limitations to this model. The model will type only in forward direction. This means that it will not correct texts. From the study done by Vizer, we know the ratio of the delete key. However, it does not how much text is deleted before the user retypes the text.

Also, the typist model is not able to alter texts that it has written before, such as rewriting a specific paragraph in the middle of the text, adding new headers or correcting an earlier spelling mistake. No studies were found that studied this user behaviour. It is expected that this behaviour will influence the fragmentation of Word files. Word will either rewrite the complete file to disk again or do a partial update.

The way the break schedule is implemented, it will only pause after the complete scenario is finished. This means that for the basic profile, the file is always saved before pausing. In real life, the user might leave his Word open during a break. However, it is not expected that this will influence the fragmentation.

Chapter 7

Case Studies using WildFragSim

7.1 Master File Table Observations

WildFragSim contains a model of the Master File Table that can be used for further research. To validate whether the model is correct and that parsing of the raw bytes on disk is done correctly, this experiment is set up.

Setup and Execution

For validation of the model, a Windows 10 virtual machine supplied by Microsoft is used[1]. WildFragSim is included as a library to analyse the disk.

Results

The virtual machine has 128,094 Master File Table-entries of which 125,146 are active, that is 93,354 files and 31,792 directories. This is the clean machine, before booting for the first time.

A Master File Table entry consists of multiple attributes that describe the size of the file. There are one or more file name attributes, which have a real size and allocated size. Also there is the data attribute for each data stream. The data attributes contain a real size, allocated size and a data stream size. The real size is the actual number of bytes in used by the content of the file. The allocated size is the sum of bytes from the allocated clusters. The data stream size is the amount of bytes that the stream is on disk, this size is different from the real size in case that the content of the file is compressed by the file system or if the file is sparse.

During analysis, it is observed that 1,341 files have an allocated size in the data attribute that is bigger than the disk itself. These files are already on the disk before the initial boot of the virtual machine and belong to the operating system or supportive software. Also, the files are uncompressed and not sparse.

The sizes in the $FileName-attributes are not always populated. For 61,955 the sizes in the file name attribute are empty.

Discussion

It is expected that the conversion from bytes to the JAVA long to represent the size is wrong. However, validation with other files - big and small - do reveal this symptom. Hard links are also ruled out as a cause, since hard links add an extra file name attribute and do not change any information of the data attribute. Since each data attribute corresponds with exactly one data stream, the is not caused by alternative data streams. Table 7.1 depicts one file that is observed to have divergent file sizes. In this specific example the allocated stream size is 115,200 times bigger than the size of the disk. According to the documentation of NTFS, the size in specified in the standard attribute header on offset 40 and is 8 bytes long[2]. See Figure 7.1

[1]`https://developer.microsoft.com/en-us/microsoft-edge/tools/vms/`

[2]`https://flatcap.org/linux-ntfs/ntfs/concepts/attribute_header.html`

File name:	hh.exe
Real size $FileName-attribute:	18 kB
Allocated size $FileName-attribute:	20 kB
Real size $Data-attribute:	18 kB
Initial stream size $Data-attribute:	18 kB
Allocated stream size $Data-attribute:	**4608 TB**
Disk size:	**40 GB**

Table 7.1: Example of file with bigger data allocation size than the disk found during analysis of the Master File Table

```
     00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00:  80 00 00 00 48 00 00 00 01 00 00 00 00 00 04 00     ....H...........
10:  00 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00     ................
20:  40 00 00 00 00 00 00 00 00 50 00 00 00 00 12 00     @........P......
30:  00 48 00 00 00 00 00 00 00 48 00 00 00 00 00 00     .H.......H......
40:  31 05 95 31 23 00 00 00                             1..1#...
```

Figure 7.1: The hex-dump of the $Data-attribute from a file that has a bigger allocation size that the disk, bytes that mark the allocation size are in bold

for the bytes that make up the allocated file size. In this case `00 50 00 00 00 00 12 00` represents the value 5,066,549,580,812,288 bytes. However, when the last two bytes are ignored, the value is only 20,480 bytes or 20 kilobytes. This corresponds with the size found in the file name attribute. Over the course of this research project, no explanation has been found for the meaning of the `12` byte in the allocation size. Not all files with an allocation size that is bigger that the disk have the same byte value on that position. There seems to be no pattern on the specific value for that byte. The same values can be observed in other tools like analyzeMFT[3]. SleuthKit does not present an allocation size with the attribute. It only displays the stream size, real size, and data runs[4].

In case the file name sizes are empty, locating the file using the Windows Explorer and enabling the detailed view will list sizes on screen. After that, the sizes are also updated in the file name attribute of that file in the MFT. From this behaviour, it looks like Windows uses the sizes in the file name attribute to present to the user and only updates them when the user requests them. Files created with an application by the user will always have the file sizes filled in the attribute. File name attributes are available for different name spaces such as POSIX, Win32, DOS, or Win32 and DOS combined. The sizes between the file name attributes in different name-spaces are not in sync while still referencing the same file. Windows will only update sizes for the Win32 or Win32 and DOS name-spaces.

7.2 Fragmentation over Time using Basic Office Model

WildFragSim is built to study the fragmentation by doing synthetic experiments. By using a user model, it simulates user behaviour. This case study runs an experiment using a basic model of an office worker using Microsoft Word.

Setup and Execution

For this experiment a Windows 10 virtual machine is used, that is fully up to date. After updating the machine, Microsoft Word is installed from the Office 365 website. During this experiment, Windows updates

[3]https://github.com/dkovar/analyzeMFT/blob/master/analyzemft/mft.py
[4]https://github.com/sleuthkit/sleuthkit/blob/4a9c6c02ac076e63f44c251241666856ba04d231/tsk/fs/ntfs.c#L4776

were disabled, so that it would not intervene with the results. To authenticate Microsoft Word, a student license from SURFspot[5] is used. The virtual machine has a disk space of 40 gigabytes.

WildFragSim was configured using a basic model of an office worker. For this experiment, WildFragSim will run 1,000 scenarios of which 60% are creations, 30% modifications and 10% deletions. For typing documents the number from Galinsky[6] and Vizer[19] are used to simulate a typist including keystroke pauses, word burst lengths, word burst pauses, and word pauses. Also, normal users would interact with other software as well, like a browser and an email client. These were left out of scope so that this experiment only focuses on Microsoft Word.

During the experiment WildFragSim will create, modify and delete Microsoft Word files randomly. Every 100 scenario's WildFragSim extracted the Master File Table to analyse the fragmentation that occurs on the virtual machine. For the internal fragmentation, files that have a size bigger than the disk space were ignored (see Section 7.1).

