NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

DATABASE CREATION AND STATISTICAL ANALYSIS: FINDING CONNECTIONS BETWEEN TWO OR MORE SECONDARY STORAGE DEVICES

by

Jennifer M. Johnson

September 2017

Thesis Advisor: Neil C. Rowe
Second Reader: Michael R. McCarrin

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704–0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202–4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY *(Leave Blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 2016	Master's Thesis 07-31-2014 to 12-01-2016

4. TITLE AND SUBTITLE

DATABASE CREATION AND STATISTICAL ANALYSIS: FINDING CONNECTIONS BETWEEN TWO OR MORE SECONDARY STORAGE DEVICES

5. FUNDING NUMBERS

6. AUTHOR(S)

Jennifer M. Johnson

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Postgraduate School
Monterey, CA 93943

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Department of the Navy

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

The views expressed in this document are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol Number: N/A.

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

We have created a database to analyze digital data to find connections between two or more different secondary storage devices. We used MongoDB and created a document for each secondary-storage image and each unique sector. Ingesting the secondary-storage images took so much time that we had to carefully consider all the reasons for the slow down and experiment on different ways to insert the data. Using a partial database, we found the fraction of space that is empty (contains NULLS), per secondary-storage image and for the entire database. We found duplicate images. Future students may continue to grow the database. Rather then make the goal a completed database, the students will analyze the current data and add to the database.

14. SUBJECT TERMS	15. NUMBER OF PAGES
	55
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2–89)
Prescribed by ANSI Std. 239–18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

DATABASE CREATION AND STATISTICAL ANALYSIS: FINDING CONNECTIONS BETWEEN TWO OR MORE SECONDARY STORAGE DEVICES

Jennifer M. Johnson
Civilian, Department of Defense
B.S., San José State University, 2005

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2017

Approved by: Neil C. Rowe
Thesis Advisor

Michael R. McCarrin
Second Reader

Peter J. Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

We have created a database to analyze digital data to find connections between two or more different secondary storage devices. We used MongoDB and created a document for each secondary-storage image and each unique sector. Ingesting the secondary-storage images took so much time that we had to carefully consider all the reasons for the slow down and experiment on different ways to insert the data. Using a partial database, we found the fraction of space that is empty (contains NULLS), per secondary-storage image and for the entire database. We found duplicate images. Future students may continue to grow the database. Rather then make the goal a completed database, the students will analyze the current data and add to the database.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

1 Introduction **1**
1.1 The Problem and Motivation 1
1.2 DoD Applicability 2
1.3 Research Questions 2
1.4 Thesis Structure 4

2 Background and Related Work **5**
2.1 Background 5
2.2 Bulk Extractor 11

3 Methodology **15**
3.1 Experimental Setup 15
3.2 Hardware 15
3.3 Software 15
3.4 Data set 16
3.5 Experimental Overview 18
3.6 Inspecting The Slow Ingest 20

4 Results **25**
4.1 Top Common Matches 25
4.2 Finding the right Shannon Entropy value 28
4.3 Speeding up the Big Database 30

5 Conclusion **33**

List of References **35**

Initial Distribution List **39**

THIS PAGE INTENTIONALLY LEFT BLANK

List of Figures

Figure 2.1 Partition Table Layout, `mmls` Command Output. 9

Figure 2.2 Example of SQL table. 10

Figure 3.1 Four pieces of useful information. 16

Figure 3.2 The `_id` command used to identify each image in MongoDB. . . 17

Figure 3.3 Sector layer schema for MongoDB. 19

Figure 3.4 MongoDB Command 20

Figure 3.5 Histogram of times for inserting secondary-storage images smaller than 500 Mb into the database. 22

Figure 3.6 Inserting secondary-storage images that are smaller then approximately 500 Mb. 23

Figure 4.1 A MongoDB Command to find most common MD5 hash. 25

Figure 4.2 Most common hash with about 980 images inserted. 25

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

B bytes

CPU central processing units

CS Computer Science

DEEP Digital Evaluation and Exploitation

DoD Department of Defense

EWF Expert Witness Compression Format

FBI Federal Bureau of Investigation

GB gigabytes

GPU graphical processing units

KiB kibibyte

MB megabytes

MD5 message digest 5

ME Mechanical Engineer

NSF National Science Foundation

NIST National Institute of Standards and Technology

NSRL National Software Reference Library

NTFS New Technology File System

NUS non United States

RAM random access memory

RDC Real Data Corpus

SHA-1 secure hash algorithm 1

SFS Scholarship For Service

SQL structured query language

TB terabytes

RCFL Regional Computer Forensics Laboratory

TSK The Sleuth Kit

Acknowledgments

I have so many people to thank because I could not have written this thesis on my own. In no particular order because I am grateful towards everyone. I thank my sister Edwarda for all her love and support, what would I do without all those phone calls? I thank my mom, Sandra, for making me a capable adult; that was not magic it was hard work. I appreciate the kindness from my good friends Casi and Joh, we cried, we laughed we exchanged way too many memes. Everyone in my cohort helped me get through this program more then once. I thank my thesis adviser Michael McCarrin for his patience during our marathon thesis meetings where he had to repeat things many, many times. I thank my thesis Adviser Neil Rowe for his hardwork and support. I thank Thao for reminding me that this program exists and giving me her GRE contact and saying, "here sign up, do this now." I thank my cat Sadie, because cat hugs are the best. She passed away before I could complete this thesis but I carry her in my thoughts. I thank the National Science Foundation (NSF) Scholarship For Service (SFS) program for its existence and this chance to earn my master's in Computer Science (CS) with an undergraduate degree in Mechanical Engineer (ME).

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
Introduction

1.1 The Problem and Motivation

We address two problems. The first is managing large-scale heterogeneous digital-forensic data. The second is finding a digitally forensic connection between two or more secondary-storage devices. The National Institute of Standards and Technology (NIST) defines digital forensics as "the application of science to the identification, collection, examination, and analysis of data while preserving the integrity of the information and maintaining a strict chain of custody for the data" [1].

The growing amount of data is our motivation. In recent years, the per-gigabyte price of data has been steadily decreasing [2]. It is common for the average consumer to purchase terabytes of digital storage space. As a consequence, law enforcement agencies and cyber divisions in the Department of Defense (DoD), have acquired terabytes of data while collecting criminal evidence. The Regional Computer Forensics Laboratory (RCFL), established by the FBI, has annual reports and they noted that the Chicago lab, just one of the 15 labs, had collected and processed 580 TB of digital data in one year [3].

Currently, examiners process data on secondary-storage images drive-by-drive using forensic tools designed to run on a single workstation. Each drive is considered separately, and little work is done to correlate information across different images. From an analyst's perspective, this approach means important information may be missed. For example, there is no organized effort to detect collaboration or communication between owners of devices acquired at different times. Likewise, little has been done to study large-scale patterns in acquired data. Studying trends in data may offer insight into longstanding forensic analysis problems. Carving deleted files, for example is a longstanding forensic problem, because it can be time intensive. File carving is the method of detecting a file signature and then extracting the data associated with it [4].

A tactic that can reduce the processing time required for file carving is matching blocks that reside in allocated space with those blocks in unallocated space. Allocation means the

file-system has assigned space for the file. When a file is deleted the file-system no longer indexes it but the data is not erased [5]. It is possible to avoid duplicating work by quickly identifying material in unallocated space that is already available in allocated space, and removing this material from consideration for further file carving and analysis. If material could be removed, the file-carving problem could become smaller and potentially faster.

