Comparing Open Source Search Engine Functionality, Efficiency and Effectiveness with Respect to Digital Forensic Search

Joachim Hansen, Kyle Porter*, Andrii Shalaginov, and Katrin Franke

NTNU Digital Forensics Group
Department of Information Security and Communication Technology
Faculty of Information Technology and Electrical Engineering
Norwegian University of Science and Technology
joachim90jh@gmail.com,{kyle.porter,andrii.shalaginov,katrin.franke}@ntnu.no

Abstract

Keyword search is one of the key components of the Cyber Crime Investigations. It has a direct influence on precision and relevance of the data found on seized data carriers. However, many of the digital forensics tools developers do not reveal the actual underlying algorithms or source code of their search engines. Therefore, there is a challenge to verify their accuracy and efficiency. On the other hand, open-source search engines are an alternative to using proprietary keyword search tools, where they have extensive functionality and perform well on large-scale datasets. The goal of this paper is to explore the applicability of such search engines in forensics search. The contribution of the paper is two-folded. First, a thorough literature review and comparison of the supported functionality documented by open-source search engines and open-source digital forensic tools was performed. In addition, a survey of existing publicly-available digital forensics datasets was conducted. Second, out of reviewed search engines, Solr and Elasticsearch were selected and compared by their functionality, efficiency in searching and indexing, and effectiveness of search results with respect to digital forensic search using relevant datasets. Our findings should assist those in the digital forensic community when choosing the appropriate open source search engines for keyword search in large-scale datasets.

1 Introduction

The current Big Data digital landscape has provided digital forensic investigations with massive amounts of structured and unstructured data to analyze and search for relevant evidence, and the quantity of data to analyze only continues to grow [32]. Such circumstances lead to important considerations such as how forensic practitioners should search through their collected data for investigations in a reasonable amount of time. Another challenge is how to best handle the storage requirements of the data. The use of relational databases to process the data has been already found to be inappropriate for digital investigations years ago. The reason for this is that the majority of the data retrieved is unstructured (for example, human written text) and, therefore, require other approaches that relational storage [31].

Information retrieval systems, such as search engines, are often used to help locate enterprise data where these enterprises oftentimes have to manage large volumes of heterogeneous data structures and formats [15]. For the digital forensic practitioner, the processing of the data must be reliable, forensically sound, and ideally the search engines and forensic tools supporting these ends should utilize algorithms with low memory and time complexities. On the other hand, we

*The author presented this paper at the NISK 2018 conference.

Proceedings of the 11th Norwegian Information Security Conference, Sep. 19-20, 2018.
NISK Journal, e-ISSN 1894-7735. https://ojs.bibsys.no/index.php/NISK

have to take into consideration so-called Forensic Search that may put specific restrictions of the keyword search process. These include lack of standardized approach to data preprocessing and formatting (for subsequent search needs), and the heterogeneous nature of the data (varied file types). In addition, the digital forensics practitioner may experience new data formats that also put demand on quality of the search results, which additionally have to follow Digital Forensics Process guidelines.

The problem with proprietary digital forensic tools is that developers do not reveal the underlying algorithms or source code. Therefore, we examine open-source search engines as an alternative method to using proprietary keyword search tools, where they have extensive functionality and perform well on large-scale datasets. There is a little chance to perform equivalent analysis of proprietary software due to license and source code availability limitations. In particular, the goal of this paper is to evaluate the performance of selected open source search engines and search functionality on forensic data.

Our first contribution is the comparison of the supported functionality documented by open-source search engines and open-source tools. The different search features of several popular open-source search engines and forensic tools are accounted for in an easy to read checklist. Additionally, we conducted a survey on existing publicly-available digital forensic datasets, six of which were used for testing. Our second contribution is an experimental comparison of *Solr* and *Elasticsearch*. This benchmark experiment tests how well they perform at indexing, searching and memory consumption during searching. Our results indicate *Elasticsearch* was generally better then *Solr* at index creation time, minimizing index size and response time for the first run of search terms. *Solr* outperformed *Elasticsearch* on second run of search terms. The difference between the search engines with respect to memory performance during searching was negligible. Information regarding which open-source search engines are available, their search features, and their searching and indexing capabilities is valuable for forensic practitioners when choosing to utilize an open source-search engine for performing keyword search in large-scale datasets.

The paper has the following structure. Section 2 presents the documented functionality of open-source search engines and existing publicly available forensic-related datasets. Experiment approach motivation, data selection and computing environment are given in the Section 3. Analysis of efficiency and effectiveness of the selected open-source search engines on given digital forensics data follow in the Section 4. Finally, Section 5 discusses implications of the study and concludes the paper.

2 Open Source Digital Forensic Tools, Search Engines, and Datasets

In this section we describe the results of a survey of the search functionality of open-source search engines and open-source forensics tools. In addition, an overview of the publicly-available digital forensics-related datasets is given.