Results

These are the results of the first experiment using WildFragSim to test the feasibility of the tool. It took over 30 hours to fully complete experiment. At the beginning of the experiment, the data fragmentation is only 0.92%, the internal fragmentation is close to nothing with 0.01%, and external fragmentation is already at 38.13%. As stated in Section 7.1, the machine has 125,146 MFT-entries. However, after the machine is set up with Microsoft Word and all the Windows updates applied, the machine has 223,015 entries in the Master File Table, of which 217,355 are not deleted. Of this 217,355 entries, 165,947 are files and the other 51,408 are directories.

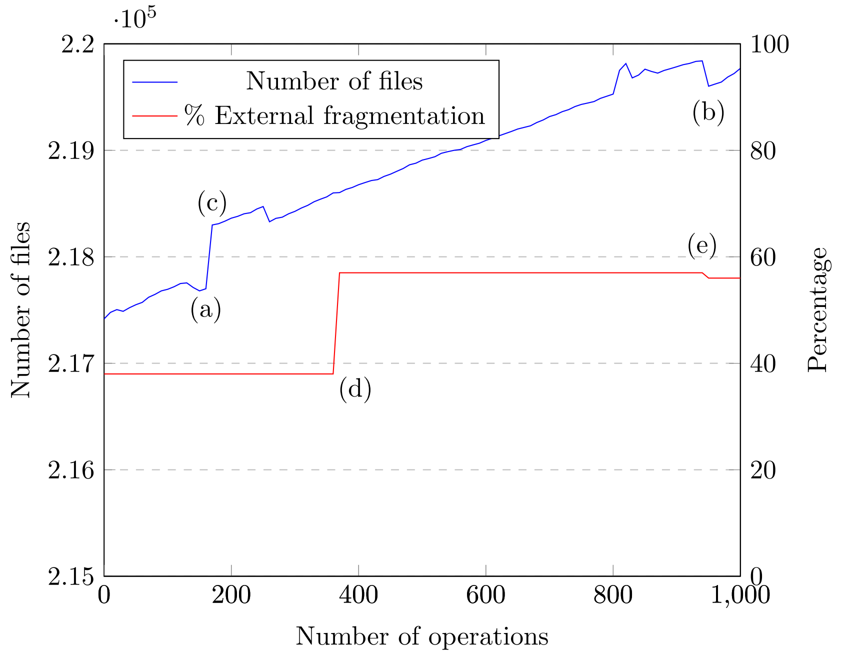


Figure 7.2: The change in number of files and external fragmentation caused by experiment with a basic user profile simulating an office worker

Figure 7.2 shows the course of the number of files and the external fragmentation. It was found that data fragmentation and internal fragmentation did not change significantly during this experiment.

At the end of this experiment, 2,353 were added to the virtual machine while only 438 files were created by WildFragSim directly. Interesting to see is that almost every scenario that involved creating a Word document creates more than five temporary and recovery files. When the virtual machine is not busy,

[5]https://www.surfspot.nl/

Windows decides to clean the temporarily created files as can be seen around scenario 160 (a) and 950 (b). Around these time, the simulation was interrupted for 15 minutes, to simulate a coffee break. It is not clear what occurred after 170 scenarios (c) with sudden boost in number of files. At the end of the experiment 415 files created by WildFragSim remained active on the machine. All 23 records of files real documents that were deleted were to be found in the Master File Table. This means none of the records were reused by the operating system.

The external fragmentation is stable with only two changes. The first one around 390 scenarios (d) where the fragmentation increased, and a small drop after 950 scenarios (e).

The 438 files created during this experiment are spread across the disk. Figure 7.3 shows the distribution of file creates across hundred evenly sized block ranges of the volume. Most files were created in the second halve of the volume, this is also where the most space was left.

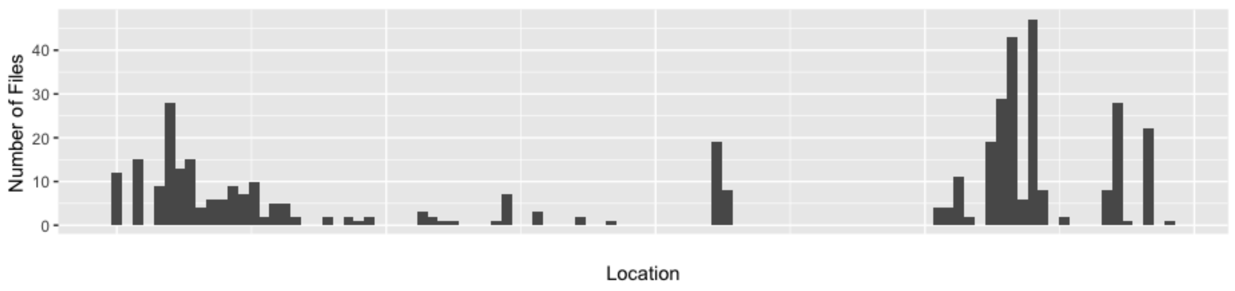


Figure 7.3: Distribution of created files across 100 evenly sized buckets of the volume (n=438)

Discussion

Initial level op external fragmentation is 38.13%. This is because the operating system keeps free space right after the Master File Table, so that it can grow when more files need to be added. At a glance, the sudden change in percentage of 20 percent point seems irrational. However, it can be explained by looking at the formula for external fragmentation from Section 2.3.2. When files are placed in gabs smaller than the biggest consecutive space of empty clusters, the percentage will not change. However, when one file is placed in the middle of the biggest gab, the percentage might be affected by a lot since the biggest gab changed in size or is no longer the biggest.

The files created during the experiment are spread across the disk. There is no distinct order in the allocation of these files, so Windows does not allocate the files from lower to higher blocks. Also, Windows does not try to write the file into the smallest gap of continuous free blocks (best fit). This is explained by the sudden jump in external fragmentation and also by the fact that gaps of the exact same number of blocks as the file are still available after the experiment while files were allocated in bigger spaces. The data suggests that the space allocated by the operating system allows the file to grow without having to fragment the file. Running the scenario of creating the file in isolation reveals that multiple recovery files are kept and that allocation seems arbitrary.

7.3 Fragmentation caused by Windows updates

This case study shows how WildFragSim can be used to study a single factor. It is clear that with 1.000 scenario's the effects are limited. But there are use cases that have a bigger impact on the number of files of the system. The fragmentation is more affected by for example installing Windows updates.

Setup and Execution

Using a clean virtual machine with Windows 10, Windows will start downloading the pending updates when the machine is started and logged in for the first time. When it has downloaded all the updates, it will install them. Sometimes, installing and activating the updates require a reboot of the machine. This is done manually over the course of the experiment. Measurements are taken at the start and end of the experiment.