An experiment was performed on some of the holdings in the Real Data Corpus (RDC), a collection of the contents of secondary-storage images held by the Digital Evaluation and Exploitation (DEEP) Lab. For each image we identified partitions within the file-system, built a sector hash database from overt files on those partitions, scanned the unallocated space for matches, and tallied up the results. On one drive containing 7.12 gigabytes (GB) of allocated space and 3.72 GB of unallocated space, we found 0.61 GB of duplicated material meaning about 16.29% of the unallocated space was duplicated.

What other statistical information can we find to reduce the processing time required for file carving or other types of forensic analysis? We propose some experiments in the methodology section to do some relevant analysis over images on the RDC.

1.2 DoD Applicability

Cyberspace is an established warfare domain for the Navy. The US Patriot Act, an anti-terrorism law, in Title VIII, section 816, says "Development and support of cybersecurity forensic capabilities" [6]. We are adding to the nation's forensic capabilities by researching techniques to increase the digital forensics processing speed.

1.3 Research Questions

We scope our thesis by concentrating on analysis of trends that may be leveraged by forensic tools. In addition, we intend to estimate the potential utility of suggested approaches in terms of data reduction.

We are looking for relevant patterns in 3,000+ secondary-storage images in the RDC. The features analyzed are divided into two categories. Category one includes basic features that can be trivially extracted from the images in the corpus:

- Device name
- Device hash
- Number of sectors
- Sector size
- Device type
- Total disk size
- Number of partitions
- Partition offsets
- Recognizability of the partition?
- Volume system type
- Block size of volume
- Partition type
- Partition allocation
- Description of partition
- File system type
- Block size of file system
- Number of blocks in files system
- Sector offset of file system

Category two is comprised of features that require more extensive analysis to measure:

- Fraction of space that is empty (or contains NULLS)
- Fraction of space that is unallocated or allocated
- Fraction of space that is unallocated and non-empty
- Fraction of non-empty unallocated space that matches allocated space
- Average (2-byte Shannon) entropy score of non-empty sectors

In order to gather statistical information on all the secondary-storage images on the non United States (NUS) portion of the RDC, we first need to create a database for our analysis. We have two important steps. Step 1a is building the database and step 1b is the analysis. We have 124,104,544,671,744 bytes (B) of data in the NUS portion of the RDC. An important research question is how long will it take to build a database of sector hashes?

1.4 Thesis Structure

In Chapter two we cover the background and related work. In Chapter three we discuss the methodology. In Chapter four we discuss our results. In Chapter five we discuss our conclusions and future work.

CHAPTER 2:
Background and Related Work

2.1 Background

In this chapter we provide a brief technical explanation of the hardware and software we use to create the database. This chapter provides a technical explanation on the media we are investigating, along with popular forensic formats and tools. In addition we explain hash matching techniques and how they are are currently used to match target files or carve files but that we need to apply them to cross drive analysis.

2.1.1 The computer

Building our database is resource intensive. It helps to describe the tools we require so that our explanation of why our hardware and software resources are strained maybe understood. Secondary-storage images are peripheral components of the computer, which consists of one or more processors and main memory [7]. The computational ability of a computer depends upon its processor and main memory. A processor's main memory executes instructions. Instructions are run using a process which is a self-contained execution environment. Threads run instructions within a process [8]. Python has global interpreter lock (GIL), which means Python cannot thread, but it can multi-process [9].

Multiprocessing

Programming in parallel is an important strategy to use when a large number of instructions need to be executed as with a large database. A processor processes instructions [10]. The speed of a processor is measured in instructions (Hz) per second. One multi-core processor may have multiple virtual processors [7]. A thread is a software-based batch of instructions executed by a processor. A thread can also be hardware-based [7].

Concurrent programming is programming that uses several threads, and allows the microprocessor scheduler to manage when they execute [11]. If the microprocessor executes the instructions from the threads quickly enough we get pseudo-parallelism [11].

Data Organization

Computers communicate with binary numbers: 0 and 1. Each digit is 1 bit in size and bits are organized into groups of 8 and are 1 byte in size. The hexadecimal number system has 16 symbols 0 to 9 and followed by A to F. Hexadecimal is often used to inspect raw data and not binary. Data is stored by allocation on a storage device. A byte is the common smallest amount of space allocated. Since a byte can only hold 256 values they are grouped to store a larger number. Computers store characters and strings by encoding them to numbers, ASCII or Unicode are common methods. The letter A is for example equal to 0x41. Each byte stores the value of a character. The NULL symbol 0x00 often signals the end of a string. A Unicode character must be stored and UTF-8 is the method often used because has the least amount of wasted space [5].

Data Structure

Data structures are used to layout data and is analogous to a map. Data structures contain fields and each field has a size and a name. The data structure is not saved with the data.

Writing data to a device requires identifying the correct data structure to define where each value should be written. Take "1 Main St." as an example, as used in Carrier's File System Forensic Analysis book. The digit 1 is written in bytes 0 to 1 of the storage space, then the string "Main St." in bytes 2 to 9 in ASCII values and then the remaining bytes are 0 [5], see Table 2.1. This data maybe located any where on the device and the byte offset is relative to the start of allocated space. Using a tool that converts binary often referred to as raw data we can examine the output.

Offset	Hex	String
0000000:	0100 4d61 696e 742e 0000 0000 0000	..Main S

Table 2.1. Example strings offset and data structure.

Sector Addresses

Read and write from the device requires creating addresses for each sector. A sector will be assigned a new address each time a partition, file-system or a file requires it. The address

relative to the start of the physical media is called the physical address. The sectors of a volume only need to give the impression that they are in consecutive order, the damaged sectors maybe skipped without the user needing to know [5].

Data Unit Viewing

Carrier defines the term data unit viewing as knowing the address or the byte offset of the data. He notes that this method maybe used to find potentially hidden data. FAT32 file-systems do not use sector 3 so if the investigator uses the `dcat` tool found in The Sleuth Kit (TSK) she can view a specific data unit in either raw or hexadecimal. If that data is non-zero then this maybe evidence of hidden data [5]. If we find a sector match and note its byte offset per hardware division which is typically 512 B in order to view the entire file we also need to know the file-system data unit, which maybe be 1,024, 2,048 or larger.

Slack Space

If the size of a file is not a multiple of the data unit size slack space occurs. This is because a file must allocate all of the data unit, even if the file only needs part of the data unit [5]. In addition to this rule most computer systems do not delete slack space so it contains data from previous files or from memory. The end of a file and the end of the sector of the file is place where we can find slack space. Also sectors that have no file content maybe an area of slack space [5]. The Operating System determines what is done with the slack space. Some fill the space with data from random access memory (RAM), or zeros.

Device Images

The NUS portion of the Real Data Corpus is raw data extracted from secondary-storage images [12]. The RDC primarily consists of flash memory and computer drives [12]. Despite the fact that the secondary-storage images had been discarded by their owners, many of the drives in the RDC were not erased by their owners [13].

There are two main types of secondary-storage images. One is raw format, an exact sector-by-sector of the original secondary-storage image (usually called “raw” format). A sector is the smallest unit that can be accessed on media [1]. The other type contains the raw data as well as a checksum and metadata; the most common form is EWF format. A checksum is a many-to-one mapping on data that can be used to detect errors when data is

copied [5]. The metadata is information about the secondary-storage image. Our forensic data set was created using secondary-storage images that have been duplicated using EnCase software which provides checksums and metadata for each image in EWF format. EnCase chunks each image into 640 megabytes (MB) and names those chunks in sequence (i.e., E01, E02, E03, E04 and so on).

Forensic Artifact Extraction

We use TSK, a library, a framework, and a collection of command-line tools for forensic investigation disk images [14]. The TSK is free to download at https://www.sleuthkit.org/. TSK is organized by layers: disk-image, volume-system, file-system, and hash-database layer [15]. The `tsk_loaddb` command populates a SQLite database with metadata from a disk image [15].

