2017-09

Database creation and statistical analysis: finding connections between two or more secondary storage devices

Johnson, Jennifer M.

Monterey, California: Naval Postgraduate School

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

1. AGENCY USE ONLY *(Leave Blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 2017	Master's Thesis 07-31-2014 to 9-29-2017

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 used MongoDB and created a database of each disk image and each unique sector found in the Real Data Corpus—a collection of disk images held by the Digital Evaluation and Exploitation Lab. 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. We also characterized some of the non-probative sectors found in our database. Future students may benchmark other databases and shard the database.

14. SUBJECT TERMS

digital forensics, sector hashing, common blocks, hash databases

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 used MongoDB and created a database of each disk image and each unique sector found in the Real Data Corpus—a collection of disk images held by the Digital Evaluation and Exploitation Lab. 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. We also characterized some of the non-probative sectors found in our database. Future students may benchmark other databases and shard the database.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

THIS PAGE INTENTIONALLY LEFT BLANK

List of Figures

THIS PAGE INTENTIONALLY LEFT BLANK

List of Tables

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

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

YCSB Yahoo! Cloud Serving Benchmark

Acknowledgments

I have so many people to thank because I could not have written this thesis on my own. This list is in no particular order, because I am grateful toward everyone. I thank my sister, Edwarda, for all her love and support. What would I have done 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 than once. I thank my thesis advisor, Michael McCarrin, for his epic patience during our marathon thesis meetings. I thank my second reader, Neil Rowe, for his hard work and support. I thank Thao for reminding me that this program exists, and for 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 for this chance to earn my master's degree in Computer Science (CS) with an undergraduate degree in Mechanical Engineering (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 digital forensic connections between two or more secondary-storage devices. The growing amount of data is our motivation. In recent years, the per-gigabyte price of data has been steadily decreasing [1]. 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, noted in their annual reports that the Chicago lab, just one of the 15 labs, had collected and processed 580 TB of digital data in one year [2].

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. With the current tools it is difficult to detect collaboration or communication between owners of devices acquired at different times. Likewise, more needs to be 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.

Analyzing trends can be divided roughly into two categories. One looking for things we already know about and two trying to understand the unknown. Trying to understand the unknown is generally much harder. The goal or 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. You might have cringed at the vagueness of that question, perhaps you are thinking only fictional characters get to explore where no woman has gone before.

Neil deGrasse Tyson wrote a book *Astrophysics for People in a Hurry* which explores dark energy and the mystery behind the force that expands the universe. On *Real Time with Bill Maher*, Maher asks why we should care and Tyson says "I don't know." He goes on to explain that about 90 or 80 years ago scientist were first discovering the atom and got asked that very same question and now atoms are the basis for all current science and technology [3]. While our work may not become the foundation for all forensic science 90 years from now the field is in serious need of exploration and innovation to find solutions for dealing with large amounts of heterogeneous data.

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. When a file is deleted the file-system no longer indexes it but the data is not erased [4]. The fact that the data in is not erased is what makes it a possibility that we would find duplicate material and that would be an interesting pattern. An experiment was performed on 150 disk images 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 (data not indexed by the file-system) 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 will build a forensics database and look for patterns over images on the RDC.

1.2 DOD Applicability

Cyberspace is an established warfare domain for the Navy. The U.S. Patriot Act, in Title VIII, section 816, identifies "Development and support of cybersecurity forensic capabilities" as a priority [5]. 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
- Characterization of non-probative 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 1 is building the database and step 2 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?

CHAPTER 2:
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 Core Concepts

2.1.1 Shannon Entropy

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

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

2.1.2 F Score

"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" [6].

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

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

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

2.1.3 Digital Forensics

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. 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" [7].

Garfinkel in his 2012 survey on lessons in digital forensics defines and describes the current and trending state of the field. A major challenge in the field of digital forensics is the growth of data diversity and data scale. Forensic analysts have a need for software that meets these challenges [8]. Our work focuses on analyzing secondary-storage images in a large scale.

2.1.4 Disk Images

The NUS portion of the Real Data Corpus is raw data extracted from secondary-storage images [9]. The RDC primarily consists of USB flash memory devices and computer drives [9]. Despite the fact that the secondary-storage images had been discarded by their owners, many of the drives in the RDC had not been erased by their owners [10].

