

Aalto University  
School of Science  
Master's Programme in Computer, Communication and Information Sciences

Niklas Harjunpää

# **Log management system technologies and methods for near real-time fault analysis systems:**

## **An exploration of log shipping and storage**

Master's Thesis  
Espoo, May 26, 2023

Supervisor: Dr. Matti Siekkinen, Aalto University  
Advisor: Dr. Matti Siekkinen

<b>Author:</b>	Niklas Harjupää	
<b>Title:</b>	Log management system technologies and methods for near real-time fault analysis systems: An exploration of log shipping and storage	
<b>Date:</b>	May 26, 2023	<b>Pages:</b> 67
<b>Major:</b>	Computer Science	<b>Code:</b> SCI3042
<b>Supervisor:</b>	Dr. Matti Siekkinen	
<b>Advisor:</b>	Dr. Matti Siekkinen	
	<p>Log management is a process that allows the collection, processing, storage, and analysis of log files generated by various applications and systems. Maintaining system security, performance, and reliability has become increasingly important. However, logging and the infrastructure supporting it are still in their early stages of development compared to log analysis. Furthermore, the existing infrastructure-related resources are becoming outdated and need to be updated to ensure the current applicability of the employed methods and technologies.</p> <p>This thesis investigates the methods and technologies employed by log management systems suitable for near real-time fault analysis. It presents a methodology used for selecting log management systems for an in-depth review and provides insights into the methods and technologies employed for log shipping, storage and archival. A technical documentation review approach of the available log management systems on the market was conducted to achieve the objectives.</p> <p>The results show that the most commonly used shipping technologies were custom agents, Fluent-based and Elastic-based shippers, creating a shift towards guaranteed delivery. In addition, the thesis found that JSON and Regex were the most supported parsing methods. Popular log storage systems are built with object storage or search engines, all supporting replication, which is critical for high-uptime systems. All assessed systems employed time-based data retention methods, and object storage with a two-tiered approach was the preferred structure for archiving data. The thesis provides practitioners and researchers with a resource for understanding and creating more robust and efficient log management systems.</p>	
<b>Keywords:</b>	log management system, log shipping, log storage, log archival, log architecture	
<b>Language:</b>	English	

Aalto-universitetet  
 Högskolan för teknikvetenskaper  
 Magisterprogrammet i data-, kommunikations- och infor- SAMMANDRAG AV  
 mationsteknik DIPLOMARBETET

<b>Utfört av:</b>	Niklas Harjupää		
<b>Arbetets namn:</b>	Logghanteringssystem teknologier och metoder för nästan realtidsfelanalysystem: En utforskning av loggsändning och logglagring		
<b>Datum:</b>	Den 26 Maj 2023	<b>Sidantal:</b>	67
<b>Huvudämne:</b>	Datateknik	<b>Kod:</b>	SCI3042
<b>Övervakare:</b>	Dr. Matti Siekkinen		
<b>Handledare:</b>	Dr. Matti Siekkinen		
<p>Logghantering är en process som möjliggör insamling, bearbetning, lagring och analys av loggfiler skapade av olika applikationer och system. Logghanteringssystem kan användas för många olika syften som systemsäkerhet, prestanda och pålitlighet. Loggning och den stödjande infrastrukturen är dock i ett tidigt utvecklingsteg jämfört med loganalys. Dessutom är existerande infrastrukturellerade resurser i flesta fall föråldrade och behöver uppdateras för att säkerställa fortsatt användbarhet av de metoder och teknologier som presenterats.</p> <p>Denna avhandling undersöker de metoder och teknologier som används av logghanteringssystem lämpade för nästan realtidsfelanalys. Avhandlingen presenterar en metodik som har använts för att välja logghanteringssystem för djupgående granskning och ger insikter i de metoder och teknologier som används för loggsändning, lagring och arkivering. Dessa mål av avhandlingen uppnåddes genom granskning av teknisk dokumentation av de tillgängliga logghanteringssystemen på marknaden.</p> <p>Resultaten visar att de vanligaste loggsändnings teknologierna var skräddarsydda agenter, Fluent-baserade och Elastic-baserade sändare, vilket visar en förändring mot teknologier som stöder garanterad leverans. Dessutom befann avhandlingen att JSON och Regex var de mest stödda loggparsnings metoderna. Logglagringsystemen är byggda med objektlagring eller sökmotorer. Alla utvärderade systemen stöder replikering av data, som är ett viktigt koncept för system som kräver hög tillgänglighet. Alla utvärderade system använde sig av tidsbaserade arkiveringsmetoder och objektlagring med två nivåer för dataarkivering. Avhandlingen ger forskare och praktiker en resurs för att förstå och skapa mer effektiva logghanteringssystem.</p>			
<b>Nyckelord:</b>	logghanteringssystem, loggsändning, logglagring, loggarkivering, loggarkitektur		
<b>Språk:</b>	Engelska		

# Abbreviations and Acronyms

ACID	Atomicity, Consistency preservation, Isolation, Durability
BASE	Basically Available, Soft state, Eventually consistent
CAP theorem	Consistency, Availability, Partition tolerance - theorem
COTS	Commercial Off-The-Shelf
DBMS	Database Management Systems
ELT	Extract, Load, Transform
ETL	Extract, Transform, Load
GCS	Google Cloud Storage
GDPR	General Data Protection Regulation
IoT	Internet of Things
KVP	Key-Value Parsing
NIST	National Institute of Standards and Technology
Regex	Regular expressions
SaaS	Software-as-a-Service
SIEM	Security Information and Event Management
SLA	Service-Level Agreement

# Contents

Abbreviations and Acronyms	4
<b>1 Introduction</b>	<b>7</b>
1.1 Problem statement	8
1.2 Structure of the Thesis	8
<b>2 Background</b>	<b>10</b>
2.1 Logging	10
2.1.1 What is a log?	10
2.1.1.1 Log file standards and severity levels	12
2.1.2 Why log?	13
2.2 Log Management Systems	14
2.2.1 Log Generation	16
2.2.2 Log Shippers	17
2.2.2.1 Data transformation and reduction	17
2.2.2.2 Data parsing	18
2.2.2.3 Data transportation	19
2.2.2.4 Delivery guarantees	20
2.2.2.5 Data processing	21
2.2.3 Log Storage	22
2.2.3.1 Data lifecycle	22
2.2.3.2 Storage foundations	24
2.2.3.3 Storage types	25
2.2.4 Log Analysis	27
2.2.5 Challenges of Log Management	27
2.2.6 Design of a log management system	28
2.2.6.1 All-in-one solutions and SIEMs	29
2.2.6.2 Best-of-breed solutions	30
2.3 Related Work	30

<b>3</b>	<b>Methods</b>	<b>33</b>
3.1	Near real-time fault analysis log management systems . . . . .	34
3.2	Use case definition . . . . .	35
3.2.1	Use case description . . . . .	35
3.2.2	Policy creation from use case requirements and description . . . . .	37
3.2.2.1	Log generation policies . . . . .	37
3.2.2.2	Log shipping policies . . . . .	39
3.2.2.3	Log transformation policies . . . . .	39
3.2.2.4	Log storage policies . . . . .	40
3.2.2.5	Log analysis policies . . . . .	40
3.3	Initial selection method . . . . .	41
3.4	Software elimination method . . . . .	42
3.5	In-depth review method . . . . .	43
<b>4</b>	<b>Results</b>	<b>44</b>
4.1	Implementation details . . . . .	44
4.2	Data shipping methods and technologies . . . . .	46
4.2.1	Shipping technologies . . . . .	46
4.2.2	Fault-tolerant shipping technologies . . . . .	49
4.2.3	Parsing methods . . . . .	50
4.3	Data storage methods and technologies . . . . .	51
4.3.1	Storage systems and models . . . . .	51
4.3.2	Storage consistency and replication . . . . .	54
4.4	Data archival methods and technologies . . . . .	54
<b>5</b>	<b>Discussion</b>	<b>57</b>
5.1	Reflection of implementation . . . . .	57
5.2	Reflection of the results . . . . .	58
5.3	Future work . . . . .	59
<b>6</b>	<b>Conclusions</b>	<b>61</b>

# Chapter 1

## Introduction

Modern software systems are essential to our daily lives, and even brief periods of system downtime may result in significant economic repercussions. Log management systems are employed as a means of mitigating system outages. These systems are designed to collect, process, and store log files generated by various applications and systems so that they may be analysed. As modern systems can produce enormous volumes of log files, it is essential for effective system management to process, store, and analyse the data adequately. In addition, log data serves various purposes, including troubleshooting, security, and audit, which are crucial to ensuring that systems are functioning properly.

System debugging has historically been done by manual human review of individual log files. However, the volume of log files being generated far exceeds the scope of manual analysis, making automation the only practical solution to this problem. Using log management systems to gather log files from distributed systems has become more common and efficient. However, organisations continue to explore solutions and techniques to effectively manage vast data streams in response to the ever-increasing data growth.

According to Cândido et al.'s study from 2021 on log-based software monitoring [36], the log analysis field is well-established. However, logging and the infrastructure supporting it are still in their early stages of development. This could be attributed to the lack of research demonstrating the methods and technologies utilised in log management systems. Furthermore, the existing infrastructure-related resources are becoming outdated and could benefit from updates to ensure the continued applicability of the employed methods and technologies.

This thesis aims to investigate the methods and technologies employed by log management systems suitable for near real-time fault analysis as it is the most common use case according to a study conducted in 2019 [57]. There-

fore, this thesis compares log management systems suitable for this purpose. In addition, the thesis focuses on exploring the infrastructure methods and technologies utilised in log management systems, emphasising log shipping and storage. The goal is to compare the different approaches used by modern commercial off-the-shelf (COTS) log management systems.

