Contents lists available at SciVerse ScienceDirect

Digital Investigation

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

Real-time digital forensics and triage

Vassil Roussev*, Candice Quates, Robert Martell

University of New Orleans, USA

ARTICLE INFO

Article history:
Received 10 November 2012
Received in revised form 30 January 2013
Accepted 2 February 2013

Keywords:
Digital forensics
Triage
Real-time forensics
Latency-optimized target acquisition
LOTA

ABSTRACT

There are two main reasons the processing speed of current generation digital forensic tools is inadequate for the average case: a) users have failed to formulate explicit performance requirements; and b) developers have failed to put performance, specifically *latency*, as a top-level concern in line with reliability and correctness.

In this work, we formulate forensic triage as a real-time computation problem with specific technical requirements, and we use these requirements to evaluate the suitability of different forensic methods for triage purposes. Further, we generalize our discussion to show that the complete digital forensics process should be viewed as a (soft) real-time computation with well-defined performance requirements.

We propose and validate a new approach to target acquisition that enables file-centric processing without disrupting optimal data throughput from the raw device. We evaluate core forensic processing functions with respect to processing rates and show their intrinsic limitations in both desktop and server scenarios. Our results suggest that, with current software, keeping up with a commodity SATA HDD at 120 MB/s requires 120–200 cores.

1. Introduction

Compute resources have grown exponentially in line with Moore's Law since the 1960s, and are projected to continue to do so for the foreseeable future. As Patterson (2004) showed, storage capacity has enjoyed its own exponential growth in parallel, with no end in sight. The average amount of data per case, as experienced by FBI's 15 Regional Computer Forensic Laboratories, has grown 6.65 times (from 84 GB to 559 GB) in eight years (2003–2011) (RCFL).

To simplify, more computing enables the cheap processing and storing of more data, which indirectly *demands* proportionally more compute resources to be deployed for forensic purposes.

Although the above outline of the problem is generally known, the reality is that the ability of forensic tools to employ a bigger "computational hammer" has not grown appreciably. We submit that the blame for this state of affairs should be shouldered by *both* users and developers.

Users—digital forensic analysts—are focused on the daily grind of following established routines and analyzing their cases using the available tools on their workstation. Let's assume that ten years ago this was an adequate approach, and an analyst could turnaround cases within a reasonable amount of time. Every year, the amount of preprocessing that needs to be done keeps growing faster than the compute facilities of a lone workstation. To some degree, the mismatch has been compensated for by better methodology, more selective processing, and more experienced analysts. However, over time, bridging the gap with ad-hoc measures has become increasingly infeasible. By now, most professionals agree that (lack of) tool performance is a central part of the problem (Hibshi et al., 2011), yet we have not seen clearly defined requirements that developers can pursue.

Absent specific requirements, developers have largely been content with their own "best effort" approach to

* Corresponding author. Department of Computer Science, University of New Orleans, 2000 Lakeshore Dr., 311 Mathematics Bldg., New Orleans, LA 70148, USA. Tel.: +1 5042806594.
E-mail address: vassil@cs.uno.edu (V. Roussev).

1742-2876/$ – see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.diin.2013.02.001

performance. With the exception of a few research projects (discussed later) developers have continued to focus primarily on providing new means to extract more data out of forensic targets. Arguably, some of these insights can help speed up processing but there is no overarching design that brings all the pieces together, such that timely processing is ensured.

At this point, it would be tempting to use the old "boiling frog" metaphor, which claims that a frog placed in a pot of water that is heated slowly will eventually be cooked without even noticing. We prefer to consider the actual behavior of a frog under these conditions as inspiration: "*As the temperature of the water is gradually increased, the frog will eventually become more and more active in attempts to escape the heated water. If the container size and opening allow the frog to jump out, it will do so.*" (Gibbons, 2002)

In other words, we believe that practitioners and developers are "feeling the heat" but the efforts so far have not been very focused, or coherent, and mere agitation is not sufficient for success. The goal of this work is to contribute to the formulation of clear and realistic goals and objectives, and to evaluate the current state of affairs with respect to them, with a particular emphasis on triage.

1.1. Real-time computing

Real-time computations are distinguished by the fact that they have formal deadlines by which they must be completed; otherwise, the computation is considered incorrect. Conceptually, any computation can be viewed as having a deadline, even if not explicitly given, as users have *implied* timeliness expectations. In practice, the term *real-time system* (RTS) is used more narrowly to describe a system that has a *short* deadline (typically on the order of milliseconds) to react to external input. For example, an airplane's autopilot must monitor input from hundreds of sensors and react accordingly. Systems in which failure of one, or more, of the system's processes to meet their deadlines can have catastrophic consequences are referred to as *hard* RTS, and are engineered such that computations meet their deadlines under *all* circumstances.

On the other end of the spectrum are *soft* RTS, in which the cost of missing some fraction of the deadlines is tolerable. For example, NTSC standard quality video playback runs at 30 frames per second, which means that every 33 ms the hardware will automatically render the current content of the frame buffer. This implies that a video player must produce a new frame every 33 ms, or else the old frame would be rendered. In practice, missing the deadline would have a negligible perceptual effect, as long as it does not happen too often.

Video playback is an example of a particular type of RTS—one in which a predictable amount of data needs to be processed per time unit. Thus, the real-time requirement is often specified implicitly as an average data processing rate (e.g., 5 MB/s). Evidently, an application could miss some of the internal frame deadlines yet meet the overall rate requirement; however, such lapses are usually acceptable.

In our view, digital forensic processing ought to be viewed the same way—as a soft real-time process with an explicit processing rate requirement. To be clear, we are only concerned here with the computational tasks—such as hashing, indexing, and filtering—performed by the forensic software. Clearly, overall investigation time is influenced by a large number of other factors that are beyond the control of the tool developer.