The simplest type of forensic image is raw format: an exact sector-by-sector of the original secondary-storage device. Another type of image contains the raw data as well as a checksum and metadata; the most common implementation is EWF format. The checksum helps ensure integrity is preserved [4]. The metadata provides information about the secondary-storage image. Our forensic data set, the RDC, splits each image into a fixed size chunk and names those chunks in sequence (i.e., E01, E02, E03, E04, and so on).

2.1.5 Forensic Artifacts

When a file-system has been compromised by an attacker we call the evidence left behind forensic artifacts [11]. In general, forensic artifacts may also refer to useful information found on the file-system. For example, *bulk_extractor* identifies credit card numbers, IP addresses, email address and many other artifacts that are often called features [12].

2.1.6 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 [13]. MD5 and other cryptographic hashes are 160 bits and are designed so that it is very unlikely for a collision to occur [14]. A hash collision happens if two different inputs produce the same hash [15]. 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.

2.1.7 Relational and Non-relational Databases

Our research uses both relational and non-relational databases to store and manages forensic data. 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 and it is a relational 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 The Sleuth Kit (TSK).

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

The SQL command provides the output result shown in Figure 2.1: a 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" [16]. A database schema consists of metadata [17]. The columns of the table label the attributes of the data, and the rows contain the data [17]. A schema created from a table is called relational. An

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 2.1. Example of SQL Table.

alternative database type is a non-relational database. An example is MongodDB which uses a document-schema database [18]. MongoDB uses BSON documents to store data records [18]. BSON is short for Binary JSON (JavaScript Object Notation) [19]. A document is similar to a Python "dictionary" or hash table. A MongoDB document is identified with `_id` a required special key that identifies the document and insures that it is unique in the collection. 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 [20].

The `tsk_file_layout` table stores the layout of a file within the image [21]. The `tsk_files` table lists every file found in the images and has the basic metadata for the file [21]. The layout of file can be connected to the metadata of the same file using a technique known as normalization [20]. 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 [20]. 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 [20]. 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 [20].

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 establishes

and enforces a link between the data in two tables. This is not available in non-relational databases [20]. MongoDB and other non-relational databases use Java script like query commands and nested documents can become complex when trying to query [20]. When creating a large database, distributing its contents among multiple servers may be necessary; non-relational databases' use of simpler data models makes this easier to do than SQL-type databases [20]. This is the main reason we chose to build our database using a non-relational database.

2.1.8 National Software Reference Library

The National Software Reference Library (NSRL) currently maintains a database of metadata consisting of a hash of the file's content, the file's origin (the software typically required to view it), original name, and size [22]. The hash is produced using, among other hash algorithms, MD5, and secure hash algorithm 1 (SHA-1) [23]. 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 re-examining known files [23].

2.2 Secondary Storage Concepts

2.2.1 File-System Storage

Writing data to a device requires consulting the correct file-system 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*. 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 [4], 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.

Table 2.1. Example Strings Offset and Data in Hexadecimal Format.

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

2.2.2 Sectors

A sector is the smallest unit that can be accessed on media [7]. They are typically 512 B or 4096 512 B, the size is determined by the manufacturer of the hardware. When needing to read or write data on a disk it is done at the sector level [4]. A file-system uses file allocation units, the smallest unit is a block, sometimes referred to as clusters, and is typically 4096 B [7].

2.2.3 Sector Addresses

Reading and writing 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. Damaged sectors may be skipped without the user transparently at the device level [4].

2.2.4 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 may be used to find potentially hidden data. For example, FAT32 file-systems do not use sector 3 so if the investigator uses the `dcat` tool found in TSK she can view a specific data unit in either raw or hexadecimal. If that data is non-zero then this may be evidence of hidden data [4]. 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 may be be 1,024, 2,048 or larger.

2.2.5 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 [4]. In addition to this rule most file-systems do not over write 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 may be an area of slack space [4]. The file-system determines what is done with the slack space. Some fill the space with data from random access memory (RAM), or zeros [4].