## 1.1 Problem statement

This thesis investigates the methods and technologies utilized in shipping and storage log management components. As noted, research on logging infrastructure is limited and prior works are relatively old. This thesis aims to rectify this issue by shedding light on log shipping and storage methods and technologies utilized by modern log management systems. The main goal is to compare the methods and technologies employed in existing log management systems to provide insights to practitioners and researchers. By studying these components, this thesis intends to increase the understanding of log management systems, the development of custom solutions, aid in system selection decisions, and facilitate further research on the components. It is important to note that this study focuses solely on identifying and describing the utilized methods and technologies and does not involve performance comparisons or other numerical metrics.

The two main research questions the thesis investigates are as follows:

- **RQ 1:** What are some of the more popular methods and technologies used for shipping and storage in log management systems?
- **RQ 2:** Which of the discovered methods and technologies are suitable for near real-time fault analysis log management systems?

This thesis analyses log management systems selected by an inclusion criteria selection method presented in more detail in section 3.4. In addition, the log management systems chosen undergo an in-depth review of their technical documentation, explained in further detail in section 3.5.

## 1.2 Structure of the Thesis

The rest of the thesis is structured as follows. Chapter 2 of this thesis gives a general description of log management systems, including the components the systems are built from, the challenges, and related work conducted in the field. Chapter 3 describes the methodology employed in this thesis to

find and review log management systems. Chapter 4 describes the implementation of the thesis methodology and the results of the in-depth review. The chapter covers the selection and elimination process for log management system candidates and analyses the methods and technologies employed by the selected log management systems. Chapter 5 analyzes and reflects on the methodology used in the thesis along with the findings of the thesis and considers possible future work. Chapter 6, the final chapter of the thesis, offers a conclusion.

## Chapter 2

# Background

This chapter provides an overview of the background information on logging and log management systems. Section 2.1.1 explains what log files are and their significance in software systems. Section 2.2 covers the different components of log management systems, challenges, and overall architecture. Section 2.3 is an overview of the previous research and material in the log management field before this thesis.

## 2.1 Logging

### 2.1.1 What is a log?

The National Institute of Standards and Technology (NIST) defines a log as follows:

*“A log is a record of the events occurring within an organization’s systems and networks. Logs are composed of log entries; each entry contains information related to a specific event that has occurred within a system or network. Many logs within an organization contain records related to computer security. These computer security logs are generated by many sources, including security software, such as antivirus software, firewalls, and intrusion detection and prevention systems; operating systems on servers, workstations, and networking equipment; and applications.” [45]*

In essence, log files record what events have happened and when [46]. Due to the plethora of log types and their varied purposes, there is still no single definition of what a log file typically contains. Log files may contain a diverse range of information, including, but not limited to, operational details, authorization records, and debugging messages. Commonly found attributes in log files are presented in table 2.1 [38]. However, logs can

contain much more information than what is presented in table 2.1, like IP addresses, Process/Thread IDs, or other metadata. Generally, a log file depends heavily on the specific system that creates it. What is recorded in the log is ultimately decided by the system generating it as well as the programmer who designed the logging system.

Attribute	Description
Timestamp	The time and date when a particular event happened
Severity level	A classification that defines the potential impact of an event (e.g. Info, Debug, Warning, Error, Alert)
Source	The identifier of the entity that generated the log file
Message	The text description of the event itself

Table 2.1: Common log file data attributes

While a software system is running, logs are generated automatically. Whenever an event occurs in the system, the logging system creates a log entry with predefined information that needs to be tracked. These logs come in different types and formats, with their content and level of detail varying depending on the specific requirements and configurations of the system. Several types of log files are commonly used in IT systems, including:

- **Application logs:** Application logs record events and messages created inside a software program. These logs contain info about events, errors, and warnings.
- **System logs:** System logs record operating system events and messages. These logs contain starting messages, errors, warnings, and unexpected shutdowns.
- **Security logs:** Security logs record details regarding security-related events. These logs typically contain information about logins/logouts, access attempts, and other actions that might compromise security.
- **Network logs:** Network logs record events and operations regarding network activities. These logs contain information on, e.g. network connections, data transfers, and problems related to networking.
- **Web server logs:** Web server logs are produced by web servers and record activities related to them. These logs typically contain information to determine who, when, and how a server is used.
- **Transaction logs:** Transaction logs record the history of the changes made to a system. These logs enable the system to execute a roll-back operation, which involves reversing the steps executed so far.

- **Audit logs:** Audit logs record changes made to a system. These logs usually contain details about the time, the responsible entity who made the change, and the impacted entity.

### 2.1.1.1 Log file standards and severity levels

Although many log file formats contain similar data, their diversity indicates that they can be used for a variety of purposes. The lack of a uniform format for log messages is one of the significant problems with log management [42]. Because log file contents depend significantly on the system generating it and the programmer who designed it, it causes issues later in the log management pipeline. One issue related to the variety is that any system wishing to ingest different log file types will need unique parsing schemes to form a uniform data structure. The closest standardized formats include Syslog (RFC-5414) [41], JSON (RFC-8259) [35] log format, Common Log Format (CLF) [5], and W3C Extended Log File Format [52], to mention a few.

Despite the lack of well-defined standards of what a log should convey, most log messages can be categorized based on severity levels [38]. The severity levels convey the urgency or relevance of a specific log message. Developers and administrators may use this to prioritize and filter out logs according to severity, enabling them to find and fix urgent problems quickly. There are several classifications, but the five common are *info*, *debug*, *warning*, *error*, and *alert*. For instance, an *alert* or *error* severity level message can indicate a potential problem that needs immediate attention, whereas an *info* or *debug* severity level message might signal a possible issue that is not a pressing matter yet.

A representation of log JSON file contents can be seen in figure 2.1. Each entry includes a timestamp, severity level, source, and message describing the event. In addition, some entries include extra information specific to the event, such as system ID, IP address, error code, and error message. The semi-structured nature of JSON data makes it easier for humans to read and for parsing tools to extract, filter and transform the data to the desired format.

A representation of syslog messages can be seen in figure 2.2. Syslog is a protocol for transporting log messages but also defines a standardized format for the messages by specifying structured data components such as the facility, severity level, timestamp, hostname, and message. Looking at the first example message in figure 2.2, it can be observed that this message has a priority level of 14, a timestamp, source, process ID, and the message of the event. The priority value is calculated from the facility and priority levels using a formula of  $Priority = Facility * 8 + Severity\_Level$ [41].

```
{
  "timestamp": "2023-03-16T16:22:00.000Z",
  "severity": "INFO",
  "source": "server1",
  "message": "Connection successful to API",
  "system_id": "123456",
  "ip_address": "127.0.0.1"
}
{
  "timestamp": "2023-03-16T16:23:00.000Z",
  "severity": "WARNING",
  "source": "server1",
  "message": "Database capacity critical: 85% of total capacity"
}
{"timestamp": "2023-03-16T16:24:00.000Z",
 "severity": "ERROR",
 "source": "server1",
 "message": "Unable to connect to API",
 "error_code": "500",
 "error_message": "Connection refused"
}
```

Figure 2.1: JSON log file contents example

```
<14>Mar 16 16:25:00 server1 CRON[1234]: Job "Database backup" started
<12>Mar 16 16:30:00 server1 DB[5678]: Database capacity at 85%
<11>Mar 16 16:35:00 server1 DB[5678]: Error: Connection timed out
```

Figure 2.2: Syslog message examples

Both figure 2.1 and figure 2.2 provide similar information in a slightly different format. As mentioned earlier, both of these logs require each message to be broken down into their individual components for analysis. The different existing formats might require their own parsing techniques to extract information from these unstructured or semi-structured logs.

### 2.1.2 Why log?

The widespread use of distributed systems, cloud computing, and Internet of Things (IoT) devices has led to system architectures that have the potential to generate massive amounts of log files. With each system and device

generating its own log files, the usefulness of these logs is tied to the ability to collect, store and analyse them effectively.

Because logs may generate enormous amounts of data, traditional monolithic system expansion cannot handle the scaling and can become exceedingly pricey. Scalability and reliability are foundational guiding concepts for modern systems. To achieve scalability and reliability, log management systems can instead use horizontal scaling, allowing them to add more nodes as needed to distribute the load across all of them. This strategy enables the system to maintain massive amounts of data while guaranteeing high availability and fault tolerance.

Log files are important as they may be used for a variety of purposes, such as security, diagnostic, troubleshooting, and auditing. The files should be collected, stored, and analysed to debug these systems effectively. Unfortunately, when logs are gathered for analysis, no single method usually tells what data is significant and what is not. A major challenge with log management is that traditional methods to store and analyse data are unable to handle the diversity of sources, the velocity of the files, and their volume.

## 2.2 Log Management Systems

Log management refers to the methods and processes used for collecting, processing, storing, and analysing log files generated by various applications and systems. Log management systems can be used for many different purposes, such as troubleshooting, security, and audit, and hence play a vital role in ensuring that different systems are running properly. In large organisations, logs are usually collected from multiple servers and must be processed to utilise the files properly. The volume of log data generated by modern systems is overwhelming and impossible for humans to comprehend manually. Some organisations are not only processing data on the kilo, mega or giga scale, but in this day and age, we are talking about terabytes and petabytes.

Log management solutions must be scalable and able to process the data in real time since log data is generated at high volumes and velocities. This is a big data-related issue and requires specialized tools, technologies, and methods that are able to process the sheer amount of complex log data. Despite great enthusiasm and buzz surrounding big data, it is important to acknowledge the critical role that architecture-related concerns play in the field. Even though infrastructure-related issues might get less attention than other big data-related topics such as analysis, these problems must be resolved without question [36]. Despite the challenges they present, solving these issues can help big data systems succeed as a whole.