1.2. Forensic computing with deadlines

How do we formulate forensic processing as a real-time task? A central part of the performance problem today is that forensics is viewed primarily as an open-ended, post-mortem analysis. This is reflected in the canonical procedural model (Kent et al., 2006), which prescribes a linear processing model of acquisition/collection, followed by examination, followed by analysis, followed by reporting. This model could be made even more detailed (Harrell, 2010):

$$\begin{aligned} &\textit{preparation} \rightarrow \textit{identification} \rightarrow \textit{acquisition} \rightarrow \textit{analysis} \\ &\qquad \rightarrow \textit{reporting} \rightarrow \textit{archiving} \end{aligned}$$

Regardless of its level of detail, this is fundamentally a sequential model in which each stage waits for the previous one to complete before it commences. Thus, the only means to improve end-to-end latency is to speed up all stages of the process. Unfortunately, exactly the opposite trend is in place. As an illustration, consider the acquisition and analysis stages, which are on the critical path for all subsequent processing.

Acquisition rates are limited by the maximum sustained throughput from the target drives. For high-capacity drives—the main pressure point—acquisition time has increased from 1 h in 2003 (200 GB at 58 MB/s) to almost 7 h (3TB SATA HDD at 123 MB/s (WD Green Desktop Hard Drive), or 4TB SAS HDD at 171 MB/s (WD Black Desktop Hard Drive)). It turns out this is optimistic—we benchmarked the acquisition rate of the ewfacquire tool (Metz) at 74 MB/s using an *in-RAM* 40 GB target, *in-RAM* output file, on a 2.6 GHz AMD processor. This turns the mere acquisition of a 3TB SATA HDD into an 11+ hour affair. The clear performance bottleneck here is the fact that the output is in the compressed *ewf* (*Expert Witness Format*) format and that the tool uses only a single CPU core. We also benchmarked the ewfexport tool, which reads and decompressed the acquired target at 150 MB/s (with in-RAM input and output).

Some of the subsequent stages, discussed later, employ even slower processing such indexing and carving. *The Sleuthkit Framework* (TSK, sleuthkit.org), the de facto reference open-source forensic architecture, defines further processing as being organized in the form of pipelines (Fig. 1).

Clearly, in this type of architecture there is no concern for *latency* (the time between the initiation of a computation and its completion) and we can neither express, nor hope to achieve, any specific processing deadlines. For that reason, the only way to achieve the necessary performance is to recast forensic processing as a real-time task and completely overhaul the architectural model of forensic processing.

Specifically, we submit that the ultimate performance objectives should be expressed as follows:

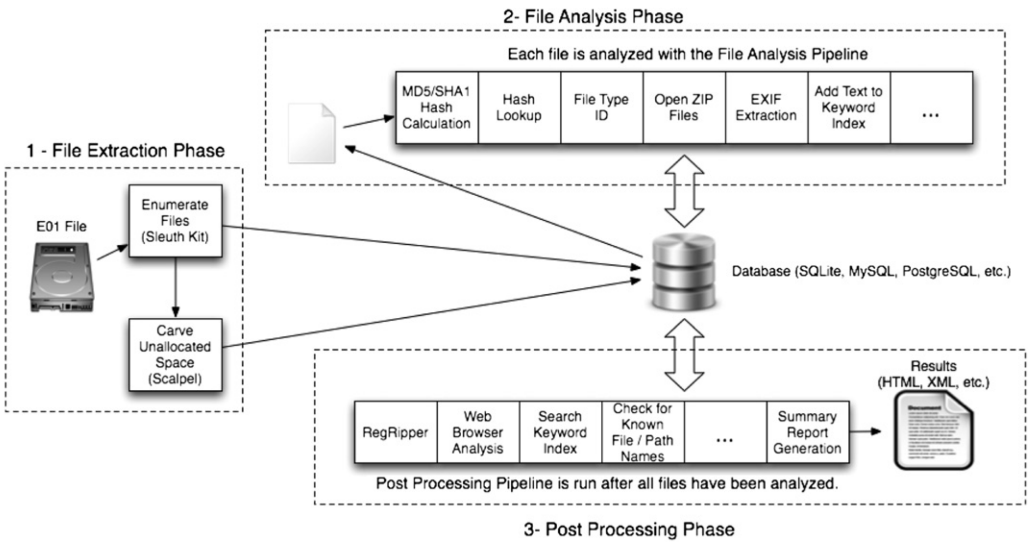
2- File Analysis Phase
Each file is analyzed with the File Analysis Pipeline
MD5/SHA1 Hash Calculation
Hash Lookup
File Type ID
Open ZIP Files
EXIF Extraction
Add Text to Keyword Index
...
1 - File Extraction Phase
E01 File
Enumerate Files (Sleuth Kit)
Carve Unallocated Space (Scalpel)
Database (SQLite, MySQL, PostgreSQL, etc.)
Results (HTML, XML, etc.)
RegRipper
Web Browser Analysis
Search Keyword Index
Check for Known File / Path Names
...
Summary Report Generation
Post Processing Pipeline is run after all files have been analyzed.
3- Post Processing Phase

Fig. 1. Phases of TSK filesystem analysis. http://www.sleuthkit.org/sleuthkit/framework.php.

a. Target acquisition and forensic processing should be performed in parallel at the maximum throughput rate afforded by the target;
b. Acquisition and processing should start and complete at (approximately) the same time;
c. Partial results should be made available as soon as they are produced.

Alternatively, we could say that the processing rate should equal the maximum target acquisition rate. Thus, for a modern SATA drive, the reference processing rate is about 120 MB/s. Note that this requirement includes *all* types of processing that we anticipate for the target. The last requirement on the list above effectively eliminates processing as a bottleneck in the investigative process, and allows analysts to begin productive work almost instantly. This is consistent with the general notion of real-time processing as perceived by users.

The given performance objective is provably optimal—it is not possible to complete processing of *all* the data before *all* the data is actually read, so finishing at the same time is the best result we can hope for. Also, once the processing is done, we (the developers) are out of the picture as a performance bottleneck, and it is up to investigators to time-optimize their analysis.