Result

Installing all the updates for the virtual machine meant downloading and locating 79,413 files and folders in 30 minutes. While this is a lot of files in a short time period, the amount of external fragmentation was barely affected. It dropped 1.1 percentage point from 38.6% to 37.5%. Internal fragmentation increased from almost nothing (0.009%) to 0.01%, a 17% increase compared to before the updates. The amount of data fragmentation on the disk decreased from 1.6% to 0.9%.

Discussion

The internal fragmentation increased after downloading and installing all the updates. The reason for this is that of the 79,413 files added none were exactly a multiple of the cluster size. This indicates that the files were written to disk as consecutive clusters. Together with the change in file fragmentation, it is to be expected that most files of the update were positioned in a small gab without splitting the files in multiple fragments.

Chapter 8

Future Work

This research presents preliminary results on the development of fragmentation over time. It shows the possibility for running synthetic experiments to investigate fragmentation using user profiles.

User Model

The minimalistic user behaviour that simulates a simple office worker should be expanded and validated. For example, other regular office tools such as an e-mail client and spreadsheets can be included. The fragmentation patterns generated using this method should be checked on realism. This can for example be done by comparing the disks from virtual machines with the fragmentation patterns of 200+ computers available in the database WildFrag[13].

Further study can be conducted in to setting up realistic profiles. For example, the approach Rosenqvist took to analyse the keystrokes of users might be usable as input for the synthetic experiments[17]. He logged every keystroke of the user including pauses. This is different from the current approach where the model will calculate pauses based and is only able to type in a forward motion. Developing the user model to better approximate real-life situations will improve the results from the synthetic experiments.

The current model lacks the ability to do partial rewrites of documents, such as the correction of spelling and altering the layout of the texts. It is expected that these actions will influence the fragmentation of Word documents. Also, the model is missing the ability to change its behaviour based on the time. It is expected that users do different things on their computer based on time.

Automation of Virtual Machine

It has occurred that when the Windows system was updated, that the icons that are being used by WildFragSim changed. This results in WildFragSim not being able to locate the buttons anymore. In this study, the option was ruled out to install supportive software on the virtual machine. In case it can be proved that the impact is small or acceptable, it is possible to use the Microsoft Active Accessibility. This will give more control over the virtual machine and the ability to validate the state more precisely. In case of Microsoft Word not responding, the Accessibility will be able to detect this. To further improve the stability of WildFragSim it is advised to incorporate a retry strategy and the ability to reset the machine when an ill state is detected. Just like a regular user would reboot his machine when it renders unresponsive.

Master File Table Model

WildFragSim is currently limited in the number of files it can inspect. This is because it uses a byte array to store the data from disk in memory. JAVA byte arrays have a maximum element count of 2^{32}. This means only files smaller than 4 GB can be parsed. In case of the Master File Table, this would mean it can only contain 4,194,304 entries of 1024 bytes in size. This is sufficient for regular use cases, since the average file count in the WildFrag data set is 305,713[13]. The biggest volume in the set has 1,742,137 files.

Real-time Limitation

Also, the current experiments run on a real-time. This means that one hour of simulation is one hour in real-time. Conducting an experiment that simulates the life-span of a computer will than take an awful lot of time. Research should determine whether it is possible to speed up these simulations. This would allow for faster results and more iterations. There are possibilities in VirtualBox to speed up the virtual machine[1]. This would also influence the speed of the typist. However, it is not possible to speed-up Windows updates. Also, it is not clear if Windows itself would behave differently. Currently, the speed of the Windows virtual machine is limited by the host machine.

Utilising WildFragSim

Further, the method can now be used to study the affects of software and behaviour on tooling. Factors that might affect fragmentation include but are not limited to:

1. different Windows versions
2. file sharing tools e.g. DropBox
3. the initial disk occupation or disk size
4. tools that use sparse files e.g. BitTorrent

Based of the results, other metrics might be more interesting such as file order on disk. As showed, internal-, external- and data fragmentation are not that heavily affected by a minimalistic office user while it produced more than 2,000 documents.

The allocation strategy for the files has not been clear. Data suggests Windows is not using purely a best-fit approach to allocate the data. Using this method, further research into the allocation strategy can be conducted.

[1] `https://www.virtualbox.org/manual/ch09.html#fine-tune-timers`

Chapter 9

Conclusion

This research strives for a method that allows research into fragmentation over time. WildFragSim can run synthetic experiments by simulating user actions for longer periods of time and analyses the system's Master File Table for the development of fragmentation. By using virtual machines, WildFragSim can be used to isolate fragmentation factors and study them in depth (R.5). This all, without any privacy concerns and without contaminating the measurements (R.2).

Using synthetic experiments based of user profiles that mimic users, WildFragSim is able to aid the research into fragmentation over time without privacy concerns because no real user information is used (R.6). The model of a user is based of the research done by others to mimic its behaviour as closely as possible and allows for further extension (R.1).

Using a basic model using pause time, keystroke burst, and break schedule, the first results showed that internal fragmentation and data fragmentation only change slowly. External fragmentation is affected and changes rapidly and suddenly, because of the allocation strategy used by the operating system. Windows Updates have a big impact on the number of files. Although the number of files is big, the impact on fragmentation is small.

The preliminary results in this research demonstrates that the synthetic experiments are able to generate fragmentation patterns using a user model. Using the Master File table, the fragmentation can be measured on an interval so that the fragmentation development over time can be analysed. Improving the user model used, will improve the fragmentation results.

However, WildFragSim also presents a problem in using images as user interactions. The tool lacks insights into the state of the application under test and of the machine as a whole. This makes experiments fragile. Changes in the software such as a new version might render the current set of images of the controls useless. Also, running the experiment for an extended time proved to be challenging. During the run, Word or Windows sometimes become unresponsive. It is sometimes uncertain whether the Microsoft Word is fully ready to accept input and the boot time is inconsistent. Also Word can crash and become unresponsive. In such a case WildFragSim will retry the action, however there were moments this did not resolve the issue. The only resolution at those moments was to reboot the virtual machine and restart the experiment. These leads to the conclusion that the requirement for running over longer periods of time unattended is not fully met (R.7).

Bibliography

[1] Rui A Alves et al. “Influence of typing skill on pause-execution cycles in written composition”. In: *Writing and cognition: Research and applications* (2007).

[2] Sebastian Bauersfeld and Tanja EJ Vos. “User interface level testing with TESTAR; what about more sophisticated action specification and selection?” In: *SATToSE*. 2014, pp. 60–78.

[3] Oren Ben-Kiki, Clark Evans, and Brian Ingerson. “Yaml ain’t markup language (yaml^{TM}) version 1.1”. In: *yaml. org, Tech. Rep* (2005), p. 23.