Figure 2.3 depicts a typical architecture of a log management system. Generally, they consist of several parts that interact together to generate, collect, store, and analyse log data. Each of these elements is essential to the architecture of a log management system. A typical log management system can be split into four separate components:

- **Log generation:** Log generation is the process of creating log files. The main issues with log generation are about how much data should be logged, choosing the proper logging places, and developing and maintaining the logging code. In essence, log file generation is about where-to-log, what-to-log, and how-to-log [37].
- **Log shippers:** Log shippers (also known as the log collection process) are software components used to collect log files from various sources and send them to a centralised storage. Log shippers usually possess filtering and transformation methods to convert the log files to a preferred format. Log shippers generally are about parsing, transforming, and delivering data.
- **Log storage:** Log storage is the process of storing the log data that the log shippers send to a centralised location to make managing and analysing the files more accessible. The storage component is responsible for storing and retrieving the data in a centralised storage, be it a relational database, NoSQL database, or some other data storage system. Simply put, log storage is about handling the data in a manner to increase accessibility and to reduce strain on the whole system.
- **Log analysis:** Log analysis is the process of analysing and visualising the data to gain valuable insights. Log analysis comprises data correlation, pattern discovery, and other statistical analysis methods. Log analysis can benefit from simple to more advanced analysis tools, techniques, and methods such as machine learning models. Fundamentally, log analysis is about interpreting data and getting value from it.

The pipeline in figure 2.3 illustrates the general process steps of turning log data into valuable insights. However, in a production system, some steps could be employed in a different sequence or dropped altogether. For example, some systems prefer to filter the data at the generation component, some might not reduce the amount of data, and some could even do data reduction and normalization on the log storage component. The most direct parallel we can make is to data warehousing. The standard processes in a data warehouse pipeline are extract, transform, and load (ETL). Most of the

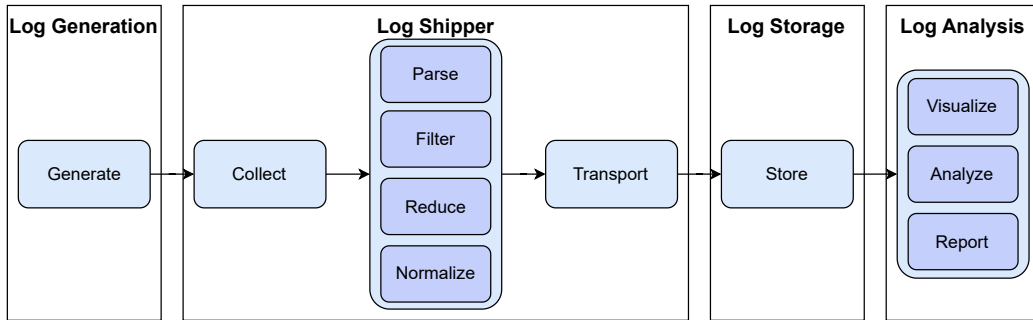


Figure 2.3: Example of a high level log management system pipeline

collection, transformation, and storage in a log management system may be viewed as streaming ETL with possible extra steps.

### 2.2.1 Log Generation

Technically, all devices and systems that produce log files can generate several files in different formats. However, because of this volume and variety of files, log generation is a tricky issue to solve without following some standards. As stated by Chen and Jiang (2017) [37], log generation can be boiled down to three main challenges: where-to-log, what-to-log, and how-to-log. This thesis considers these three aspects more closely tied to logging policies than architecture-related issues. How to design a logging policy is out of the scope of this thesis as it focuses more on infrastructure-related issues. While the logging policies are unquestionably crucial to a log management system, the policies only make up a portion of the design. Logging policies specify what data should be gathered and how it shall be utilized to achieve specific goals like debugging, performance, security, or compliance. A considerable part of log analysis is effectively made possible by having clearly established logging policies, which guarantee that logs are produced at regular intervals, always in the same style, and comply with business requirements and laws.

In log management systems, there is an argument to be made about where log data should be parsed. According to Kreps (2014) [46], log data cleanup or transformation should be handled on the producer side before the data is pushed to the log file. It is usually the case that the log owner knows best about the contents and hence should do the cleanup within the defined logging policies. In a perfect world, only the essential lines would be logged, but as we know, there is no correct way of determining what is relevant and what is not for analysis. Hence, we will assume that most of the sources will

not know exactly what to clean up or how to do it, leaving the cleanup to later stages of the architecture like the log shippers, storage or analysis.

As stated in the section 2.1.1, we listed a few of the most common log file types. Usually, the device or system generating the logs defines the log file type. For example, network devices produce network logs, and software applications produce software logs. These different types and formats play an essential role in parsing and analysis. Therefore, it is crucial to understand the various types of log files and their formats to store and manage them effectively.

One format for generating log files instead of using human-readable text is to save the log files as binary. This instantly reduces the file sizes and increases the logging performance. However, one of the significant issues with generating binary log files is that you will need a conversion tool to translate them for any meaningful purpose. From the standpoint of log analysis, it is highly recommended to utilize log files in a human-readable text format. As expressed by Chuvakin et al.'s (2013) in the "Logging and Log Management" [38] book, the possible repercussions of a software system failure dwarf the negative performance impact of logging. Without sufficient logging, it can be significantly more difficult to diagnose and fix system problems, and the subsequent effects on system performance may be worse than a possible logging-related slowdown.

## 2.2.2 Log Shippers

Log shippers are the components used for collecting, transforming, and transporting log files from sources to the centralized storage. To generalize, a shipper is a simple way of delivering a file to a location. Log shippers usually work on a scheduled time, be it batched jobs or real-time, by taking the log files from the sources and then processing the data by removing and adding information.

### 2.2.2.1 Data transformation and reduction

The creation of valuable data is the key to successful log analysis. Therefore, it is essential to use data transformation and normalization to achieve this. As stated earlier in section 2.2.1, data cleanup and transformation should preferably be undertaken on the producer side. Transformation and reduction are standard data-cleaning methods. Transformation is done to simplify the identification of relevant data by removing, changing, or adding data. Data reduction methods like filtering, aggregation, and sampling are used to counteract the challenges of increasing volumes.

- **Filtering:** This method eliminates useless data from the log files. For instance, a log file might have a lot of debug lines used by programmers to find errors in their code that are unnecessary for the production environment logs. Filtering out this type of information can greatly decrease the file size. Some of the most common tools used to filter logs are Regular expressions (Regex), Grep, and AWK. Specific data can be extracted from a log file based on predefined criteria using any of the three tools, each of which is an effective text-processing tool.
- **Aggregation:** This method involves summarizing several rows of data to give an overview. Aggregation may summarize several logs by severity level, time, or some other data attribute. This method loses the fine details of the data but, in turn, reduces several lines of text to one summary.
- **Sampling:** This method only stores a subset of the log data. This subset can be selected randomly or by systematic approaches such as storing every tenth row. This method can significantly decrease the amount of data stored but is not practical for analysis because the data stored does not necessarily reflect the whole dataset.

From a storage viewpoint, data reduction can increase the efficiency of the storage and save significant storage space. Reducing the amount of data stored reduces the amount of data needed to query, resulting in faster queries. However, determining what constitutes irrelevant or redundant data can be difficult, rendering reduction methods impractical. For a near real-time fault analysis system, the data is more worth than the storage space costs if the data can be used for debugging.

### 2.2.2.2 Data parsing

Parsing is a method for extracting only the relevant information from a log file. Generally, the first step of parsing is to know or identify the log data format. The format (structured, semi-structured, or unstructured) and the particular use case influence the selection of the parsing method.

Once the format is known, we can parse the data. This step splits each log event into its individual components. For example, the first message of figure 2.1 would be split into its individual fields, which are timestamp, severity, source, message, system\_id, and ip\_address. Some well-known parsing methods are delimiter, Regex, and key-value parsing (KVP).

Delimiter parsing works by splitting a log event into its individual components using a specific character or sequence of characters known as a delimiter. CSV files commonly use a comma as a delimiter between each value.

Regex parsing works by searching, matching, and manipulating text. Regex has its own syntax that creates a pattern for searching, matching, and manipulating text. For example, regex patterns can extract emails, IP addresses, and other variables. Regex is an extremely powerful tool for text data because text data does not need to follow any format making it hard to extract specific information.

Key-value parsing looks for pairs of objects that are grouped by a key and a value. JSON and XML are prime examples of key-value formats. Key-value data is also simple and intuitive for humans to understand at a glance, especially when compared to other semi-structured data like CSV files.

Once an event has been split into its individual components, each field can be further processed by parsing, filtering, or modifying. The final parsing step is usually normalization which takes the parsed data and turns it into a standard format or structure. Log data normalization is done to standardize the data for a more straightforward comparison analysis.

The order of the stages is not an absolute order that must be followed but a general order that works for most cases. In data warehousing, ETL is arguably the order that is more familiar to most and utilized more. However, the variation ELT (Extract, Load, Transform) exists because it has its unique uses. The ordering of extracting, transforming, and loading the data depends heavily on the system's use case. The steps can be performed at many points throughout the log management pipeline, such as the generation, shipping, or storing steps.

### 2.2.2.3 Data transportation

In simple terms, log transportation is a way to move a log message from one place to another. Several log transportation methods exist, such as file transfer protocols which are built upon network protocols.