2.1 Functionality of Open Source Digital Forensic Tools and Search Engines

Our choices of open source search engines to analyze was based on popularity of use (Google Trends) and mentions within the scientific literature. To narrow down our selection of open source forensic tools and search engines we selected them based on their degree of documentation and tool category. The inspection process was performed as a combination of targeted manual

inspection, keyword searching of technical documentation and source code comments of the software being inspected. By targeted manual inspection, we mean looking at portions of the documentation more likely to include relevant information. Some documentation pages might be very old or indicate that the documented feature is experimental, and as such were excluded from our review. One issue with the inspection process is that inferences often had to be made on out of context images and text that describes the search capabilities. Moreover, the description was often quite short. Ideally, the software capabilities would be confirmed by practical tests, but perhaps this can be done as future work. Ultimately, Sleuth Kit - Autopsy, Volatility, Mozilla InvestiGator, and Hachoir were the forensic tools selected based on their degree of documentation and catalogue of tools. Elasticsearch, Solr, and Sphinx were the search engines selected based on popularity. Table 1 shows a summary of the open source forensic tool and search engine functionality analysis. A checkmark indicates that the given program has the capability and an empty cell means that it does not. This Table is a result of extensive literature review aimed at understanding of all possible search functionalities existing in digital forensics tools.

While we do not define every capability of each forensic tool and search engine, we do go over the one we utilized for our experiments. *Full text search* is suitable for finding relevant documents in a large set of unstructured data [14, 16]. A document in full text search is considered a list of searchable terms (e.g. words and numbers) [16]. The terms are usually indexed in order to have faster subsequent searches.

2.2 Publicly available digital forensics-related databases

To support this study, an overview of the publicly-available datasets related to digital forensic was performed. In contrast to the overview of all possible Digital Forensics datasets performed in 2017 [9], this work focuses only on publicly-available, which means that anybody can fetch them and repeat the experiments. It is important to understand that there is no way of getting real-world data from crime investigation, however, there are plenty of datasets created by researchers for data analysis purposes. The systematic literature review included following the iterations:

1. Search digital libraries, scan scientific articles for names, direct links or sources related to the datasets below, and use this information on the Google search engine to identify individual datasets or repositories of datasets.
2. Document search phrases that resulted in identifying new datasets.
3. Repeat step 1 and 2 with other resources like github, Kaggle and figshare to locate more datasets.

Table 2 is a summary of review process to identify datasets candidates to be part of the experiment set in the next section. We identified 83 datasets that are publicly available and attributed to Digital Forensic. This number excludes biometric datasets such as images of fingerprints, hand signature, gait, voice recognition and iris. However, the review will include authorship attribution corpus. To our knowledge, there has not been performed any comprehensive enumeration of forensics-related datasets.

Table 1: Comparison of search capabilities and functionality
Source: [2, 6, 10, 19, 20, 23, 25, 27]

Capability	Sleuthkit	Volatility	Mozilla Invest-Gator	Hachoir	Elasticsearch	Solr	Sphinx
Regular expression	✓	✓	✓	✓	✓	✓	✓
Decide/Insensitive case	✓	✓	✓	✓	✓	✓	
Concurrent search	✓				✓		✓
Automate search, with respect to keywordlist	✓						
Import keywords	✓						
Export keywords	✓						
Periodical search	✓						
Substring matching	✓				✓	✓	✓
Export search results	✓				✓	✓	
Match highlighting	✓				✓	✓	✓
UTF-8 Encoding support	✓		✓	✓	?	✓	✓
UTF-16 Encoding support	✓			✓	?		
ISO-8859-1 Encoding support				✓	?		
Deduplication support	✓					✓	
Approximate hash based matching	✓						
Orphan/deleted file search	✓						
RAM search	✓	✓	✓				
Matching memory structures (pre-made)		✓					
Hash database lookups	✓						
Wildcard		✓			✓	✓	✓
Binary search	✓	✓					
HTML renderer for search results		✓					
Support for masking sensitive fields			✓				
Exact hash matching			✓				
System provided keyword suggestions						✓	
AND, OR, NOT, GROUP boolean operators					✓	✓	✓
+ boolean operator (term must exist)						✓	
File search filter			✓				
Retrieval of documents not matching filters			✓				
Set max search hits			✓		✓	✓	
Stripping senstive metadata			✓				
Increase search priority of important indexes					✓		
Terminate search after a given elapsed time					✓	✓	
Sorting search results					✓	✓	✓
Customized message/ post-search action					✓	✓	
Aggregated summary of search results					✓		
Narrow search results with post filter					✓	✓	
Set relevancy weight for field					✓	✓	
MoreLikeThisQuery					✓	✓	
Search result clustering						✓	✓
Minimum matching criteria						✓	
Fixed relevancy score						✓	
Field collapsing					✓	✓	
Support for TF-IDF					✓	✓	✓
Language detection on index time						✓	
Fuzzy matching					✓	✓	✓
Faceted search					✓	✓	
Phonetic search					✓		
Geospatial search					✓	✓	
Streamed search						✓	

Name	Catagory
BAC	Authorship
CTFAC	Authorship
PCSN	Authorship
TBGC	Authorship
Personae	Authorship
PAN-Enron	Authorship
PAN/CLEF12	Authorship
PAN13	Authorship
PAN14	Authorship
PAN15	Authorship
RCTAC	Authorship
MUD03	Authorship
netre-sec	Collection of misc
MTA13-17	Collection of misc
pcapr	Collection of misc
PCAPsDB	Collection of misc
CAIDA	Collection of misc
csmining	Collection of misc
AZSecure-data	Collection of misc
Digital Corpora	Collection of misc
DFCF review	Collection of misc

Name	Catagory
GIfiles	Email
USHCE	Email
419 fraud dataset	Email
MLE200	Email
Enrondata	Email
RAISE	Files
SherLock	Files
AndroZoo	Files
CTD15	Financial fraud
UCSD-FICO-09	Financial fraud
CMS	Financial fraud
PaySim	Financial fraud
BankSim	Financial fraud
MICC	Forgery
Brian Carrier	Forensic Images
RDC	Forensic Images
CFReDS	Forensic Images
VirusShare	Malware
BIG15	Malware

