Handling Big Data using a Distributed Search Engine

Preparing Log Data for On-Demand Analysis

NIKLAS EKMAN
Abstract

Big data are datasets that are very large and computational complex. With an increasing volume of data the time a trivial processing task can be challenging. Companies collects data at a fast rate but knowing what to do with the data can be hard. A search engine is a system that indexes data making it efficiently queryable by users. When a bug occurs in a computer system log data is consulted in order to understand why, but processing big log data can take a long time. The purpose of this thesis is to investigate, compare and implement a distributed search engine that can prepare log data for analysis, which will make it easier for a developer to investigate bugs. There are three popular search engines: Apache Lucene, Elasticsearch and Apache Solr. Elasticsearch and Apache Solr are built as distributed systems making them capable of handling big data. Requirements was established through interviews. Big log data of totally 40 GB was provided that would be indexed in the selected search engine. The log data provided was generated in a proprietary binary format and it had to be decoded before. The distributed search engines was evaluated based on: Distributed architecture, text analysis, indexing and querying. Elasticsearch was selected for implementation. A cluster was set up on Amazon Web Services and tests was executed in order to determine how different configurations performed. An indexing software was written that would transfer data to the cluster. Results was verified through a case-study with participants of the stakeholder.

Keywords
Big Data, Distributed System, Search Engine
Abstract


Nyckelord
Stordata, Distribuerat system, Sökmotor
# Table of Contents

1 **Introduction** ................................................................. 1  
   1.1 Background ................................................................. 1  
   1.2 Problem statement ........................................................ 2  
   1.3 Purpose ................................................................. 2  
   1.4 Goal ................................................................. 2  
   1.5 Benefits, Ethics and Sustainability .................................. 2  
   1.6 Methodology/Methods .................................................. 3  
      1.6.1 Philosophical Assumptions ...................................... 3  
      1.6.2 Research Methods ................................................. 4  
      1.6.3 Research Approaches ............................................. 4  
      1.6.4 Literature Study ................................................... 5  
   1.7 Stakeholders ............................................................ 5  
   1.8 Delimitations ............................................................ 5  
   1.9 Outline ................................................................. 6  

2 **Distributed Systems** ....................................................... 7  
   2.1 Big Data ................................................................. 7  
   2.2 Node Cooperation ........................................................ 8  
   2.3 Replication and Ensuring Consistency ................................ 9  
   2.4 Scaling ................................................................. 10  

3 **Search Engines** ................................................................. 11  
   3.1 Querying Data ............................................................ 11  
   3.2 Text Analysis ............................................................ 11  
   3.3 Indexing Data ............................................................. 11  
   3.4 Distributing Search Engines ........................................... 12  
   3.5 Popular Choices .......................................................... 12  
      3.5.1 Apache Lucene ....................................................... 13  
      3.5.2 Apache Solr .......................................................... 14  
      3.5.3 Elasticsearch ........................................................ 16  
   3.6 Related Work ............................................................ 20  
      3.6.1 Usage of Related Work ......................................... 21  

4 **Methods** ........................................................................... 22  
   4.1 Research Strategies and Design ....................................... 22  
   4.2 Data Collection ............................................................ 23  
   4.3 Data Analysis ............................................................. 23  
   4.4 Quality Assurance ....................................................... 24  
   4.5 Software Development Methods ....................................... 25  
      4.5.1 Waterfall Method ..................................................... 25  
      4.5.2 Scrum ................................................................. 26  
      4.5.3 Kanban ............................................................... 26  
      4.5.4 Software Development Method Chosen for this Thesis ...... 27
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Handling Big Data using a Distributed Search Engine</td>
<td>28</td>
</tr>
<tr>
<td>5.1</td>
<td>Gathering Requirements</td>
<td>28</td>
</tr>
<tr>
<td>5.2</td>
<td>Established Requirements</td>
<td>28</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Log File Generation Rate</td>
<td>29</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Log File Format</td>
<td>29</td>
</tr>
<tr>
<td>5.3</td>
<td>Search Engine Selection</td>
<td>30</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Evaluation of Distributing the System</td>
<td>31</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Evaluation of Text Analysis</td>
<td>31</td>
</tr>
<tr>
<td>5.3.3</td>
<td>Evaluation of Indexing</td>
<td>31</td>
</tr>
<tr>
<td>5.3.4</td>
<td>Evaluation of Querying</td>
<td>32</td>
</tr>
<tr>
<td>5.4</td>
<td>Stakeholder Decision</td>
<td>32</td>
</tr>
<tr>
<td>6</td>
<td>Setting Up an Elasticsearch Cluster</td>
<td>33</td>
</tr>
<tr>
<td>6.1</td>
<td>Decoding Log Files</td>
<td>33</td>
</tr>
<tr>
<td>6.2</td>
<td>Indexing Software</td>
<td>33</td>
</tr>
<tr>
<td>6.3</td>
<td>Hardware</td>
<td>33</td>
</tr>
<tr>
<td>6.4</td>
<td>Provisioning a Node</td>
<td>34</td>
</tr>
<tr>
<td>6.5</td>
<td>Indexing Performance</td>
<td>35</td>
</tr>
<tr>
<td>6.5.1</td>
<td>Horizontal Scaling</td>
<td>35</td>
</tr>
<tr>
<td>6.5.2</td>
<td>Vertical Scaling</td>
<td>36</td>
</tr>
<tr>
<td>6.5.3</td>
<td>Total Indexing Time</td>
<td>37</td>
</tr>
<tr>
<td>6.6</td>
<td>Real-Time Requirements</td>
<td>38</td>
</tr>
<tr>
<td>7</td>
<td>Verifying the Elasticsearch Implementation</td>
<td>39</td>
</tr>
<tr>
<td>8</td>
<td>Conclusions</td>
<td>40</td>
</tr>
<tr>
<td>8.1</td>
<td>Evaluation of the Elasticsearch Implementation</td>
<td>40</td>
</tr>
<tr>
<td>8.2</td>
<td>Discussions</td>
<td>40</td>
</tr>
<tr>
<td>8.2.1</td>
<td>Importance of a Distributed Architecture</td>
<td>42</td>
</tr>
<tr>
<td>8.3</td>
<td>Future Work</td>
<td>42</td>
</tr>
<tr>
<td>8.3.1</td>
<td>What to do next</td>
<td>42</td>
</tr>
<tr>
<td>References</td>
<td></td>
<td>44</td>
</tr>
<tr>
<td>Appendices</td>
<td></td>
<td>48</td>
</tr>
<tr>
<td>A</td>
<td>Software Licenses</td>
<td>48</td>
</tr>
<tr>
<td>A.1</td>
<td>Apache License 2.0</td>
<td>48</td>
</tr>
<tr>
<td>B</td>
<td>Requirements Interview 1</td>
<td>49</td>
</tr>
<tr>
<td>C</td>
<td>Requirements Interview 2</td>
<td>51</td>
</tr>
</tbody>
</table>
1 Introduction

Data is being collected at an increasing rate from an ever expanding list of devices such as smartphones, Internet-of-Things devices and laptops. The yearly data collection volume is predicted to increase from approximately 0.8 ZB to 35 ZB\(^1\) between the years 2009 to 2022 \([1]\). IBM generates 2.5 quintillion bytes\(^2\) every day and Facebook gathers 6 TB of user activity per day \([2,3]\). The term big data describes the phenomenon of large volumes of computational complex data. Big data leads to new challenges but also possibilities \([4]\).

Logbooks have been used for a long time to log events such as maintenance of an airplane or how much distance a boat has covered \([5]\). Computer systems generate log data in order to provide an audit trail, which is a document containing step-by-step history of events occurred in the system. Audit trails are used for investigative purposes in order to improve e.g. quality or performance of a system \([6]\).

1.1 Background

Processing data is a trivial task. For instance, using a Linux terminal users can count the amount of words in a text document by typing `wc -l <filename>` \([7]\). Processing big data is more complex due to problems such as processing time and data storage. A solution is to migrate the processing to a distributed system forming a cluster of computers (nodes) that is working in parallel which results in load balancing and reduction in the processing time. Migrating a trivial processing task to a distributed system introduces new challenges \([8]\). Nodes needs to cooperate by passing messages through unreliable communication links which means assuming that they can, and will, fail at any time. Another challenge is keeping a synchronized state between all nodes \([9]\).

When a bug is discovered in a computer system the audit trail has to be examined in order to understand it. The audit trail may be several gigabytes in storage and spread out across multiple nodes. Manually processing/investigating Big Log Data (BLD) is time-consuming and tedious for humans. Audit trails needs to be processed quickly and be searchable in real-time on-demand for investigate purposes \([3]\). A search engine may be used for aggregating, indexing and querying log data \([10]\).

![Diagram](image.png)

Figure 1: Processing log data in a distributed system using a buffer node.

\(^1\) 1 zettabyte (ZB) = 10\(^9\) terabytes (TB)
\(^2\) 1 quintillion bytes = 10\(^6\) terabytes (TB)
In order for a search engine to process log data a node will need to transfer its log data to the search engine. As the computer system grows so will the volume of log data and a solution is to add a buffer node, that will temporarily store the log data from other nodes while relaying it to the search engine when it is ready for it. See figure 1 for an architecture using a buffer node. [3, 11]

### 1.2 Problem statement

Being able to effectively analyze will make identifying bugs an easier task for developers. A developer can probably draw a conclusions as to what is causing the bug from an error description, but being able to validate the conclusions with facts is a task that can take a long time. Basing conclusions on facts increases the development quality of the systems, which decreases the likely hood of reoccurring bugs.

How can software developers handle big data using distributed search engines and make it searchable on-demand, for analytical and investigative purposes?

### 1.3 Purpose

Investigate, compare and implement a distributed search engine that can prepare log data for analysis, which will make it easier for a developer to investigate bugs. Easing the process of understanding bugs will lead to computer systems of higher quality and decrease the time it takes to patch bugs as well the likelihood of reoccurring bugs.