Later, we will discuss the question of *how* to achieve our performance objective. Before that, let us use this performance-centric point of view to take a fresh look at how to define and satisfy the requirements of forensic triage.

2. What is triage?

Over the last several years, the concept of triage has entered the digital forensic vocabulary and practice. Although its specific interpretation varies, it generally refers to a fast, initial screen of (potential) investigative targets in order to estimate their evidentiary value.

Triage is generally perceived as a separate, almost throwaway, effort that is not formally connected to the main forensic investigation. We see this entirely as the result of the performance inadequacies of standard forensic methods, which forces practitioners to hack together separate triage tools, such as Carvey's *Forensic Scanner*.[1]

Many practitioners in law enforcement refer to this initial investigative step as triage, *only* if it happens outside the lab and before admitting the media as evidence (Parsonage). In our view, such procedural distinctions are largely inconsequential from a software engineering perspective. Instead, building on the discussion in the previous section, we define triage as an optimization problem:

Digital forensic triage is a partial forensic examination conducted under (significant) time and resource constraints.

In other words, given a certain amount of time (e.g., 60 min), computational resources (CPU/RAM—8 cores, 16 GB RAM) and I/O resources (120 MB/s HDD throughput), there is a finite amount of work that can be performed on the target. The goal of triage is to get to the richest and most relevant information content within these constraints, and present the investigator with the best available information to make decisions. This formulation covers all practical scenarios, regardless of whether the triage is performed in the field, or in the lab. Any case-specific legal restrictions can be modeled as placing yet more restrictions on the optimization task.

Triage is almost indistinguishable from early forensic investigative steps, and it closely follows what experienced analysts do with a target in the beginning. Given more time, triage naturally transitions into deeper forensics. In that

[1] http://code.google.com/p/forensicscanner/.

sense, attempting to formulate and delineate triage as a completely separate process is neither necessary, nor useful.

From a technical perspective, it becomes clear that *latency*—the elapsed time between a query, and a response—is *the* primary performance requirement in triage. Since low latency is also an important design requirement for *any* forensic tool, there are no inherent trade-offs in optimizing *all* tools for latency. In other words, we can use the acute needs of triage as a reason to broadly rethink and improve digital forensic tool design.

Given the *low latency* (LL) requirement, a triage tool has several choices:

- *Employ existing LL methods*, such as examining filesystem metadata.
- *Develop new LL methods*, such as block forensics (Garfinkel et al., 2010) and similarity digests (Roussev et al., 2010).
- *Adapt high latency (HL) methods to a LL setup*, e.g., by sampling data and/or optimizing the implementation.
- *Turn HL into LL by applying parallel processing.* In a lab environment, it is becoming quite feasible to utilize tens/hundreds of CPUs; the vast majority of current tools are not ready for that. In the field, parallelization opportunities are likely to be less generous.
- *Use higher level knowledge*, whereby LL methods are combined with the occasional use of higher latency methods. In this setup, a tool could use an inexact LL tool to obtain a *hint* as to the relative information value of different objects (such as files) are worth examining with high latency methods.

By exploring in detail these different approaches, we can identify the most promising avenues for research and development. Fundamentally, total latency is the sum of (data) *access* latency—the time is takes to retrieve the input data—and *processing* latency—the time it takes to perform the computation. In the next section, we examine these concerns for different classes of forensic methods.

Although it would be nice, for the sake of completeness, to cover all forensic tools of note in the study, closed source tools present at least two critical methodological problems that have no clear remedies:

- We have no means by which to isolate and benchmark individual processing functions; thus, it is unclear what we would be measuring.
- By extension, we have no means to reliably measure the effects of parallelism in performance; indeed, most commercial tool have very limited concurrency capabilities.

There is also the practical concern of how to choose the tools that ought to be included in a representative sample. For all of the above reasons, we chose to break down the evaluation by function and use open, state-of-the-art implementations. Although it is not inconceivable that some commercial implementations of individual functions *might* hold a slight advantage, our overall experience does not point to that being user observable. It is also notable that many of the performance limitations stem directly from hardware constraints and filesystem design, so there is a hard limit on how fast some of the functions *could* be performed.

3. Rate classification of forensic methods

The study in this section is an attempt to quantify different methods with respect to their inherent latency costs, as well as identify methods that can be sped up. Since our main focus is triage, we are interested in what can be accomplished by working directly with the target media. Our discussion assumes that the analyst can afford about an hour worth of processing, and we are evaluating the applicability of various techniques in this context.

3.1. Choosing the target

Modern drives have split into two distinct categories—larger capacity hard disks (2–3TB), and smaller capacity solid state drives (128–512 GB)—that are very different with respect to price and performance. Table 1 summarizes these differences for two representative drives—the Western Digital Green 3TB and the Samsung 830 256 GB.

As the numbers clearly show, the real problem rests with large HDDs—they have larger capacities and notably weaker bandwidth and latency (IOPS) characteristics. For SSDs, we can comfortably read the entire target during the 1 h reference period. Therefore, we assume for most of our discussion that the target is an HDD.

However, we do not lose sight of the fact that SSDs have become affordable and are the new standard for laptops and mobile devices. We will return to this point later in our discussion and will show that the increased performance of these storage devices exposes to an even greater degree the performance deficiencies of forensic tools.

3.2. Filesystem metadata extraction

3.2.1. File attributes

Volume metadata (partitions) is a trivial amount of data, which can be accessed instantaneously. Filesystem metadata, specifically file attributes, can be obtained rather quickly, even for a large hard disk, as the following experiment shows:

Table 1
Price and performance comparison of HDD and SSD.