Name	Catagory
Drebin	Malware
DroidWare	Malware
MILCOM16	Malware
Kharon	Malware
Mudflow	Malware
ISOT2010	Malware
ECML/PKDD07	Malware
CSIC10	Malware
BlogPcap	Malware
Malwarerec	Malware
CTU-13	Malware
ISCX	Malware
ISCXAB	Malware
DAROA98/99	Network
DARPA2000	Network
MAWILab	Network
KDD Cup99	Network
UNSW-NB15	Network
NSA-CDX	Network
ADFA	Network

Name	Catagory
Kyoto data	Network
crawdad	Network
ICS-pcap	Network
Common Crawl	Network
NSL-KDD	Network
ISCXTNT	Network
ISCXVNV	Network
ISCXIDS	Network
YPFC	Password
VincentPassword	Password
MBT08	RAM
NUSSC	SMS
WebbSC11	SPAM
DITSSC	SPAM
TREC05-07	SPAM
Hewlett spam	SPAM
WEBSPAMUK07	SPAM
microblogPCU	SPAM
TREC11	SPAM
SPAM/HAM	SPAM
phishtank	Phising
millersmiles	Phising
PWDS15	Phising

Table 2: Dataset names and category

3 Experiment Methodology

The focus of practical evaluation in this study is to provide understanding of how well the open-source search engines perform on digital forensics-related data and whether their effectiveness can be considered acceptable for criminal investigations.

3.1 Experimental Approach

To measure the difference in effectiveness and efficiency for both search engines, the following experiments were conducted:

- Experiments with fulltext searching
- Experiments performed on a set of search engines (*Solr* and *Elasticsearch*).
- Set of keywords based on domain knowledge of datasets and a search for strings that are not present in the dataset
- Searching within an index (i.e. not searching across all indexes or multiple indexes at the same time).
- Search time
- Cache temperature
- Memory measurement during search
- Search Accuracy - count of clear cut misses
- Out of box configurations (default values)

There are two main limitations of the experiment. The first being that the experiments are performed on only one virtual machine. This environment does not allow testing for how well the search engines perform at distributed search. The second issue is that only the default configurations was tested (out-of-the-box setup) with *Solr* [11] and *Elasticsearch* [7].

3.2 Data selection

When working with and evaluating digital forensics tools it is difficult to obtain a real-world dataset due to sensitivity of the information, which might be related to criminal cases. Addition-

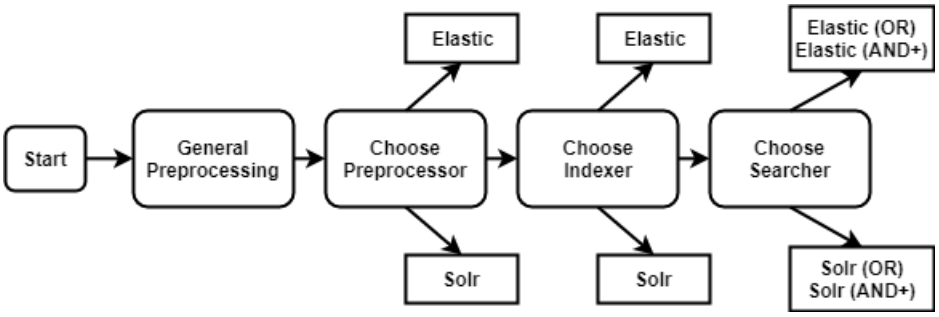
ally, digital forensics experts may experience a variety of data formats, even when considering primarily text datasets. One of the criteria was also to look for large-scale datasets to have a better performance testing, therefore, the datasets were selected based on dataset size and forensic category. The list is shown below.

1. **Fraud:** Enron email dataset [4].
2. **Network:** Snort IDS log file [26].
3. **Email:** Hillary Clinton emails [13].
4. **Malware:** VirusTotal and PE32 reports [21].
5. **Spam:** DITSSC [3].
6. **SMS:** NUSSC [5]

3.3 Dataset Pre-processing, Indexing and Search Queries

A short description of the experimental design including the preprocessing, indexing and searching steps in the experiment are given here. As shown in Figure 1, the first step is running the datasets though a general preprocessing step.

Figure 1: Flowchart experimental design



The data pre-processings step includes following sub-steps:

- Remove *JSON* structure, *XML* structure and/or dataset specific structure from all the datasets.
- Removes characters that cannot be processed by the *JSON* parser in *Solr* or *Elasticsearch*.
- Removes junk (e.g. terms that is not interesting, such as many repeated characters)
- Removes unnecessary whitespace
- Removes empty lines
- Separate dataset entries on their own lines.
- Split the dataset into **bulk files** of 1000 lines each (each line is considered as a document for importing).

Then after general pre-processing the dataset, bulk files have to go through another two pre-processing steps, one for *Elasticsearch* and one for *Solr* as shown in the Figure 2a and Figure 2b respectively.

A single content field entry is 1 *JSON* document to be indexed. We call the output from these steps bulk *Elastic* and bulk *Solr* respectively. The next 2 steps are *Elastic* and *Solr* specific bulk indexer. These indexers take the *Elastic* and *Solr* bulk files as input and indexes all the bulk files/the entire dataset. During these steps, the following measurements are taken:

```
{ "index":{} }
{"content":"bulk file line n" }
{ "index":{} }
{"content":"bulk file line n+1" }
```

(a) JSON format for ElasticSearch

```
[
{"content":"bulk file line n" }
{"content":"bulk file line n+1" }
]
```

(b) JSON format for Solr

- The *Linux command time* is used to capture total elapsed time of indexer.
- *QTime* in *Solr* and *took* in *Elasticsearch* are used as response time.
- A size command in *Elasticsearch* are used to get the size of the index. Additionally, the filesystem in our environment is inspected to find the index size in *Solr*.