[4] Hal Berghel and Natasa Brajkovska. “Wading into alternate data streams”. In: *Communications of the ACM* 47.4 (2004), pp. 21–27.

[5] Adam Chapweske. “The PS/2 mouse/keyboard protocol”. In: *electronic file available: http://www.computer-engineering.org/ps2protocol* (2003).

[6] Traci L Galinsky et al. “A field study of supplementary rest breaks for data-entry operators”. In: *Ergonomics* 43.5 (2000), pp. 622–638.

[7] Simson L Garfinkel. “Carving contiguous and fragmented files with fast object validation”. In: *digital investigation* 4 (2007), pp. 2–12.

[8] Cheng Ji et al. “File Fragmentation in Mobile Devices: Measurement, Evaluation, and Treatment”. In: *IEEE Transactions on Mobile Computing* (2018).

[9] W Volvano Jonathan. *Embedded Microcomputer Systems-Real Time Interfacing*. 2011.

[10] Nikolaos Kaklanis et al. “Towards standardisation of user models for simulation and adaptation purposes”. In: *Universal Access in the Information Society* 15.1 (2016), pp. 21–48.

[11] Peter S Magnusson et al. “Simics: A full system simulation platform”. In: *Computer* 35.2 (2002), pp. 50–58.

[12] Marshall Kirk McKusick, George V Neville-Neil, and Robert NM Watson. *The design and implementation of the FreeBSD operating system*. Pearson Education, 2014.

[13] Vincent van der Meer. *WildFrag*. Data set containing file information of 200+ computers. 2018.

[14] Bruce J Nikkel. “Forensic analysis of GPT disks and GUID partition tables”. In: *Digital Investigation* 6.1-2 (2009), pp. 39–47.

[15] Anandabrata Pal and Nasir Memon. “The evolution of file carving”. In: *IEEE signal processing magazine* 26.2 (2009), pp. 59–71.

[16] Jens Palsberg and C Barry Jay. “The essence of the visitor pattern”. In: *Proceedings. The Twenty-Second Annual International Computer Software and Applications Conference (Compsac’98)(Cat. No. 98CB 36241)*. IEEE. 1998, pp. 9–15.

[17] Simon Rosenqvist. *Developing pause thresholds for keystroke logging analysis*. 2015.

[18] Russell Sears and Catharine van Ingen. “Fragmentation in large object repositories”. In: *arXiv preprint cs/0612111* (2006).

[19] Lisa M Vizer, Lina Zhou, and Andrew Sears. “Automated stress detection using keystroke and linguistic features: An exploratory study”. In: *International Journal of Human-Computer Studies* 67.10 (2009), pp. 870–886.

[20] Christopher C Yang et al. *Intelligence and Security Informatics: IEEE ISI 2008 International Workshops: PAISI, PACCF and SOCO 2008, Taipei, Taiwan, June 17, 2008, Proceedings*. Vol. 5075. Springer, 2008.

[21] Tom Yeh, Tsung-Hsiang Chang, and Robert C Miller. "Sikuli: using GUI screenshots for search and automation". In: *Proceedings of the 22nd annual ACM symposium on User interface software and technology*. ACM. 2009, pp. 183–192.

Appendix A

Implementation Challenges of WildFragSim

Booting Windows in a Virtual Machine

Most experiments will need to boot and/or reboot the machine as part of a user profile. This is for example need to install updates in Windows. On a regular computer the user will press a button to boot the machine and then wait for the machine to be fully booted.

Virtual Machines in VirtualBox behave like regular computers. There is a "switch" that turns it on and from the outside there is now automatic way do detect when the machine is ready for interaction. The same thing is true for a Virtual Machine. It is not clear when a Virtual Machine is fully booted and interactive. An operating system is interactive when it is ready to receive key presses and mouse clicks.

Normally a user would use to screen to tell whether the machine is ready. WildFragSim applies this same technique. All VirtualBox images supplied by Microsoft are password protected. The password is `Passw0rd!` by default. This forces that Windows shows a lock screen after it has booted. WildFragSim checks whether this screen is visible and looks for the input field. When the control is visible, the machine is expected to be interactive. More about finding the controls on the screen can be read in Subsection A.

Microsoft Office

Text editing is a common practice on a computer and should therefore be able to be handled by WildFragSim. However, Microsoft has no free version of Word available. Thereby, later versions of Microsoft Office - of which Word is part of - encourages users to use the cloud version, Office 365. This version stores all files that the user works on in the cloud. It is expected that the fragmentation patterns are affected since there will be less activity on the disk.

To use WildFragSim in combination with Word a private (student) license was acquired to run experiments. When Word is not an option, LibreOffice or OpenOffice might be a good alternative. Both tools support the Microsoft Word Office file formats. This does not mean that they generate the same fragmentation patterns as Microsoft Word. WildFragSim is extensible enough to accommodate for other text editors.

Python vs JAVA

WildFragSim relies on automating the Virtual Machine through the API provided by VirtualBox. The bindings that VirtualBox provides are there for programming languages including but not limited to Python, C and JAVA.

Different Python libraries have been created that interact with VirtualBox. There exists an abstraction layer called pyvbox[1] that delivers conveniences around the API such as converting string into scan codes.

[1] `https://pythonhosted.org/pyvbox/`

So the first option was to create WildFragSim in Python. However, Python has two version that are widely used, Python 2.7 and Python 3. Python 2.7 is deemed deprecated.

It is generally ill-advised to use deprecated tooling. Since Python 2.7 is deprecated, it is preferred to use the newer Python 3. This version has also support for type hints[2]. Unfortunately Python 3 is not supported by VirtualBox for all platforms[3]. Oracle does not ship the bindings for Python 3 on MacOS at this moment. For Linux based systems, the developer needs to compile the bindings himself. It is preferred that WildFragSim can run on all operating systems. This currently rules out Python as language for WildFragSim.

Next to Python, JAVA is supported by VirtualBox. It also includes a remote binding that allows for the control of virtual machines that run somewhere else. However, no layer such as pyvbox exists for JAVA. This makes converting strings to input for the virtual machine harder. The VirtualBox API makes use of virtualised keyboards and mice. These keyboards do not work with regular strings. The strings need to be converted to scan codes like real input devices. For multiple key presses such as a capital letter or special character multiple scan codes need to be send. Also when keys are released, there are separate scan codes that need to get send. The scan codes are available in the PS/2 protocol [5]. More about the protocol can be read in Section 2.5.

These scan codes differ per keyboard layout. So, a QWERTY-keyboard has different scan codes than an AZERTY-keyboard. However, the scan codes are not affected by the language settings of the computer. Special characters could get mangled when the layout of the keyboard does not match the language settings. This would be exactly the same when a normal user would misconfigure its keyboard settings. The interpretation of some of the scan codes is different depending on the NumLock state. For example the arrow up is an eight when NumLock is enabled. To ensure the correct meaning, WildFragSim checks the NumLock-state based of the exposed keyboard LED status and changes this state accordingly.