The disk-image layer includes the entire secondary-storage image. The creation of a volume-system is required before most secondary-storage images can be used to store files. Logical volumes are created from partitions in the image [1]. A partition is a logical division of the disk-image into separate units [1]. A file-system is one or more partitions that has been formatted with a file-system [1]. A file-system determines file names, and how they are stored, organized, and accessed on logical volumes [1]. A lot of different file-systems exist, however all have some common attributes. They use directories and in most cases sub-directories to organize and store files [1]. File-systems make use of a data structure to point to location of files on the image. One, or more, file allocation unit is used to store a file. A cluster is a common name for the file allocation unit [1].

A file-system may hold data from deleted files or earlier versions of existing files. This data can still provide useful forensic information. A deleted file means the data structure that had pointed to that file has been removed, not the data itself. The data will remain as free space and in many cases is not over written until the space is required [1]. If a file uses less space then required by the file allocation unit, it is still reserved by the file-system and called slack space. Slack space may still have some useful forensic information [1]. Free space has not been allocated to a partition, perhaps unallocated clusters or blocks. This includes space where files or volumes have been deleted, free space may also contains forensically useful information. The reason why we hash at the sector level is to grab all of the small bits of forensic data that would other wise be lost in deleted, free or slack space.

The `mmls` command of the TSK tool displays the partition layout of a volume system [14]. Example output of `mmls`:

```
Partition Table
Offset Sector: 0
Units are in 512-byte sectors

     Slot    Start        End          Length       Description
00:  Meta    0000000000   0000000000   0000000001   Primary Table (#0)
01:  -----   0000000000   0000000062   0000000063   Unallocated
02:  00:00   0000000063   0078108029   0078107967   NTFS (0x07)
03:  -----   0078108030   0078165359   0000057330   Unallocated
```

Figure 2.1. Partition Table Layout, `mmls` Command Output.

In this example we see that the sector size is 512 B. The image uses New Technology File System (NTFS) and the sections that are unallocated space are labeled. Some forensics tools require being able to understand the partition, file-system or file type. However, other software like *bulk_extractor* "operates on disk images, files or a directory of files and extracts useful information without parsing the file-system or file system structures" [16].

Hashes

Hashes provide a fixed-sized identifier for a variable amount of data. Our work used the message digest 5 (MD5), a cryptographic message-digest algorithm used to create hashes because it is extensively used within the forensic community and it is computationally fast. MD5 and other cryptographic hashes are 160 bits and are designed so that it is very unlikely for a collision to occur [17]. A hash collision happens if two different inputs produce the same hash [18]. With the MD5 algorithm, 3.40×10^{38} hashes can be generated on the average before a collision occurs. Secure Hash Algorithm 1 (SHA-1) is another popular hash method, and EnCase can create both.

Databases

This work will store forensic data including hashes in a database. A database is a collection of information organized for quick random access. The structured query language (SQL) is a programming language designed to manage a database. For example, the following SQL command says select five rows and all columns from the `tsk_file_layout` table; `tsk_file_layout` is created by TSK.

```
sqlite> SELECT * FROM tsk_file_layout LIMIT 5;
```

The SQL command provides the following output: Figure 2.2 shows the result, a table

```
obj_id      byte_start  byte_len    sequence
----------  ----------  ----------  ----------
0           67182592    8192        0
6           2672295526  8192        0
13          2248798208  16384       0
13          2248814592  4096        1
13          2248818688  4096        2
```

Figure 2.2. Example of SQL table.

with attribute columns: obj_id, byte_start, byte_len, and sequence. Each row represents a secondary-storage image.

Metadata is data that "provides information about other data" [19]. A database schema consists of metadata [20]. The columns of the table label the attributes of the data, and the rows contain the data [20]. A schema created from a table is called relational. An alternative database type is a non-relational database. An example is MongodDB which uses a document-schema database [21]. A document is similar to a Python "dictionary" or hash table. In an SQL database the schema for the table must be designed before data is added, changes are possible but can become complicated. In a non-relational schema data can be added to documents at any time and documents are easy to change, however a poor design is still possible [22].

The `tsk_file_layout` table stores the layout of a file within the image [23]. The `tsk_files` table lists every file found in the images and has the basic metadata for the file [23]. The layout of file can be connected to the metadata of the same file using a technique known as normalization [22]. Normalization connects two different tables with a reference, in this case with the `obj_id` column. Normalization, or connecting two or more documents with a reference field is also possible using non-relational MongoDB [22]. SQL queries use the JOIN command to relate multiple tables, non-relational databases do not have that command so normalized documents have to retrieve all documents associated with `obj_id` and then manually link the two [22]. Denormalization means that rather then

using a reference, data is repeated in each table or document. Denormalization allows for faster queries, the reason that non-relational databases are said to be faster, but with slower updates [22].

It is common for SQL databases to enforce data integrity rules using foreign key constraints. A foreign key constraint is a column or combination of columns that i establishes and enforces a link between the data in two tables. This is not available in non-relational databases [22]. MongoDB and other non-relational databases use Java script like query commands and nested documents can become complex when trying to query [22]. When creating a large database distributing it among multiple servers maybe necessary, non-relational databases use of simpler data models makes this easier to do then SQL type databases [22]. This is the main reason we choose to build our database using a non-relational database.

A Bloom filter is another way to store hashes when no additional information need be stored with them. False positives matches are a problem [24]. A tool, *hashdb*, can chunk files in 4 kibibyte (KiB) blocks, then hashes them and builds a Bloom filter from them. It also uses HASH-SETS, an algorithm that reports the fraction of blocks associated with each target file that is present on the disk image [25].

2.2 Bulk Extractor

Bulk_extractor reads a secondary-storage image from the image level [26]. It can be used to create sector hashes and then hashdb can put them in a database. Hashdb has been used previously on some secondary-storage images and common sector hashes have been identified [25].

Digital Forensic

Digital Forensics analysis is defined as gathering information that may be found on a computer, any data-carrying device, and data sent over a network. Garfinkel in his 2012 survey on lessons in digital forensics defines and describes the current and trending state of the field. The field of digital forensics software has the challenge of requiring growing with the growth of data diversity and data scale. People who analyze digital forensics are therefore challenged to also develop software to meet the challenge of diversity and

scale [27]. Software starts off as scripts and our aim is to find solutions worthy of being developed into software. Our work focuses on analyzing secondary-storage images in a large scale.

File Carving

The National Software Reference Library (NSRL) currently maintains a database of meta-data consisting of a hash of the file's content, the file's origin (the software typically required to view it), original name, and size [28]. The hash is produced using, among other hash algorithms MD5, and secure hash algorithm 1 (SHA-1) [29]. It is common to find hundreds of thousands of files during a digital forensics analysis and the goal of the database is to reduce the time spent examining the computer [29].

File carving, is a recovery technique that searches for a file's signature in a given image. A file's signature contains the file's header and footer. Carving extracts the file's contents, or the blocks between the header and footer [5]. The file-system meta-data is not required and this means that files maybe carved from unallocated space [5].

Full file identification and carving, are limited because the hash that makes each file's content unique can not deal with similar. Therefore a small change to a file or a corrupt block means the hash will change and the file will no longer be identifiable [30]. In order to solve this problem in 2009 Garfinkel explores using cryptographic hash functions on sectors or blocks of data in order to search for target files [31]. The term hash-based carving means searching for the a target file in a given secondary-storage image by first hashing blocks of the file, rather then the entire file [31].

Garfinkel et al. developed a tool and called it *frag_find* because it is a hash-based carver that identifies files using sector-by-sector hash comparisons. The tool can identify files because "there exist distinct data blocks that, if found, indicate that the entire file from which the block was extracted was once resident on the media in question" [31].