The last two steps are searching with *Solr* and *Elasticsearch*. The Solr(OR)/Elastic(OR) and Solr(AND+)/Elastic(AND+) are similar. The (OR) search queries are using the Boolean OR operator between multiple search terms. The (AND+) search queries requires that all search terms in the search string are present in the matching string, and that the terms are in the right order for them being a match. At these steps, the following measurements are taken: Clear cut misses, Memory stats are captured with the *Linux top* command, *QTime* and *took* response time.

3.4 Computing Environment

To perform benchmarking of effectiveness and efficiency of the selected search engines, the following setup was used: Virtual Machine (6 cores, 40GB RAM and 2TB storage) with Ubuntu 16.04.3 LTS, *Openjdk 1.8.0_131*, *Elasticsearch 6.0.1* and *Solr 7.1.0* and *Solr Cloud*.

4 Results & Analysis

This section is devoted to quantitative comparison of the efficiency in indexing and searching of *Solr* and *Elasticsearch* with subsequent analysis of the obtained results.

4.1 Indexer performance

The benchmark of the indexer considers the change in size starting from the collection of documents to be indexed to the resulting index size, as well as the real time, response time and delta (difference between the times) when measuring index creation time. Real time (in milliseconds) is the total elapsed time by the indexer process measured by the Linux time command. Response time is the time it takes from the indexer getting the index request until completion of the indexer, given in milliseconds by QTime (*Solr*) and took (*Elasticsearch*). Delta is the the remaining time when subtracting the response time from real time (i.e. delta = real time - response time). We measure for delta since the response time and real time may not be the same. This is because the real time includes I/O overhead (request creation, context switching, writing to console, etc.). For each benchmark test we run the indexer twice on each data set, denoted first run and second run respectively. The Figure 3 shows the percentage size change from "to be indexed" data-set to "indexed data-set" for *Solr* and *Elasticsearch*.

Figure 4 shows a comparison of the response time for *Solr* and *Elasticsearch* indexer for each individual dataset. In Figure 6 shows the real elapsed time for indexing and how much time was spent on I/O for *Solr* and *Elasticsearch*.

Figure 3: The change in index size

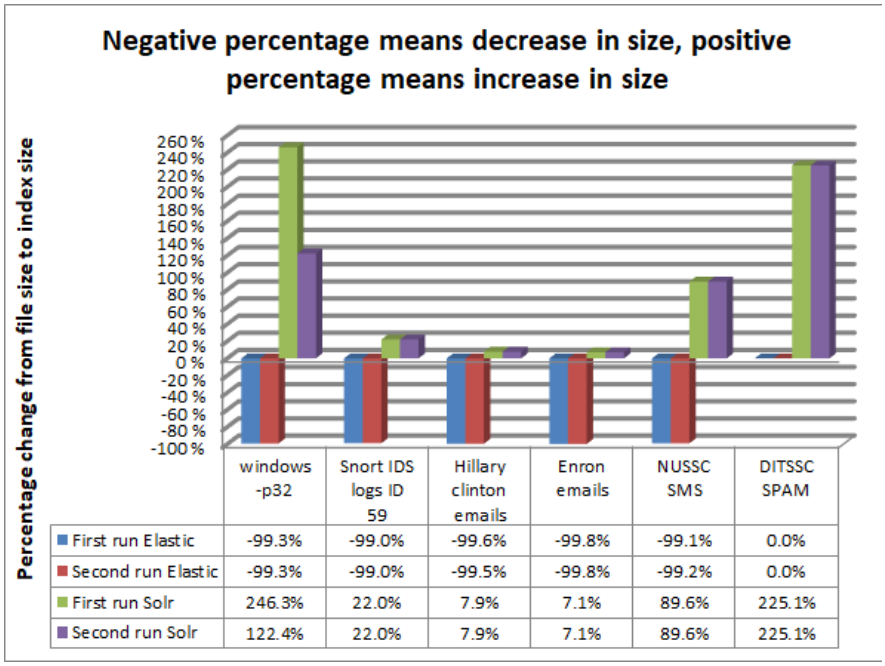
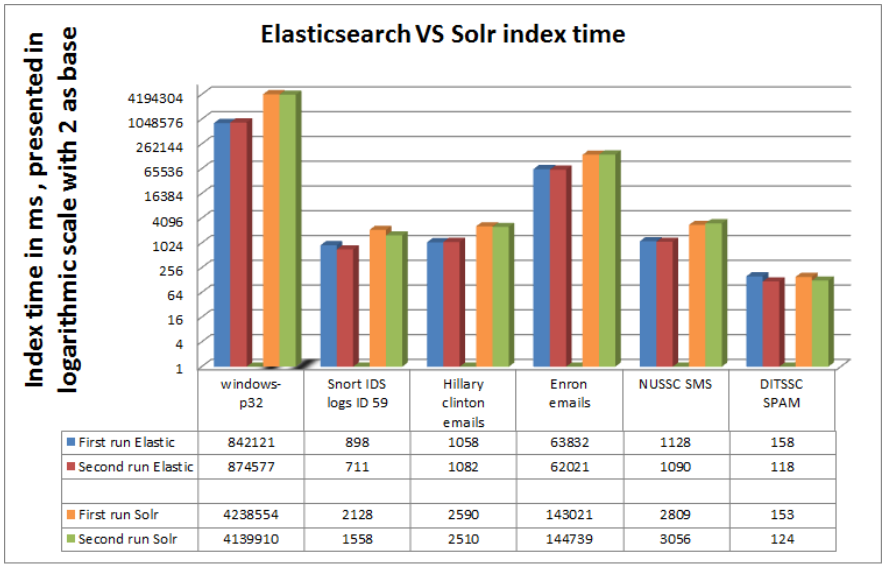