### 1.4 Goal

Implement a computer system that can to gather big log data from nodes in a distributed system and prepare them for on-demand investigation. The solution must be scalable to compensate for increasing amounts of log data an organization may incur over time. It must be fault tolerant and have a high availability.

A stakeholder organization has been selected and the solution will be implemented for them.

### 1.5 Benefits, Ethics and Sustainability

As the data volume of companies grows, revisiting archived data and processing it may yield new value that was not discovered previously. Handling big data poses a security risk. Database management systems contains access control to secure the data from people that are not authorized, but big data frameworks usually does not have this safeguard. For ethical purposes data that is confidential or contains personal data, such as social security numbers, must be cleaned and the sensitive data stripped away. [1]
Anonymizing big data needs careful considerations and is a challenge. Research that was conducted on Facebook data went through steps for anonymization. However, it was still possible to trace the data back to individuals. [12]

The computer system will be sustainable by using hardware with low carbon footprints and third party providers that has a conscious environment policy. Creating an optimized solution will also decrease the amount of resources needed in order to support it, which means less impact on the environment.

1.6 Methodology/Methods

Using a proper research method, and knowing why, is important when researching in order to assure the quality. There are two basic categories of methodologies that is used when researching: [13]

1. **Quantitative research** means writing hypothesis that are clear and verifiable. The research method uses statistics and large quantities of data to test the hypotheses.

2. **Qualitative research** means that the researcher tries to understand meanings, opinions and behaviors to build theories, inventions or computer systems. The method uses data sets that are small enough to reach a reliable result.

This thesis uses a qualitative- and quantitative research method. By doing a literature study, going through others works and research a method will be constructed of how to prepare BLD for analysis. Thereafter, an implementation will be carried out and tested using big data.

What follows in this chapter is a summary of steps taken when conducting research, what they entail, which has been chosen in this thesis and why.

1.6.1 Philosophical Assumptions

A philosophical assumption is made at the research beginning and will guide future assumptions. There are four philosophical assumptions that should be considered: [13]

1. **Positivism** assumes that the researcher cannot influence the observations.

2. **Realism** means that the researchers tries to understand observations with regards to a defined reality.

3. **Interpretivism** tries to understand peoples perception of events.

4. **Criticalism** tries to understand the reality in a social and cultural way, such as "why does racism exist?".

This thesis uses the realism philosophical assumption. The purpose of the research is to find a method to prepare BLD for analysis. Ultimately, the reference implementation of
the method will depend on what the stakeholder is in need of. Choosing a method may vary with other stakeholders.

1.6.2 Research Methods

Research methods define rules how a task is carried out. There are seven research methods:

1. **Experimental** tests causalities between variable relationships.
2. **Non-experimental** draws conclusions from existing data.
3. **Descriptive** means that the researcher assigns characterizing properties to data.
4. **Analytical** validates existing hypotheses using already collected data, such as validating someone else's research.
5. **Fundamental** discovers new insights by observations and theory testing, also known as **basic research**.
6. **Applied** involves answering questions or solving practical problems.
7. **Conceptual** means trying to understand existing concepts or develop new using methods such as literature studies.
8. **Empirical** discovers knowledge by collecting data, analyzing and evaluating it. Literature study is an example of empirical research.

This thesis will be using a combination of non-experimental, conceptual and empirical method. The research will start with a literature study that serves two purposes; explaining background information for the reader and identifying search engines that will be usable to solve the problem. Literature study will be an important step of the research where conclusions has to be drawn, concepts has to be understood and new knowledge has to be discovered by evaluation multiple literature sources.

1.6.3 Research Approaches

There are three approaches for drawing conclusions:

1. **Inductive** derives hypotheses from data, e.g. "I like the food, therefore you might like it."
2. **Deductive** verifies or falsifies hypotheses often from large volumes of data, e.g. "Do I like the food? I like the food."
3. **Abductive** is a combination of inductive and deductive that tries finding the simplest most probable explanation.
Inductive approaches are often used with qualitative methods and deductive with quantitative. [13]

This thesis uses an inductive approach. The reference implementation will induce a hypothesis from a set of search engines, i.e. which serves the purpose best?.

1.6.4 Literature Study

The purpose of a literature study is to educate the researcher with missing knowledge and to identify existing research. It is also used to convince the reader that appropriate consideration has been taken into existing research and that the new research expands on the problems. There are different types of literature studies such as: [14]

1. **Systematic reviews** is a quantitative method involving shortening existing results.

2. **Secondary data analysis projects** begins with a research question that is answered after reviewing a wealth of information.

3. **Introduction to a primary research topic** means setting a broad context and narrowing it down to a specific research problem. It is used to convince the reader that a research project has taken existing problems and knowledge into consideration.

This thesis uses a systematic reviews literature study method. Different search engines will be investigated, summarized and evaluated.

1.7 Stakeholders

The research will be carried out at the Cybercom Group AB. Cybercom is a consultancy company operating in Sweden, Finland and Denmark with customers in the public and private sector, such as Stockholms Läns Landsting, Ericsson and Saab. [15]

Cybercom maintains internal systems for their customers which generates lots of log data each month. Whenever a problem occurs the developers will investigate the log files manually using a text editor. Solving complex problems can take weeks and they want to optimize this process. They have provided the data used in this research and are interested in a solution how to prepare big log data for on-demand analysis.

1.8 Delimitations

This research will be limited to preparing log data for analysis. The focus is to access log data and not to process it and discover new value using methods such as data mining and/or statistical analysis.
The literature study will be limited to researching three distributed search engines. There may be many adequate softwares on the market, but a limit has been set in order to set an appropriate scope for the research.

1.9 Outline

Chapters 2 Distributed Systems and 3 Search Engines contains a literature study in order to gain deeper knowledge into big data, distributed systems and search engines. Chapter 4 Methods describes what research methods was applied and why. In 5 Handling Big Data using a Distributed Search Engine the method of selecting a distributed search engine is presented. The results of creating an implementation using a selected search engine is described in chapter 6 Setting Up an Elasticsearch Cluster. 7 Verifying the Elasticsearch Implementation explains the steps taken to verify the results. Ultimately, the entire research is concluded and discussed in chapter 8 Conclusions and it presents interesting future work.
2 Distributed Systems

This thesis will be investigating distributed search engines. The following chapter contains knowledge of big data.

2.1 Big Data

Big data is a technique of handling large volumes of data. These sets of data are characterized by five properties:

1. **Volume** - It is large in *volume*. The largest data set available is 12.1 petabytes as of 2014.

2. **Velocity** - Data is received and sometimes acted upon in a high *velocity*. Some systems require real-time processing of the data, while some just save it to disk and handle it later.

3. **Variety** - Some data may be aggregated and stored without knowing what to do with it at the time. The structure of the data *varies* and may even change in the future, which is known as "schema on read".

4. **Value** - The data has *value* after processing. There are multiple ways of discovering value, some is dependent on humans researching and exploring the correct variables.

5. **Veracity** - Data needs to be able to be validate the *veracity* of it. Anyone who uses a data set must be able to answer questions such as "How was this data generated?".

These five properties are the basic ones that is commonly used to describe big data. There are additional properties that are sometimes applied to the concept of big data, but not used as widely:

- **Variability** - Data does not only vary in structure but also in context. E.g. the context of blog posts may vary from food to computer games.

- **Visualization** - Large volumes of data is hard to visualize. A multitude of statistical methods needs to be applied in order to reduce the data to a *visualizable* data set.

- **Validity** - It is common for researchers to pre-process a data set before using it. *Validity* refers to how close the structure of the data set is for its intended use.

- **Vulnerability** - Systems containing big data needs to take more responsibility to keep it secure. A data breach will mean that large amounts of data will be vulnerable.

- **Volatility** - Refers to how long it will take before the data becomes outdated and useless.

---

31 petabyte (PB) = $10^4$ terabytes (TB)
The optional properties has been introduced in order to describe big data more deeply.

Large volumes of data introduces new challenges of storing and processing. Tasks that was once trivial is more complex since it takes longer time to execute procedures and takes more memory to store it. In order to process the data in a feasible amount of time the workload must be distributed to multiple nodes. If workloads can be split into many smaller isolated tasks that does not need to be share state, distribution becomes a lot easier. Also, distributed systems are favorably abstracted away from the user, only exposing programming interfaces that makes processing appear trivial. Big data embraces localization which means moving the algorithms closer to the data eliminating costly and time consuming move operations. [4]

2.2 Node Cooperation

In a distributed system, nodes cooperatively carry out the purpose of a computer system by passing messages using unreliable communication links, usually using protocols such as Transmission Control Protocol (TCP) or User Datagram Protocol (UDP). A node can be anything from one process on a computer to a computer itself. Nodes are expected to crash at any time, making the system designed to handle failures which leads to higher availability. [9]

The Consistency-Availability-Partition Tolerant theorem (CAP theorem) states that every distributed system may only have exactly two out of three properties:

1. **Consistency** - A system is consistent if an operation at time \(t\) can be seen in the system during any subsequent operation at \(t + 1\). If the system fails to see the operation at time \(t\) it is eventually consistent. [18]

2. **Availability** - A system is available if it can respond to requests within a short time window. [18]

3. **Partition tolerance** - Nodes are connected using different topologies. A node in a system most likely does not know of every participant, but only a subset. Distributed systems are built with the assumption that any node can fail at any time and a distributed system tries to avoid having a single-point-of-failure. One way of doing this is by giving each node the same privileges. If nodes starts failing it is possible for the cluster to be accidentally split into two isolated sub-clusters called partitions. Partitions may also occur if network links between certain nodes fails, creating the illusion of nodes failing. A distributed system must be able to handle partition problems or ensure that they never happen. [9, 18]

Designing a distributed system contains no "one-solution-fits-all" and the theorem can be used as a guideline when choosing between different distributed systems for the current problem.
Figure 2: Explanation of the CAP theorem using two nodes and a client.