2.3 Forensic Tools and Techniques

2.3.1 Artifact Extraction

We use TSK, a library, a framework, and a collection of command-line tools for forensic investigation disk images [24]. 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 [25]. The `tsk_loaddb` command populates a SQLite database with metadata from a disk image [25].

The disk-image layer includes the entire secondary-storage image. Many system configurations use a volume-system. In [7], NIST SP800-86 guide observes that logical volumes are created from partitions in the image. The guide also explains that a partition is a logical division of the disk-image into separate units. The guide describes how a file-system resides on one or more partitions and determines how files are stored, organized, and accessed on logical volumes. The guide writes that there are many different file-systems; however all have some common attributes. They use directories and in most cases subdirectories to organize and store files. File-systems make use of a data structure to point to location of files on the image. File allocation units are used to store a file. A cluster is a common name for the file allocation unit [7].

The NIST SP800-86 guide discusses how a file-system may hold data from deleted files or earlier versions of existing files. This data can 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 [7]. Space that has not been allocated to a partition, perhaps unallocated clusters or blocks, or space where files or volumes have been deleted, may also contains forensically useful information. The reason we hash at the sector level is to grab all of the small bits of forensic data that would otherwise be lost in deleted, free, or slack space.

The `mmls` command of the TSK tool displays the partition layout of a volume system [24], as shown in Figure 2.2. 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" [26].

```
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 2.2. Partition Table Layout, `mmls` Command Output.

2.3.2 File Carving

File carving is a data 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 [4]. The file-system meta-data is not required and this means that files maybe carved from unallocated space [4].

Full file hashes are limited with respect to their ability to identify carved files because the hash that makes each file's content unique will only match identical content. Therefore, a small change to a file or a corrupt block means the hash will change and the file will no longer be identifiable [27]. In order to solve this problem, Garfinkel explores using cryptographic hash functions on sectors or blocks of data in order to search for target files [28]. The term hash-based carving means searching for the target file in a given secondary-storage image by first hashing blocks of the file, rather then the entire file [28].

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" [28].

A "probative," or distinct block, is a block that indicates a high probability that the entire

targeted file was on the device at some point. A common block, the most common being a set of all NULLs, is a block that does not give strong evidence of a correlation between the data region's in which it is found. "Non-probative" is another term for common block [29], [30].

Hash-based carving inherently increases the size of the data a forensic analyst must process. If we for example make a gross assumption that each file needs to be sectioned into 1,000 blocks and if we had been dealing with 10 million files, we are now dealing with 10 billion 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 factors we can adjust to attempt to speed up matching are the hardware, the type of database in which the blocks are indexed, the algorithm to search the database or all those methods in combination.

Collange et al. in their 2009 study noted 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 brute force method of comparing the contents of each sector on a given secondary-storage image with the target file sectors is time consuming. 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 [31].

In 2012, Foster examined whether sector hashing is effective for identifying given forensic artifacts. 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 could be 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. Foster notes

the limitation that files must be sector aligned on the disk for successful identification [32]. The *bulk_extractor* scanner was created as a tool that builds and search the Bloom filter database [8], [12].

CHAPTER 3:
Methodology

3.1 Experimental Setup

First, we build a database designed to inspect unique individual sectors of the 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 also match and compare individual sectors with metadata from volume, partition and file-systems, as well individual files.

3.1.1 Hardware

We ran our experiments on a server of 64-cores and a 512 GB main memory node that is dedicated for Digital Evaluation and Exploitation Lab, or DEEP, use.

3.1.2 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 command line tools. TSK can create SQLite databases of metadata extracted from each image and we used schema version 2. Rather than use SQLite, leave this information in, we import it into MongoDB because the flexible documents of MongoDB allow for larger collections to be split across multiple servers. We used the library libewf to access the Expert Witness Compression Format (EWF), the pyewf bindings allows us to do this using Python [33]. The pyewf library allows us to convert EWF to raw format, which we divide into 512 B sectors.

3.2 Designing Schema

To set up our database, we first constructed a non-relational schema for the secondary-storage images. We designed a schema to contain metadata about each image. This meta-

data was extracted first using TSK and stored in SQLite files, one for each image. The `ewfinfo` command from TSK gives four pieces of useful information, (see Figure (3.1)): the MD5 hash of the image, the size of the image, the name of the device, and whether the partitions of the device's volume-system are recognizable. We used the MD5 hash of the image as a key to track which sector we are referring to. We used the size as a way to sort the images so that we could use the smaller images first.

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

Figure 3.1. Four Pieces of Useful Information.

3.3 Data Set

Our data set consists of the secondary-storage images in the non-U.S. 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. 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 inves-tigation 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 attributes we retrieve from the TSK `tsk_loaddb` command are as follows:

- TSK`tsk_loaddb` produces a SQL table named `tsk_image_info` that 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. TSK `tsk_loaddb` produces a SQL table named `tsk_vs_info` and

holds the following 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. 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.
- 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.

If the file-system starts at an address that is not evenly divisible by our block size, then 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 [29]. This is a problem if we choose to hash 4086 B blocks and the file-system starts on sector 63, and the underlying sectors are 512 B and not 4096 B. If the sector size is the same as the block size we are hashing there is no alignment problem. It is typical to see file-systems start at sector 63 with the images in our holdings.

3.4 Database Creation

In order to create our database of hashed sectors for the entire media on the non-U.S. portion of the Real Data Corpus we first considered using hashdb. It is easy to configure for use with hash blocks of 512 B. However, although hashdb can handle billions of hashes, it cannot easily scale to the approximately 240 billion hashes required to represent all 512 B sectors in the RDC. In addition, if we wanted to do a cross drive hash match, hashdb may not be the ideal tool, since it relies on a tree-based storage structure that would require $O(n^2)$ look ups.

MongoDB has the advantage of being more flexible but the disadvantage of not being as fast. We started the database by successfully importing the image, partition and file-system information from TSK output. Having that information made it easy to find that all of the images use 512 B sector as the smallest division. We know that file-systems are sector aligned because we used 512 B sectors and not 4096 B.

The bulk of our database consists of MD5 hashes we created from secondary-storage image sectors. To create these documents, we open and read each image at the byte level, section the image into 512 B, and create an MD5 hash of each sector. Each hash is used to create a document in MongoDB. The resulting document contains a list of source hashes in the key `src_id`. We can use this field to 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. This value is capped at ten because, while most hashes are rare, a few repeat thousands or millions of times. Stating every offset for these pathological cases can cause the document to grow too large, as seen in Figure 3.3.

In order to create the MongoDB documents as shown 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 on 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.