Figure 4: Index time Took and QTime



From our indexing experiments, we find that Elasticsearch is generally much faster than *Solr* at indexing (index response time) for both the first and second runs with the exception of the DITSSC SPAM dataset (as seen in Figure 4). Furthermore, we can see that the index response time is not static from the first run to the second. So there is no consistent reduction in response time from run 1 to run 2. From Figure 6 we see that the real time and response time are proportionally closer in *Solr* than in *Elasticsearch*. Therefore, more time is spent on delta in *Elasticsearch* than *Solr*.

Figure 5: One iteration of the indexer

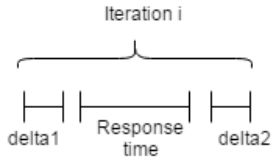
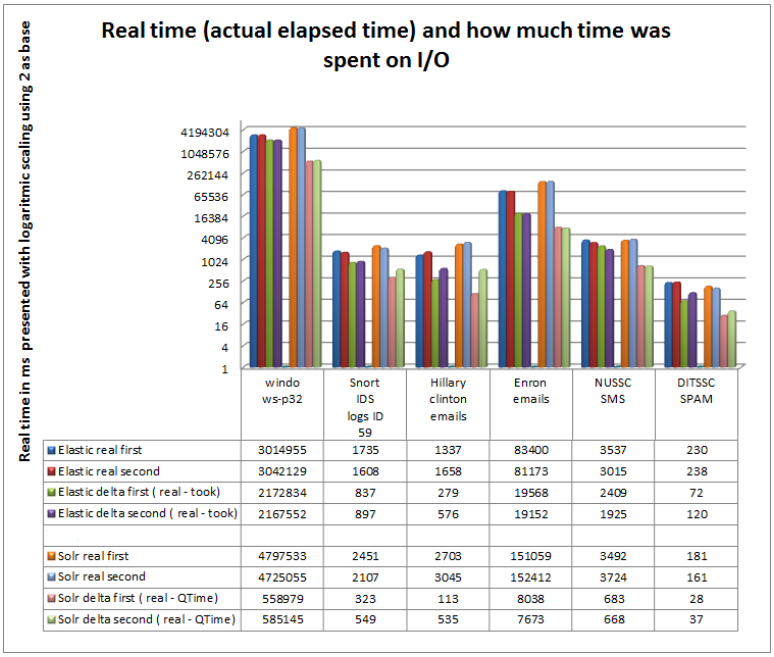


Figure 6: Index real time and delta



	windows-p32	Snort IDS logs ID 59	Hillary clinton emails	Enron emails	NUSSC SMS	DITSSC SPAM
Elastic real first	3014955	1735	1337	83400	3537	230
Elastic real second	3042129	1608	1658	81173	3015	238
Elastic delta first (real - took)	2172834	837	279	19568	2409	72
Elastic delta second (real - took)	2167552	897	576	19152	1925	120
Solr real first	4797533	2451	2703	151059	3492	181
Solr real second	4725055	2107	3045	152412	3724	161
Solr delta first (real - QTime)	558979	323	113	8038	683	28
Solr delta second (real - QTime)	585145	549	535	7673	668	37

Figure 5 shows a single iteration of the *Solr* or *Elasticsearch* indexer. The indexer would run i=(number of bulk files for dataset d) times. The indexer is still working but the actions is not part of the request handler. It is assumed that delta is composed of some actions that is done both before and after the actions of the request handler [1,8,30]. The second assumption is that the actions that can be attributed to delta and response time is similar for *Elasticsearch* and *Solr*, with some minor exceptions. Possible actions that could contribute to the delta time are reading the bulk Elasticsearch/Solr files to be indexed, transferring with curl, sending request to the request handler, writing to console, etc. Furthermore, the request handler is responsible for creating the index structure, creating ID automatically for the JSON document, commit

(*Solr* specific command), and writing index to disk.

The two main reasons were identified as to why *Elasticsearch* uses more time with Delta than *Solr*. The first is that the *JSON* bulk index file format in *Elasticsearch*, with the { "index":{} } lines, resulted in a "Elastic bulk index file" that was about twice as large as in the case of *Solr*. The second is that it has been observed that *Elasticsearch* writes far more information to the console than *Solr* during indexing. We consider the latter to be the highest contributing factor.

Elasticsearch outperformed *Solr* on index creation time as more emphasis was put on the request handler actions as opposed to I/O. One aspect that influences indexing creation time is the frequency of commits. The commit operation was performed at every iteration of the indexer script and may therefore be partially responsible for slowing down the overall indexing process. While the commit operations are needed in *Solr* in order to make the documents available for search, the operation is not free [17,30]. It was recommended to limit the frequency of commits to improve indexer performance with respect to index creation time. Another influencing factor is the document size and their line lengths. Large gaps in response time between *Solr* and *Elasticsearch* may be seen in Figure 4, but in particular the largest gaps occurred when there are many lines with low character counts (for example the PE32 dataset). So, if *Elasticsearch* is slightly faster at indexing smaller documents over *Solr* (where there are many of these smaller documents), then it can explain the gap. Given the big difference in size between the indexes in *Solr* and *Elasticsearch* (see Figure 3), the longer index creation time in *Solr* can partly be explained by having to write more to disk and more processing of the documents to create the index.