The mouse in VirtualBox accepts coordinates, so no complex algorithms are needed to move the mouse across the screen.

Considering that the compatibility problems between Python and VirtualBox on different operating systems are blocking and JAVA is a strong typed language with the support of multiple bindings of the VirtualBox API, WildFragSim is constructed in JAVA. In the design of WildFragSim, an abstraction is available for the keyboard scan codes.

Find Controls on Screen

One of the requirements of WildFragSim is that it should not influence the fragmentation patterns on the machine (R.2). To find controls on the virtual machine without having to install extra software on the machine, WildFragSim uses image recognition. This means the tool will look for a control on the screen and then interact with it, like a real user would do. This eliminates the need of knowing the exact location of the control and to install extra software. This concept has been applied by the tool Sikuli before [21].

Since Sikuli uses OpenCV to do template matching, it is more flexible and faster to re-implement this behaviour in WildFragSim, so it can be tailored to its needs. OpenCV is able to detect subtle difference between the template or haystack and the needle[4]. However, differences in size are not detected. WildFragSim will therefore only be able to find controls that match in size.

Figure A.1 depicts the two states of the recycle bin in Windows 10. When template matching supplied by OpenCV is used, WildFragSim is able to recognise both images as the recycle bin. It will not see the differences between a full or empty recycle bin. This makes it simpler to locate the recycle bin or other icons on the screen. In cases that it is necessary to know whether the icon matches exactly, OpenCV supplies a confidence score. The higher the score, the more the templates matches.

Another option for finding controls on the screen is the Microsoft Active Accessibility API[5] that comes with every Windows installation. It is also used by TESTAR for testing software applications[2]. However, this API is not accessible from the outside of the virtual machine. To use this API, it would mean that software needs to be installed on the machine to expose this API to WildFragSim. Installing this component,

[2]`https://docs.python.org/3/library/typing.html`
[3]`https://www.virtualbox.org/ticket/18366`
[4]`https://docs.opencv.org/2.4.13.7/doc/tutorials/imgproc/histograms/template_matching/template_matching.html`
[5]`https://docs.microsoft.com/en-us/windows/desktop/winauto/microsoft-active-accessibility`

(a) Empty recycle bin (b) A full recycle bin

Figure A.1: Two states of the recycle bin in Windows 10

would affect the fragmentation measurements on the machine. It therefore is not used by WildFragSim, because it does not ad-here to requirements set (R.2). It could definitely be an option to make WildFragSim more versatile and robust.

Measure Fragmentation on Virtual Machine

To measure the fragmentation that is happening on the disk we need to analyse the file system of the virtual machine. The hard disk of virtual machines used by VirtualBox are stored as files on the host system. These file are a (sparse) compressed representations of a file system and cannot be read directly. The files are in VDI format and can be accessed through the VirtualBox cli and API. The VDI file of a clean system with a 40GB virtual disk is already 11GB on the host system.

The Master File Table of the virtual machine will be used by WildFragSim to detect fragmentation. The MFT can be read from the raw disk. More on the structure of the Master File Table and locating it on disk can be read in Section 2.2.2. To extract the MFT from the VirtualBox disk, there are three options.

1. Convert to RAW disk so it can be analysed by existing tooling such as SleuthKit.
2. Use direct access to the medium of VirtualBox.
3. Use the explorer provided by VirtualBox

Option one is to clone and convert the complete disk into a readable format. This readable format can then be analysed by SleuthKit. SleuthKit is able to extract the MFT for further analysis.

This has two major disadvantages. The file needs to be cloned and converted to raw. This method, to extract the body file from the VirtualBox image is described by Adrea Fortuna[6]. The scale of the raw disk image will be one on one. So a 40GB virtual disk will be 40GB on the host machine. This is next to the VDI file, which is already 11GB for an empty Windows 10 installation. Next to the amount of space required for this option, cloning and converting takes more than 15 minutes every time the analysis is run. The time will increase when the virtual file system is bigger.

The second option is to use direct access to the disk. VirtualBox supplies a method to access the bytes on the system of the virtual machine directly. This is limited to only 256 kilobytes per call. The downside is that the virtual machine needs to be stopped in order to have the disk accessible. Stopping the virtual machine and constructing the MFT directly from disk, takes only a minute. Since the later option is faster than the first one it will stall the simulation far less.

Thirdly, according to the API documentation supplied by VirtualBox, there is ability to access files on a disk of virtual machine directly by their file name. There is an implementation for a file explorer in the API[7]. Unfortunately, the explorer is not implemented for the VirtualBox machines that run a Windows operating system. This would also be against the requirement of not installing extra software.

Considering the disk space and time option one needs to convert a disk and option three is not provided, WildFragSim uses the direct access to extract the Master File Table from the virtual machine. It therefore needs to analyse certain parts of the disk on a byte level.

[6]`https://www.andreafortuna.org/dfir/extract-filesystem-bodyfile-from-a-virtualbox-vm/`
[7]`https://www.virtualbox.org/sdkref/interface_i_v_f_s_explorer.html`

First, the Master Boot Record is read from the medium. The MBR need to be interpreted in order to gain the address of the main partition on the disk[8]. The Master Boot Record is located in the first 512 bytes of any disk. Using the analysis done by Nikkel, the Master Boot Record is bisected [14]. It consists of the executable code that is used for bootstrapping the computer. Next to that is a partition table that consists of four entries. It finishes with a consistent boot signature that is always the same.

Modern computers use GPT since this is able to accommodate for bigger and more partitions. Also the virtual machines supplied by Microsoft use GPT. When a machine is using GPT, the first sector of partition table in the MBR will contain a directive to flag that the system should use the GPT table which will be located on the second sector of the disk. The MBR should have only one record in its partition table, which is not bootable and has type `0xEE`, according to Apple[9]. This is called a protective MBR, see also Section 2.1.

GUID Partition Table uses GUID to define the type of partition. GUIDs are Globally Unique Identifiers consisting out of 128 bits. They are also referred to as UUID what stands for Universally Unique Identifiers in UNIX operating systems and most programming languages including JAVA. Transforming the byte information into the correct GUID has proven to be more complex. The byte-order or endianness of the GUID on disk do not match the one of the JAVA ecosystem. The GUID of the disk are in mixed-endian. The first two groups of respectively four and two bytes are little-endian, the remaining two groups are big-endian. However, JAVA expects all bytes to be big-endian when using the `java.util.UUID` class. WildFragSim converts the format. A more in depth documentation of the GUID Partition Table is provided by Microsoft[10].