```
{'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.

```
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.

3.5 Calculating the F Score

To calculate the F (see the equation in Section 2.1.2) score to screen simple patterns and catch complex ones, we first took a random sample of 500 sectors. Then we identified the interesting sectors and label them as "positives." The sectors are interesting if they have not been screened by the characterizations we made by examining the top 1500 matching sectors we discovered, as shown in Table 4.1 of Section 4.1. We created a file and arranged it so the first 250 are complex or "positives." Then we calculate the Shannon entropy, see Section 2.1.1, for each sector in our sample and used this as our threshold. 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. True negatives are not required for the calculation of our F score. However, they occur when the sector is "non-interesting" and they fall below the Shannon threshold. Then we computed how many "non-interesting" were over the threshold; these were false positives. Then, we computed how many positives are below the threshold; these were false negatives (see Table 3.1).

Table 3.1. Definitions of TP, TN FP, FN.

True Positive (TP)	"interesting" sector w/ entropy > threshold
True Negative (TN)	"non-interesting" sector w/ entropy < threshold
False Positive (FP)	"non-interesting" sector w/ entropy > threshold
False Negative (FN)	"interesting" sector w/ entropy > threshold

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 also examined the other most common matches. We used the command in Figure 4.1, which took about 15 minutes to complete. The counts for the top three sectors are seen in Figure 4.2. The nice thing about using MongoDB is that we also could have examined the first 10 or 100 most common matches. For our analysis, we examined the first 1500 sectors that had a match of 1 or more.

```
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 [34]. They are sectors with very simple patterns and therefore are not strongly correlated with forensic artifacts. 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 1,537 of the 3,000 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 1,537 sectors. It is clear that many of these common sectors contain no

information that would be helpful to a forensic analyst. We will now discuss in more detail what these patterns look like.