The theorem can be explained using three examples. Figure 2 illustrates each example (from left to right): [19]

(A) **Consistency** - A client sends data to a consistent system which means that the nodes needs to synchronously synchronize the state before other nodes can accept requests again, i.e. it cannot be available.

(B) **Availability** - A client sends data to an available system which means that the receiving node must answer quickly, i.e. it cannot be consistent.

(C) **Partition tolerant** - A client sends data to a partition tolerant system which means that the system will not fail if nodes are divided into isolated sub-clusters. The system will either have to answer quickly, i.e. it cannot be consistent, or wait for the sub-cluster to merge again before synchronizing states, i.e. it cannot be available.

Understanding these properties is a basic fundamental in order to understand distributed systems. [19]

### 2.3 Replication and Ensuring Consistency

In order to ensure higher availability and fault tolerance the nodes in a distributed system may create backup copies of itself called *replicas*. One replica is elected the leader, it is the *primary replica*. Having many replicas can be used to speed up computations, check for data corruption or may exist for the sole purpose of being elected leader if the primary crashes [20]. [9]

Updating the state of a node that has replicas, while ensuring consistency, is complex. The quorum $W + R > N$ has to be fulfilled in order to ensure consistency, where: [20]

- $W$ are how many nodes that must execute a write in order to consider it successful.
- $R$ are how many nodes must execute a read in order to ensure that the most up-to-date value will be seen.
- $N$ is the amount of replicas a node must have.
If you have a distributed system where $R = 3$ and choose $W = 2$ then $R = 2$, which means all nodes must have three replicas and a successful write execution is defined as two nodes successfully executing the write. Read executions requires reads from at least two nodes. Note that there is rarely the need for reading from all nodes or writing to all nodes. [20]

### 2.4 Scaling

A computer system that is built as a cluster of cooperating nodes can be scaled efficiently by adding or removing nodes as throughput requires [8]. There are two types of scaling strategies: *scaling-up* or *scaling-out*, also known as *vertical- and horizontal scaling*. Scaling-up means upgrading the existing hardware and scaling-out means adding more computers to the cluster. Scaling-up is usually more expensive than scaling-out and also suffers from the inability to easily downscale, i.e. preventing over-provisioning, and hardware limitations [21]. [22, 23]

Relational database management systems are an example of systems that can effectively be scaled-up, but scaling-out is hard. The CAP theorem (see chapter 2.2 Node Cooperation) has been a driving factor in developing new database systems called *Not only SQL*, built as distributed systems with the ability to scale-out and therefore better suited for handling big data. [21].
3 Search Engines

A search engine is a system containing documents that are indexed. By building an index the system can efficiently provide the results of a query. Documents can be anything from text to images. This chapter will explain the theory behind search engines, then present some of the most popular alternatives on the market and lastly conclude with related work.

3.1 Querying Data

Users queries a search engines expecting relevant results. Some search engines supports advanced querying, using filters and special syntax, while others can only handle a simple query [25]. It may return a certain set of documents, some relevant considering a context and some not. Search engines will also sort the results by relevance using a scoring function. The implementation depends on the search engine [26].

3.2 Text Analysis

The purpose of text analysis is to split the data into smaller parts called terms. It also executes sub-procedures on the terms such as normalization by stemming, i.e. removal of common words (such as and and or). Text analysis is applied during both querying and indexing of data since the system must interpret the query similar to the index. There are many different text analysis techniques and there is no "one-solution-fits-all". It is the users responsibility to find an existing or creating a new text analysis strategy for his or her purpose. [24]

For instance running a text analyzer on the data The day is sunny and beautiful may yield day, sunny, beautiful. Words such as is and and has been removed since it does not describe the data. If a user queries the data with sunny day it will match the document because it contains those exact word.

3.3 Indexing Data

Indexing occurs after text analysis and is the process of analyzing the terms for meta-data. The process may look for words that occurs frequently or a dominating color in an image. The meta-data can later be used to provide users with relevant documents based on a search query. When a user queries it only needs to look through the index and not the contents of each document which makes searching for data efficient. [27]

An inverted index is a data structure that maps a term to the documents it appears and is the core mechanism that allows for quick retrieval of documents. In order for a search to provide relevant results, an inverted indexes needs to handle challenges such as synonyms
and words in different tenses. Such challenges are handled when a document goes through text analysis (see chapter 3.2 Text Analysis). [23, 24]

<table>
<thead>
<tr>
<th>Term</th>
<th>Document A</th>
<th>Document B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beautiful</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>little</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>puppies</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>Dogs</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>are</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>cute</td>
<td>-</td>
<td>X</td>
</tr>
</tbody>
</table>

Table 1: Basic inverted index used for retrieving relevant results in a search engine.

Consider two documents with the following contents:

1. Beautiful little puppies.
2. Dogs are cute.

A basic way of analyzing the documents would result in the inverted index described in table 1. Querying for puppies would return document 1 and a query for dog would result in document 2. Arguably, dog and puppies are considered the same, i.e. a search for dog or puppies should both return document 1 and 2. The same can be considered for the terms beautiful and cute. Correlations like these are built when documents are processed during text analysis and later stored in the index.

### 3.4 Distributing Search Engines

Being able to query big data on-demand is challenging. Indexing takes time, especially if there is a requirement to process the data in real time. Distributed search engines has been built to solve the problem of distributing an index [11]. There are two common ways of distributing the indexes in a search engine: document- and term partitioning. Document partitioning creates shards that function as fully functional independent indexes, only containing a subset of the entire index. Workload is distributed by executing a query on all shards in parallel. Term partitioning delegates responsibilities for terms to shards, e.g. shard 1 is responsible for the term apple. It requires at most one shard per term which is efficient and queries only have to execute on the shard responsible for the term, but it creates higher network traffic. [28]

### 3.5 Popular Choices

Presented in this chapter are the three popular search engines Apache Lucene, Apache Solr and Elasticsearch. The search engines has been picked due to being well established
on the market and used by many large companies. Apache Lucene is used by the company Twitter and is the oldest solution [29], Apache Solr is used by Apple and Bloomberg [30, 31] and Elasticsearch is used by Netflix, SoundCloud, GitHub, Facebook and Adobe Systems [32].

Each engine will be compared with respect to the following points:

- What is the software license?
- How is it distributed?
- How does it analyze text?
- How does it index data?
- What kind of query language does it support?

The questions have been chosen due to covering all core concepts of a search engine.

3.5.1 Apache Lucene

Lucene is an open-source text search library that was created by Doug Cutting in 1997 and later donated to the Apache Software Foundation [29, 33]. The software is distributed under an [Apache License 2.0](see appendix A Software Licenses). Official enterprise support does not exist [34].

Distributing Lucene  Lucene is not a distributed system, it is a programming library for users that want to incorporate search into their existing application. There are projects that has created distributed search engines using Lucene, such as Apache Solr, Elasticsearch and Twitter's Earlybird. [28, 24, 23, 29]

Text Analysis  During text analysis Lucene converts a documents into tokens, which are stored in an inverted index or used in queries. A token is a collection of attributes describing it, e.g. position in the document. The text analyzer will process the documents in three steps: [33]

- Tokenization
- Token filtering
- Normalization

Lucene will use a user-defined strategy when creating tokens, e.g. the whitespace strategy will split the document into an array of words. After the tokenization it will apply optional filters that will modify the token according to user-defined rules. Users can use an existing token strategy, filtering and normalization or write a custom one. The library has built-in language support for 32 different languages, which in turn contains specific strategies. [33]
Indexing  Lucene uses inverted indexes (see chapter 3.3 Indexing Data) to store document data. The inverted indexes are immutable and stored in-memory as smaller parts called segments. Segments are periodically persisted and merged with existing data on disk in order to endure failures. 

Documents are broken down into fields that have name, data, weight and other attributes. The system assigns a unique identifier to a document. Fields belongs to one of two categories: indexed and stored. An indexed field is stored as an analyzed version of the raw data, i.e. a collection of tokens. A stored field stores the raw contents of the data as an array of bytes. Fields can belong to both categories, but only stored fields are used during querying. 

Query Language  Lucene uses query objects to express queries. Querying can be done in many ways such as boolean, where, proximity queries, position-based, wildcard, fuzzy and regular expression. By combining the building blocks of the query objects, complex queries can be expressed. Queries executes a sequential search in each segment applying a score method on each results to rank the relevance. 

A user can configure what score method to use, e.g. term frequency is a common one. The results from each segment is processed by a collector that will merge, sort and retrieve as many results as the user wants. 

3.5.2 Apache Solr

Apache Solr was created on top of Apache Lucene in 2004 and donated to the Apache Software Foundation. Apache Lucene and Apache Solr was later bundled together meaning development on Apache Solr also affects Lucene. The system is distributed using a ALv2 license (see appendix A Software Licenses). Apache Software Foundation does not provide any enterprise support.

Apache Solr contains a basic out-of-the-box graphical interface for issuing queries. Third-party interfaces exists, but not provided officially by the Apache Software Foundation.

Distributing Apache Solr  Apache Solr was not designed as a distributed system. There has been efforts to adapt the system into a distributed architecture. There are four distribution methods: 

1. Sharding  splits a large index into shards that are distribute on a single machine. Query requests from clients are distributed to all shards in parallel and the results are merged before replying.

2. Master-slave  writes all indexing requests to a master node. Slave nodes will poll the master for updates, which will respond with the entire index. Query requests are issued directly to a slave node. Clients issues requests directly to a node making it their responsibility to load balance the requests.
3. Hybrid contains many master-slave architectures each responsible for a single shard. It is the clients responsibility to load balance requests. The method became popular and SolrCloud was introduced in order to reduce the complexity and problems of maintaining this architecture.