TCP and UDP are two examples of network protocols that may be used. These protocols enable real-time log data transmission via networks. Network protocols generally are used in solutions that require higher throughput and low latency. UDP is a lightweight, low-overhead, minimal latency method. Because UDP is connectionless, UDP is faster and less resource-intensive than TCP. This also makes UDP prone to data loss because it has no delivery checking or retransmission methods. TCP requires communication for transporting information, which means that it establishes a reliable connection between two locations before sending the data. TCP ensures that

the data is received with acknowledgements, and if something is missing or corrupted, it will resend the data. This makes TCP a bit slower than UDP but ensures that the data will arrive at the location. According to Chuvakin et al.'s (2013) book "Logging and Log management" [38], even with multiple weaknesses, syslog UDP is the most popular log transport method.

File transfer protocols like FTP, SFTP, and SCP are a few examples that may be used. FTP is a protocol for transferring files over a network. The underlying protocol FTP uses is TCP. The file transfers using FTP are reliable and efficient but are lackluster in other areas, such as security, leaving the protocol vulnerable to network security threats. SFTP is the encrypted version of FTP. SFTP is a more secure way of transferring files but causes overhead with encryption and decryption, which leads to a slower process.

Some log management solutions even work by writing the log data directly to a columnar database. However, it is advised to avoid writing directly to a columnar database in practice because network protocols like FTP and SFTP are designed to transfer files efficiently in bulk. In most cases, the system will need to support various transportation methods, as different software systems and devices may not be compatible with the same transportation methods.

Generally, the decision of transportation methods will rely on the particular needs of the log management system. The requirements include the volume of data, latency, throughput, reliability, and security required.

#### 2.2.2.4 Delivery guarantees

In the context of log message or log file transportation, three common delivery guarantee concepts are used to describe the delivery requirements of a message:

- **At-most-once delivery:** This guarantee assures that a message is sent just once, if at all. In other words, the sender will not try to resend it again if it is lost in transit. This guarantee provides the greatest speed with the cost of possible missing deliveries.
- **At-least-once delivery:** This guarantee assures that a message is received at least once but may be delivered several times until the recipient responds with an acknowledgement. This guarantee ensures that a message will not be lost due to network problems or other issues, but it may lead to duplicate messages. This method provides reliability at the cost of speed.

- **Exactly-once delivery:** This guarantee assures that a message is sent and delivered once. Typically exactly-once delivery is the most challenging to do because it requires additional message processing steps compared to at-least-once delivery. Hence, at-least-once delivery tends to be faster.

Depending on the log management systems policies and requirements, a specific delivery requirement may be required. Suppose a system can afford to lose single events and want to prioritize speed over guaranteed delivery. In that case, it may opt for protocols like UDP, an example of an at-most-once delivery guarantee. On the other hand, FTP is an at-least-once delivery guarantee. There is no single correct delivery guarantee that would work on all systems. The system policies and requirements decide what guarantee is correct. Generally, at-least-once and exactly-once delivery guarantees are commonly used for systems where message loss is not tolerated. In contrast, the at-most-once delivery guarantee is suited for systems where message loss is tolerated.

### 2.2.2.5 Data processing

Log shippers are also often responsible for scheduling shipping times for the files. Stream and batch processing are the two main methods used for processing data.

Batch processing has been an industry-standard way of processing data. Batching allows systems to process the data when they want to, meaning they can avoid putting unnecessary strain on the system during peak business hours. Often batching is scheduled to run nightly or on weekends. A significant downside of batching is that the data is available later than it is produced. Using an only batching approach in a real-time fault analysis system where debugging of issues needs to happen as soon as possible can lead to data being old or even borderline useless before it even arrives in the system. Batch processing is more suited to complex analysis systems that need to do an in-depth analysis of non-time-sensitive data, such as sales validation and prediction.

Stream processing is considered "real-time" processing, meaning the data produced will be instantly ingested without unnecessary delay. Stream processing is suitable for systems that favour the time it takes for data to be used for analysis. A solution that is in-between batching and streaming is micro-batching. Micro-batching involves segmenting a stream into smaller batches, usually gathered within a time window or size limitation. This approach of gathering data over a time window, which can be in seconds to minutes,

allows for near real-time processing of data streams. However, determining what qualifies as near real-time can often be ambiguous. The distinction between batching and near real-time processing has a fine line differentiating the two and is usually defined by the system's requirements. As such, the definition of near real-time processing is up to each use case.

Fault analysis systems are a great example of a system that needs stream processing, be it real-time speed or near-real time, with a relatively small window. Each minute of downtime can be a considerable revenue loss for organizations if they are not able to upkeep their service-level agreements (SLA).

### 2.2.3 Log Storage

The ability to store and retrieve logs is essential for a smooth log management system. A centralised storage place for decentralised sources is beneficial in many ways. A central place with a clean copy of every piece of data is an extremely useful asset for analysing massive amounts of data. One benefit is that storing all logs in a centralised storage reduces the strain on the decentralised hosts that produce the logs. Accessing the files with only a single interface reduces the time to analyse and find insights when everything is accessible from one place. Even if the files were not used for analysis and were only stored in the log management system, the system will act as a backup of the files.

Data storage is a typical big data issue, and we can draw several parallels between data warehousing and log management systems. Both data warehouses and log storage are typically created for high write throughput and for handling large volumes of real-time data. This requires both systems to be scalable and reliable storage solutions. Data warehouses usually contain ETL pipelines to modify the data. Log management systems also have similar steps for cleaning the incoming data, but usually are less transform-heavy. On a high level, both systems share remarkably similar qualities, but to differentiate the two, log management systems are created to store and retrieve log files. In contrast, data warehouses are mainly created to store structured data from various sources.

#### 2.2.3.1 Data lifecycle

To draw further connections to data warehouses, data storage benefits of having a lifecycle. As data becomes older, the relevance of that data also often reduces. Regular analysis and reporting tasks do not require old data, so it should be deleted. Although organizations desire to save costs and

increase storage space, it is still crucial to follow retention policies and laws to prevent data deletion before the allowed time. Hence a lifecycle policy is important for log storage. The concept of *hot* and *cold* data relates to data's significance and how often it is used. *Hot* and *cold* tiered storage does have different variations, like *hot*, *warm*, *cold*, and *frozen* storage, which has additional storage layers.

Figure 2.4 depicts how the *hot/warm/cold/frozen* tiered storage usually works. Data referred to as "warmer data" is found on the left-hand side, whilst data referred to as "colder data" is found on the right-hand side. Moving from left to right reduces the storage's performance and costs. The *hot* layer is often accessed and used in real-time applications. The *hot* layer usually has more nodes than the colder layers, enabling higher performance and reliability. The data in the *warm* layer is usually accessed less than the *hot* data but kept in a more accessible format than the *cold* or *frozen* layer. Data is rarely accessed on the *cold* layer, while the *frozen* layer is a long-term storage that serves more compliance and legal reason. In general, the warmer layers are usually the database or object storage used for daily analysis. In comparison, the colder layers are usually object storages with fewer replication factors and processing power. Therefore, accessing data on colder layers requires more time than the warmer layers.

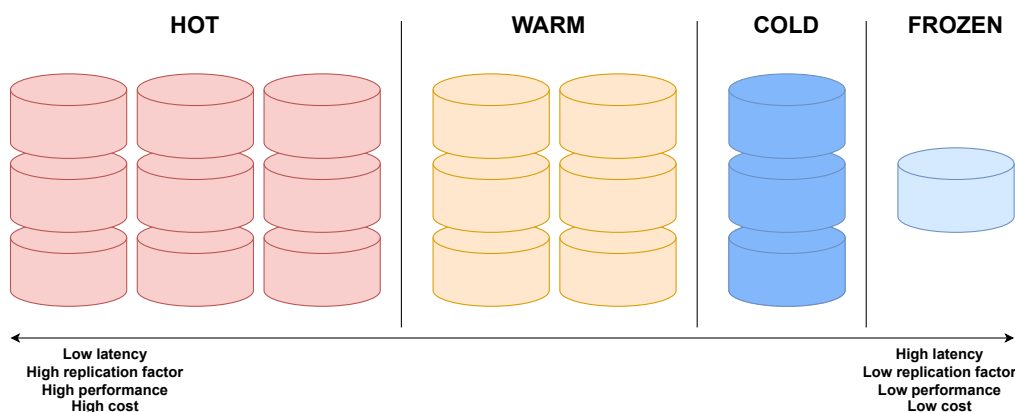


Figure 2.4: Depiction of hot/warm/cold/frozen tiered storage

Data retention policies and laws like GDPR (General Data Protection Regulation) usually define the period needed to store data in the different layers. The data retention policy usually defines how long data should be stored in each layer. After the data has been stored for the defined period in a layer, it is moved down to a colder layer. On the last layer, the data

is deleted when the storage period is over. Both the *cold* and *frozen* data layers are typically considered archiving.

Archiving is the practice of transferring infrequently used data to long-term storage, such as the *cold* or *frozen* layer. Log management systems can retain higher performance and costs by clearing up disk space on the frequently accessed layers. In addition, data is often stored in a compressed or encrypted format to decrease the storage space in archives. For example, the compression format Parquet can significantly reduce the size of the logs, up to six times smaller than conventional logs [42].

### 2.2.3.2 Storage foundations

The CAP theorem is a fundamental concept in distributed systems. According to the CAP theorem, a distributed system can only possess two out of the following three properties: consistency (C), availability (A), and partition tolerance (P) [40]. This theory carries over to data platforms, where database designs differ in their basic conceptual designs. The two important concepts are known as ACID and BASE.