Table 4.1. Summary Counts of Different Types of Sectors Found in the 1,537 Recognized Sectors of the 3,000 Most Common Sectors in Our Hash Collection.

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

We would like to eliminate the non-probative matches from our database. 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 4.2.

Table 4.2. Example of 512 Bytes of the Same Exact Character.

Single Character
13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 ...

Unnecessary NULLs in a sector can often be eliminated. However, if the NULLs are randomly distributed within the sector, there can be $512!/500! \approx 10^{33}$ possibilities—too many to specify in advance.

We characterized the condition where 25% or more of the sector has the exact same character. If all of the repeating characters are at the beginning, and we make that character NULL, this gives a lower bound of 255^{128} sectors that end with 384 NULLs. We are accounting for a lot of scenarios with one algorithm. As an example, we show in Table 4.3 a pattern of mostly ASCII characters 255 with a few intervening NULLs.

Table 4.3. Example in which Twenty-Five Percent or More of the Sector is the Same Exact Character.

25% > same character
... 255 255 255 255 255 255 255 255 0 0 0 255 255 255 255 255 255 255 255 ...

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 [32].

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, sometimes it is a character and two NULLs. We can also describe this as an incrementing 4-byte integer, see Table 4.4.

Table 4.4. Example in which a Byte Value Increases by 1 Every 3 Characters.

Progressive Difference
0 0 0 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** 11
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

Repeating characters are another pattern we found frequently. We wrote a script to count if it found two characters repeated. We also found repeating strings of 16 characters, see Table 4.5.

Table 4.5. Example Repeating Sequence of Characters.

2 > more characters repeating
31 3 31 3 **31 3**
88 80 65 68 68 73 78 71 80 65 68 68 73 78 71 88
88 80 65 68 68 73 78 71 80 65 68 68 73 78 71 88

We found heuristics to identify sectors that are likely to be common. We did this by investigating sectors 1,500 that had a match of one or more. We successfully found patterns

Randomly repeating characters > five
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.

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 five or more repeating characters, but the repeating characters are random. An alternative to using heuristics is to calculate the entropy of a sector and classify as uninteresting all sectors with low entropy. While this computation is simple, it is not as individual as heuristics based on observation. Thus forensic investigators have a decision to make. Sometimes perfection is necessary, and sometimes it is not.

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 measure missed some of the patterns we referred to in the previous section.

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

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

4.3 Investigating Ingestion Rate

We started building our database by ingesting 100 of our images. We sorted from our smallest of 2,490,368 B to our largest at 1,000,204,886,016 B. It took about 8 minutes to ingest 100 images that are about 60 Mb. We have 118 images that are about 60 Mb or less. To be exact, the first 100 images totaled about 4 Gb and that is 7,981,752 sectors of 512 B.

Then, we increased our ingest size to 500 images. They happen to be about 500 Mb in size or less. It took about 8 hours to finish. Those 500 images equal about 118 Gb total or 231,530,983 sectors of 512 B. This means we increased the ingest size by about a factor of 29. Yet, the time increased by about a factor of 60. To look for patterns we created a scatter graph of time to ingest versus the size of the image, as shown in Figure 4.3. We observe that with the exception of one outlier at 8 hours that both processing time and size of the image increase linearly. We also observe that the same size image has range to its insertion rate in four places: 60 Mb, 130 Mb, 255 Mb and 500 Mb. We created Table 4.8 so that we can examine the range more closely.

When looking at the range of values as in Table 4.8, we asked whether there was something unique about the data that took a long time. We reexamined the secondary-storage images as seen in Table 4.8 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. The image with the longest processing time was not always inserted last.