4. SolrCloud builds on the hybrid setup and abstracts the load balancing away from the client and eliminates the need for slaves to poll the master by pushing updates. Apache ZooKeeper is used for node coordination. SolrCloud replicates shards onto nodes and one replica is the leader.

There are many ways to distribute Apache Solr which are results of developing the software to fit new requirements from the market. [24]

Text Analysis During text analysis the data is converted into tokens which are indexed as terms. An analyzer is a pipeline of procedures that converts the data into tokens. Tokens and terms stores additional meta-data such as:

- Start offset in document.
- End offset in document.
- Position increment information - Stemming text removes words from the text, which is reflected in the position increment information.
- Byte payload.

This is the data that is later used when results are passed through the scoring function.

Apache Solr provides many built-in filters that can be used during text analysis. Some of them are: [24, 37]

1. Whitespace tokenizer will remove all whitespace in a text and converts it into an array of strings.
2. ASCII folding filter will remove any non-ASCII characters from the text.
3. Stop filter marks words as stop words that will be discarded by the system. It can be used to remove commonly occurring words from a language such as and.
4. Keep word filter keeps only the words specified by the user.
5. Porter stem filter will convert terms to their base form, e.g. cars to car.
6. Trim filter will remove whitespace before and after a term (but not in the middle of it).

These filters are the basic building blocks for filtering data during text analysis. A user can create his own for the problem at hand. Apache Solr contains more features such as identifying phonetically similar words, e.g. the words "you" and "u" are considered the same. Additionally, Apache Solr provides support for 36 language out-of-the-box. [24, 37]
Indexing  Like Apache Lucene, Apache Solr associates fields to documents and stores data using an inverted index. A schema file can be created in order to describe the data that is about to be indexed. It helps Apache Solr to index fields such as dates. If a schema is not given the system will automatically try to detect the type of the field. A field type is a name of an analysis pipeline to be used during indexing and querying. For instance the class "Boolean" will interpret the field value as true or false and "String" will interpret it as raw text (no text analysis). There is a flag called stored which will determine if the value is retrievable. A non-stored field can never be displayed to the user.[24]

Indexing a document involves going through multiple steps: [24]

1. Preparation
2. Upload
3. Pre-processing
4. Field analysis
5. Index

The document must first be prepared so that Apache Solr can understand the data. Steps involves parsing the data into a format that the system understands such as, Extensible Markup Language (XML), JavaScript Object Notation (JSON), Comma Separated Values (CSV) or javabin. Once the data has been prepared it can be uploaded to the system. [24]

Apache Solr will pre-process documents which involves tasks such as detecting duplicates and the content language. Users can write their own procedure in order to manipulate any value. During field analysis the document is split into tokens and terms (see chapter 3.5.2 Text Analysis). [24]

Query Language  Queries are executed by sending a request to a web API and are written as a URL parameter called q. The syntax is similar to that of SQL. Words can be prefixed with the field where Apache Solr should look for the data or it will use the schema default field. A user can use the operators "OR", "AND", "NOT", "+" and "-". The plus operator describes a required term is included and the minus an optional. Omitting the operator inserts an implicit "OR". A query that searching for the James Bond movies "Golden Eye" and "Tomorrow Never Dies", with a default field of actor, could look like this: [24]

q=title:(golden OR tomorrow) AND character:(james AND bond) AND (connery brosnan)

3.5.3 Elasticsearch

Elasticsearch is an open-source distributed search engine maintained by the company Elasticsearch BV. It is built as a layer on-top of Apache Lucene (see chapter 3.5.1) in order to
package it into a standalone system. Elasticsearch is distributed with ALv2 (see appendix A Software Licenses) [38]. [23]

Elastic provides enterprise support with service level agreements offering coverage during business hours or any time of the day, any day of the week with response times down to one hour. It is a free software that can be downloaded and self-hosted but elastic offers hosting services of the system if needed. [39]

Kibana is an officially developed graphical user interface for Elasticsearch [40]. It is also open-source and distributed under the same license as Elasticsearch [41].

**Distributing Elastic Search** Elasticsearch is designed as a distributed system. A shard contains subset of an entire index and works as a complete search engine on its own. Nodes contains shards and the shard can be either primary or replica. Each document belongs to a single primary shard and a configurable amount of replicas. [23]

A node is one running instance of Elasticsearch. There is one master node in the system, but any node can be elected master. Requests can be issued to any node in the system which then assumes the role of coordinating node. The coordinating node will route the request to a shard using round-robin in order to load balance within the cluster. [23]

See figure 3 for an example scenario. There is a single primary and replica on each of the nodes. A client issues an indexing request to node A which is then the coordinating node. Node A determines that the document should belong to the primary shard hosted on node B. The request is handed over to the responsible primary shard after being routed to node B. The primary shard coordinates the indexing with its replica on node A. When the primary node determines that both shards has indexed the document it responds to node B, which will route the response back to node A. Ultimately, node A can respond to the entire request. [23]

The system supports eventual and full consistency. The user can configure to use one of the three write semantics to: [23]

1. **One** will write the to the requested node.
2. **All** will write to all nodes.
3. **Quorum** will write to as many nodes as necessary in order to ensure consistency (see 2.3 Replication and Ensuring Consistency). Elasticsearch uses the formula \( \text{primary} + n\cdot\text{Replicas} + 1 \).

---

17
If needed, then Elasticsearch supports full consistency but it may be disabled for other benefits. \[23\]

Queries are executed locally and in parallel on all shards and the result is transferred to the coordinating node. The size of each execution result depends on how many results the client wants. A final response is created by merging all results. If a coordinating node cannot reach the amount of nodes needed for the write semantics the request will fail and the client will receive a timeout response \[23\]

**Text Analysis**  When analyzing a document Elasticsearch splits each word into tokens using different user-configurable rules. A user may also create new sets of rules. There are four different rules built-in: \[42\]

1. *Standard analyzer* splits the text on word boundaries, removes punctuations and lowercases.
2. *Simple analyzer* splits the text on non-letters and lowercases.
3. *Whitespace analyzer* will split the text on whitespace.
4. *Language analyzer* splits the word with a language taken into account, removes common words such as and and stems words such as calling becomes call.

33 languages are supported out-of-the-box. *Multifields* are fields that has been analyzed using multiple rules, but interpreted as different fields. This is useful for situations where the language analyzer removes words that is important for the meaning, e.g. the word not will be remove but the phrase I am happy and I am not happy have two different meanings. The solution would be to analyze the phrases using both an English- and standard analyzer and to take both into account when querying the data. \[42\]

**Indexing**  Documents are indexed in an inverted index. It is recommended to create a separate index for each categories of data, for instance *magazines* and *cars*. An index consists of *types* that contains documents. Types are used to distinguish data within the index and is used to decrease the total amount of indexes at the expense of larger ones \[43\]. Schemas are attached to the type, not index. Documents are immutable meaning updates to a document will internally retrieve it, create a new copy of it and then re-index the new document. \[23\]

$$shard = hash(id) \% nPrimaryShards$$ \ (1)

During index creation the user configures how many shards it has and replicas each shard has. Number of shards is immutable after creation. A coordinating node will detect which primary shard a document belongs to by passing it through a hashing function, see equation \[1\]. \[23\]
Elasticsearch supports near real-time search, it trades real-time for performance. Since Elasticsearch is built on-top of Apache Lucene the inverted index is stored as smaller parts called segments (see chapter 3.5.1 Indexing). Therefore, it is not until segments are persisted from memory to disk that they are queryable. [44]

**Query Language** There are two kinds of queries: full-text and exact-value. Full-text analyses the entire document and retrieves results based on relevance. Exact-value functions more like an SQL query where the user specifies which fields should have a certain value, e.g. `date = 2017-01-01`. [45]

Listing 1: Example query using Elasticsearch’s internal query language.

```json
{
  "query": {
    "constant_score": {
      "filter": {
        "term": {
          "author": "niklas",
          "date": "2017-01-01"
        }
      }
    }
  }
}
```

Elasticsearch features its own query domain specific language, which is based on the JSON format. An example of a query can be viewed in listing 1 that matches all documents with a specific value on the `author` and `date` fields. In addition to queries there are filters. The difference being queries are applied before relevance evaluated while filters after. A filter will look up all documents containing the term and build a bitset for each filter, an array of 1s and 0s where 1 indicates that the term is present. It will then find the documents that have a 1 in each bitset. The last 256 queries which did not contain more than 10 000 documents are cached in-memory. [46]

Elasticsearch takes these factors into account when evaluating the score: [46]

- Term frequency
- Inverse document frequency
- Field-length norm

\[
tf(t) = \sqrt{\text{frequency}}
\]  

(2)

The more often a term appears in a field the more relevant it is also known as term fre-
**quency (tf).** The formula can be seen in equation 2 for term t.

\[
idf(t) = 1 + \log\left(\frac{nDocs}{docFrequency + 1}\right)
\] (3)

Terms that appear more often in multiple documents is considered less relevant. This is called the *inverse document frequency* (idf) and is calculated according to equation 3 for term t. \(docFrequency\) is how many docs the term appears in and \(nDocs\) how many documents there are in the index in total.

\[
fln(f) = \frac{1}{\sqrt{nTerms}}
\] (4)

Longer fields are less relevant which is called *field-length norm* (fln). Equation 4 describes the fln for field \(d\) that has \(nTerms\) terms.

### 3.6 Related Work

The search engines Apache Lucene, Apache Solr and Elasticsearch (see chapter 3.5 Popular Choices) are considered related work since they are the result of research within the field of handling big data.

Data warehousing is a concept that pre-dates big and focuses on pre-processing data from multiple sources and storing it in a database. The purpose of data warehousing is to archive large volumes of data efficiently so that it can later be used for analysis. Business Intelligence is the concept of understanding big data by analyzing and visualizing it using methods such as data mining and dashboards. It is often used in the context of understanding users. Data warehousing is considered a foundation to Business Intelligence.