	HDD WD Green	SSD Samsung 830
Capacity (GB)	3000	512
Cost (dollar/GB)	0.04	0.74
Throughput (MB/s)	123	500
IOPS	135	80,000
Acquisition time (min)	407	17

Target:	2TB SATA drive, 7200 rpm, *etx4* filesystem, 1.8M files.
Test:	Obtain complete file listing from a freshly mounted drive (by executing the Linux *find* command).
Result:	Total execution time is 36 s. Using the Linux *blktrace* utility we monitored the device at the block level to understand the block-level access pattern. The observed workload consisted of ∼47,000 I/O read operations for a total of 188 MB of data; average rate was 5.2 MB/s.

The latter number shows that the access pattern is clearly “inconvenient” for the mechanical drive, which, based on our benchmark tests, is capable of 104 MB/s of sequential throughput.

In a separate experiment, we created 500,000 empty files on each of two 10 GB partitions, formatted as *etx4* and *ntfs*, respectively. Mounting and reading the file attributes took 1.2 s on *etx4* and 4.8 s on *ntfs*. The difference can be explained by examining the workloads: on *etx4* it was ∼5000 reads (20 MB), whereas *ntfs* generated ∼22,000 reads (88 MB). These numbers are supported by the fact that, overall, *ntfs* uses more storage for the filesystem metadata than *ext4*: 170MB with *etx4* vs. 620 MB with *ntfs*.

Despite the differences across different filesystems, the amount of data accessed is small enough that, even for large targets with millions of files, it is practical to obtain the file attributes for *all* files on the system during triage.

3.2.2. MS Windows registry

The Windows registry contains a significant amount of system and user behavior metadata, and often retains trace data well beyond its intended duration. As such, it is a prime candidate for inclusion in any triage analysis. We use *Registry Decoder* (Marziale and Case) as the reference tool in our performance benchmarks; our reference target system is a Windows 7 desktop, in use for a year. In acquiring/analyzing the registry data, we have the choice of working with the current state only, or also consider prior snapshots maintained by the OS; we start with the former.

Acquiring the registry hives takes 2 min and results in *seven* files (100 MB total) containing ∼280,000 keys. Preprocessing (parsing) the hives takes another 6 min on a 2.9 GHz CPU; running all the available plugin components (which produce targeted analytical reports) requires another 2 min. Thus, the total time for processing of the *current state* is about 10 min.

Including the prior state of the registry requires the processing of an additional seven snapshots, which brings the total to 63 files and 770 MB. Since processing is linear, it would take over an hour to parse all files, and run all plugins. This number suggests that, with its current implementation, registry processing must be applied selectively. The alternative point of view is that registry tools need to be sped up, with parallelization as the obvious first step.

3.3. File metadata extraction

Most file types contain metadata information—such as authors, keywords, and tools—as part of the file format and those can be used to perform high-level filtering of the data without having to process the content of the file. For example, EXIF data is frequently embedded in JPEGs; author and keyword information is used routinely in office documents. File metadata is a small fraction of the overall size of the file, so we can expect that it is significantly cheaper to extract and process. Further, most of it is located in the file header, so we would expect expedient access.

To verify these conjectures, we ran the *ExifTool* (Harvey) in a variety of scenarios. First, against a hard drive:

Target:	2TB SATA drive, 7200 rpm, *etx4* filesystem.
Test:	Obtain all metadata for 20,000 files from the GovDocs corpus[2] (Garfinkel et al., 2009) (10.6 GB) from a just-mounted HDD, using *exiftool* 8.6.
Results:	Total execution time: 5m33s Total data read: 757 MB Total block read operations: ∼47,000 Average (per file) data read: 38 KB/file Effective processing rate (files): 60 files/s Effective processing rate (data): 2.27 MB/s

This is a somewhat unexpected result in that CPU processing *is* a limiting factor. Rerunning the experiment with *four* CPU cores cuts the execution time in half to 2m38s, bringing the processing rate up to 4.8 MB/s—comparable to the rate for file attribute extraction. Further increase in parallelism does not yield benefits as the task becomes I/O-constrained.

The above results suggest that file metadata extraction needs to be applied more selectively in a triage scenario, as we can only hope to process about 200,000 files/h.

3.4. File content extraction

File content extraction simply means reading the full content of a file. On a mechanical drive, this creates some challenges as the *operating system* (OS) alternates between reading from disk blocks containing filesystem metadata and blocks containing actual file content. This translates into non-sequential access pattern at the block level, which forces expensive *seek* operations. In the extreme, when the pattern is completely scattered, the throughput of the drive becomes limited by the IOPS rate, which is around 100 IOPS for the average SATA drive. Therefore, the OS makes a significant effort to ensure that most data is laid out sequentially, and if the workload consists of long sequential reads, throughput would approach the maximum sustained rate for the hardware (104 MB/s for our reference HDD).

Given the above observations, we run our scenario several times targeting different files for extraction. Namely, we split the files into several groups based on size—0–4 KiB, 4–16 KiB, 16–64 KiB, 64 KiB–1 MiB, and 1 MiB–128 MiB—and extract 10,000 files from each group. (Before every run, we remount the drive to clear the cache, and warm up the metadata cache by reading the file attributes for all files.) Table 2 summarizes the results:

The numbers clearly bring home the point that, under the constraints of triage, file extraction ought to be planned

[2] http://digitalcorpora.org/corp/nps/files/govdocs1/.

Table 2
Execution time and throughput for 10,000 files in five categories (HDD).

	Size total (MB)	Avg file (KB)	Time (s)	Rate (MB/s)	Rate (files/s)
0–4K	41	4	79	0.52	127
4–16K	121	12	112	1.08	89
16–64K	398	40	123	3.24	81
64K–1M	3587	359	176	20.38	57
1–128M	26,772	2677	450	59.49	22

carefully as data throughput for small files on HDDs is abysmal. Nonetheless, it is still quite feasible to obtain 10–20,000 small files within 2–3 min. (It is worth noting that our reference HDD represents a best case scenario as it was created in one shot, and the OS has had the opportunity to organize the data sequentially with no subsequent delete/create cycles.)