A probative, or distinct block, means that if the block is found there is a high probability that the entire targeted file will also be found. A common block, the most common being a set of all NULLs, means that detecting it on two or more images does not signify a correlation. Non-probative is another term for common block [25] [32].

Hash based carving inherently increases the size of the data a forensic analyst must deal with. If we for example make a gross assumption that each file needs to be sectioned into 2 blocks and if we had been dealing with 10 million files, we are now dealing with 20 million hash blocks. In addition the algorithms required to match the blocks take up a considerable amount of RAM and central processing units (CPU) resources. The variables we can attempt to speed up are the hardware, the type of database, the algorithm to search the database or all those methods in combination.

Collange et al. in their 2009 study notes that the "ability to detect fragments of deleted image files and to reconstruct these image files from all available fragments on [a] disk is a key activity in the field of digital forensics". The task is time consuming with the brute force method of comparing the contents of each sector on a given secondary-storage image with the target file sectors. The study showed that this problem maybe solved using graphical processing units (GPU) in parallel. They chose to use the djb2 hash algorithm (named after Daniel Julius Bernstein) for its computational speed even though they found a .33% collision rate. The research found that their parallel implementations of GPU hardware enabled them to search for deleted file fragments at a rate of 500 MB/s [33].

In 2012 Foster examines if sector hashing is more effective for file carving then file hashing on a large scale. She finds that a custom B-tree key-value store with a Bloom filter is the most effective type of database to query sector hashes, looking for distinct blocks. She shows that even over a large set of data (Govdocs, OCMalware and NSRL) that distinct blocks still exist and can be used to ID files and software. In order to scale the distinct blocks method the database must be able to store the file block hashes of every file disk at I/O speed. In 2012 that speed was calculated at 150 K sectors/second because that is how fast a 1 TB drive of 512 B sectors can read. However, with media sampling the rate drops to few thousand transactions per second because a 72000 RPM hard drive can perform 300 seeks per second. If the addresses are non linear then it takes longer to seek, seek means looking up the addresses. They note the limitation that files must be sector aligned on the disk for successful identification [34]. *Bulk_extractor* was created as a tool that implements the Bloom filter database [35] [27].

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 3: Methodology

3.1 Experimental Setup

The goal of our research is to find interesting patterns across the hashed sections of the secondary-storage images of the Non-US portion of the Real Data Corpus. We have under our collective forensic belt hash-based file carving and software such has *bulk_extractor* and *hashdb*. However, those tools are limited to looking for targeted files, and we know there is more to secondary-storage images such as the setup of the file-system, and information hidden in slack and deleted space. Also most computers do not work in isolation, and we want to know, can we find cross drive patterns that will flag computers that contain similar files or user activity. In addition these database tools do not easily combine more then two secondary-storage images.

First we make it our goal to build a database designed to inspect unique individual sectors of the secondary-storage images in our collection. Then we investigate the fraction of sectors that are empty, compare matches in allocated and unallocated space within the same image and across multiple images. We can also match and compare individual sectors with metadata from volume, partition and file-systems as well individual files.

3.2 Hardware

We used a private data server, it has a configuration of 64-cores and a 512 GB main memory node that is dedicated for Digital Evaluation and Exploitation Lab, or DEEP, use.

3.3 Software

We used Python version 3.5.1 to automate our tools. We used MongoDB version 3.0.14 for our database. We used Pymongo version 2.5.2 as the interface between Python and the MongoDB software. We are using The Sleuth Kit or, TSK, version 4.1.3. TSK consists of a static C/C++ library in addition to a command line tools. TSK can create SQLite schema of each image and we used schema version 2. Rather then use SQLite schema we import them

into MongoDB because the flexible documents of MongoDB allow for larger collections to be split across multiple servers. The library libewf is used to access the Expert Witness Compression Format (EWF) and pyewf allows us to do this using Python [36]. The EWF is a compressed format that allows us to store secondary-storage images on a server. We used the pyewf version 2 library to interface between Python and EnCase® or the original name: EWF. The pyewf library allows us to convert EWF to the raw format or the binary level of the secondary-storage device. Using the raw format we can divide the secondary-storage image into 512 B sectors.

3.4 Data set

Our data set consists of the secondary-storage images in the Non-US portion of the Real Data Corpus. At the time of our experiment we had 3,196 images in EWF format (with the EnCase extension) on the NUS portion of the RDC. We found this number by writing a script that looks through the NUS directory recursively and counts all the files with EWF extension, E01. To set up our data base we needed to construct a non-relational schema for the secondary-storage images. Our schema needed to contain metadata about each image. We gathered our metadata from the TSK SQLite schema of each image. We looked at each device with the `ewfinfo` command from TSK using the pyewf library to find which devices were created with MD5 hashes. This gives four pieces of useful information, see Figure (3.1): the name of the device, the MD5 hash, the size of the image, and whether the partitions of the device's volume-system are recognizable.

```
['4f14ece14e4e6276da1f20cc9c9e8818', 2490368,
 '/corp/nus/drives/AE/AE10-0023/AE10-0023.E01', 'yes']
```

Figure 3.1. Four pieces of useful information.

Before we begin building the database we checked for duplicate MD5 hashes on the images, so as to not duplicate work. We found that we have 2,914 unique hashes and 122 non-empty images that require further investigation because they appear to be duplicates. We measured 124,104,544,671,744 bytes of data total.

See Figure 3.2 for an example of how we defined a document by MD5 hash.

```
{'_id' : '02ba1d4a12333a833218538b8dab9cfd'}
```

Figure 3.2. The `_id` command used to identify each image in MongoDB.

The following list are the attributes we retrieve from the TSK `tsk_loaddb` command.

- The TSK `tsk_loaddb` produces a SQL table named `tsk_image_info` and holds the metadata of the type of disk image format, the sector size, the sequence of image parts and the time zone. We also include the image name, the number of sectors in an image, and the image size.
- The volume layer key-value pair is nested in the event that we have more than one volume. The TSK `tsk_loaddb` produces a SQL table named `tsk_vs_info` and holds the metadata: type of volume-system, the byte offset where the volume-system starts in bytes, and the block size in bytes.
- The partition layer key-value pair is nested in the event that we have more than one partition. The TSK `tsk_loaddb` produces a SQL table named `tsk_vs_parts` and holds the metadata. The address of the partition, the offset of the partition start in bytes (zero being the start of the image), the number of sectors in the partition, and a description of the partition type including allocation.
- The file-system layer key-value pair is nested in the event that we have more than one file-system.
- The TSK `tsk_loaddb` produces a SQL table named `tsk_fs_info` and holds the meta-data of the offset of the file-system start in bytes (zero being the start of the image), the type of file-system, the block size in bytes, the block count or the number of blocks in the file-system and the address of the root directory and the first valid address and the last.

A sector is the smallest division of a secondary-storage image and is hardware defined [1]. A file-system uses file allocation units, the smallest unit is a blocks, sometimes referred to as clusters, and is typically 4096 B [1]. Starting to hash the sectors at the beginning of the image, or 0, means ignoring file-system alignment. If the file-system alignment is not taken into account the sector hashes will not be aligned with the file block hashes and matches will not be found [25].

MongoDB uses BSON documents to store data records [21]. BSON is short for Binary JSON (JavaScript Object Notation) [37]. A MongoDB document is identified with `_id` a required special key that identifies the document and insures that it is unique in the collection. We have two different categories of documents one of the entire image and one of each sector found. See the Appendix for an example of both.

3.5 Experimental Overview

Our experiment summary:

1. Create a database of sector hashes.
   (a) Investigate reasons for slow ingestion rate.
   (b) Test methods for increasing the ingest.
2. Single and Cross image analysis.