ACID is an acronym for the following properties: Atomicity, Consistency preservation, Isolation, and Durability [56]. A system guaranteeing the four properties of ACID assures the reliability and correctness of a system. Atomicity means that actions on the database should be treated as a single unit of work. A transaction is carried out entirely or not at all. In other words, all transaction steps are completed without errors, or the database will roll back to the state before any steps were executed. This guarantees that the database continues to be consistent. Consistency preservation means that a transaction transitions from one consistent state to another. Inconsistent intermediate states are inevitable and accepted, but the end state must be consistent. Isolation means that each transaction should be carried out separately. To avoid inconsistency, this concept prevents concurrent transactions from interfering with one another. Durability means that once a transaction has been completed, its modifications are irreversible such that it will withstand any system failures, be it hardware or software failures. Because it can endure failures, this ensures that the modifications are persistent until a subsequent transaction affects the same data.

Alternatively, BASE is an acronym for the properties Basically Available, Soft state, and Eventually consistent. Basically Available means the database is always accessible for reads and writes even when partitions, nodes, or other components are down. This can mean that the system returns a partial response or even breaks consistency. This property ensures the operability and responsiveness of a system when some of its components are having

issues. Soft state means that even without input from outside, the system's state can vary over time. This is often a result of things like network lag or eventual consistency. This property is associated with systems that are able to respond more quickly to failures and adapt to changes in network latency or system load. Eventually consistent means that the state will eventually become consistent given enough time. Across the nodes of a system, there can exist temporary inconsistencies caused by networking or other issues. Scalability and fault tolerance are made possible by this property.

To summarize, ACID is a strong integrity and consistent system, while BASE is highly available, flexible, and fault-tolerant. Depending on the system's required guarantees, either ACID or BASE will be preferable because no database can fulfil all three properties of the CAP theorem. While ACID properties are crucial for applications using transactional data like financial systems, some properties need to be more relaxed for many applications. For example, a more relaxed consistency method like BASE creates availability, scalability, and performance instead of strict consistency.

### 2.2.3.3 Storage types

For log management, a variety of storage methods can be employed for different storage layers, each with its benefits and drawbacks. The following are the most typical storage types for log management:

- **Relational databases:** Relational databases order data in tables with a predefined structure. Tables consist of rows and columns, where columns represent data attributes. Due to log data's semi- or unstructured nature, the formats are often incompatible without changes. Hence, the variety in logs presents a big issue when storing the data in the schemas demanded by relational databases. In addition, log messages also exhibit variability in length, necessitating the use of schemas with excessively large character counts for the message column. One of the main benefits of using a columnar database is the ability to create queries and execute them efficiently. Unstructured data is difficult to handle for relational databases, especially regarding queries. Many relational databases offer limited full-text search or real-time analysis, two features sought after by log management systems. Another benefit of a relational database system is the strong access and permission system. In essence, relational databases work best with data that is cleanly structured.
- **Document-oriented databases:** Document-oriented databases store data in a semi-structured format. Data is stored in a key-value format,

where each document has a unique key. Their flexible schemas allow for easy storage of varying file structures and formats. Document-oriented databases are designed to scale. They are able to grow horizontally. A drawback of document-oriented databases is that data redundancy may occur when many documents contain the same data and no schema checks for duplication. Another drawback is that document-oriented databases do not support transactions, making maintaining integrity more difficult.

- **File storage:** File storage is a type of hierarchical data storage that stores data in files and folders. Both structured and unstructured data may be stored quite effectively using file storage. Furthermore, file storage is relatively affordable compared to other data storage types, such as databases. However, this type of storage is rarely optimized for fast data querying and is often less fault tolerant than other methods. These properties make file storage mainly usable as an archiving method.
- **Object storage:** Unlike file storage, object storage utilizes rich meta-data and flat address space to store data as objects in blocks. These objects are often files or unstructured data, perfect for log management systems. To easily retrieve the data, objects are assigned IDs, making it simple to retrieve and manage them without knowing the exact location, unlike in a hierarchical space. Because they are built on a distributed architecture, allowing them to scale horizontally rather than vertically by expanding the cluster's node counts, making them highly scalable. Object storage is a suitable alternative for archiving.

Because storage space is precious, data reduction methods are sometimes employed to reduce the amount of data that needs to be stored for analysis without losing critical information. According to a survey by ERA Software about the log management domain, 76% of respondents are taking active steps to reduce the storage volume [32]. However, cutting down on stored logs can cause issues later on when the log files could have proven helpful for troubleshooting. Therefore, organizations that do not employ the proper tools risk suboptimal system performance and increased costs.

While proper data reduction methods are important to reserve precious storage volumes, data filtration should only be performed if it is certain that the data can be reduced or deleted. Examples of data reduction methods used in log management systems are filtering, aggregation, and sampling, as mentioned in section 2.2.2. These methods can be effective if implemented correctly but are also challenging to implement without impacting analysis

results. Applying data reduction methods before storage is generally advisable, although this depends on the specific use case. Pre-processing and removing unnecessary data early in the pipeline can improve the efficiency of subsequent steps.

### 2.2.4 Log Analysis

Log analysis is the practice of analysing and interpreting log files to derive meaningful insights or spot system problems and abnormalities. Therefore, log management is an essential component, and the potential value of logs can be considered trivial if they are never used or reviewed, rendering them useless since they serve no real purpose.

As the primary goal of log analysis is to interpret the data, various approaches can be taken to log analysis, such as searching the data manually, visualising it, or using other methods like machine learning. Depending on the underlying data, different methodologies can derive meaning from that data. Typical approaches include data correlation, pattern discovery, and other statistical methods. In addition, standard statistical methods, like measuring events over a given time, can be used to assess the health and stability of a system.

Going through different log analysis methodologies is out of the scope of this thesis. The methodologies range from simple methods and tools to more advanced rule-based and machine-learning models. When building and designing a near real-time fault analysis log management system, the essential aspect of log analysis is that the visualisation methods support querying, reporting, and alerts in near real-time.

### 2.2.5 Challenges of Log Management

The previous sections have highlighted infrastructure challenges and methods to tackle them. Because log management systems must be capable of handling huge volumes, velocities, and varieties of data, the systems are considered to be a part of the big data ecosystem. Big data opportunities and challenges commonly share the five V's of big data (Volume, Variety, Velocity, Veracity, and Value) [31]. Therefore, these five V's of big data may be used as a basic framework for summarising and understanding the main challenges of managing and analysing massive datasets. The five V's of big data can be mapped to the following main issues in log management:

- **Volume:** The size and amount of log files being generated are growing in proportion to the number of systems and their complexities.

- **Variety:** Log files come in many forms. Systems generate them automatically, and developers can even create custom log formats. A few universal log standards and formats exist, but they are not widely used
- **Velocity:** Modern systems can generate log files rapidly. Depending on the use case, some systems require a huge file amount to be processed in real-time or at least in near-real time to assist their operation.
- **Veracity:** Log files are bloated with information. Not everything generated in the log files is useful for analysis. Log files that come in must be cleaned so that the data can be efficiently used.
- **Value:** To be able to gain value from log files, proper log analysis methods need to be employed by organisations to understand the data.

A significant challenge for analysis is the pace of data ingestion. The data ingestion rate is directly linked to the analysis's usefulness. Log analysis depends on fast and reliable access to data. For near real-time fault analysis, quick reaction times are essential for preventing major outages in service. As a result, this emphasises the infrastructure's scalability and reliability.

## 2.2.6 Design of a log management system

This section provides a quick overview of how a log management system is designed, a domain needing more research.

Log management systems are used for IT operational purposes to prevent downtime and ensure smooth operations. IT platforms have SLAs requiring the system to maintain uptime, response time, and other performance measures to be met. To ensure smooth operations, speed is crucial for enabling the teams to identify and resolve issues immediately. As the variety, velocity, and volume of log data continue to increase, problems with crucial system requirements become increasingly apparent. Hence, the design of the log management system must be scalable and fault-tolerant to tolerate the growth of the data.

Any system can be built from the ground up using custom tooling, which is usually operationally expensive. This thesis will focus on analysing technologies and methods employed by COTS software. There are two primary ways to go about it when designing a log management system without writing custom software for it. The first solution is to pick and choose an all-in-one log management system software solution that encompasses all aspects of a log management system. These software have functionality for all aspects

from collection and storage to analysis. The other method uses a best-of-breed design solution where tools are chosen for the individual components of the log management software. This design allows the mixing and matching of the best individual components from several vendors rather than depending on one vendor's solution for the entire system.

Even though this thesis will not cover how to effectively create log management system policies, the policies are a crucial part of the design and should be included in the design phase. A log management system policy creates guidelines on essential details of the system, such as what and how log files should be collected, stored, and analysed. Sources [38], [42], [45], and [33] cover several factors to consider while creating a log management system, which inherently help to create the log management system policies.

### 2.2.6.1 All-in-one solutions and SIEMs

All-in-one solutions provide all features and functionality required for a log management system, including log collection, storage, and analysis. There has been a surge in the development of specialized log management systems. However, many all-in-one solutions market themselves as security log management tools.

Many of these security-marketed log management tools are also better known as Security information and event management (SIEM) tools. SIEMs are typically used by security teams to monitor and respond to threats. SIEMs usually have more advanced correlation, alert, and analysis tools than pure log management system solutions. Even though SIEMs are designed for security usage, they can be used as log management systems. However, SIEMs, as a standalone log management solution, can provide too many tools and features, which may be overkill for use cases that only need to collect, store and search logs for troubleshooting or auditing purposes. Therefore, a simpler log management tool set may be more cost-effective and efficient than a SIEM.

The benefits of using an all-in-one solution include simplified usage and management of the system. Most all-in-one systems are plug-and-play, requiring only that sources are connected and set up to adhere to established policies to produce a functioning system. When all components are under the one and same infrastructure, maintenance and management of the system becomes far more straightforward. Typical drawbacks of using all-in-one solutions are vendor lock-in and limited customization. Vendor-lock-in can make it harder to switch to another solution, and the lack of customization may lead to limited analysis because of lacking features. In summary, an all-in-one solution provides an easily maintainable quick solution.