Hadoop is an alternative way of processing big data. It uses the MapReduce programming model where users specifies map and reduce tasks. A map function takes a key value pair and typically aggregates them to produce an intermediate value. A reduce function then executes using the intermediate value as input creating a reduced output. MapReduce makes it easy to split the processing into isolated tasks which can be distributed over many nodes while executing in parallel. Hadoop is more focused on processing then on-demand analysis meaning the execution time may be long and it does not take storing data into consideration.

One of the most commercially successful research into searching large amounts of data is the one that resulted in Google. The researchers have built a distributed system that, apart from easy scaling, have been optimized in many aspects such as minimizing disk seeks in order to cope with the future demands of crawling the ever growing Internet. Google is not only built for efficient searching but also designed to crawl the web in order to dynamically discover documents on the Internet. When Google was researched the field had not come so far regarding scaling data storage solution which therefore had to be looked at. The
researchers were pioneers within the field that would later be known as big data. Google's PageRank algorithm rates documents so that they can later be retrieved based on relevance. The algorithm was at the time of creation a huge improvement to other similar algorithms. [11]

There has been research that used a distributed search engine in order to make spatio-temporal data searchable. They have applied Elasticsearch as the search engine and set up a cluster of virtual machines where each node contains an Elasticsearch instance. The researchers used servers with 32 GB of RAM, indexing over four million records with a response time less than a second. [50]

Apache Solr has been used in order to allow crime data to be searched by the public. A smartphone application was created where users can report new suspicious crimes and plan safe walking routes that take reported crime locations into account. Crimes were uploaded to an Apache Solr cluster that would index the data and make it searchable for the user. Although, the computer system did not use Apache Solr as the primary data storage solution but as a complement in order to allow quick searching. [51]

3.6.1 Usage of Related Work

The most popular choices of distributed search engines have been studied. Depending on the stakeholders' requirements on their analysis problem, an appropriate distributed search engine will be chosen and an implementation will be set up for the stakeholder. Choosing a distributed search engine is a delicate task and options have to be considered carefully.

There are a lot of information about data warehousing and Business Intelligence to widen the understanding of big data. However, the subjects have a different focus on big data since they focus more on understanding data, apart from making it ready for on-demand analysis.

Researchers have used distributed search engines to solve research related tasks. They will be taken into account with regards to what kind of problems they have come across. Difficulties they have come upon may be for instance configuration, performance or other problems that may be of use learning about. By preventing issues with distributed search engines more time can be spent elsewhere.
4 Methods

There are many research methods and all are suitable for different purposes. This chapter explains what methodologies exists, which has been chosen in this thesis and why.

4.1 Research Strategies and Design

Research strategies and designs are guidelines for conducting research methods, as opposed to research methods that define rules for carrying out a research task (see chapter 1.6.2 Research Methods). Different available research strategies and designs include:

- **Experimental research** examines and discovers causality in large volumes of data using statistical analysis. This method is designed to verify or falsify hypotheses.
- **Ex post facto research** is the same as experimental research differentiating itself by trying to find hidden causality by looking back in time of already collected data.
- **Surveys** discover frequency and relationships between variables by collecting data of a population using cross-sectional or longitudinal method. Cross-sectional collects data at a single point of time and longitudinal over a time period. Can be used with quantitative and qualitative methods.
- **Case studies** tries to understand a real-life event where the context is not entirely known.
- **Action research** investigates problems by executing a systematic action and observing the outcome followed by an evaluation. Improves how issues are addressed and solved.
- **Exploratory research** finds as many relationships in different variables as possible by exploration. Mostly identifies issues (not answers) using surveys and qualitative methods.
- **Grounded theory** constructs a theory by collecting and analyzing data. Discovers inductive theories that allows development of general features for a topic.
- **Ethnography** researches peoples to discover relationships and commonalities using descriptive studies of cultures.

This thesis will use the research strategy *grounded theory* since it will investigate search engines and implement the one that is found best suited to solve the problem for the stakeholder.
4.2 Data Collection

Collecting data for a research project is an important activity. There are six methods for collecting research data: [13]

1. **Experiments** are used for collecting a large data volume for variables.

2. **Questionnaires** collects data by asking questions that are either quantifying or qualifying. Quantifying are closed multiple choice questions with alternative follow-up questions and qualifying are open questions where subjects enter reviews.

3. **Case studies** are in-depth analysis of a small group of participants. Used together with the case study research method (see chapter 4.1 Research Strategies and Design).

4. **Observations** collects data by observing behavior with focus on participation in situations or behavior in a culture.

5. **Interviews** gives a deep understanding and captures the participants point of view. Can be structured, semi-structured or unstructured. Structured interviews prepares questions and gives participants exactly the same interview, semi-structured allows new ideas to arise during a structured interview and unstructured does not prepare questions at all.

6. **Language and text** interprets meanings in language, i.e. conversations, texts and documents.

Data will be collected using a literature study. To understand the stakeholders requirements semi-structured interviews will be conducted with employees. The data will be interpreted and filtered through a set of established requirements, in order to find a theory of what method that will solve the problem of analyzing large volumes of data. The method will be implemented and validated by issuing a case study for employees at the stakeholder.

4.3 Data Analysis

After data collection the it should be analyzed in order to review, clean, transform and model it. There are five methods for analyzing data: [13]

1. **Statistics** does calculations on a data population and analyzes the results.

2. **Computational mathematics** uses algorithms, numerical- and symbolic methods to perform calculations or to build models or simulations.

3. **Coding** turns qualitative data into quantitative by analyzing using statistics, such as analyzing transcripts of interviews and observations.

4. **Analytic induction and grounded Theory** collects data and analyzes it to verify or falsify a hypothesis. The method is iterated until a solution has been found.
5. **Narrative analysis** analyzes test and documents using hermeneutic or semiotic methods using literary discussions and analysis. Usages includes tracing requirements and interfaces.

Data will be analyzed using a *analytic induction and grounded theory approach*. Data will be collected to build the solution theory, which will then be verified.

### 4.4 Quality Assurance

Quality assurance is the last step to test and verify the outcome of a research. The experimenter must always consider how ethically correct the research is.

Quantitative research needs validation and assessment to make sure that it is valid, reliable and reproducible. For clarification, these are word definitions with respects to quantitative research: [13]

- **Valid** - Make sure that tests are measuring what it is expected to measure.
- **Reliable** - How consistent the results are for every test.
- **Reproducible** - Possibility to reproduce the results by repeating the same research process.
- **Ethics** - Moral principles of conducting and reporting research, such as protection of participants, maintaining privacy, avoiding forceful participation, written consents from participants and making sure the research is confidential.

Qualitative research must be assessed to make sure that the outcome is valid, dependable, able to be confirmed and transferable. For clarification, these are word definitions with respects to qualitative research: [13]

- **Valid** - A valid qualitative research result should not be open for interpretation. The results are validated and confirmed in order to make sure no other interpretation exists, also known as trustworthiness.
- **Dependable** - Process of judging how correct the conclusions are.
- **Able to be confirmed** - Research should not be conducted with personal assessments affecting the results.
- **Transferable** - Research descriptions should be elaborate so other researchers can use it during their research.

In order to ensure quality a quantitative method will be used in order to build a theory of what system will solve the problem of preparing [BLD] for on-demand analysis. The theory will then be lastly verified using quantitative methods. Tests will be established and valid, reliable and reproducible. Ethical aspects will be considered in order to make sure that no confidential or private data exists.
4.5 Software Development Methods

The following chapter contains brief descriptions of different software development methodologies. Only the basics of each method will be covered.

4.5.1 Waterfall Method

Figure 4: Phases of the waterfall method used in software development.

The waterfall method splits software development into the five phases. A completed phase is final and no time should be allocated going back to a previous phase, see figure 4. The waterfall model phases are:

1. **Requirements**
2. **Design**
3. **Implementation**
4. **Test**
5. **Support**

During the requirements phase (1) the team communicates with all stakeholders in order to establish a complete set of requirements. The team then designs the computer system, during the design phase (2), by splitting it into components and creates use cases, UML-diagrams and flowcharts for each component. The design phase purpose is to make it easy to translate the computer system into code. When it is time for implementation phase (3) the team then translates the design into code. During the test phase (4) all components are integrated with each other. The support phase (5) starts after deployment,
when end-users are using the software. Maintenance is now required. It is common that the implementation team hands over the responsibility to a support team. [52]

4.5.2 Scrum

Scrum is an agile development method that can be broken down into three “concepts”:

1. **Three roles** - Product owner, team and scrum master.
2. **Three documents** - Product backlog, sprint backlog and sprint results.
3. **Three meetings** - Sprint planning meeting, daily scrum meeting and sprint review.

A development team is in charge of developing and testing the system. One product owner acts as the team and communicates with the stakeholders. A scrum master solves any team issues or problems, supervises the scrum process and adapts it to best fit the organization. The product backlog contains all the stakeholder requirements. Development is done in iterations also known as *sprints*. Each sprint contains a *sprint backlog* which is the subset of product backlog requirements that are chosen to be solved during the current sprint.

Each sprint starts with a meeting called the *sprint planning meeting*. The sprint backlog is decided based on entry priority set by the product owner. Every day the scrum master gathers the team for a short meeting called the *daily scrum meeting*, during which each team member answer the question: [52]

- What have I worked on?
- What issues have occurred?
- What am I working on next?

Sprint ends with a *sprint review* where everyone discusses the results. Each sprint should end with a working deliverable version of the product. Scrum iterates this process until the product is finished and deployed. [52]

4.5.3 Kanban

The Kanban software methodology revolves the *Kanban board*. The board is used to provide transparency of what needs to be done, ensure focus, provide traceability and to detect blocking work items fast. [53]
Work items are split into cards that is attached to the board. A card has the state to do, in progress or done (see figure 5). Teams may use as many states as they want. A card also contains information such as:

- Who is responsible for it.
- Short description of the work.
- Estimated time it takes to finish.