In order to create our database of hashed sectors for the entire media on the Non-US portion of the Real Data Corpus we first considered using hashdb. It is easy to configure creating hash blocks of 512 B however creating a hash database of more then 2 images is not what the tool was created for and neither is documenting the source offset. In addition if we wanted to do something like do a cross drive hash match we find it is not easy to do. This is because hahsdb is a wrapper that at first used Bloom Filters then used the dense hash store in conjunction with other forensic tools to do very useful tasks such as extracting files on large images.

MongoDB has the advantage of being more flexible but the disadvantage of not being as fast. We started the database with successfully importing the image, partition and file-system information. Having that information made it easy to find that all of the images use 512 B sector as the smallest division. While we know that file-systems are sector aligned we start at the image offset, and this may cause some misalignment that must be considered in our analysis.

Our MongoDB document comprised of each unique hashed sector encountered contains a list of source hashes in the key `src_id`. We can then track if we have seen the same MD5 hash in multiple secondary-storage images. We also track the number of times we have seen the MD5 hash on a secondary-storage image, and the total number of times we have seen

it. We also add the ten most recent offsets at which we have seen the MD5 hash so that no one document grows too large. We add to the document as seen in Figure 3.3.

```
{'src_id' : [
        '4f14ece14e4e6276da1f20cc9c9e8818',
        'ce8fc1ed372d69cfb94f0cb20f479e62',
        '574b0bb13cf3c2a1e234945def480eb7',
        '2df68f24df5411556bf1d829bd142b02',
        'e7f90c5e0d3d54bf8374414193d6b835',
        'a859e3562f0bd4d14749d4e3878894de'],
'per_source_count' : {
        '4f14ece14e4e6276da1f20cc9c9e8818' : 1,
        'ce8fc1ed372d69cfb94f0cb20f479e62' : 1,
        '574b0bb13cf3c2a1e234945def480eb7' : 1,
        '2df68f24df5411556bf1d829bd142b02' : 1,
        'e7f90c5e0d3d54bf8374414193d6b835' : 1,
        'a859e3562f0bd4d14749d4e3878894de' : 1},
'total_count' : 6,
'offset' : {'4f14ece14e4e6276da1f20cc9c9e8818' : [
                314880],
        'ce8fc1ed372d69cfb94f0cb20f479e62' : [
                9941504],
        '574b0bb13cf3c2a1e234945def480eb7' : [
                379369472],
        '2df68f24df5411556bf1d829bd142b02' : [
                488855040],
        'e7f90c5e0d3d54bf8374414193d6b835' : [
                6919168],
        'a859e3562f0bd4d14749d4e3878894de' : [
                250661888]}}
```

Figure 3.3. Sector layer schema for MongoDB.

As we open and read each image at the byte level we section the image into 512 B and created an MD5 hash of each byte. Each hash is used to create a document in MongoDB. The bulk of our database consists of MD5 hashes we created from secondary-storage image sectors. In order to create the MongoDB documents as seen in Figures 3.2 and 3.3, we used the MongoDB `UpdateOne` command to insert our dictionary into our database. We perform the task in parallel per each image using our 64 available cores. The MongoDB

`UpdateOne` command is used in conjunction with MongoDB's bulk write commands. Each command is put into a list and looks as seen in Figure 3.4.

```
UpdateOne({'_id': md5_hash},
{'$addToSet' : {'src_id': src_id},
 '$push': {
     'offset.%s' % src_id: {
         '$each': [offset],
         '$slice' : 10}},
 '$inc':{
 'per_source_count.%s' % src_id: 1,
 'total_count': 1 }},
 upsert=True)
```

Figure 3.4. MongoDB Command

Creating the database this way immediately is slow. As we build the database it is good to keep in mind some logical limitations. It is poor etiquette to use all 64 cores for an extended period of time. This project is not the only one being run on our server. Our speed is also bound by the read and write speeds of Domex's hard drives. The RDC is on NetApp; cloud storage. MongoDB has granular locks and when a document is being written, only one instance of MongoDB can write to it [38]. Write applications are atomic. MongoDB has concurrency control. Each document has a unique index. In our case it is the MD5 hash of each 512 B sector [39]. In the case of multi-document transactions, or concurrency, MongoDB uses a two phase commit. The actions are initialized and then applied [39]. This is how we can use the multiple cores available.

3.6 Inspecting The Slow Ingest

We ingested 500 images. Ingesting the secondary-storage images took so much time that we had to carefully consider all the reasons and experiment on different ways to insert the data. After finding that building our database was not going to be done in one run of our script we sorted the images by size and limited the number of images that we would be inserting

at once; we sorted from our smallest of 2,490,368 B to our largest at 1,000,204,886,016 B. We logged timing data for each image. We started by inserting the images that were approximately 500 Mb or smaller in size.

We noticed that starting at approximately 60 Mb the same size secondary-storage image takes an increasingly amount of time to process. Considering the fact that our data set is continuous, and because the insertion time can take any value, we can use a histogram to search for a pattern. We created a histogram of frequency versus insertion time. We created bins (data split into intervals) of 50 seconds. We can see from Figure 3.5 that most of the 500 Mb or smaller secondary-storage images (about 360) can be inserted in 50 seconds or less. That lends some evidence to the observation that opening, reading and hashing each sector is not the root of our slow down. If it were then all of the images would take a long time to ingest. Opening, reading and hashing can still become a problem for larger images, but for now we can say the problem is with the database.

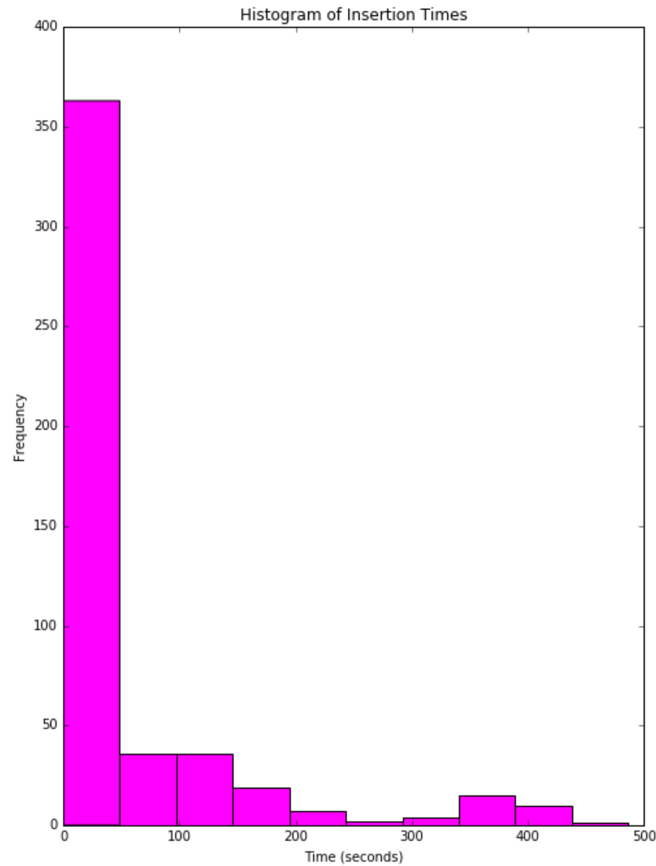

Figure 3.5. Histogram of times for inserting secondary-storage images smaller than 500 Mb into the database.

To look for other patterns, we created a scatter graph so that we could observe how the same size images took a different range of time to ingest, see Figure 3.6. A 60 Mb secondary-storage image took as little as 2 minutes and as long as 40 minutes, and a 130 Mb secondary-storage image took as short as approximately 2 minutes and as long as 1 hour and 40 minutes.