### 2.2.6.2 Best-of-breed solutions

Instead of depending on a single tool to supply all the required functionality, a best-of-breed solution selects the best tools for each component in the log management system.

Using a decoupled solution can offer many benefits. Each component is intended to do a single task well rather than several tasks passably. This usually leads to better performance. Scaling is simple since each component is decoupled from the other. Parts of the pipeline are also easier to replace, making the whole system more flexible to change. Picking and choosing solutions can also be very cost-effective, and replacing underperforming parts is more accessible than in an all-in-one solution.

The downside of using different tools from different vendors is compatibility related. Integration between these tools can take time and effort. A lot of research needs to be done beforehand to know what components are compatible. In addition, software upgrades in one tool can break the integration between them, creating more operational work and costs. In summary, a best-of-breed solution provides excellent flexibility and scalability, which are sought-after traits in big data platforms.

## 2.3 Related Work

In order to find studies related to this thesis, a search was conducted on various online databases, including the ACM Digital Library, IEEE Xplore, Science Direct, and Google Scholar. Our search included keywords such as "log", "log architecture", "log methods", "log management", "log management system", and "log management methods". The term "log" is used widely in computer sciences and has a variety of meanings, including the mathematical logarithmic function. In the domain of log management, logs are broadly quoted for log generation, mining, analysis, and much more. Because of this, it is challenging to find earlier work that pursues the same objectives as this thesis. As far as we can tell from these results, little to no prior research has provided a thorough overview of log management system methods and technologies in the past years.

Logs are crucial in software system development, maintenance, and security. Log management systems can be used for many different purposes, such as troubleshooting, security, and auditing. Log management systems used for real-time fault analysis enable the ability to diagnose issues [39], predict possible issues [44], help detect system faults [54], and hence play a vital role in ensuring that different systems are running correctly.

A study conducted in 2021 [36] about log-based software monitoring found that the log analysis field has matured. However, logging and log infrastructure are still in the early stages of development. The search conducted on the digital libraries confirms this by mostly finding publications about log analysis. There exists some research on more practical implementations of different Log management systems using a variety of tools such as Elastic-stack [53], Splunk [49], Hadoop [47], and other solutions readily available on the market. In addition to this, there exists a wide range of research on evaluating software tools [55], architectures [48], and individual components of log management systems [58].

Most of the previously listed research related to log management systems has tended to focus on specific aspects or products, ending in a narrow perspective of the issues surrounding log management solutions. Previously listed research does an excellent job of addressing the issues. However, some of the research relies on their experience when selecting solutions without always providing clear reasoning on why the methods or technologies are appropriate for the issue. Therefore, even with the growing interest in log management systems, potential research areas still need to be explored, such as an overview of the methods and technologies that support real-time fault analysis.

The paper "Advances and challenges in log analysis" [50] clearly states that effectively managing a log management system is still challenging. The paper approaches log management from a log analysis perspective and the difficulties associated with issues like log generation, parsing, and mining. These issues are undoubtedly crucial to maintaining logs, although they comprise a small percentage of the log management system. Consequently, infrastructure-related problems with log storage, scaling, and reliability are barely mentioned in the publication.

Perhaps the closest publications related to infrastructure-related challenges associated with log management can be found in the book "Logging and Log management - The Authoritative Guide to understanding the concepts surrounding logging and log management" [38], NIST's "Guide to Computer Security Log Management"[45] or the paper "A methodology for building a log management infrastructure" [33].

The goal of Chuvakin et al.'s (2013) book "Logging and Log management" [38] is to provide readers with a deeper understanding of the many ideas and problems associated with logging and log management. The book covers various topics such as log generation, log storage, log parsing, log normalisation, log analysis, data visualisation, and log monitoring. Throughout the book, the authors emphasise the significance of log management from an analytical standpoint while overlooking most of the architectural challenges.

The NIST's "Guide to Computer Security Log Management" [45] presents challenges related to the number, volume, and variety of computer security logs. The guide states several times that the major challenge with log management systems is issues related to "*balancing a limited amount of log management resources with a continuous supply of log data*". The guide also identifies issues like inconsistent log formats, a high number of sources, and the volume of the files. Finally, the guide focuses on how policies, procedures, and proper planning of a log management system can help create and maintain infrastructure.

The paper "A methodology for building a log management infrastructure" [33] proposes a methodology for designing and building a log management infrastructure. The suggested technique involves gathering requirements, designing, implementing, and evaluating. In addition, the authors discuss various challenges and solutions with the collection, storage, and analysis.

Because most similar publications are a decade old, having an updated perspective on the challenges, methods, and technologies would be valuable. This thesis aims to fill some gaps in the existing literature by presenting current methods and technologies employed in log shipping and storage by log management systems available on the market.

## Chapter 3

# Methods

In this chapter, the research methodology employed in this thesis is explained. The chapter begins with an overview of near real-time fault analysis log management systems, followed by an imaginary use case featuring its requirements and policies. Finally, the chapter concludes with the log management system selection and the in-depth review methodology.

To provide practitioners and researchers with insights into the methods and technologies log management systems use, we need to do an in-depth analysis of the tools available on the market.

According to Jadhav and Sonar (2009) [43], finding a software package that meets the requirements involves a comprehensive evaluation of numerous competing aspects, which is a difficult task. To reduce the amount of software compared, the thesis uses a step-by-step selection method that is based on their research. The methodology presented by them is meant as a guideline and tool that may be modified to meet the needs of the evaluation. Their method has been simplified and adjusted since all steps cannot be applied in this thesis without implementing the log management system in a real-world target environment. Figure 3.1 presents the process flow steps. Step 1 is to capture the requirements and create policies based on the use case. Step 2 is an initial selection of software found with the help of market research and product review sites. Step 3 is software elimination using inclusion criteria based on the use case requirements. Finally, step 4 is an in-depth review of the selected software systems' methods and technologies. These steps are explained in more detail in sections 3.2 through 3.5.

Anastopoulos and Katsikas (2014) [33] proposed that capturing the requirements is the first step in building a log management system. Log management systems can be used for various use cases, each having different requirements. Unfortunately, a broad generalization of the most suitable methods and technologies cannot be made as they depend on the specific use

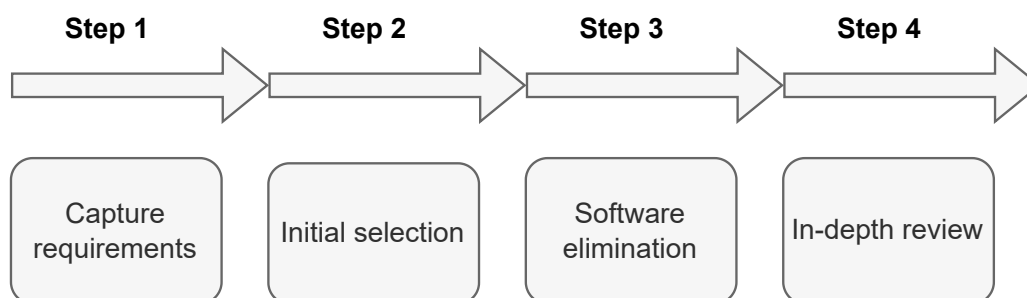


Figure 3.1: Depiction of the thesis methodology

case. Therefore, a single use case must be defined for this thesis to assess the methods and technologies for a log management system.

A hypothetical use case has been proposed to compensate for the lack of a real-world company. This thesis uses a near real-time fault analysis system as the use case for the log management system. The use case was chosen because, according to a study conducted in 2019 (Yachin, 2019) [57], IT operations troubleshooting related was the most popular use case for a log management system in an organization. For IT operations to function smoothly, IT operation teams must quickly detect and respond to issues. A near real-time fault analysis log management system allows IT operation teams to act proactively and fix potential upcoming problems. Settling on a specific use case enables the determination of the system's requirements and features. This strategy makes it easier to determine the suitability of the methods and technologies based on the identified requirements.

### 3.1 Near real-time fault analysis log management systems

Let us start by clarifying what a near real-time fault analysis log management system is. In IT operations, a near real-time fault analysis log management system is a software system used to monitor various other systems' and components' events to predict and prevent faults.

The system has the same capabilities as most log management systems: collect, store, analyse, and visualise log data with an emphasis on providing all these features in near real-time. A near real-time system allows IT operation teams to recognise issues faster and reduce system downtime. In

addition, with near real-time feedback, IT operation teams can proactively fix issues before they escalate. For example, suppose the problems escalate and cause an outage in service. In that case, the troubleshooting capabilities are much more extensive with near real-time feedback than with manual log analysis of each source component.

The fault analysis system is helpful for troubleshooting and can also be used to get other insights, such as usage patterns. By analysing system loads and performance, IT operations teams can optimise the system during peak hours by increasing performance capabilities and reducing performance allocation during non-peak hours.

To summarise, near real-time fault analysis log management systems are useful for root cause analysis, proactive fixing, and performance optimisation. This powerful tool is a great addition to any IT operations team looking to streamline the monitoring and maintenance of their products.

## 3.2 Use case definition

The first step of the methodology is to capture the requirements. This thesis has been conducted independently, without a company's cooperation. An imaginary but realistic general use case has been proposed to simulate a realistic scenario. As stated earlier, our log management system will use a near real-time fault analysis use case. Because methods and technologies employed in a log management system may vary depending on the use case, an appropriately designed use case will aid the comparison. In this context, the use case serves as a reference point for assessing the methods and technologies.