Solr cloud has a dependency on a server/program called *Apache ZooKeeper* and its database [24]. In our experiment where the *Zookeper* server and *Solr* was located on the same virtual machine, it might cause *Solr* and *Zookeper* to fight over I/O resources and therefore trigger a *Zookeper* timeout [24]. This could explain the gap in indexing performance.

There is a consideration to be made in *Solr*, if the segment count is set to a low number, then more merges will occur when indexing, that negatively effects indexing performance [22]. On the upside, queries will be faster as there are fewer places to search. If on the other hand the segment count is high, then we get the opposite situation with improved indexed speed and degraded search performance. A low segment count can be the reason why *Solr* seems to perform badly with indexing yet be good at searching.

4.2 Search performance

Search performance was evaluated with three different cases on both *cold* (first run) and *warm* search phrases (second run). The meaning of *"cold"* is the use of search phrase for the first time against a given data set, while *"warm"* infers that the query has been run before. The search cases for evaluation is described below.

1. Single term search (e.g. a word or sequence of symbols)
2. Multiple term search (for AND+ and OR queries)
3. Searching for something that is not present in the dataset (i.e. MD5 hash of the string "Joachim Hansen" = "02edbd94746bc69677e969a89c4eb0d8").

Figure 7 shows the aggregated search hits and Figure 8 presents the aggregated search time for the three search cases, where the timings and hits for the different dataset experiments were

aggregated with respect to the different search cases.

Figure 7: Aggregated search hits

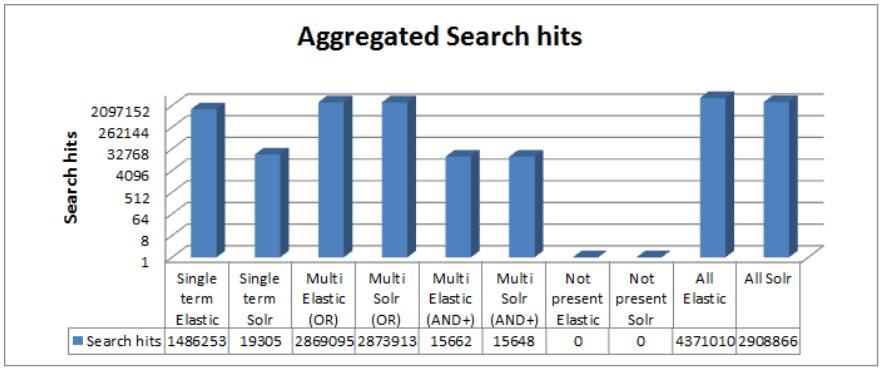
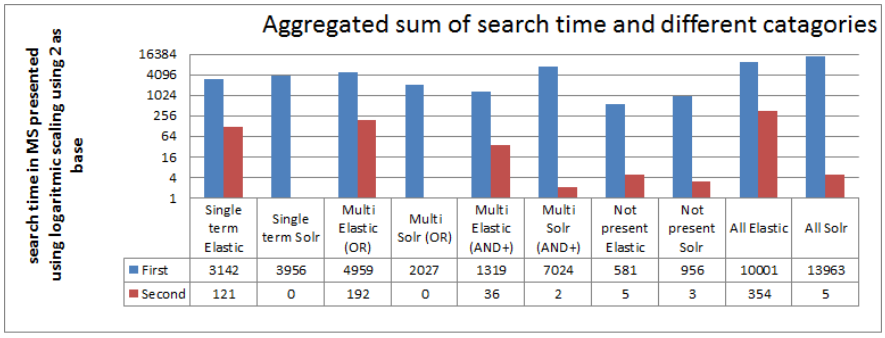


Figure 8: Search time aggregated



From Figure 7 we can see that for the majority of the search categories that there is only a small difference between the number of search hits between *Solr* and *Elasticsearch*. The exception to this finding is the single term search where *Elasticsearch* had 7,598% more search hits over *Solr*. The reasons for this was either that the search queries do not behave the same way, parsing index data was done differently in the search engines, or one search engine parses more information (more exhaustive search) than the other. It was also assumed that OR and AND operators did not affect single terms search. However, this assumption could be incorrect and can be the reason for the large difference between the number of search hits in *Solr* and *Elasticsearch* with respect to single term search. Interestingly, both *Solr* and *Elasticsearch* has 2 clear cut misses, meaning that no search hits were returned when there is at least one matching occurrence of the search string present in the dataset.

Figure 8 shows the aggregated sum of the search time for all datasets categorized by search case. Our experiments show that *Elasticsearch* is better than *Solr* for 3 out of the 4 search categories for the first run, but *Solr* proved to be much better than *Elasticsearch* for the second run. For instance, *Elasticsearch* has a 6,980% increase of search time as an aggregate for the second run over *Solr*. We believe more importance should be placed on the second run, as it is more like a real operating environment with a warm cache.