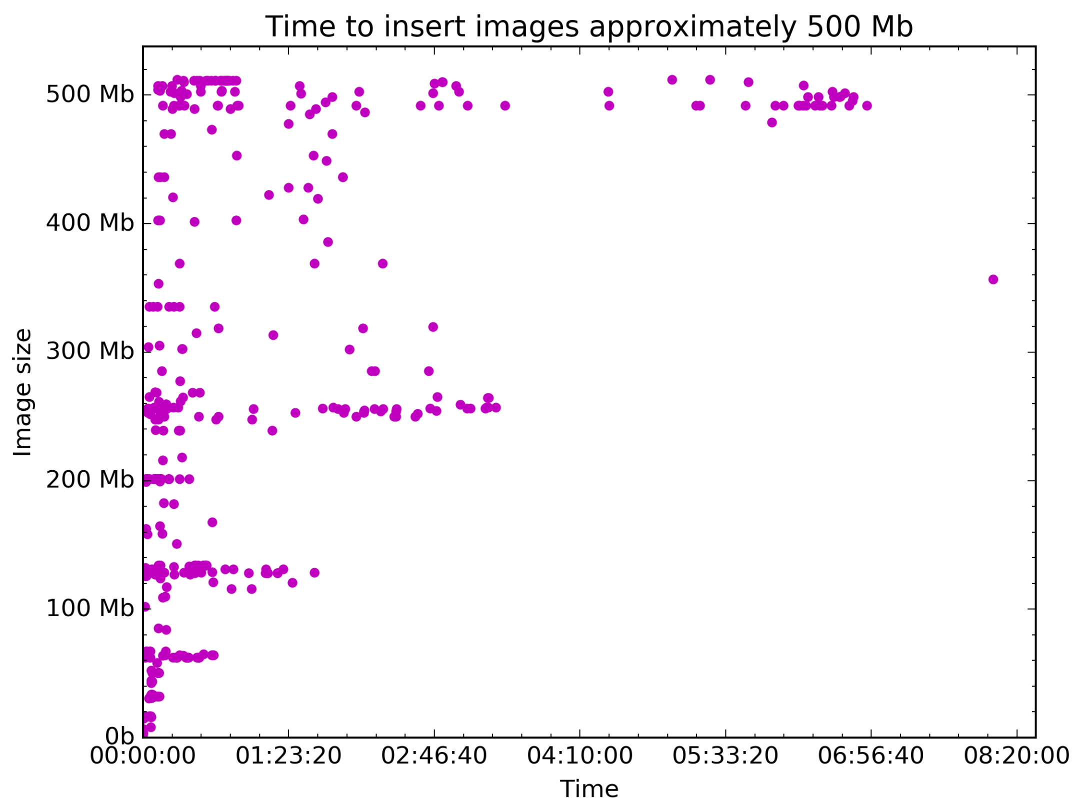

Figure 3.6. Inserting secondary-storage images that are smaller then approximately 500 Mb.

When looking at the range of values as in Table 3.1, we asked whether there was something unique about the data that took a long time. We reexamined the secondary-storage images as seen in Figure 4.2 to see if there was something unique about the secondary-storage images that took the longest to process. These secondary-storage images had the shortest and longest insertion times per the same size of image.

Table 3.1. A closer look at differing insertion times for the same image size.

Names	~ Size	Min Time (H:M:S)	Max Time (H:M:S)
CN32-04 and IN10-0229	64 Mb	00:01:42	00:39:51
CN27-57 and CN21-01	128 Mb	00:01:49	01:37:56
CN32-51 and IN10-02014	255 Mb	00:02:33	03:21:46
CN32-85 and CN6-12	350 Mb	00:08:42	08:06:25
CN19-12 and IN133-1018	500 Mb	00:08:29	06:46:26

While the high volume of images that take a short time indicate there is no problem with opening, reading and hashing most of the images, perhaps some of the images are damaged. Perhaps the image is corrupted. We can tell from Table 3.2 that there is no problem with any of the images that took a long time to ingest initially. When creating the database with the target images non of the images took a long time. Which is good news, more evidence that there is nothing too slow about opening, reading and hashing each image.

Table 3.2. A closer look at differing insertion times for the same image size re-inserted.

Names	~ Size	Min Time (H:M:S)	Max Time (H:M:S)
CN32-04 and IN10-0229	64 Mb	00:00:15	00:00:55
CN27-57 and CN21-01	128 Mb	00:00:27	00:01:53
CN32-51 and IN10-0214	255 Mb	00:01:13	00:02:48
CN32-85 and CN6-12	350 Mb	00:01:24	00:03:48
CN19-12 and IN133-1018	500 Mb	00:01:44	00:04:39

CHAPTER 4:
Results

4.1 Top Common Matches

After ingesting 980 secondary-storage images we saw that the most common sector hash had 181,976,293 matches. We can also look at the other most common matches. We used the command in Figure 3.7 which took about 15 minutes to complete. The counts for the top 3 sectors are seen in 4.2.

```
db.RDC_NUS.find({}, {"_id":1, "total_count":1}).
    sort({"total_count" : -1}).limit(3)
```

Figure 4.1. A MongoDB Command to find most common MD5 hash.

```
{"_id" : "de03fe65a6765caa8c91343acc62cffc", "total_count" : 181976293}
{"_id" : "bf619eac0cdf3f68d496ea9344137e8b", "total_count" : 128869202}
{"_id" : "bde3baf7bc52f4db657ef3f8c47bdcbb", "total_count" : 19254824}
```

Figure 4.2. Most common hash with about 980 images inserted.

We know from previous experiments that most top matches are not probative. They are sectors with very simple patterns and because they cannot link a sector to a file or link two images to one another they should not be considered interesting.

We were able to identify 1537 of the 3000 most common sectors by comparing against sectors on a set of computers we had in our laboratory. Table 4.1 is a breakdown of the major kinds of 1537 sectors. It is clear that many of these common sectors are useless. We will now discuss in more detail what these patterns look like.

Pattern	Count
Single Repeating Character	68
Progressive Difference	74
25 % > Same Character	369
Repeating Sequence $\geq$ 2 characters	6
Consecutive Random Number	156
Zero block of > 20 in middle	337
Shannon Entropy > 4	518
Interesting Patterns Remaining	9

Table 4.1. Summary counts of different types of sectors found in the 1537 recognized sectors of the 3000 most common sectors in our hash collection.

We would like to eliminate the non-probative matches. An easy example is a pattern consisting entirely of one character. The most common sector, for instance, consisted of 512 NULL characters. We found other characters repeated 512 times (Table 3.4).

Unnecessary spaces in a sector can often be eliminated. However, if the spaces are randomly distributed within the sector, there can be $512!/500! \approx 10^{33}$ possibilities, too many to specify in advance. As an example we show in Table 4.3 a pattern of mostly ASCII character 255 with a few intervening NULLs.

Pattern Name	Single Character
Example of Pattern	13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 ...

Table 4.2. Example of 512 Bytes of the same exact character.

However, if the spaces are randomly distributed within the sector, since there are $512!/500! \approx 10^{33}$ possibilities they take more thought to identify. As an example we table 4.3 with a pattern of mostly ASCII character 255 and a few NULLs in-between.

Pattern Name	25% > same character
Example of Pattern	... 255 0 0 0 255 ...

Table 4.3. Example in which twenty-five percent or more of the sector is the same exact character.

We saw a number of sectors consisting of 511 occurrences of the same character and one occurrence of another character. For instance, we saw a sector of NULLs followed by a single 255 character. We found it was useful to test if a quarter or more of a sector had the same character. Similarly, if a 4-byte pattern repeated for more than a quarter of the sector then that sector is most likely non-probative or common [34].