The information that a card contains should be so sufficient that a person such as a product owner can understand the work, how long it will take and prioritize it accordingly.

4.5.4 Software Development Method Chosen for this Thesis

Software developed during the thesis will be using the Kanban software methodology. Kanban uses a continuous workflow, instead of shorter iterations like Scrum, and is therefore a better fit for this thesis.
5 Handling Big Data using a Distributed Search Engine

In this chapter the process of selecting a search engine will be presented. The choice will vary depending on stakeholder requirements. The chapter will explain how requirements was gathered, what the established requirements are and ultimately which distributed search engine that will be chosen for implementation.

5.1 Gathering Requirements

In order to understand which search engine to choose the stakeholder requirements has to be established. They will be gathered by interviewing stakeholder employees in order to get a wide understanding of the problem.

The interviews that was conducted were semi-structured (see chapter 4.2 Data Collection). The questions that was prepared before the interviews were:

Q1. What kind of log data is gathered within the system?
Q2. To what purpose do you use the log data?
Q3. What manual procedure do you follow when analyzing the log data?
Q4. How long does it take to complete the manual procedure?
Q5. How much data is being generated?
Q6. Can the data contain sensitive information, such as can it be tied to an individual?
Q7. What kind of hosting do you allow (cloud, in-house, etc.)?
Q8. What is the most important feature of this system?
Q9. Is it important that there exists some kind of enterprise support (official or third-party)?
Q10. Do you require third-party software to adhere to a specific license?

Before an interview the interviewer asked for interviewee consent and explaining the research purpose. The interviewee was made aware that the end results of the research will be a public document, but the interview will be kept strictly confidential and no personally identifiable information will be documented. Records were kept during the interviews and can be found in appendix B Requirements Interview 1 and C Requirements Interview 2.

5.2 Established Requirements

The interviews was conducted on two employees at the stakeholder. These are the most important features of the system:
• Parsing log files that stores key/value data.
• Strong filtering capabilities for keys.
• Able to self-host the system.
• Must be distributed with a liberal license.

The important features have no flexibility and must be fulfilled. For instance, the data acquired by the stakeholder may be highly sensitive so the search engine must be self-hostable since the stakeholder is bound by contract to host it themselves. The filtering possibilities that has been deemed extra important is:

• Which user that executed the operation.
• At what time the operation was executed.
• The type of operation that it was.

The stakeholder expressed that they needed as strong filtering capabilities as possible.

5.2.1 Log File Generation Rate

The amount of data that the stakeholder generates is approximately 350 MB each day (see equation 5), which means 10.5 GB each month (see equation 6). Numbers can be referenced to the second interview (see appendix C Requirements Interview 2).

\[ 5MB \times 70 = 350MB \]  
(5)

\[ 350MB \times 30 = 10.5GB \]  
(6)

5.2.2 Log File Format

The log files that will be are generated by operations executed on a database. An operation can be e.g. a delete or update action. The log files are stored in a proprietary binary format and therefore has to be processed by a decoding software in order to convert it into text. The decoding process can be configured to output all data or a subset of it. If the process is not configured the standard format of each operation looks like this:

```
----------------- OPERATION 000001 ----------------
Create Time :Sat Feb 4 22:03:15.813482 2017
Start Time :Sat Feb 4 22:03:15.813503 2017
End Time :Sat Feb 4 22:03:15.813659 2017
OpUUID :2ed26cce-426f-4443-8767-db02eedbc668
DapBindId :286c0053
Concurrency :1
```
OpStackSize : 1
OpFlow In/Out : 0/0
Duration : 0.000156 sec
User : cn=portalproxyuser, ou=users, ou=system_logins, o=stakholder, o=SE, o=EDIRAroot
IP+Port+Sd : [127.0.0.1]+39308+77
Op-Name : LDAP_Con888850_Op0
Operation : BIND
Version : 3
MessageID : 1
Bind-Type : simple
Security : normal
DAP-Share-Count: 4
Bytes Received : 90
Bytes Returned : 29
Socket Mode : plain
Abandoned : no
Result Code : 0 (success)
Error Message : Bind succeeded.

The data in an event is stored as key/value pairs and the keys varies depending on the operation. Configuration of the decoding process includes specifying the output format and keys to extract. The decoding result can be formats such as CSV. The stakeholder provided 976 files that occupied approximately 40 GB of memory. Each log file contains at the most 50 000 events.

### 5.3 Search Engine Selection

This chapter will go through the process of selecting a distributed system reported in previous chapters (see Search Engines). The selection will be governed by the stakeholder requirements (see chapter Established Requirements).

<table>
<thead>
<tr>
<th>Search engine</th>
<th>Distrib.</th>
<th>Analysis</th>
<th>Index</th>
<th>Query</th>
<th>License</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache Lucene</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Elasticsearch</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Apache Solr</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Which distributed search engine fulfill the requirements of the stakeholder.

Table summarizes all the distributed search engines and what features that is considered to be fulfilled with regards to stakeholder requirements. The rest of this chapters will review each search engine and conclude why they fulfill the requirements or do not.
5.3.1 Evaluation of Distributing the System

Apache Lucene is a programming library and provides no out-of-the-box distributed search engine. Developers can use it to integrate search functionality into an existing application, but in this thesis its abstraction level is too low, compared to the alternatives.

Apache Solr is built onto of Apache Lucene. The architecture of Solr can be setup in many different ways. It provides a distributed system architecture using SolrCloud. However, it still contains a single-point-of-failure since it uses Apache ZooKeeper to manage nodes in the system.

Elasticsearch is designed as a distributed system on top of Apache Lucene. All nodes in Elasticsearch is considered equals and if one fails, the responsibility can be passed on to another node and the system can start to rebalanced its replicated data.

It is concluded that both Elasticsearch and Apache Solr meets the stakeholder requirements.

5.3.2 Evaluation of Text Analysis

Apache Lucene provides a powerful search interface. It can analyze text both Swedish and English. It is the users responsibility to use the appropriate text analyzer or create a custom one. Both Solr and Elasticsearch are extensions of Apache Lucene and contains similar features.

Apache Solr supports phonetic text analysis. It includes Swedish and English text analysis and other advanced features.

Elasticsearch is built on top of Apache Lucene and extends its text analyzers. It supports phonetic analysis, Swedish and English analysis.

All search engines will provide enough features in order to fulfill the text analysis requirements of the stakeholder.

5.3.3 Evaluation of Indexing

Apache Lucene uses inverted indexes to store documents. After text analysis, Parsed data is stored in indexed fields so that it can be used in queries.

Apache Solr allows the usage of schema files which will make it easier for the system to understand data. A document can adhere to a schema that, describes the expected fields and its data type. Solr can also index schema-less data. It will then try to draw conclusions what data type a field is.

Elasticsearch stores documents in an inverted index. It recommends splitting indexes into logical groupings of data called types. Types are used to to reduce the amount of indexes.
All search engines can index key value pair of data and provide querying capabilities on that data.

### 5.3.4 Evaluation of Querying

It is required that the solution supports real-time analysis and can filter data in as many ways as possible.

Apache Lucene is a library that exposes an API for integrating search into an application. It only provides primitive data types. More complex types can be utilized by combining primitive types.

Elasticsearch has its own query language using where users can describe a query in **JSON**. It can query data in a scoring and non-scoring way. If formatted correctly, a **JSON** string can be very readable.

Apache Solr also has its own query language that describes queries using an SQL-like syntax, provided as a URL parameter. By combining many primitive building blocks is it possible to filter anything, although providing queries in the URL may be hard to read and, although a suggestion, it is subject to length limits in as stated in the HTTP specification [54].

Apache Lucene provides a low abstraction level on its query component. Elasticsearch and Solr provides a higher abstraction level which will reduce the time it takes to provide powerful filtering capabilities. Although, all search engines provides adequate querying functionality.

### 5.4 Stakeholder Decision

Elasticsearch has been chosen for implementation. Reasons for selecting it is:

- Designed as a distributed system. All nodes have equal responsibility.
- Supports satisfactory text analysis.
- Can index key/value pairs of data in near-real time.
- Contains a powerful **JSON** query language.
- Has an liberal open source license and enterprise support. Users can contribute to the project.
- A widely used system by many large companies.
- When investigating software problems, as much information regarding the problem as possible is of interest.
6 Setting Up an Elasticsearch Cluster

This chapter describes the steps taken in order to setup an Elasticsearch cluster that can be used to index and query data.

6.1 Decoding Log Files

Log files generated by the stakeholders system was saved in a proprietary binary format (see chapter 5.2.2 Log File Format). An application was created that would explore a folder of log files and then hand them over to the decoding software. From the binary proprietary data as much data as possible was extracted. A configuration file for the decoding software was created by reading the official documentation.

Output of the decoding software could be configured. It was decided that the Comma Separated Values (CSV) format was to be used, since it is a generic text format that is easy to parse when indexing the data. CSV is a format that stores data in rows, where each row is an entry that contains values separated by a comma. There exists no official specification and therefore many permutations of the format has emerged. Deciding the exact permutation to use was an iterative task. After a couple of iterations it was determined to use a tab separator.

Decoding all 40 GB of data resulted in a dataset that was approximately 30 GB and the same amount of files. The decoding process took 1 hour which means that each file was decoded in approximately 0.271 seconds (see equation 7).

\[
\frac{976}{60 \times 60} \approx 0.271 \text{ seconds}
\]  

(7)

6.2 Indexing Software

In order to index data in the Elasticsearch cluster an indexing software was created. The purpose was to read the decoded CSV log files and upload them to the cluster. The software would read all decoded CSV files in a folder and execute a PUT request to the Elasticsearch cluster for each log event.