When the GPT has been parsed, each record record of the Basic Data Partition type can be checked for a Volume Boot Record. However, not all Windows VirtualBox images use GPT. In that case the MBR is not protective and contains a regular partition table.

On the first sector of a volume contains the Volume Boot Record. This VBR describes the file system that is used on the volume. Also, it contains information of where the Master File Table is located[11].

The first MFT entry on a disk is always the entry for the MFT itself. Since the MFT behaves like a normal file, the MFT itself can be fragmented. To get the complete Master File Table, the first entry is parsed. This entry tells where all parts of the MFT are located. This is done by inspecting the data runs of the first record. More about the structure of an MFT entry can be read in 2.2.1.

All bytes for the MFT are than read and parsed to an internal model of the MFT in WildFragSim. WildFragSim is than able to deduct the fragmentation.

[8]`https://www.cgsecurity.org/STDMBR.htm`

[9]`https://developer.apple.com/library/archive/technotes/tn2166/_index.html#//apple_ref/doc/uid/DTS10003927-CH1-SUBSECTION11`

[10]`https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2003/cc739412(v%3dws.10)`

[11]`https://medium.com/@bromiley/a-journey-into-ntfs-part-3-5e197a0cab58`

Appendix B

Usage of WildfragSim

Setting up an Experiment

To run an experiment in WildFragSim, the researcher sets it up using a YAML file. In this YAML file, the researcher specifies the virtual machines that are used, the actions that need to be executed at the beginning of the experiment such as booting the virtual machine, the actions for tearing down the experiment at the end, the measurements that need to be collected, the interval of these measurements, and the scenarios and their ratios. These ratios allow tuning the distribution of scenarios. WildFragSim will weight the scenarios and chooses random what scenario it is going to execute. Scenarios are made up out of actions. Therefore in the YAML is a list of actions per scenario that are executed as an atomic operation.

The researcher needs to collect images for the controls of that application. The profile can reuse the existing actions and supply a different images or when the action is not supported a complete new action must be implemented.

Before the experiments can actually run, the researcher also needs to set up the initial virtual machines. It is to be advised to have the machines up to date clean machines and use clones to create virtual machines with the correct software installed and configuration. It is useful to create a snapshot at that moment, so the experiment can be reset to run again.

Running an Experiment

To run WildFragSim, JAVA 8 needs to be installed and available on the machine. Maven is used to download the dependencies and package the application.

Running an experiment using WildFragSim can be done from the command line. It is important to specify the location of the VirtualBox binding through command line arguments, this is because the location of the VirtualBox bindings change per operating system and the way VirtualBox got installed.

Since WildFragSim parses a lot of big byte arrays that make up the Master File Table, WildFragSim runs best if the heap space is set tot 8GB or more. At start-up, WildFragSim needs an output location for the CSV and profile to run. Both are passed in as command line arguments. There is an option to print the measurements to the screen.

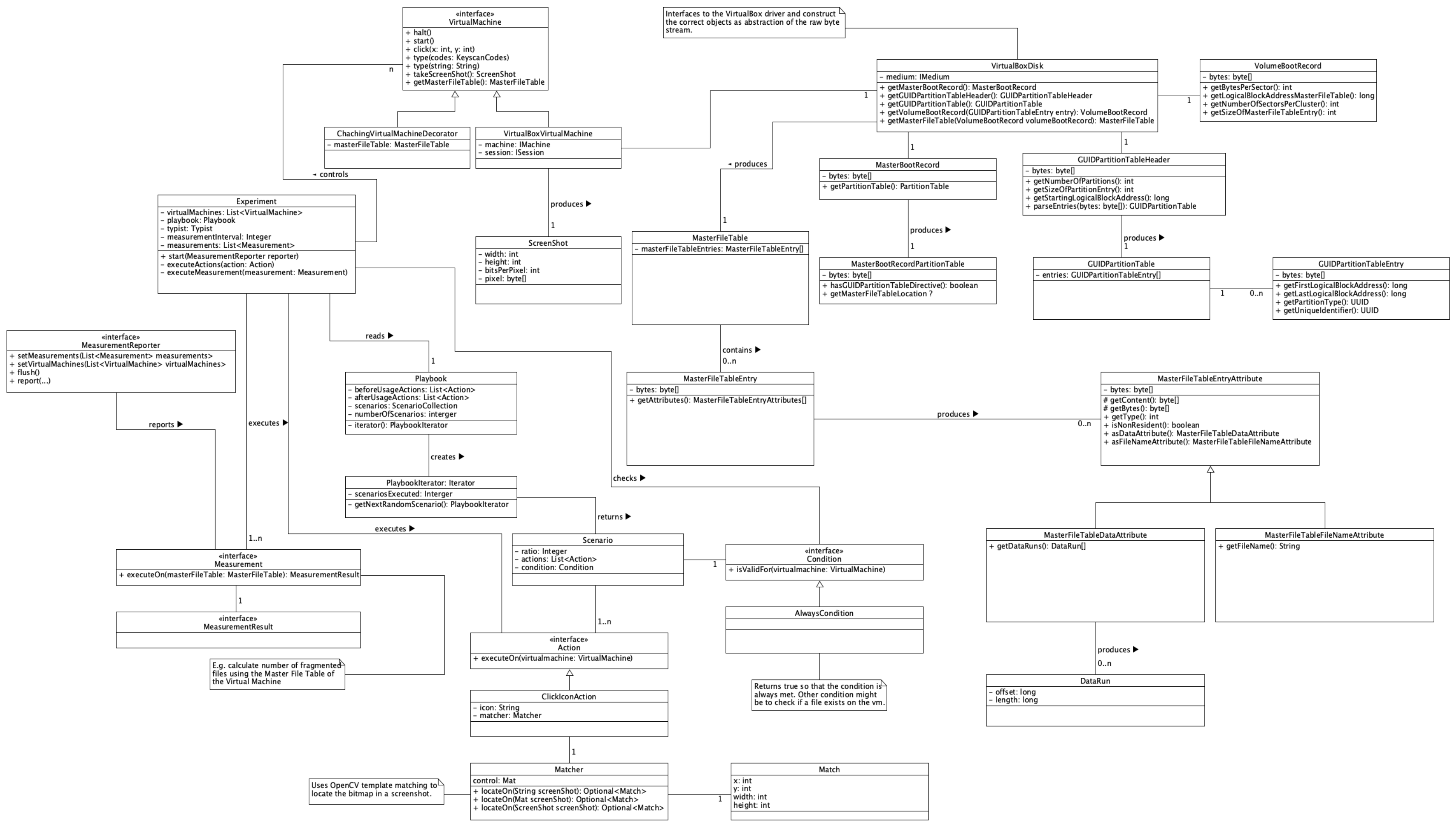
Output of an Experiment

Depending on the settings, WildFragSim will report measurements on a set interval of number of scenarios. It will write them to the specified CSV file. This allows other tools to visualise or analyse this information further. Each measurement will be in its own row in the final CSV. There is a column for each measurement virtual machine combination. Next to the raw data, the date time of the measurement and the amount of scenarios done are added as extra reference point.