3.5. *Latency-optimized target acquisition (LOTA)*

The observations in the previous section should make it clear that optimizing data extraction on large HDDs needs a more systematic approach to maximize the amount of data extracted from the target. We built a prototype system that implements *latency-optimized target acquisition* (LOTA), which reconciles the need to read data sequentially from the target with the need to process it in the form of files.

The rationale of the system is simple—before we start cloning a target, we parse its filesystem metadata to build an inverse map of data blocks to files. After that, we start reading the disk blocks sequentially from beginning to end, and use the map to reconstruct on the fly the files whose contents have been acquired. Once a file is complete, it is made available through the regular filesystem interface to file processing tools. Disk cloning proceeds in parallel.

The filesystem *parser* reads and analyzes the filesystem metadata of the target image/device and returns a list of file entries; each contains the full-path of the file, its *inode* (number), its parent inode, its mac times, the actual size of the file on-disk, the list of clusters used by the file (in order of usage) and finally its resident data (in case the file is small enough to fit within a single entry).

The resulting map is read by the (master) *server* component and distributed evenly to the pool of (agent) *client* processes, which may be local, or remote. Once the target processing begins, all blocks are routed by the server to the responsible client for on-the-fly file reconstruction. Completed files are written out to the local filesystem and the available file handlers are notified; a handler can be any piece of software installed on the system. Fig. 2 illustrates the LOTA architecture.

A couple of features are worth pointing out—there is no hard limit on the number of clients and handlers so the system can be scaled up to the necessary degree to keep up with acquisition rates (alternatively, the system will automatically throttle back to a sustainable throughput rate). Since blocks are sent one at a time, there is no network flooding effect and, since 1Gbit Ethernet approximately matches the 120 MB/s HDD throughput, the system does not require any special hardware.

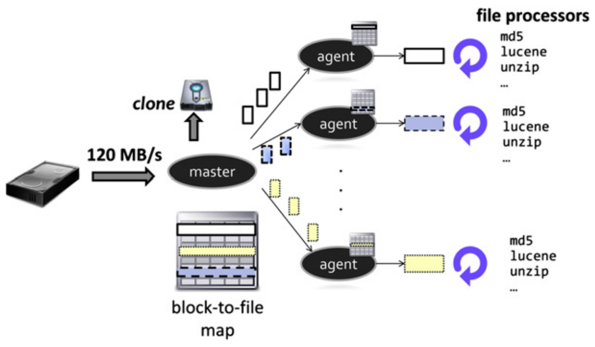

Fig. 2. Latency-optimized target acquisition architecture.

The benchmarked results match our expectations—we can perform I/O-bound operations at line speed. Specifically, we used a 300 MB/s RAID target to understand the performance of the system. With a single client, file extraction can be performed at 100–110 MB/s, with two (and more) it reaches 160–180 MB/s and becomes I/O-bound as the RAID is also used for writing out files. Unsurprisingly, adding crypto-hashing as a file handler does not affect the observed throughput rate.

Overall, LOTA can increase the average file extraction rate (by volume) 3–4 times: from 25 to 30 MB/s to 100–120 MB/s. Over an hour, this means an increase from ∼100 GB of data to ∼400 GB. The benefits to smaller files are substantially larger, which opens up a new avenue of research—optimizing the read sequence to maximize the number of files recovered by the deadline.

3.6. *File content processing*

3.6.1. *Crypto-hashing*

Cryptographic hashing, typically of the MD5 and/or SHA1 variety, is routinely employed during processing. These are fast computations with bulk rates (on a 2.9 GHz Intel Xeon X5670 single CPU core) of 475 MB/s and 340 MB/s for MD5 and SHA1, respectively. Block hashing in increments of 4KiB is only marginally slower at 415 MB/s and 295 MB/s, respectively.

In a triage scenario, crypto-hashing is I/O-bound and should be applied on all data read from a target. Lookup in a properly-organized in-RAM database—based on a hash table, or Bloom filters—is practically free. For example, a 4 GiB Bloom filter could represent 4TiB of reference data at 4KiB blocks with a false positive rate of 1 in 10,000; lookup requires no more than five memory accesses.

3.6.2. *Similarity hashing*

Similarity hashing using similarity digests (Roussev et al., 2010) is a technique that allows the correlation of objects (such as blocks, files, and network packets) that share common data. The commonality is detected at the bitstream level and does not require parsing or understanding of the data being compared. There are two basic scenarios in which similarity hashing is useful: a) identifying the presence of a small piece of data (block, file) inside a bigger container (file, volume); and b) identifying *versions* of an object (file) that are comparable in size. The algorithm is robust with respect to the alignment and distribution of the common data. In practice, this means that we can find, for example, all files (pdf, doc, docx, zip, etc) that include a particular (jpeg/png) image, such as a corporate logo. Similarly, we can find versions of known executable files and libraries.

Earlier work on *fuzzy hashing* by Kornblum (2006) produced fixed size signatures, which are applicable in a relatively narrow range of scenarios and are much less reliable than similarity digests (Roussev, 2011), which produce signatures proportional to the size of the target ($\sim$1.5–2.5% of the original).

The reference similarity digest implementation consists of a core library—*libsdbf*—with multiple language bindings, a self-contained command line tool—*sdhash*—for all major platforms (Window/Linux/MacOS), a service implementation—*sdhash-srv*—with a language-neutral protocol, and a web-based client. All tools are designed for high throughput by taking advantage of multi-core parallelism.

Detailed description of *sdhash* capabilities is beyond the scope of this article; instead, we highlight some benchmark results relevant to triage. Specifically, we focus on two scenarios—streaming identification of content, and differential analysis of multiple targets.

In the streaming scenario, we want to read a target HDD at its sustainable throughput rate of 100–120 MB/s and, for each data block (4/16 KiB), query a reference database of known files. The question we ask is—given the target rate, what it the maximum size of the reference database that can be queried online?