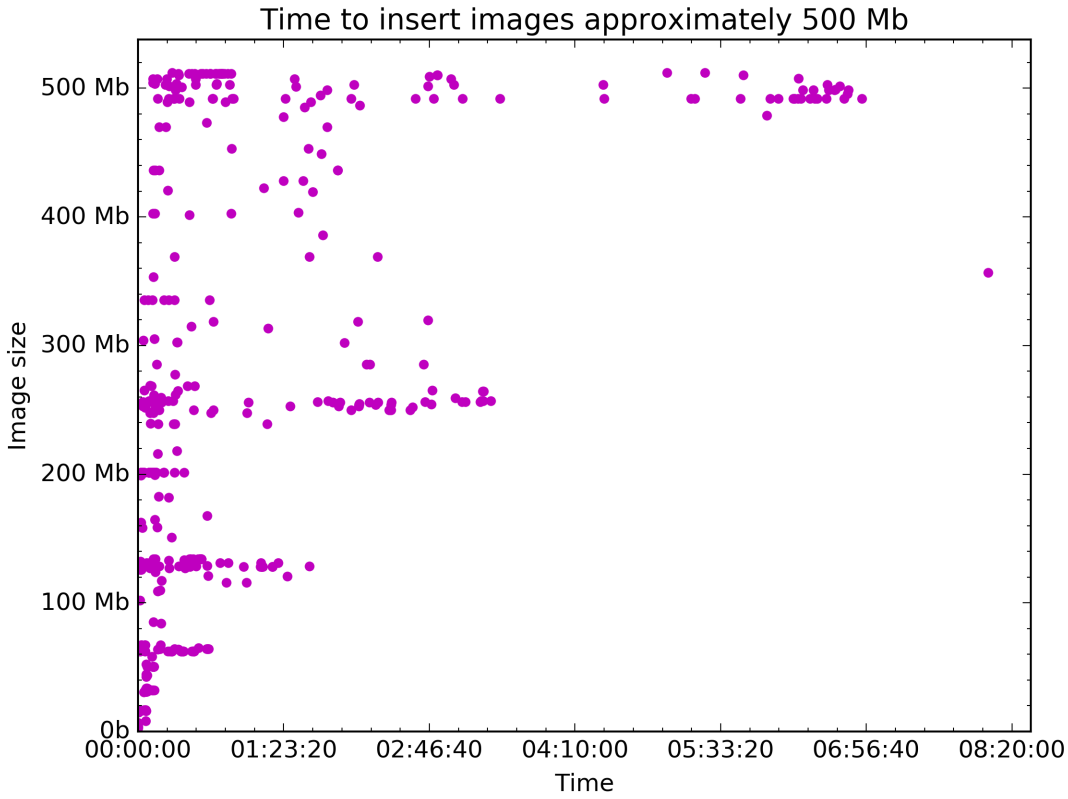

Figure 4.3. Inserting Secondary-Storage Images that Are Smaller than Approximately 500 Mb.

Table 4.8. 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 4.9 that there is no

problem with any of the images that took a long time to ingest initially. We created a new test database using only the targeted images. This means we are not accounting for the possibility that those images simply have a large amount of the same exact hash. When creating the database with the target images none of the images took longer than 5 minutes to process. This result provides more evidence that there is nothing too slow about opening, reading and hashing each image.

Table 4.9. 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

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 logged timing data for each image. We started by inserting the images that were approximately 500 Mb or smaller in size.

Creating the database this way immediately is slow. We will look at the numbers in detail in Section 4.4. As we build the database, it is good to keep in mind some logical limitations. Our speed is also bound by the read and write speeds of our private server's hard drives. MongoDB has granular locks and when a document is being written, only one instance of MongoDB can write to it [35]. Write applications are atomic. MongoDB has concurrency control. Each document has a unique index, which is the MD5 hash of each 512 B sector [36]. In the case of multi-document transactions, or concurrency, MongoDB uses a two phase commit. The actions are initialized and then applied [36]. This is how we can use the multiple cores available.

4.4 Speeding up the Database

We analyzed our ingestion rates in terms of disk size over time in order to search for a pattern that would allow us to calculate how long it will take to build our database. The disk image that took the longest to ingest into the database was approximately 350 Mb, and it took eight hours and 20 minutes. This is an outlier and when we re-ran the same image it took just under 4 minutes to digest. This instance is extreme, but it points out the reason why we need to run our scripts multiple times and take the average. We also divided the overall insertion job into discrete jobs that include reading, creating the hash, creating the MongoDB documents and inserting them into the database.