Another pattern we saw was where every three characters the following character increased by 1. The in-between characters tend to be 3 NULL characters; however, we can generalize that to check for the in-between characters having a length between 1 and 4.

Pattern Name	Progressive Difference
Example of Pattern	1 0 0 0 2 0 0 0 3 0 0 0 4 0 0 0 5 0 0 0 6 0 0 0 7 0 0 0 8 0 0 0 9 0 0 0 10 0 0 0 0 11
Example of Pattern	129 7 0 0 130 7 0 0 131 7 0 0 132 7 0 0 133 7 0 0 134 7 0 0 135 7 0 0 136 7 0 0 137 7 0 0 138 7 0 0 139

Table 4.4. Example in which a byte value increases by 1 every X characters.

Repeating characters are another pattern we found frequently. We made the algorithm to count if it found two characters repeated, as for example in 4.5.

Pattern Name	2 > more characters repeating
Example of Pattern	31 3 31 3 31 3
Example of Pattern	80 65 68 68 73 78 71 88 88 80 65 68 68 73 78 71

Table 4.5. Example repeating sequence of characters.

A long block of nulls in the center of a sector also suggests a non-probative sector since it suggests misalignment of the view of the data.

We investigated our sectors for simple patterns and created algorithms to characterize those patterns. We successfully found patterns to eliminate from our database because common blocks will not help us cross drives or to find useful files.

4.2 Finding the right Shannon Entropy value

After creating algorithms to eliminate some of the common blocks we encountered we were still left with simple patterns to consider. We can use an entropy algorithm to find many other simple non-probative patterns. For instance, Table 4.6 shows a pattern that has 5 or more repeating characters, but the repeating characters are random. An alternative to regular expressions is to calculate the entropy of a sector and classify as uninteresting all sectors with low entropy. While this is simple to compute it is not as perfect as regular expressions. Thus forensic investigators have a decision to make, and sometimes perfection is necessary and sometimes not.

Pattern Name	Randomly repeating characters > 5
Example of Pattern	71 71 71 71 71 146 71 146 210 210 210 174 174 174 174 69 69 69 69 69 69 93 93 93 93 239 239 239 239 239 239 117 239 117 239 117 117 117 57 57 57 57 57 57 57 57 57 57 57 57 57 57 57 17 17 57 17 17 17 17 17 17 17 17 17 17 17 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 88 20 17 174 30 34 252 252 252 252 252 252

Table 4.6. Repeating sequence of 5 or more characters where the character repeated appears random.

Patterns in successive differences (e.g. 0, 1, 2, 3, 4, 5, ...) are another type of simple pattern that we can remove from consideration.

In thermodynamics entropy is the measure of randomness. In information theory we can measure the randomness with Shannon values. If we set X as a random variable the Shannon entropy equation is

$$H(X) = -\sum_{x} p(x) \log p(x).$$

To find a $H(X)$ threshold that will screen simple patterns and catch complex ones, we first took a random sample of 100 sectors. Then we identified the interesting sectors and label them as “positives”. We created a file and arranged it so the first 50 are complex or “positives”. Then we calculate the Shannon entropy for each sector in our sample. We then counted the number of true positives by using the set of representative thresholds and computing how many positives were over the threshold; these were true positives. Then we computed how many positives are below the threshold; these were false negatives. Then we computed how many non-positives were over the threshold; these were false positives. “The F score can be interpreted as a weighted average of the precision and recall, where an F score reaches its best value at 1 and worst at 0.” [40]

$$\text{Precision} = \frac{tp}{tp + fp}$$

$$\text{Recall} = \frac{tp}{tp + fn}$$

$$F = 2 \cdot \frac{1}{\frac{1}{\text{recall}} + \frac{1}{\text{precision}}} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

According to our Table 4.7 we can see that a Shannon Value of 4 will screen simple patterns and catch complex ones, so we recommend this value. But not everything above the threshold was correct, and this measured missed some of the patterns we referred to in the previous section.

Shannon Value	TP	FN	FP	F-Score
4	491	9	12	0.9791
4.5	443	57	7	0.9326
5	391	109	6	0.8718
6	146	354	3	0.4499
7	11	489	0	0.0430
8	0	500	0	NA

Table 4.7. Calculated F-Score given TP, FP, FN, and Shannon values.

4.3 Speeding up the Big Database

The secondary-storage image that took the longest to ingest into the database was approximately 350 Mb and it took eight hours and 20 minutes. It is encouraging that a smaller database does not take as long to create as a large one because now we know there is problem with how we set up our database.

Taking a second look at Figure 3.6 we see that the first 500 images total 119 GB and it took about 8.33 hours. At minimum the processing rate is 14.3 GB per hour and the total image size is 124,000 GB. When timing how long it takes our script to perform a task we must remember that the times are an approximation. Every time we run the script we get a different time. What is import here is to observe outliers.

When doing the same script in parallel if we keep the number of jobs at max three and then we estimate the ratio of one minute per GB. We find for example that nine secondary-storage images at that are one GB in size take about three minutes. Insertion time takes 30 minutes for one GB, so we can see that MongoDB is our bottleneck. Insertion time for the hashed sectors of one GB secondary image can take between seven minutes and 40 minutes. If we keep the max number of jobs to two then we can keep that time range when inserting sector hashes of each image into MongoDB. If not we run the risk of one GB taking two hours or more. It took about six hours to process 16 one GB hard drives.

We have 124,104,544,671,744 B of data or about 124,104 GB of data. Best case scenario it will take 1 minute to make a list of commands and 7 minutes to insert those commands per

GB of data. 8 minutes per GB of data. 124,104 GB divided by 8 minutes equals a speed of 15513 GB per minute or 86 days; which is 2.88 months.

It could be that the secondary-storage images that took hours to insert have a lot of exact same MD5 hashes. So we examined CN19-12 and IN133-1018. Remember CN19-12 an approximately 500 Mb image took eight and half minutes to ingest. We found that it had 972 of the exact same sector hashes `bf619eac0cdf3f68d496ea9344137e8b`. IN133-1018, also an approximately 500 Mb image, and it took close to 7 hours to ingest, it has 2,532 of the exact same sector hash `bf619eac0cdf3f68d496ea9344137e8b`,
and has 940,636 of the sector hash `96c8e709c96dce8f9ca6f3d760479345`. It is encouraging that we see an increase in matching numbers in the image that takes the longest. We now know we need to consider how to deal with a large number of matching sectors.

While finding this information we observed that we had to search through all of the MongoDB documents because the per source count key has nested values. It took 5,951 seconds or over an hour and half to search through all of the documents. This is a problem because when updating the document it will also take a long time to find the correct sub document to update. MongoDB works fastest when it can use its index value.

We updated the MongoDB documents so that there is nesting. With the updated schema we were able to process 1,000 of the secondary-storage images, sorted by size in six hours and 40 minutes; a significant improvement. That was an ingest of 646 GB out of 124 TB. Or a rate of $646GB/1840.23Minutes \approx 0.35GB/Minutes$ so it would take roughly $124000Gb/0.35GB/Minutes* \approx 354285.71minutes$. or about 246 days. Still quite sometime but an improvement of 115 days. It would be best to create the database in chunks and do an analysis in steps.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 5: Conclusion

We started this project with the idea that creating the database would be the first step and that doing statistical analysis would be the second step. Creating a large database takes a large amount of time, and our results suggest that the insertion rate per byte of data depends on too many factors to be very consistent. This however need not mean that using MongoDB is not viable.