6.3 Hardware

Servers from the information technology infrastructure provider Amazon Web Services (AWS) was used. A Virtual Private Cloud (VPC) created an isolated environment within AWS, with its own firewall, where the cluster could reside. Three ports was allowed through the firewall:
1. **Port 22** - Connect to the node using Secure Shell (SSH) in order to provision and administer the node.

2. **Port 9200** - Default port for external clients to communicate with the Elasticsearch Representational State Transfer (REST) Application Programming Interface (API).

3. **Port 9300** - Default port for node communication within an Elasticsearch cluster.

The amount of ports to open was kept at a minimal and only the most important ports was opened.

Elastic Compute Cloud (EC2) is a service from AWS that provides technological infrastructure. Three nodes was set where each had 1 CPU, 2 GB memory, Ubuntu Server 16.04 LTS and 58 GB General Purpose SSD (GP2). The resource configuration is identified as t2.small by AWS [56]. The resources was chosen in order to fit a budget of approximately $100. The chosen operating system was Ubuntu long term support (LTS) by Canonical. Canonical releases a new version of Ubuntu every six months, but LTS versions are only updated once every two years which results in a more stable operating system [57].

Programming, indexing and testing tasks was run on an development computer, which had 2 CPUs, 12 GB memory, Windows 10 and a 246 GB General Purpose SSD.

### 6.4 Provisioning a Node

A newly launched node had to be provisioned in order to provide the needed functionality. A process for provisioning the nodes was automated by creating a script in order to allow for easier scaling (see chapter 6.4 Provisioning a Node). The following steps was taken on a node in order to provision it:

1. Update the operating system to the latest version.
2. Download and install Java and Elasticsearch.
3. Download configurations for Elasticsearch.
4. Download a run script.
5. Restart the node.

The operating system was updated to the latest version as it can be considered good practice when setting up new servers. The installed applications are the minimum that are required in order for the implementation to work. Configurations was downloaded, as opposed to configured, in order to create a way of batch configuring nodes and to eliminate the possibility of errors. The run script would download the latest configurations before running the software in order to easily be able to reconfigure the nodes.

A private package repository was set up for hosting the software and configurations needed in this thesis. The reason for privately hosting them was to make sure that new versions of the software would not break the provisioning process. Also, be able to host out-of-the-box configuration files, as opposed to configuring each node independently.
Java was the only package that was installed from Ubuntu's official repository. Altering the default configuration was kept at a minimum. The changed configurations were: [58]

- **Cluster name** - Can be set to anything.
- **Node name** - Unique name to the node. Configured to be the hostname of the node.
- **Network host** - Which network interfaces to bind the server to. Configured to bind to all interfaces.
- **Unicast hosts** - When a node starts it uses a list called unicast hosts in order to establish a connection to the cluster. The list can contain a subset of the nodes available in the cluster or all of them.
- **Minimum master nodes** - Minimum master configuration dictates how many nodes that needs to be connected in order for the cluster to be valid. This is done in order to ensure that the cluster does not get partitioned.
- **Java Virtual Machine (JVM)** - The JVM had to be configured and half of the node RAM was allocated and locked to the JVM, as recommended by the official documentation. [59]

The amount of changes to the configuration was kept at a minimum. The changed configurations are the most important and required changes in order to have a functional cluster.

### 6.5 Indexing Performance

Tests were executed in order to determine how the setup performed. 1000 log events would be indexed on three different cluster configurations using both a horizontal- and vertical scaling technique.

#### 6.5.1 Horizontal Scaling

Three tests, each with the following cluster configuration, would be executed to test the horizontal scaling test:

- Cluster of 1 node
- Cluster of 2 nodes
- Cluster of 3 nodes

A node would have the same hardware as described in chapter 6.3 [Hardware].
The test took 1176 seconds to complete on a cluster of one node, 1156 seconds on two nodes and 1181 on three. Test results of the horizontal scaling test can be viewed in table 3. The results has also been plotted and can be viewed in figure 6.

### 6.5.2 Vertical Scaling

Three tests were executed using vertical scaling, each run on a cluster consisting of one and the same hardware as described in chapter 6.3 Hardware. However, the amount of memory available to the node would be altered:

- 1 GB memory
- 2 GB memory
• 4 GB memory

<table>
<thead>
<tr>
<th>Processing time (s)</th>
<th>1 GB</th>
<th>2 GB</th>
<th>4 GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1176</td>
<td>1156</td>
<td>888</td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Processing time when vertically scaling an Elastic Search cluster.

Figure 7: Processing time when horizontally scaling an Elastic Search cluster.

The test took 1176 seconds to complete on the 1 GB node, 1156 seconds on two and 888 on three. Results of the horizontal scaling test can be viewed in table 4. Test results has also been plotted and can be seen in figure 7.

6.5.3 Total Indexing Time

Each log file contains 50 000 events and there were about 974 log files. Therefore there was about 48700000 log events to index, according to equation 8.

\[ 50000 \times 974 = 48700000 \]
Assuming that the best indexing test result, i.e. an time of 888 seconds for 1000 events, scales linearly. It would then take $43245600$ seconds to index the all data (see equation (9)).

$$\frac{888}{1000} \times 48700000 = 43245600$$ \hspace{1cm} (9)

Which means that it would take approximately 500 days to index the data (see equation (10)).

$$\frac{43245600}{60^2 \times 24} \approx 500$$ \hspace{1cm} (10)

6.6  Real-Time Requirements

Real-time processing of events was not implemented. However, the indexing time for a real time requirement can be theorized upon. Assume the best indexing test result, i.e. an time of 888 seconds for 1000 events, scales linearly and that the decoding process can be neglected. That would mean one event takes $0.0148$ minutes to index as seen in equation (11).

$$\frac{888}{1000 \times 60} = 0.0148$$ \hspace{1cm} (11)

The test consisted of a node with 4 GB RAM. Assuming that the RAM scales linearly, it means that the cluster would need 185 GB of RAM (see equation (12)) in order to handle 50 000 events every minute.

$$\frac{50000 \times 0.0148}{4} = 185$$ \hspace{1cm} (12)
7 Verifying the Elasticsearch Implementation

In order to verify the solution, a case study was conducted with a participant from the stakeholder. As preparation for the case study, the graphical user interface Kibana was set up (see chapter 3.5.3 Elasticsearch for more information on Kibana). Kibana was launched on a separate node in AWS using the hardware described in the results (see chapter 6.3 Hardware). The software was hosted in the private package repository and downloaded onto the node and these configurations was changed:

1. Bind to all IPs.
2. URL to an Elasticsearch cluster.
3. Server name
4. Log file destination

Kibana was used by the case study participants in order to query the log data, instead of issuing calls directly to the REST API. A demonstration conducted where these two queries was shown:

- user:*cybercom*
- opType: BIND

The demonstration purpose was to educate participants the basic behind querying. Hopefully, the partaker would find it intuitive to issues their own queries afterwards. Participants thereafter got time to get accustomed with Kibana before giving an oral review. The case output was positive.
8 Conclusions

This thesis has presents how to select a distributed search engine with the purpose of being able to investigate on-demand. The problem statement of "How can software developers handle big data using distributed search engines and make it searchable on-demand, for analytical and investigative purposes?" is answered by having implemented an Elasticsearch cluster that has indexed big log data making it efficiently searchable.

This chapter discusses the results, evaluates the verification and work that can further expand on the research.

8.1 Evaluation of the Elasticsearch Implementation

Elasticsearch scales best horizontally. It could be an indication that the cluster did not have sufficient resources. It would have been more fair to the results if more resources was used and bigger ranges RAM and CPUs when comparing. Due to economical reasons it was not possible and the only possibility was to theorize on the scaling. Theories results are quite naive as they assume that the scaling would be linear compared to what the results showed. More testing is need in order to verify or falsify that hypotheses. The total cost of the cluster used in this was approximately 100 USD.

Due to more resources not being available the focus shifted to providing a basic approximation of how the system would scale. Elasticsearch themselves recommends nodes that has 4-32 GB memory [59] which might be an indication that more resources should have been used. The test that outperformed was the one within the lower bounds of the recommended RAM range, i.e. 4 GB of RAM.

The Elasticsearch cluster was validated using a case study with participants from the stakeholder. It was a good way to verify if the Elasticsearch implementation would suffice for their needs. Details regarding the implementation will depend on the stakeholder requirements, there are many solutions to handling big data and not just only distributed search engines. Elasticsearch only provides a REST API to query data. The graphical user interface Kibana is a separate product. If Kibana would not have been set up before validating the solution it would have risked the participants focusing on the tediousness of executing REST API queries. Instead the focus was intentionally placed on demonstrating the Elasticsearch implementation.

8.2 Discussions

The entire data set of 40 GB given by the stakeholder was never indexed as it was not expected to take 500 days. A first iteration of the indexing software was executed on a dedicated server on AWS. The data set was transfered to the instance so that indexing could execute locally. After a day or two the server was checked for progress and it was concluded that the indexing would take too long. Some tests was done in order to try and enhance the
indexing performance. What if multiple threads could be utilized in order to load balance requests sent to the cluster. Requests was load balanced in a round-robin fashion to the clusters so that all nodes would get a subset of the indexing work. Multiple threads did not yield any noticeable performance gains. After experimentation it was decided to instead index 1000 events and theorize the total indexing time and hardware requirements. A theoretical study that determined total indexing time and hardware requirements should have been done as part of this thesis, before starting to index the entire data set.

The decoding process is not common in these kind of scenarios which increased the problem complexity. Decoding could be configured to output different formats. The CSV format was chosen since it is simple and comprehensible. Writing a customer parser results in a flexible solution that can be adapted to many situations.