Using *sdhash 3.0* (sdhash.org) and a 48-core server, the answer is $\sim$15 GB. Specifically, we ran a 14 GB disk image from the M57 set[3] against a 10 GB reference set in 92 s, corresponding to a processing rate of 152 MB/s. This is equivalent to processing at 100 MB/s against a reference set of 15 GB.

In Roussev and Quates (2012), we showed that *sdhash* can be used to perform large-scale automated differential analysis, by correlating multiple hard disk images and RAM snapshots. We applied it to a data set that is 1.5TB in size, and consists of 78 disk images and 84 RAM snapshot, and showed that we can sketch out the solution of the three main scenarios—contraband, eavesdropping, and espionage—in approximately 2 h (on a 24-core box) beyond the time to clone and hash (in parallel) the targets.

This is a somewhat different triage/early investigation scenario from the one we considered so far but has the same urgency and requires that results be produced in minutes. Given more processing power, in a lab setting, it is entirely possible to produce results sooner, *during* the cloning/hashing process.

3.6.3. *Indexing*

(*Search engine*) *indexing* is the process of parsing, analyzing, and storing of data to enable fast and accurate *information retrieval* (IR). This is a commonly employed preprocessing step during forensic investigation, and one not known for speed and efficiency. This is a technique that is borrowed from the IR field, and its *primary* performance requirement is the speed and efficiency of retrieval, *not* indexing.

Table 3
Performance of a single Solr instance with 12 threads.

File type	File count	Avg file size (KB)	Total size (MB)	Time (s)	Rate (MB/s)	Rate (files/s)
txt	25,378	800	19,826	1294	15.32	20
html	47,499	62	2898	197	14.71	241
rtf	229	174	39	3	13.00	76
pdf	46,957	613	28,091	971	28.93	48
doc	14,776	598	8624	277	31.13	53
xls	7607	914	6790	213	31.88	36
Total/avg	142,446	465	66,268	2955	22.43	48

To quantify typical indexing performance, we used the *Apache Solr* search platform[4] and the SolrJ client[5] to get representative statistics. We used a 66 GB sample of documents from the NPS Govdocs corpus; the results are presented in Table 3.

We established experimentally that the *Solr* instance's throughput tops out using 12 threads, although all processing is done on in-RAM targets and 48 cores are available. The numbers represent end-to-end performance, which includes parsing the target documents.

We can see two performance clusters around 15 MB/s and 30 MB/s, with more complex files (*pdf/doc/xls*) offering—somewhat counter-intuitively—the higher rate. The 15 MB/s number should be considered most relevant in that it shows how fast *text* gets processed. The higher processing rate for *pdf/doc* files is, to some degree, misleading as these often contain images, which are skipped over by the indexing process. Finally, *xls* files contain relatively little textual data, so their performance is atypical as well.

3.6.4. *Decompression*

Although not a forensic function, per se, decompression often stands on the critical path of file processing as a large number of formats are compressed. Unpacking of compressed archives creates potential new (and larger) data sources. The vast majority of lossless compression is based on the zlib/deflate/gzip standards (RFC 1950–52) so understanding their performance is a good proxy for overall decompression performance.

We use 1 GB in-memory input and output targets to estimate the decompression rate using the Linux *gzip*, and *gunzip*, which both yield an average decompression rate of 28–30 MB/s on a 2.6 GHz processor.

3.7. *Block-level processing*

Block-level techniques completely ignore the filesystem metadata and treat the target merely as a sequence of blocks. That has the advantage of requiring only a sequential pass over the target media, thereby ensuring maximum throughput. As well, they naturally include all data present, including remnant data.

[3] http://digitalcorpora.org/corpora/scenarios/m57-patents-scenario.

[4] http://lucene.apache.org/solr.

[5] http://wiki.apache.org/solr/Solrj.

Bulk Extractor[6] is a forensic tool which scans the raw disk images, or any data dump, for useful patterns (emails, URLs, IP addresses, etc). It uses precompiled scanners to speed up processing, relative to *grep*-like tools, and heuristics to reduce false positives and noise. It is designed to take advantage of available multi-core capabilities and aims to unearth useful clues early in the forensic process.

For the benchmarking purposes, we used one of the M57 disk images, and worked through the available parameters to optimize the processing rate.

Target:	10.24 GB *NTFS* volume, cached in RAM	
Test:	Bulk feature extraction with bulk_extractor v1.3; tested with 8/48 CPU cores @2.6 GHz	
Results:	Execution time	8 cores: 661 s, 15.5 MB/s;
		48 cores: 172 s, 60 MB/s.

It is clear that this is a somewhat expensive procedure and a *dedicated* workstation can achieve only 10–15% of the available I/O throughput rate. In an actual triage where CPU capacity has to be shared, the tool will fall further behind. On the server, it becomes feasible to operate at 50% of the throughput, which is getting closer to what we need.

Data carving is another tool routinely employed in forensics to reconstruct logical objects, primarily files, from the block view of the storage device. Our initial intent was to include such processing in the survey; however, despite efforts by DFRWS,[7,8] and NIST[9] to provide test cases, we found it difficult to construct a plausible average case. The problem is that there is too much variability and performance depends not only on the data but also on the specific tool used *and* the specific execution parameters.

In our view, it is precisely this unpredictability that is a strong argument against routinely deploying carving during triage. Due to high false positive rates, carving tends to generate a lot more data, which is the last thing we need during triage. It is time to rethink why we use carving and whether we could achieve its most important benefits by other means.

4. Putting it all together

The purpose of the discussion so far has been to attach some performance numbers to some typical early investigation/triage tasks. This section is an effort to understand what is possible today, where the bottlenecks are, and what are the most promising solutions. We consider a 1-h triage on an 8-core workstation (field triage) and on a 48-core server (lab triage) of a 2TB HDD.

4.1. *File attribute extraction (2 min)*

This should be the first task of any triage tool and can be completed within the first 2 min even for drives with large number of files. The extracted metadata is the only viable starting point for more intelligent triage.