As mentioned before, a high segment count could negatively affect search performance and that

both *Solr* and *Elasticsearch* emphasize search performance over compression as their default behavior. If multiple shards are present on the same node/host/computer (not distributed) then search performance can be negatively affected according to [12]. This is the case with *Elasticsearch* where 5 shards are on the same host machine. *Elasticsearch* outperforms *Solr* in 4 out of 5 search categories but *Solr* performs better at the second run. We argue that the latter observation was because *Solr* is better (e.g. caching more items or caching the right set of items) at using the cached items from the previous search then *Elasticsearch* [29].

The memory statistics in Table 3 are captured just prior to initiating a search query, during first and second attempt of the same search and the capturing process is ended just seconds after the searches complete. There was in total 84 searches (44 for *Solr* and 44 for *Elasticsearch*) that contribute to the statistics in the table below.

Table 3: Memory stats Elasticsearch and Solr during search

Elasticsearch					**Solr**				
Virtual memory(VIRT): Size in GiB					Virtual memory(VIRT): Size in GiB				
Average	Max	Min	Delta	Mode	Average	Max	Min	Delta	Mode
42.831	42.831	42.831	0	42.831	29.144	29.144	29.144	0	29.144
Physical memory (not swappable) - RES:size in GiB					Physical memory (not swappable) - RES: size in GiB				
2.807	2,898	2.666	0.232	2.898	2.936	2.964	2.881	0.083	2.964
shared memory (SHR): size in GiB (rounded up)					shared memory (SHR): size in GiB				
0.34	0.43	0.2	0.23	0.43	2.296	2.323	2.241	0.082	2.323

While there are differences in memory performance, they are not significant. One question that had arisen is what was considered when the memory was measured. The code should have summed up all threads for the process under question. But what about forking short lived sub processes that are helping with a search task, and furthermore, what part of *Solr* are captured. *Solr* can be divided into *Solr*, *Solr Cloud* and *Zookeper*. So, which one of these was parts of the memory summary? *Solr* did perform better, but if not all of these processes were taken into account, then the memory performance might not favour *Solr* after all.

5 Discussions & Conclusion

In this work we performed a review of the functionality of popular and well documented open source forensic tools and search engines, conducted a literature review of publicly-available forensic datasets, and then performed a benchmarking experiments. These included memory and time consumption comparison of the indexing and fulltext searching processes of *Elasticsearch* and *Solr*.

Solr and *Elasticsearch* supported much of the same functionality, but there were some important distinctions. From one side *Solr* has support of deduplication [28], approximate hash based matching, keyword suggestions, and search results clustering [18]. On the other hand, *Elasticsearch* has support for capabilities such as phonetic search. All of these functions are important for digital forensic search, but *Solr* has more unique capabilities that would assist search in large-scale datasets.

The benchmarking experiments show that *Elasticsearch* index creation time was faster than *Solr* for 11 out of 12 trials of the indexer. The one exception was the first run of the smallest dataset. Furthermore, in *Elasticsearch* the resulting index size is reduced from the original dataset by around 99%, while we see an increase in index size in *Solr* ranging from a few percentage to up

to around 240%. Our experiments indicate that *Elasticsearch* is favorable over *Solr* regarding both index size and index creation time.

Elasticsearch was better than *Solr* at 3 out of 4 search test categories for the first run, but *Solr* was performed better than *Elasticsearch* for the second run. There was also the same number of clear cut misses for both search engines. The second run should be more like a real operating environment with a warm cache. With respect to memory consumption, there was a big difference between *Elasticsearch* and *Solr* usage of Virtual memory. *Elasticsearch* uses around 13 more GiB on Virtual memory than *Solr*.

Future work can improve our results in the following ways. A comparison on specific search algorithms should be performed on the specific indexing and search methods used by *Elasticsearch* and *Solr*. What would greatly assist our results are performing our experiments in a multi-virtual machine environment and spread the databases, shards and servers amongst the hosts. This environment would be suitable for testing how well the search engines perform at distributed search. The experiments could also test different import methods, importing using multiple threads and using different bulk sizes to test which approach minimizes index creation time.

References

[1] Wiki Apache. General, 2013. Last accessed (DD/MM/YYYY) 18/11/2017. URL: https://wiki.apache.org/solr/SolrTerminology.

[2] Autopsy. Autopsy - the sleuth kit. accessed: 09/10/2017. URL: http://www.sleuthkit.org/autopsy/.

[3] Tao Chen and Min-Yen Kan. Creating a live, public short message service corpus: the nus sms corpus. *Language Resources and Evaluation*, 47(2):299–335, Jun 2013. URL: https://doi.org/10.1007/s10579-012-9197-9, doi:10.1007/s10579-012-9197-9.

[4] William Cukierski. The enron email dataset - 500,000+ emails from 150 employees of the enron corporation, 2016. Last accessed (DD/MM/YYYY) 21/11/2017. URL: https://www.kaggle.com/wcukierski/enron-email-dataset.

[5] Sarah Jane Delany, Mark Buckley, and Derek Greene. Sms spam filtering: Methods and data. *Expert Systems with Applications*, 39(10):9899 – 9908, 2012. URL: http://www.sciencedirect.com/science/article/pii/S0957417412002977.

[6] Elastic. Elastic search. accessed 24.04.17. URL: https://www.elastic.co/.

[7] Elasticsearch. Install elasticsearch. Last accessed (DD/MM/YYYY) 4/12/2017. URL: https://www.elastic.co/guide/en/elasticsearch/reference/current/deb.html#deb-repo.