We ran the same script in parallel and kept the number of jobs at max three and then we calculated the rate in GB per minute. We found that disk images that are one GB in size take about three minutes to to open and read and hash. Inserting the hashes of those one GB images into MongoDB can take between seven minutes and 40 minutes. 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 terabytes (TB) of data. Best case scenario it will take 1 minute to create the MongoDB documents and 7 minutes to insert those commands per GB of data—8 minutes per GB of data. We calculate that 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 disk images with exceptionally long insertion times have a lot of the same MD5 hashes. This could produce a delay because MongoDB locks that can occur at the document level [18]. To investigate this possibility we examined CN19-12 and IN133-1018. Recall that CN19-12 an approximately 500 Mb image, took 8.5 minutes to ingest. We found that it had 972 of the exact same sector hash.

`bf619eac0cdf3f68d496ea9344137e8b`

This sector hash is all NULLs. IN133-1018, also an approximately 500 Mb image, which took almost 7 hours to ingest, 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 repeated hashes in the images that take 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 a 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 no 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 646 $Gb \div 4000\ min \approx 1.615\ Gb/\ min$ so it would take roughly 124000 $Gb \times \frac{1\ min}{1.615\ Gb} \approx 53$ days. Still quite some time but an improvement of 86 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 were able to build a partial database and characterize some of the non-probative sectors that we found. It was our research goal to find interesting patterns across the hashed sections of the images of the Non-US portion of the Real Data Corpus. What we found was a way to find and therefore eliminate many of the common blocks that not only slow down the ingestion of a large forensic database but also overwhelm the observation of the interesting sectors.

When the developers of MonogDB decided to focus their efforts on improving performance from their 2.0 to 3.0 release, they focused on write performance and hardware utilization [37]. As they set up their experiment for creating a benchmark they noted "cases for MongoDB are diverse, and it is critical to use performance tests that reflect the needs of your application and the hardware you will use for your deployment. As such, there's really no 'standard' benchmark that will inform you about the best technology to use for your application. Only your requirements, your data, and your infrastructure can tell you what you need to know" [37]. In the best case scenario, we would have used Yahoo! Cloud Serving Benchmark (YCSB), "a framework and common set of workloads for evaluating the performance of different 'key-value' and 'cloud' serving stores," and tried it on a few different non-relational databases in an attempt to judge the best-suited database for our hardware [38].

In addition, we could have used sharding on the database. Sharding is used to distribute data over multiple servers [18]. Sharding works on large databases because it is meant to spread CPU capacity and the I/O capacity over more than one disk drive [18].

THIS PAGE INTENTIONALLY LEFT BLANK

List of References

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

[2] 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

[3] B. Maher, "Episode 426," over time with Bill Maher, May 2017. Available: http://www.hbo.com/real-time-with-bill-maher/episodes/15/426-episode/index.html

[4] B. Carrier, *File System Forensic Analysis*. Boston, MA: Pearson Education, 2005.

[5] "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

[6] D. Olson and D. Delen, *Advanced Data Mining Techniques*. New York, NY: Springer, 2008.

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

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

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

[10] 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.

[11] J. R. Vacca, *Computer Forensics: Computer Crime Scene Investigation*. Newton Centre, MA: Delmar Thomson Learning, 2002.

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

[13] T. Bolan and G. Fisher, "Selection of hashing algorithms," June 2010, cited on September 21, 2017. Available: https://www.nist.gov/sites/default/files/hash-selection.pdf

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

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

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

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

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

[19] "bson," June 2017. Available: http://bsonspec.org/

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

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

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

[23] 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

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

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

[26] 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

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

[28] 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

[29] 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

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

[31] 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.

[32] 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.

[33] 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

[34] F. J. Gutierrez-Villarreal, "Improving sector hash carving with rule-based and entropy-based non-probative block filters," Master's thesis, Naval Postgraduate School, Monterey, CA, March 2015.

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

[36] "Mongodb documentation," cited on May 27, 2016. Available: https://docs.mongodb.com/

[37] "Performance testing mongodb 3.0 part 1: Throughput improvements measured with ycsb," March 2015, cited on September 18, 2017. Available: https://www.mongodb.com/blog/post/performance-testing-mongodb-30-part-1-throughput-improvements-measured-ycsbm

[38] "Yahoo cloud serving benchmark," April 2010, cited on September 18, 2017. Available: https://research.yahoo.com/news/yahoo-cloud-serving-benchmark/

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