4.2. *(Current) registry extraction/parsing (10 min)*

The extraction of the current registry takes only 2 min; the actual processing takes another 8 min but can be done in the background while other I/O proceeds. Since the parsing of the data is readily parallelizable, we can hope to slim the processing down to 2 min on 8 cores. Processing prior versions of the registry can multiply that number by a factor 6–7, bringing (parallel) registry processing closer to 15–20 min. This is starting to look expensive on a 60 min budget so either delaying this step, or perhaps processing the earliest and current versions could be a better default.

4.3. *File content extraction (60 min, background)*

The optimal solution to file content extraction is to dedicate at least one core to systematically retrieving and reconstructing files. Our LOTA prototype shows that we can achieve close to optimal throughput. An even better solution would be to make sure that important files are retrieved first.

4.4. *Hashing*

This is the only processing that can keep up with I/O—a core should be dedicated to hashing and lookup. Similarity digests could only be used in the field in a selective manner, e.g., using a reference database of up to 1 GB. In the lab, much more extensive use is possible, especially when multiple targets are concerned.

4.5. *Indexing*

Indexing is essentially out of reach for the workstation case—it is too computationally taxing, even if we dedicate all the cores to it. In the lab, it would take approximately the capacity of an entire server with 48/64 cores in RAM-optimized configuration to keep up with line speed.

4.6. *Decompression*

Decompression is one of the silent performance stumbling blocks in forensic analysis—a lot of common file formats are compressed and the common compression methods used can only yield 25–30 MB/s of decompression per core, under the best of circumstances. On an 8-core budget, that is not easily affordable.

As we survey the results from the various tools, it becomes clear that we definitely need a systematic, highly automated approach to triage. Simply putting together a collection of tools, such as TAPEWORM[10] and SIFT,[11] and handing them to an investigator to use in an ad-hoc manner is woefully inefficient. Parallel processing is critical to any effort to maximize the information extracted and

[6] https://github.com/simsong/bulk_extractor.
[7] http://dfrws.org/2006/challenge/.
[8] http://dfrws.org/2007/challenge/.
[9] http://www.cfreds.nist.gov/.
[10] http://feedthetapeworm.com/.
[11] http://computer-forensics.sans.org/community/downloads.

processed from a target, and a human in the loop is not equipped to deal with that. This is *not* suggesting that analysts should be turned into spectators—on the contrary, they *must* steer the overall direction of the process and their experience can greatly speed things up. At the same time, we have to take them out of the task of controlling and optimizing the low-level processing details; this should be done by a run-time engine that understands tool performance and how to maximize it.

For 2–3TB HDDs, we can only hope to access about 10–15% of the data within an hour. To even achieve that, we need tools that are disciplined about using almost exclusively sequential access. Our prototype system for latency-optimized target acquisition shows that it is, indeed, possible to achieve much better throughput and faster processing.

For current SSDs—with access rates already passing 500 MB/s and 80k IOPS—the good news is that it is entirely feasible to access all of the data on a target within 20–40 min for 256–512 GB drive. The bad news is that almost *none* of our basic forensic methods, with the exception of crypto-hashing, are in a position to keep up. In that sense, the situation actually looks worse than the HDD case since we no longer have the fig leaf of slow I/O.

4.7. Allocating the CPU cores: workstation

Based on the observed numbers, an optimized 1-hour triage allocation could look as follows:

• File attribute extraction + registry processing	1 Core
• File content extraction	1 Core
• Crypto-hashing and lookup	1 Core
• File metadata extraction	1 Core
• Decompression	1 Core
• Bulk processing and/or similarity hashing	3 Cores

In the above schedule, the top three processing tasks are essential and can largely keep up with sustainable I/O throughput (we are assuming LOTA). The distribution of CPU resources across the next two cores—metadata extraction and decompression—is somewhat unpredictable as it largely depends on the content of the target, so two cores is the minimum allocation. The last three cores are dedicated to selective processing with stream-oriented tools that can work relatively fast on block-level data, yet, they will still need to be employed selectively to keep up.

4.8. Allocating the CPU cores: lab server

Now let us consider what is possible in a 48-core lab server to which we have attached our reference drive:

• File attributes + registry processing	4 Cores
• File content extraction	2 Cores
• Crypto-hashing and lookup	2 Cores
• File metadata extraction	2 Cores
• Decompression	2 Cores
• Bulk processing	12 Cores
• Similarity hashing	12 Cores
• Indexing	12 Cores

All the processes *above* the line we expect to keep up with line speed—we have increased the allocation to two cores per task to ensure that. For registry processing, we budgeted for highly parallel processing to extract important metadata quickly. The processes *below* the line can provide rates in the 15–30 MB/s range and will need to be applied more selectively.

On balance, the lab server can do quite a bit more, but even with 48 cores, the more sophisticated tools will not be able to run at line speed. We are assuming a RAM-rich configuration that will keep all processing in main memory—otherwise, I/O will destroy the performance of the system. Nevertheless, we have not verified experimentally that all of this processing can take place on one box at the predicted rates; it is not unreasonable to expect some drop off due to competition for shared resources.

5. Conclusions

In this work, we focused on understanding what type of processing is feasible during forensic triage. To that end, we bring several contributions to the field.

We argued that digital forensic processing ought to be treated as a soft real-time problem with performance (measured as processing rate) as a first-class concern. From a technical perspective, this leads to a demonstrably optimal solution and allows forensic turnaround times to remain constant (and adhere to real-world deadlines). Further, even if optimal solutions are not currently available, posting specific numerical objectives empowers users to measure progress and compare tools objectively.

We formulated triage as an optimization problem, specifically, as a time- and resource-constrained subset of a full forensic investigation. This is the first time that triage has been formalized in this generic fashion and allows for the solutions to be developed for *both* on-the-spot and lab-based triage scenarios.