Also disk image that is produced after running the experiments can serve for further investigation. The file can either be shared in its virtual box format or converted to a raw disk image. The downside of the raw disk image is the space required on the host system.

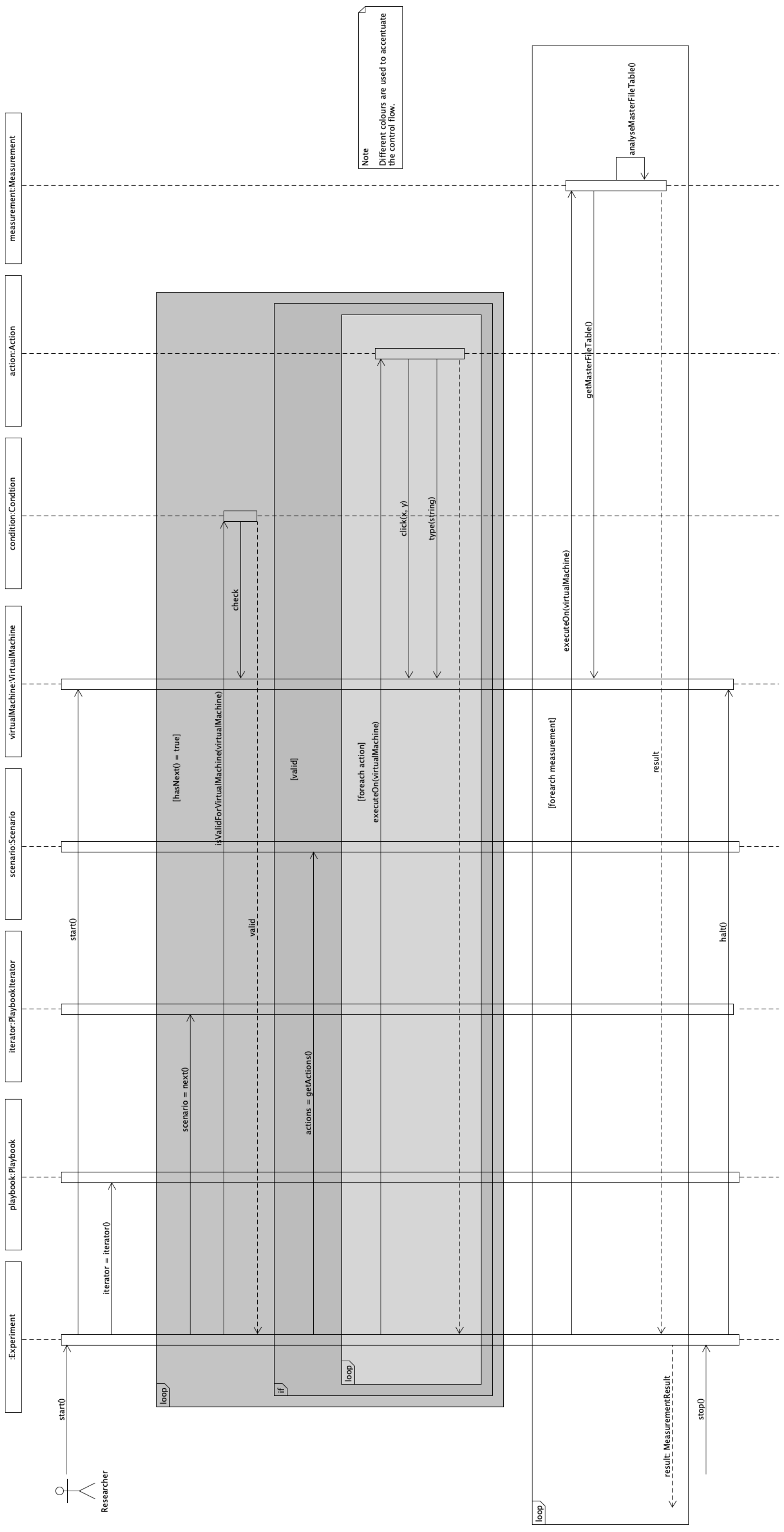
Appendix C

Class Diagram


«interface» VirtualMachine
+ halt()
+ start()
+ click(x: int, y: int)
+ type(codes: KeyscanCodes)
+ type(string: String)
+ takeScreenShot(): ScreenShot
+ getMasterFileTable(): MasterFileTable
n
ChachingVirtualMachineDecorator
– masterFileTable: MasterFileTable
VirtualBoxVirtualMachine
– machine: IMachine
– session: ISession
◂ controls
Experiment
– virtualMachines: List<VirtualMachine>
– playbook: Playbook
– typist: Typist
– measurementInterval: Integer
– measurements: List<Measurement>
+ start(MeasurementReporter reporter)
– executeActions(action: Action)
– executeMeasurement(measurement: Measurement)
produces ▶
1
ScreenShot
– width: int
– height: int
– bitsPerPixel: int
– pixel: byte[]
Interfaces to the VirtualBox driver and construct the correct objects as abstraction of the raw byte stream.
VirtualBoxDisk
– medium: IMedium
+ getMasterBootRecord(): MasterBootRecord
+ getGUIDPartitionTableHeader(): GUIDPartitionTableHeader
+ getGUIDPartitionTable(): GUIDPartitionTable
+ getVolumeBootRecord(GUIDPartitionTableEntry entry): VolumeBootRecord
+ getMasterFileTable(VolumeBootRecord volumeBootRecord): MasterFileTable
1
1
VolumeBootRecord
– bytes: byte[]
+ getBytesPerSector(): int
+ getLogicalBlockAddressMasterFileTable(): long
+ getNumberOfSectorsPerCluster(): int
+ getSizeOfMasterFileTableEntry(): int
◂ produces
1
MasterBootRecord
– bytes: byte[]
+ getPartitionTable(): PartitionTable
produces ▶
1
MasterBootRecordPartitionTable
– bytes: byte[]
+ hasGUIDPartitionTableDirective(): boolean
+ getMasterFileTableLocation ?
1
GUIDPartitionTableHeader
– bytes: byte[]
+ getNumberOfPartitions(): int
+ getSizeOfPartitionEntry(): int
+ getStartingLogicalBlockAddress(): long
+ parseEntries(bytes: byte[]): GUIDPartitionTable
produces ▶
1
GUIDPartitionTable
– entries: GUIDPartitionTableEntry[]
1
0..n
GUIDPartitionTableEntry
– bytes: byte[]
+ getFirstLogicalBlockAddress(): long
+ getLastLogicalBlockAddress(): long
+ getPartitionType(): UUID
+ getUniqueIdentifier(): UUID
1
MasterFileTable
– masterFileTableEntries: MasterFileTableEntry[]
contains ▶
0..n
MasterFileTableEntry
– bytes: byte[]
+ getAttributes(): MasterFileTableEntryAttributes[]
produces ▶
0..n
MasterFileTableEntryAttribute
– bytes: byte[]
# getContent(): byte[]
# getBytes(): byte[]
+ getType(): int
+ isNonResident(): boolean
+ asDataAttribute(): MasterFileTableDataAttribute
+ asFileNameAttribute(): MasterFileTableFileNameAttribute
MasterFileTableDataAttribute
+ getDataRuns(): DataRun[]
MasterFileTableFileNameAttribute
+ getFileName(): String
produces ▶
0..n
DataRun
– offset: long
– length: long
«interface» MeasurementReporter
+ setMeasurements(List<Measurement> measurements>
+ setVirtualMachines(List<VirtualMachine> virtualMachines>
+ flush()
+ report(...)
reports ▶
executes ▶
reads ▶
1
Playbook
– beforeUsageActions: List<Action>
– afterUsageActions: List<Action>
– scenarios: ScenarioCollection
– numberOfScenarios: interger
– iterator(): PlaybookIterator
creates ▶
PlaybookIterator: Iterator
– scenariosExecuted: Interger
– getNextRandomScenario(): PlaybookIterator
checks ▶
returns ▶
executes ▶
1..n
«interface» Measurement
+ executeOn(masterFileTable: MasterFileTable): MeasurementResult
1
«interface» MeasurementResult
E.g. calculate number of fragmented files using the Master File Table of the Virtual Machine
Scenario
– ratio: Integer
– actions: List<Action>
– condition: Condition
1
«interface» Condition
+ isValidFor(virtualmachine: VirtualMachine)
AlwaysCondition
Returns true so that the condition is always met. Other condition might be to check if a file exists on the vm.
1..n
«interface» Action
+ executeOn(virtualmachine: VirtualMachine)
ClickIconAction
– icon: String
– matcher: Matcher
1
Matcher
control: Mat
+ locateOn(String screenShot): Optional<Match>
+ locateOn(Mat screenShot): Optional<Match>
+ locateOn(ScreenShot screenShot): Optional<Match>
Uses OpenCV template matching to locate the bitmap in a screenshot.
1
Match
x: int
y: int
width: int
height: int

Appendix D

Sequence Diagram



:Experiment
playbook:Playbook
iterator:PlaybookIterator
scenario:Scenario
virtualMachine:VirtualMachine
condition:Condition
action:Action
measurement:Measurement
Researcher
start()
iterator = iterator()
start()
loop
[hasNext() = true]
scenario = next()
isValidForVirtualMachine(virtualMachine)
check
valid
if
[valid]
actions = getActions()
loop
[foreach action]
executeOn(virtualMachine)
click(x, y)