### 3.2.1 Use case description

A company that has launched a successful software-as-a-service (SaaS) product is exploring ways to optimize its operational processes. Currently, they have a customer base of over a million active users. The infrastructure for the service is made up of several servers, databases, and networking components.

The company is fixing production issues through issue reports and manual log analysis. However, reactive maintenance strategies alone are insufficient. Solely relying on reactive maintenance strategies is like relying only on guards to protect a castle without any proactive measures like a moat around the castle. A good strategy implements both proactive and reactive methods. For example, a near real-time fault analysis log management system is able to predict and prevent system faults by providing alerts of possible future

issues. Without a log management system, the teams typically start by collecting log data before they can proceed with the analysis. However, a centralized log management system allows the teams to jump directly into analyzing data, thereby streamlining the process and saving entire man-hours as well as reducing downtime. Thus the company desires to implement a log management system that helps them monitor suspicious events, provide 24/7 incident management and prevent critical faults in its product.

Data is critical for incident management, and any lost log data can lead to the loss of crucial information needed for locating and fixing system issues. Furthermore, incomplete or inaccurate analysis results may arise due to the loss of log data which could lead to pointless and time-consuming efforts to resolve the underlying issue. Therefore, all data logged is deemed essential for troubleshooting, making it necessary for the system to utilize methods to prevent data loss. As described in section 2.2.6, custom software development can be time- and resource-consuming. Considering the company's resources and time limitations, COTS software is more practical. Hence the log management system candidates will only consist of COTS software solutions.

The methods and technologies of the system should satisfy the following architectural properties:

- **Scalability:** The system should be able to scale for future demands. Horizontal scaling is preferred as monolithic vertical scaling is technically limited.
- **Reliability:** The system should also function properly when no errors occur. The system should employ methods to become more resilient to failures. For example, log data transportation should use methods that ensure data is not lost in transit.
- **Fault tolerance:** The system should be designed to handle component failures. For example, databases should be able to retain a copy of the data in a number of nodes. A higher replication factor can help mitigate the damage caused by failing nodes.
- **Near real-time analysis:** The latency of data from the source to the analysis component should be in seconds. Before deploying the log management system in production, an upper limit cannot be specified. In addition, the requirement of data to arrive in near real-time does not allow batch processing and thus requires the system to adopt either a stream processing or micro batching approach.

In essence, the core design principles for the log management system should include near real-time data processing and zero data loss.

### 3.2.2 Policy creation from use case requirements and description

Organizations should establish standards and processes to achieve a successful log management system as discussed in the "Guide to Computer Security Log Management" [45]. As a part of the planning, an organization should specify its logging policies. Policies are helpful because they provide a uniform strategy for the system that ensures legal and regulatory requirements will be fulfilled. In addition, the policies define requirements and recommendations for the log management system. The thesis's main focus is on policies that address architectural challenges. The policies cover all aspects of a log management system, including log generation, shipping, storage, and analysis.

It is essential to understand the difference between requirements and policies. Requirements are standards the system must satisfy, whereas policies are directives that aid decision-making or offer system design recommendations. In order to perform a comprehensive selection and analysis, further elaboration of the use case is necessary by creating policies. To create the policies for the use case, a combination of policy design guides sourced from [38], [42], and [45] was utilized. These resources provide guidance for designing policies that are generally relevant to log management systems. The policies created for the use case in this thesis are presented on a high level in table 3.1. Column 1 includes the main category for the log management policy, column 2 is the policy aspect impacted, and column 3 provides a more detailed description of the policy.

Sections 3.2.2.1 through 3.2.2.5 details the policies and the rationale behind them.

#### 3.2.2.1 Log generation policies

Log generation policies define the where-to-log, what-to-log, and how-to-log. Architecturally, the focus is on the types of logs generated and how fast.

- **Log sources:** All components of the SaaS product should generate logs. This includes all operationally critical components. As a result, the log management system will ingest a wide variety of log file types.

Category	Policy	Description
Generation	Log sources	All operational components in the SaaS system should generate logs
Generation	Log events	All event severity levels should be logged
Generation	Logging frequency	All occurrences of events should be logged instantly when the event happens
Shipping	Delivery guarantee	Log data delivery should use a lossless method
Shipping	Connection security	Data transportation does not need to use secure transportation methods
Shipping	Data processing	The system should use stream processing or micro-batching
Transformation	Data parsing	Data should be parsed for storage
Transformation	Data reduction	Data should not be aggregated, sampled or reduced by any other means
Transformation	Data filtering	The system should be able to filter data
Storage	Data retention model	The system should support a method for reducing the data in the data storage
Analysis	Analysis type	Reports, alerts and manual querying should be supported by the analysis component

Table 3.1: Log management system use case policies

- **Log events:** All event severity levels should be logged. Different severity levels convey differently detailed information that can be of help in debugging.
- **Logging frequency:** Events should be recorded at the moment they happen. The system must be in near real-time, so the maximum interval for an event to be recorded should be in seconds.

The rationale for these policies is based on the assumption that all data

can be critical for debugging, as stated in section 2.2.1. Requiring that all components generate logs helps ensure that every aspect of the system is monitored and that no crucial information is missed. Additionally, by recording events at the moment they happen, the system can provide a near real-time view for quicker and more efficient troubleshooting. These policies guarantee that the log files generated provide a detailed and near real-time view for efficient troubleshooting and maintenance. Since log generation methods have been widely studied, this thesis will not go into detail about them.

### 3.2.2.2 Log shipping policies

Log shipping policies define guidelines and rules for transporting log files from one location to another. Shipping policies can cover shipping frequency, transportation methods, and delivery guarantees.

- **Delivery guarantee:** Each log file can be critical for troubleshooting, making it important to have a guaranteed delivery of each file. Therefore, the system should use a lossless method to transfer log files, despite the additional overhead. This way, the system ensures the integrity of logs during shipping.
- **Connection security:** Since all logs are generated and collected within an internal network, a secure transport method is not required for data transportation.
- **Data processing:** From an architectural standpoint, the system needs to use either stream processing or micro-batching. Micro-batching and stream processing can keep the processing delays within seconds, a requirement for the use case.

The rationale for these policies is based on performance requirements. The amount of time it takes to be able to analyse data is a critical element of the system's design. Therefore, the policies were created to use methods that support the speed of data transportation for analysis since near real-time analysis is a crucial component of the log management system.

### 3.2.2.3 Log transformation policies

Log transformation policies define how the log data should be structured and stored in the storage.

- **Data parsing:** The incoming data can be in a structured or unstructured format. It is necessary to parse the incoming data to enable easier storage and faster analysis. Parsed data can be structured to better fit a predefined schema or template, increasing the effectiveness of storage and retrieval. Preprocessing the data in this way also enables the storage system to index the data in different ways, which can lead to a faster system.
- **Data reduction:** The incoming data should not be reduced to improve the processing performance or storage space, as all data can be critical for troubleshooting.
- **Data filtering:** The system should support filtering methods to filter data eventually. For example, when data is identified as irrelevant to fault analysis, it can be filtered out during the parsing step.

The rationale for these policies is to guide the data formatting in a way that supports storage architecture for faster retrieval while storing all necessary data.

#### 3.2.2.4 Log storage policies

Log storage policies define aspects of the storage system, such as how long to retain data, the availability level of the storage system, and how to ensure compliance requirements.

- **Data retention model:** The system should support a method for reducing the amount of actively queried data to improve system performance which is essential for a near real-time system. Size-or time-based removal methods can be used to avoid storing all data in the database management system forever.

The rationale for this policy is to meet regulatory requirements and legislation, as well as provide data storage that is fast.

#### 3.2.2.5 Log analysis policies

Log analysis policies generally define how the collected log data should be analyzed. This comprises methods, tools, and technologies such as correlation and pattern design, among others.

- **Analysis type:** The analysis tools should at least support reporting, alerts, and manual querying. Reporting can produce summaries of the status or events of the SaaS product. Alerts can be used to inform operational teams of possible and current issues that require immediate attention. Manual querying enables deeper ad hoc exploration and analysis of log data.

The rationale behind this policy is to provide the necessary methods to remediate issues. This thesis will not dive into how to support analysis for incident response effectively.

### 3.3 Initial selection method

The second step of the thesis methodology is the initial selection of software. The research by Bandor (2006) [34] presents several methods for performing an initial selection. The research reports that one successful method uses a selection team of several developers and engineers. However, since this thesis was conducted without the cooperation of a company, it was necessary to rely on other methods that did not require opinions from experts in the field. Therefore, the method chosen for this thesis is a systematic search approach.

In order to find log management software for evaluation, a query search approach was used. Using query searches to find relevant candidates is a challenging process. To mitigate the challenges of search queries, queries were restricted to only include the most important terms related to log management components. Queries used were: "log management software", "log collection software", "log storage software", "log shipping software", "log analysis software", and "SIEM".

In addition to searching through search engine results, software review sites provided by software technology research companies, like Capterra [4], G2 [16], Gartner [17], and TrustRadius [30] were used. Furthermore, "Magic Quadrant Reports" [51] and [3], provide lists of potential candidate software. The software from these lists was used in this study to expand the potential candidate list.

Utilizing multiple review sites and sources can maximize the variety of found log management software. It can also mitigate potential bias as different review sites and search queries may find log management software that are not featured on other sites.

### 3.4 Software elimination method

The methodology's third step is the process of refining the list of candidates to be subjected to an in-depth review. Numerous choices are available on the market for log management systems and tools. However, comparing and assessing every product in a single thesis study can be challenging when the market is stocked with alternatives. Therefore, choosing fewer log management tools to research allows for a more in-depth assessment and collection of the methods and technologies. Hence, there is an incentive to reduce the amount of software used for comparing their methods and technologies.