[8] Elasticsearch. The search api. Last accessed (DD/MM/YYYY) 18/11/2017. URL: https://www.elastic.co/guide/en/elasticsearch/reference/current/_the_search_api.html.

[9] Cinthya Grajeda, Frank Breitinger, and Ibrahim Baggili. Availability of datasets for digital forensics and what is missing. *Digital Investigation*, 22:S94 – S105, 2017.

[10] hachoir. hachoir-subfile program. Last accessed (DD/MM/YYYY) 12/10/2017. URL: http://hachoir3.readthedocs.io/subfile.html.

[11] howtoforge. How to install and configure solr 6 on ubuntu 16.04. Last accessed (DD/MM/YYYY) 4/12/2017. URL: https://www.howtoforge.com/tutorial/how-to-install-and-configure-solr-on-ubuntu-1604/.

[12] Ben Hundley. Small static dataset, 2-3 gb, 2015. Last accessed (DD/MM/YYYY) 7/12/2017. URL: https://qbox.io/blog/optimizing-elasticsearch-how-many-shards-per-index.

[13] Kaggle. Hillary clinton's emails, 2016. Last accessed (DD/MM/YYYY) 02/10/2017. URL: https://www.kaggle.com/kaggle/hillary-clinton-emails.

[14] Mark Krellenstein. Starting a search application, 2009. accessed 25.04.17. URL: https://whitepapers.em360tech.com/wp-content/files_mf/white_paper/lucid2.pdf.

[15] Yunyao Li, Ziyang Liu, and Huaiyu Zhu. Enterprise search in the big data era: Recent developments and open challenges. *Proc. VLDB Endow.*, 7(13):1717–1718, August 2014. URL: http://dx.doi.org/10.14778/2733004.2733071, doi:10.14778/2733004.2733071.

[16] Lucidworks. Full text search engines vs. dbms (whitepaper). accessed 25.04.17. URL: https://lucidworks.com/2009/09/02/full-text-search-engines-vs-dbms/.

[17] lucidworks. Understanding transaction logs, soft commit and commit in solrcloud. Last accessed (DD/MM/YYYY) 7/12/2017. URL: https://lucidworks.com/2013/08/23/understanding-transaction-logs-softcommit-and-commit-in-sorlcloud/.

[18] S. Mascarnes, P. Lopes, and P. Sakhare. Search model for searching the evidence in digital forensic analysis. In *2015 International Conference on Green Computing and Internet of Things (ICGCIoT)*, pages 1353–1358, Oct 2015. doi:10.1109/ICGCIoT.2015.7380677.

[19] MIG. mig/conf/mig-agent.cfg.inc, 2017. Last accessed (DD/MM/YYYY) 11/10/2017. URL: https://github.com/mozilla/mig/blob/a2fe0fed53fb75d6c1ece5f917268c79774ca0ef/conf/mig-agent.cfg.inc.

[20] mig. mig/doc/concepts.rst, 2017. Last accessed (DD/MM/YYYY) 11/10/2017. URL: https://github.com/mozilla/mig/blob/master/doc/concepts.rst.

[21] Andrii Shalaginov, Lars Strande Grini, and Katrin Franke. Understanding neuro-fuzzy on a class of multinomial malware detection problems. In *Neural Networks (IJCNN), 2016 International Joint Conference on*, pages 684–691. IEEE, 2016.

[22] Solr. Indexconfig in solrconfig. Last accessed (DD/MM/YYYY) 7/12/2017. URL: https://lucene.apache.org/solr/guide/6_6/indexconfig-in-solrconfig.html.

[23] Solr. Learn more about solr. Accessed on 22.03.2017. URL: http://lucene.apache.org/solr/.

[24] Solrwiki. Solrcloud, 2017. Last accessed (DD/MM/YYYY) 7/12/2017. URL: https://wiki.apache.org/solr/SolrPerformanceProblems.

[25] Sphinx. Sphinx - open source search server. accessed:22/10/2017. URL: http://sphinxsearch.com/.

[26] United states military academy west point. Data sets, 2009. Last accessed (DD/MM/YYYY) 26/09/2017. URL: Unitedstatesmilitaryacademywestpoint.

[27] Volatility. Volatility framework, 2017. accessed:11/10/2017. URL: https://github.com/volatilityfoundation/volatility/.

[28] W. B. Wang, M. L. Huang, L. Lu, and J. Zhang. Improving performance of forensics investigation with parallel coordinates visual analytics. In *2014 IEEE 17th International Conference on Computational Science and Engineering*, pages 1838–1843, Dec 2014. doi:10.1109/CSE.2014.337.

[29] wikiApache. Cache autowarm count considerations, 2014. Last accessed (DD/MM/YYYY) 7/12/2017. URL: https://wiki.apache.org/solr/SolrPerformanceFactors.

[30] wikiapache. How can indexing be accelerated, 2016. Last accessed (DD/MM/YYYY) 7/12/2017. URL: https://wiki.apache.org/solr/FAQ#Why_does_the_request_time_out_sometimes_when_doing_commits.3F.

[31] W. M. S. Yafooz, S. Z. Z. Abidin, N. Omar, and Z. Idrus. Managing unstructured data in relational databases. In *2013 IEEE Conference on Systems, Process Control (ICSPC)*, pages 198–203, Dec 2013. doi:10.1109/SPC.2013.6735131.

[32] S. Zawoad and R. Hasan. Digital forensics in the age of big data: Challenges, approaches, and opportunities. In *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*, pages 1320–1325, Aug 2015. doi:10.1109/HPCC-CSS-ICESS.2015.305.