type(string)
Note
Different colours are used to accentuate the control flow.
loop
[foreach measurement]
executeOn(virtualMachine)
getMasterFileTable()
analyseMasterFileTable()
result
result: MeasurementResult
stop()
halt()

Appendix E

Reflection

The past eight months have been an incredible roller-coaster for me. Working with Vincent and Hugo on an interesting topic was very satisfying. Their motivation and enthusiasm was very encouraging throughout the project. Their feedback was very helpful to improve the final result.

Writing

During the course of this project, I have learned an incredible amount about forensics, working of file systems and academic writing. Especially, academic writing was something that I stated as something that I wanted to improve. Compared to my level before this graduation project, I would say I definitely improved my skills in writing. Refining the text and incorporating the invaluable feedback has proven to be very time consuming. The feedback really helped to better explain the choices that I made. Based on my hour sheet, I can conclude that most time has been spent on writing and rewriting the VAF and AF reports. Keeping a close watch on the time spent between the tool and writing was hard. For me, developing the tool very interesting. This meant I had to push myself to writing the reports. The recurring meetings with Hugo and Vincent helped a lot in that. Also, I thought it should have been possible to reuse more from our initial VAF in the AF. However, it resulted in that I only used the articles that we had found during the initial part of the project. Overall I found the website created by Hugo[1] incredibly helpful while writing. This information is something I would have expected in the course *Wetenschappelijke schrijfvaardigheden*[2].

Team

Before the beginning of the project, I was hesitant in doing it as I team. I valued to personal growth more than the team work. This changed during the first meeting, where it became clear how such project would came to be. However, I had hoped that the team work would have brought more cohesion during the project. For me, the contrary was true. I felt that because of long distance and life phases of other members, they valued other things within the project. As a result, the different parts became more separate and the team felt less as a team but more like a group of individuals. But, I really liked the discussions we had about results and the sharing of information between team members. Also, thinking with each other about certain problems, helped to improve myself.

Engineering

In retrospect, I might have underestimated the difficulty of making a solid interaction with the virtual machine through the API. At the end lot of minor problems arose, that were not difficult but impacted the progress. Also, I could have easily spend much more time on improving the tool on itself. However, the tight schedule made it so that I had to to stop in order to keep within the planning. Because of that, I have chosen to leave that as future work. Also, it is incredible to see how many operations are needed before some tangible result becomes available. It was expected that it would take a long time. However, running

[1]`http://www.open.ou.nl/hjo/writing/`
[2]`https://www.ou.nl/opleiding?sku=IB2002`

WildFragSim for almost a day and still with only some minor fluctuations in fragmentation percentages was a bit disappointing to me.

At the end, WildFragSim has grown to fairly complex simulation harness to investigate fragmentation without any privacy concerns. The tool counts over 3700 lines of code excluding the tests. To model the Master File Table correctly in the tool, a lot of research was done into the workings of NTFS. Also, it made me solve a whole lot of problems in creative ways. The extensive model of the Master File Table is something I did not find in other tools. It is not a complete model, however, I think it can prove helpful in further research and analysis of Master File Tables in the future.

Side-tracks

While doing this project, I came across a lot of very interesting side investigations. Sometimes, I took the liberty to dive into these side tracks such as the weird data attribute sizes in the master file table. Finding out that other tools have similar behaviour or just ignored the size altogether made me even more curious. However, these side tracks took some valuable time. Scoping the project and keeping track of the scope was sometimes hard.

Planning

At the beginning we made a planning altogether to try and finish this project in nine month. This was tight, as already foreseen by the tutors, especially next to my normal day job. All in all, I am proud of the final result.