Nodes on AWS downloaded software hosted in a private repository. The purpose being that softwares are regularly updated which could break the solution mid-research. A thesis research stretches multiple months and time could possibly be saved by keeping software versions static throughout it. Java was downloaded from the official Ubuntu repository since it was an LTS versions of the operating system and deemed less likely to bring breaking updates.

Early on in the results development an automated script was created. It was a prioritized task since being able to efficiently add and remove nodes from a cluster is an important aspect of scalability. It defeats the purpose if a new node takes long time to configure and deploy.

It would be interesting to look at the option of comparing the Elasticsearch implementation to a similar solution of data warehousing or Hadoop. Distributed search engines needs to store big data which arguably makes the field a superset of the data warehousing field. It would not be surprising if the theory behind distributed search engines builds upon data warehousing since the field is much newer. Hadoop focuses more on the processing part of big data and leaves the storing entirely up to the user. Hadoop has its own file system called Hadoop Distributed File System which can be used to store big data, and the output of Hadoop processing, in a replicated and fault tolerant way. Using Hadoop Distributed File System is not mandatory and it is built as a separate system which can be used for anything the user may need it for. Maybe a distributed file system is the result of combining the theory behind data warehousing and big data processing such as Hadoop.

The research has been conducted ethically. No personal identifying information from interviews has been leaked and consequences of the results is not unethical or immoral. The stakeholder had no interest in interfering with the research and it was conducted in an objective way.
8.2.1 Importance of a Distributed Architecture

Elasticsearch was chosen to be used when setting up a search engine. Apache Lucene was not an option since it was not a distributed search engine, but Apache Solr were considered. Elasticsearch is designed as a true distributed system, no single point of failures and each node has the same privileges. Scaling is important when dealing with big data as the computer system will eventually run out of resources. The fact that Apache Solr is not designed as a distributed system was considered a minor weakness. However, both choices are used by large companies that most likely need to handle big data so both would probably have sufficed.

8.3 Future Work

The entire data set could be indexed in order to get a better understanding of the hardware requirements to index BLD. The tests could be more thorough by using nodes with more resources. It would be interesting to investigate how 32 GB memory nodes could perform as opposed to servers with 4 GB.

There are other search engine that can be of interest to investigate. Splunk is a proprietary search engine that is created solely for the purpose of indexing log files. Elasticsearch is marketed as an open-source search engine while Splunk is a software-as-a-service. Splunk can also be self-hosted if needed but it is entirely proprietary.

More investigation has to be put into researching real-time requirements. There exists plenty of softwares on the market that purposely indexes data in real-time. For instance, Elasticsearch recommends Logstash (developed by the same company) which is an agent that lives on the server. It monitors files and as new events are generated it can upload the data into an Elasticsearch cluster. Logstash has to be configured in order to fit the log data. Logstash was not used in this research since the decoding process existed. The agent is not built for pre-processing files, although it possibly exists a way to extend the software to do this. However, the problem scope becomes bigger. Having to decode files log files has a performance impact that needs to be investigated. [62]

The research took no consideration of data archiving. Eventually log data will become outdated and in need of storage or removal. A process that dictates how this is done needs to be setup. How this should be implemented is left as future work.

8.3.1 What to do next

To use the results in a real world setting the search engine would have to be secured. Anyone that has access to the Internet address can execute requests on the cluster. It would have to be secured behind a firewall or integrated with the stakeholders single sign on solution.
There would have to be an investigation on how much resources the cluster needs in order to handle the requirements of 50,000 events per minute. The theory from this thesis could be used as a baseline for an investigation but they are too naive viewed upon as final numbers.

Events should be visible in real-time as they are generated. There would need to be research on how to efficiently decode log files and then uploading them to the cluster. Instead of using the indexing software from this thesis, there already exists computer programs for the purpose. They are agents that lives on serve, monitoring log files and executes procedures if it detects new events.
References


Appendices

A Software Licenses

Systems described in this thesis will have a license attached to it for usage. This appendix describes the licenses that are mentioned in this thesis.

A.1 Apache License 2.0

Apache License 2.0 (ALv2) is a liberal license that allows the usage of software under these conditions:

- The top directory of the software needs to include a file named LICENSE that must contain the ALv2.
- The top directory of the software needs to include a file named NOTICES that must contain the copyright information and the name of the product.

The full license can be viewed at the Apache website.

4https://www.apache.org/licenses/LICENSE-2.0.html
B Requirements Interview 1

This is the first interview conducted when gathering the stakeholders requirements on the system. It was a semi-structured interview conducted in a controlled environment. Text in italic represents follow-up questions made by the interviewer.

Q1. What kind of data is gathered within the system? We gather log data from one of our systems. It logs operations executed in the system containing all kinds of information such as:

- Where the call came from/originated.
- If it was issued by the system or a user.
- If it was an update, what was the old and what is the new value.
- If it was a search, what did it look like.

Does the data contain stack traces? No, it is only database (LDAP) operations.

Some data is logged for performance reasons. Since our customers are also developing against this database we need to be able to see how many connections they are opening and how many they close, so that the database does not get flooded.

Q2. To what purpose do you use the log data? We use the data to investigate errors and to check that everything is in order.

Q3. What manual procedure do you follow when analyzing the log data? We e-mail the user that detected the error and try to get as much information as possible from them, such as exact time it occurred, date, how did you interact with the program etc. Getting detailed information from a user is hard.

The name of the log file contains a timestamp. We retrieve all the logs from the date that we are investigating and decode them.

What is decoding? LDAP saves log files in there own binary proprietary format. Included in the program is a decoder that will convert the logs in a text format.

Anyway, we then search through the logs, CTRL+F style using our favorite editor, using queries such as ”user=niklas” and try to find the operation(s) that failed. If the error report from the user was detailed we can filter and narrow down the search much better.

Q4. How long does it take to complete the manual procedure? Between 15 minutes and 2 hours to localize the problem. After that we determine if the cause was due to our code or others code.
Q5. How much data is being generated? I do not know exactly, but it is a lot.

Q6. Can the data contain sensitive information, such as can it be tied to an individual? On a scale of one to five it is a three. Hm, well since update operations can contain values it may be a five. Our customers are municipalities (kommuner) and country councils (landsting), so the data can be highly sensitive.

Q7. What kind of hosting do you allow (cloud, in-house, etc.)? This system is hosted locally by our customers. We used to host it but it was later moved to another SUA classified third-party hosting.

What is SUA classification? In order to host this data from government entities an organisation needs to be SUA classified\(^5\).

Other systems that is developed in-house are usually hosted in the cloud.

Q8. What is the most important feature of this system? A powerful filter so that the query can be narrowed down as much as possible. It should support phonetic searching. The log data should be able to be queried on the same day.

Q9. Is it important that there exists some kind of enterprise support (official or third-party)? Not overly important since we can go back to our old procedure whenever. In general we evaluate the need for paid support from project to project.

Q10. Do you require third-party software to adhere to a specific license? It depends on the customer and is included in the project agreement if they exist.

\(^5\)SUA classification is a safety protocol issued by the security police (SÄPO).
This is the second interview conducted for the purpose of gathering the stakeholders requirements. It was a semi-structured interview conducted in a controlled environment. Text in *italic* represents follow-up questions made by the interviewer.

**Q1. What kind of data is gathered within the system?**  We gather audit logs. It contains information of all operations executed against the catalog/database. Operation details contains information such as:

- Who executed the operation.
- At what time was the operation executed.
- How long time the operation took to execute.

*Do you have any kind of alert if a slow operation occurs?* We used to have it when the system was hosted in-house. There was a TV showing performance information that would blink red and make sounds when slow operations occurred. But we removed it because people found it annoying.

**Q2. To what purpose do you use the log data?**  We use it to make sure everything is working as intended. If errors occurs they are used for investigative purposes, to locate the problem. They are stored for one year before being archived.

**Q3. What manual procedure do you follow when analyzing the log data?**  First we decode the data. The hosting provider automatically decodes the logs and uploads them to a repository that we have access to. There logs are stored with the date in the name and compressed using the GZIP algorithm. Locate the log that you are after and use whatever tool to search in the file, it is a Linux server so you can use whatever suits you.

**Q4. How long does it take to complete the manual procedure?**  From 2 minutes to 2 weeks. It depends on the problem and handling this huge amount of data is clumsy. Some errors could be related to external systems which makes the investigation more time consuming.

*Does every developer have their own way of searching in this data? Your answer is quite different from my other interview. I have also heard that one of your developer created a tool for searching these log files.* There is an official program that are used for searching when we need to search a lot of log files.

**Q5. How much data is being generated?**  Let’s find out! Every file is about 5 MB compressed and there are about 70 files every day.
Q6. Can the data contain sensitive information, such as can it be tied to an individual? There can be sensitive data, but you can not see any values in the operations.

Can you see the old and new value of an update operation? No... I do not think so at least. As far as I can remember you can not.

If there is any sensitive information in the operation it is written in a very obvious way.

Q7. What kind of hosting do you allow (cloud, in-house, etc.)? We use the cloud for general systems. This system is hosted by a third-party hosting company issued by the client.

Q8. What is the most important feature of this system? To be able to search. Important filters are timestamp, which user that executed the operation and what kind of operation it was. Overall good filtering possibilities.

We use a program called ClickViewer to display the operations in a statistical way.

Q9. Is it important that there exists some kind of enterprise support (official or third-party)? It depends if the system is integrated with a business critical application or if it is more of a utility system, working independently. It is always good to be able to buy support.

Do you have a general policy for including open-source and enterprise support? We use a lot of open-source and there is no general policy. It depends on the project.

Q10. Do you require third-party software to adhere to a specific license? We check the licenses of every open-source project we include in our daily work. Especially if we are to hand over the system to a customer that deploys it themselves.

Do you have a general policy for license? Like we only include MIT? We do not have a general policy. As a developer we check the licenses ourselves. If there are uncertainties whether that something can be used we hand the license over to a lawyer.