We surveyed the performance of the most commonly used forensic methods as represented by their most common open-source implementations. We used a 2TB HDD as the reference target and measured their processing rates under 8-core "workstation" and 48-core "server" assumptions. From the results, it becomes clear that only a few basic methods—file/metadata extraction, crypto-hashing, registry extraction—can fit the computational/time budget in a workstation triage. More sophisticated methods—indexing, bulk processing, similarity hashing—become somewhat feasible on the server.

We introduced the notion and an experimental implementation of a *latency-optimized target acquisition* (LOTA) scheme, which enables *file* processing at maximum HDD throughput rates during initial target acquisition. This is an improvement of a factor of two for files of 1M+ and a factor of 100 for small files in the 4–64K range. The scheme is conceptually simple and should be employed routinely by forensic environments going forward.

Although the main focus here has been triage, by adopting a unified conceptual framework with deep forensics, we can draw some more general conclusions:

- The “classical” forensic hardware—the workstation, which we equate roughly to an 8-core box—simply does not offer enough processing power to even keep up with a SATA HDD. Although this is known in the field from experience, we have shown numerically that no amount of tweaking will help it; it is time to move on.
- The way forward clearly requires the *routine* utilization of *substantially* more computational power, RAM, and efficient cluster computation. We estimate that to achieve our goal of completing forensic *HDD* target preprocessing (as understood today) at the same time as cloning requires the resources of 2–4 48-/64-core servers. Note that this is to process to a *rate* of about 120 MB/s and would be the same for any size SATA drive.
- Processing of latest generation commodity SSDs presents both new opportunities and new challenges. High IOPS performance allows the flexible investigation unconstrained by drive latency considerations; the TRIM (King and Vidas, 2011) command may eliminate the need to look for deleted data; however, the high bandwidth of these drives (500 MB/s) exposes the fact that we have no readiness to process at these rates. This is an open challenge and we hope that researchers and developers embrace it.
- We need to bring *robust* data reductions techniques to bridge the gap between processing rates and data volume. Currently, this is done manually by the investigator in an ad-hoc manner. This is not sustainable as the soundness of the results is heavily dependent on the experience of the investigator. Instead, tools need to adopt sound statistical methods, as suggested by Garfinkel (Garfinkel et al., 2010), which would allow them to quantify the effects of data sampling on the final results.
- For triage in the field, we should revisit some early ideas for on-the-spot forensics (Gao et al., 2004), where investigators temporarily take over the local infrastructure (boot into a LiveCD forensic environment) and use it to speed up the triage. This is particularly interesting in a data center setup, where a service provider could loan (a rack of) servers in exchange for speedier and less disruptive search and seizure process.

References

Gao Y, Richard G, Roussev V. “Bluepipe: a scalable architecture for on-the-spot digital forensics.” International Journal of Digital Evidence Summer 2004;3(1).

Garfinkel S, Farrell P, Roussev V, Dinolt G. Bringing science to digital forensics with standardized forensic corpora. In: Proceedings of the digital forensic research conference (DFRWS); 2009. p. S2–11.

Garfinkel S, Roussev V, Nelson A, White D. Using purpose-built functions and block hashes to enable small block and sub-file forensics. In: Proceedings of the tenth annual DFRWS conference, Aug 2010, Portland, OR. p. S13–23.

Gibbons W. Legend of the boiling frog is just a legend, but does have environmental value. Athens Banner-Herald, http://onlineathens.com/stories/121202/hga_20021212022.shtml; Dec 12 2002.

Harrell C. Overall DF investigation process, http://journeyintoir.blogspot.com/2010/10/overall-df-investigation-process.html; Oct 2010.

Harvey P. ExifTool, http://owl.phy.queensu.ca/~phil/exiftool/.

Hibshi H, Vidas T, Cranor L. Usability of forensics tools: a user study. In: Sixth international conference on it security incident management and it forensics; 2011.

Kent K, Chevalier S, Grance T, Dang H. Guide to integrating forensic techniques into incident response. National Institute of Standards and Technology, http://csrc.nist.gov/publications/nistpubs/800-86/SP800-86.pdf; Aug 2006.

King C, Vidas T. Empirical analysis of solid state disk data retention when used with contemporary operating systems. In: Proceedings of the eleventh annual DFRWS conference, Aug 2011, New Orleans, LA. p. S111–7.

Kornblum J. Identifying almost identical files using context triggered piecewise hashing. In: Proceedings of the 6th annual DFRWS, Aug 2006, Lafayette, IN.

Marziale L, Case A. Registry Decoder, http://code.google.com/p/registrydecoder/.

Metz J. libewf, http://code.google.com/p/libewf/.

Parsonage H. Computer forensics case assessment and triage, http://computerforensics.parsonage.co.uk/triage/ComputerForensicsCaseAssessmentAndTriageDiscussionPaper.pdf.

Patterson D. Latency lags bandwidth. Communications of the ACM 2004;47(10).

Roussev V. An evaluation of forensics similarity hashes. In: Proceedings of the eleventh annual DFRWS conference, Aug 2011, New Orleans, LA. p. S34–41.

Roussev V. Data fingerprinting with similarity digests. In: Chow K, Shenoi S, editors. Research advances in digital forensics VI. Springer, ISBN 978-3-642-15505-5; 2010. p. 207–26.

Roussev V, Quates C. Content triage with similarity digests: the M57 case study. In: Proceedings of the eleventh annual DFRWS conference, Aug 2012, Washington, DC. p. S60–8.

RCFL. RCFL annual reports FY2003–FY2011. http://www.rcfl.gov/DSP_N_annualReport.cfm.

WD Black desktop hard drives, drive specification sheet. http://www.wdc.com/wdproducts/library/SpecSheet/ENG/2879-771434.pdf.

WD Green desktop hard drives, drive specification sheet. http://www.wdc.com/wdproducts/library/SpecSheet/ENG/2879-771438.pdf.