It could be that our problem is that we do not have enough computing power. NPS has a super computer called Hamming that we could use. The downside is that because more people share the computer processing power has to be scheduled and data maybe deleted. We will save for future work the potential of using Hamming in addition to finding out if the number of concurrent workers or maximum sectors to be inserted will increase the speed.

As noted by Garfinkel when reflecting on the challenges of managing large-scale forensics data, solutions from other fields do not easily transfer because of the heterogeneous nature of the data. Large amounts of forensic data cannot be easily locally processed and reduced [27]. We must instead sample data (for example restrict tracking all of the offsets of matching MD5 hashes to limit the size of the MongoDB record and make the database viable). Also being patient and finding a safe place to stop without losing progress when inserting the data is something that we will add in future work.

THIS PAGE INTENTIONALLY LEFT BLANK

List of References

[1] *Guide to Integrating Forensic Techniques into Incident Response*, NIST SP800-86, 2006. Available: http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-86.pdf

[2] M. Komorowski, "a history of storage cost (update)," cited on September 3, 2016. Available: http://www.mkomo.com/cost-per-gigabyte-update

[3] Regional Computer Forensics Laboratory, "Regional computer forensics laboratory annual report for fiscal year 2014," 2014, cited on September 3, 2016. Available: https://www.rcfl.gov/downloads

[4] E. Casey, *Digital Evidence and Computer Crime: Forensic Science, Computers and the Internet*. Academic Press, 2004. Available: https://books.google.com/books?id=Xo8GMt_AbQsC

[5] B. Carrier, *File System Forensic Analysis*. Pearson Education, 2005.

[6] "Uniting and strengthening america by providing appropriate tools required to intercept and obstruct terrorism (usa patriot act) act of 2001," US Government Publishing Office, Tech. Rep., 2001. Available: http://purl.access.gpo.gov/GPO/LPS39935

[7] D. Gove, *Multicore Application Programming: For Windows, Linux, and Oracle Solaris* (Developer's Library). Addison-Wesley, 2011. Available: https://books.google.com/books?id=NF-C2ZQZXekC

[8] R. Gallardo, S. Hommel, S. Kannan, J. Gordon, and S. B. Zakhour, *The Java Tutorial: A Short Course on the Basics (6th Edition)*, 6th ed. Addison-Wesley Professional, 2014.

[9] D. Beazley and B. Jones, *Python Cookbook*. O'Reilly Media, 2013. Available: https://books.google.com/books?id=S_SJ2LaZH8EC

[10] L. Null and J. Lobur, *The Essentials of Computer Organization and Architecture*. Narosa Publishing House, 2003. Available: https://books.google.com/books?id=fge4kgEACAAJ

[11] J. Palach, *Parallel Programming with Python*. Packt Publishing Ltd, 2014.

[12] "Digitalcorpora," cited on July 25, 2016. Available: http://digitalcorpora.org/

[13] S. Garfinkel, P. Farrella, V. Roussev, and G. Dinolta, "Bringing science to digital forensics with standardized forensic corpora," *Digital Investigation*, vol. 6, pp. S2–S11, September 2009.

[14] "Open source digital forensics," cited on May 27, 2016. Available: http://www.sleuthkit.org/index.phpl

[15] "Sleuthkitwiki," cited on July 25, 2016. Available: http://wiki.sleuthkit.org/index.php?title=Main_Page

[16] J. R. Bradley and S. L. Garfinkel, "Bulk extractor 1.4 user's manual," Naval Postgraduate School, Monterey, CA, Tech. Rep. NPS-CS-13-006, Aug. 2013. Available: http://hdl.handle.net/10945/36027

[17] "Md5 and hmac-md5 security considerations," RFC 6151, Internet Engineering Task Force, Mar. 2011.

[18] K. H. Rosen, *Discrete Mathematics and Its Applications 7th edition*. McGraw-Hill, 2011. Available: https://books.google.com/books?id=C2c6twAACAAJ

[19] "Merriam-webster," June 2017, metadata. Available: https://www.merriam-webster.com/dictionary/metadata

[20] A. Taylor, *SQL All-in-One For Dummies*. Wiley, 2011. Available: https://books.google.com/books?id=373J4FVF4wkC

[21] "Mongodb," cited on May 27, 2016. Available: http://www.mongodb.org/

[22] C. Buckler, "Sql vs nosql: The differences," June 2017. Available: https://www.sitepoint.com/sql-vs-nosql-differences/

[23] "Sqlite database v2 schema," cited on May 27, 2016. Available: http://wiki.sleuthkit.org/index.php?title=SQLite_Database_v2_Schema

[24] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, July 1970. Available: http://doi.acm.org/10.1145/362686.362692

[25] S. L. Garfinkel and M. McCarrin, "Hash-based carving: Searching media for complete files and file fragments with sector hashing and hashdb," *Digital Investigation*, vol. 14, Supplement 1, pp. S95 – S105, 2015, the Proceedings of the Fifteenth Annual {DFRWS} Conference. Available: http://www.sciencedirect.com/science/article/pii/S1742287615000468

[26] S. L. Garfinkel, "Digital media triage with bulk data analysis and bulk_extractor," *Computers & Security*, vol. 32, pp. 56–72, 2013.

[27] S. Garfinkel, “Lessons learned writing digital forensics tools and managing a 30tb digital evidence corpus,” *Digital Investigation*, vol. 9, pp. S80–S89, 2012.

[28] National Institute of Standards and Technology. (2016, May). National Software Reference Library Reference Data Set. [Online]. Available: http://www.nsrl.nist.gov.

[29] S. Mead, “Unique file identification in the national software reference library,” *Digital Investigation*, vol. 3, no. 3, pp. 138 – 150, 2006. Available: http://www.sciencedirect.com/science/article/pii/S1742287606000958

[30] J. Young, K. Foster, S. Garfinkel, and K. Fairbanks, “Distinct sector hashes for target file detection,” *IEEE Computer*, vol. 45, pp. 28–35, 2012.

[31] S. Garfinkel, A. Nelson, D. White, and V. Roussev, “Using purpose-built functions and block hashes to enable small block and sub-file forensics,” *Digital Investigation*, vol. 7, Supplement, pp. S13 – S23, 2010, the Proceedings of the Tenth Annual {DFRWS} Conference. Available: http://www.sciencedirect.com/science/article/pii/S1742287610000307

[32] Wikipedia, “Photorec — wikipedia, the free encyclopedia,” 2016, [Online; accessed 5-July-2016]. Available: https://en.wikipedia.org/w/index.php?title=PhotoRec&oldid=727327617

[33] S. Collange, Y. S. Dandass, M. Daumas, and D. Defour, “Using graphics processors for parallelizing hash-based data carving,” in *System Sciences, 2009. HICSS’09. 42nd Hawaii International Conference on*. IEEE, 2009, pp. 1–10.

[34] K. Foster, “Using distinct sectors in media sampling and full media analysis to detect presence of documents from a corpus,” Master’s thesis, Naval Postgraduate School, Monterey, CA, September 2012.

[35] Forensics Wiki (2016, May). Bulk extractor. [Online]. Available: http://www.forensicswiki.org/wiki/Bulk_extractor.

[36] K. Mastwijk, “Libewf is a library to access the expert witness compression format (ewf),” cited on June 19, 2016. Available: https://github.com/libyal/libewf

[37] “bson,” June 2017. Available: http://bsonspec.org/

[38] MongoDB, “Model data for atomic operations,” cited on Sept 9, 2016. Available: https://docs.mongodb.com/manual/tutorial/model-data-for-atomic-operations/

[39] “Mongodb documentation,” cited on May 27, 2016. Available: https://docs.mongodb.com/

[40] D. L. Olson, *Advanced Data Mining Techniques*. Springer, 2008.

THIS PAGE INTENTIONALLY LEFT BLANK

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California