Inclusion criteria elimination was used in the thesis to refine the viable log management software solution candidates. The use case description and related policies served as the foundation for the inclusion criteria. Product technical documentation was reviewed to assess the methods and technologies. While inclusion criteria elimination provides a quick and effective way to eliminate options that do not meet the use case requirements, it also has drawbacks. If the criteria are defined wrongly, potential software will be left out of the analysis. The inclusion criteria were grouped into three separate groups:

- **Criteria 1:** The log management system must have accessible documentation. This is because the thesis methodology uses a technical product specification review approach requiring documentation access.

Furthermore, the log management systems selected for the thesis must either be all-in-one solutions or a stack of programs from the same vendor that collectively satisfy every aspect of a log management system. Evaluating possible integration possibilities is a highly complex task. Limiting the selection allows for a more accurate analysis.

Additionally, the system must have clear documentation demonstrating its suitability for IT operational purposes, whether in the overview section, marketing material, or customer stories. This is particularly important since SIEMs can be considered overkill solutions for log management, as explained in section 2.2.6.1.

- **Criteria 2:** The log management system must support near real-time or real-time processing of log files. The reasoning for this criteria is that the use case requires the system to be able to process data in near real-time.
- **Criteria 3:** The log management system must be a unique product and may not be built using another existing log management solution.

The reasoning for this criteria is that if several selected log management systems are built using the same software, the resulting data may show a bias toward their methods and technologies.

After the inclusion criteria elimination, a list of log management system products and vendors is left. This list will act as the starting point for the in-depth review.

### 3.5 In-depth review method

To answer both RQ1 and RQ2, the methodology conducts an in-depth review of the selected software. This entails a comprehensive analysis of the methods and technologies each software uses. Again, the software documentation review approach was used to gather information. The attributes to be collected for final analysis are presented in table 3.2. Additionally, there are useful websites for validating information about different database management systems, such as the "Database of Databases" [7] and "DB-Engines" [10]. These resources offer important details on different methods, features, and functionality. Finally, the gathered data on the methods and technologies involved in the shipping and storage of logs will serve as the foundation for the final analysis and results.

<b>Attribute</b>	<b>Description</b>
Shipping technologies	Log shipping technologies supported
Fault-tolerant shipping technologies	Shipping technologies that support guaranteed delivery
Parsing methods	Log parsing methods supported
Storage system	Storage system utilized for log files
Storage model	Storage model of the storage system
Storage consistency method	The consistency concept utilized by the storage system
Replication support	Storage system support for replication
Retention method	Storage system data retention method
Archival storage model	Data archival storage model
Archival lifecycle structure	The structure of archival layers

Table 3.2: List of attributes to extract for analysis

## Chapter 4

# Results

This chapter presents the implementation of the thesis methodology process and the results of the in-depth review. The section covers the methodology's execution process and challenges encountered during steps 2, 3, and 4. Figure 4.1 illustrates the sequence of events taken during the selection and elimination of log management system candidates. The results of the in-depth review are analysed to answer both RQ1 (*What are some of the more popular methods and technologies used for shipping and storage in log management systems?*) and RQ2 (*Which of the discovered methods and technologies are suitable for near real-time fault analysis log management systems?*). The log shipping, storage, and archival methods and technologies analysed are presented in tables 4.2, 4.3, and 4.4, respectively.

### 4.1 Implementation details

A total of 97 potential log management software were identified from the initial selection, several of which were obtained from product review websites. These log management software were a combination of application performance monitoring tools, log management systems, and SIEMs.

These 97 candidates were subjected to the next step, inclusion criteria elimination. The majority of the candidates did not meet the first requirement group, which was explained in section 3.4. Most of the products eliminated by the first criteria lacked accessible documentation or had only a community page for information. This level of information was insufficient for a technical product specification review approach, which resulted in the removal of these candidates. In addition, many eliminated products were either part solutions with no complimentary parts created by the same vendor or custom-tailored solution services. As such, these candidates did not

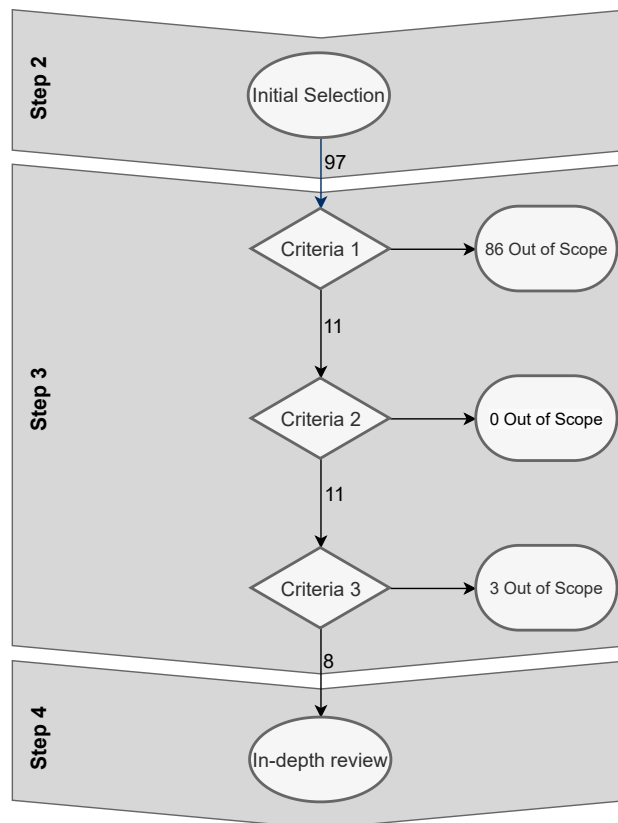


Figure 4.1: Overview of the thesis software selection process

meet the explained criteria for inclusion in this thesis. A handful of software were found to be marketed solely as SIEM solutions. As indicated in section 2.2.6.1, these solutions may have excessive features requiring extensive configuration and knowledge to set up and operate effectively. Upon a deeper examination of the documentation for a few of these solutions, it became apparent that they were overly complex for the requirements of a log management system. Therefore, such products were excluded from consideration. In addition, the product site was sparsely detailed in some instances, making it challenging to find crucial information. In extreme cases, the product site did not respond or was linked to an entirely different domain.

After the first inclusion criteria elimination, the result was that 86 candidates were scoped out of the initial 97, leaving only 11 candidates for further review. The 11 candidates were moved on to the second criteria elimination, the requirement of near real-time processing. According to each product's documentation, every product supported real-time or near real-time process-

ing. As a result, no candidates were scoped out in this step.

The third and last criterion was that the log management systems should not be based on another solution. A thorough analysis revealed that three of the eleven candidates under evaluation were built using the same open-source solution called Elastic-stack [11]. The three products, Graylog [21], Logz.io [25], and Sematext Logs [27], were all Elastic-stack-based SaaS log management solutions. These three products were not further considered to avoid bias toward one solution’s methods and technologies.

After all inclusion criteria eliminations, eight log management systems were left. The methods and technologies were then extracted for the eight log management systems: Coralogix [6], Datadog [8], Elastic Stack [12], Grafana Stack [20], InsightOps [22], DataSet [9], Splunk [28], and Sumo Logic [29]. The selected systems for in-depth review are also clearly presented in table 4.1 with the vendor and the product system name.

<b>Vendor</b>	<b>Product</b>
Coralogix LTD	Coralogix
Datadog	Datadog
Dataset Inc	DataSet
Elastic NV	Elastic Stack
Grafana Labs	Grafana Stack
Rapid7	InsightOps
Splunk Inc	Splunk
Sumo Logic, Inc.	Sumo Logic

Table 4.1: Software selected for in-depth review

The product documentation was inspected to extract the methods and technologies for the attributes presented in table 3.2. However, only information explicitly stated on the official product site was considered to ensure the extracted data’s accuracy and reliability.

## 4.2 Data shipping methods and technologies

### 4.2.1 Shipping technologies

The *shipping technologies* column in Table 4.2 are the technologies used for log shipping that the storage system supports or can accept as input.

Finding relevant information is made more difficult by the variety of names given to shipping technologies

<b>Product</b>	<b>Shipping technologies</b>	<b>Fault-tolerant shipping technologies</b>	<b>Parsing methods</b>
Coralogix	Filebeat, Fluent Bit, Fluentd, Logstash, Vector, Winlogbeat	Filebeat, Fluent Bit, Fluentd, Logstash, Vector, Winlogbeat	JSON, Regex
Datadog	Fluentd, HTTP, Logstash, NXLog, Rsyslog, syslog-ng, TCP	Fluentd, Logstash, NXLog, Rsyslog, syslog-ng, TCP	Grok, JSON, Regex
DataSet	Custom agent		Regex
Elastic Stack	Beats, HTTP, TCP, UDP, syslog	Beats, TCP, syslog	CSV, Grok, JSON, KVP
Grafana	Docker driver, Fluent Bit, Fluentd, Lambda Promtail, Logstash, Promtail	Fluent Bit, Fluentd, Logstash	JSON, Regex
InsightOps	Custom agent, REST API, syslog	Syslog	JSON, KVP, Regex
Splunk	Custom agent, HTTP, REST API	Custom agent	CSV, JSON, KVP, Regex
Sumo Logic	Custom agent, HTTP, syslog	Syslog	CSV, JSON, Regex, XML

Table 4.2: Data Shipping methods and technologies

The multitude of names used for shipping technologies, including "agents, collectors, forwarders, senders, shippers", makes finding relevant information challenging. It is worth noting that the technologies listed in Table 4.2 are only some of the available input methods, as all systems support direct integrations to other sources, such as databases, cloud services, and programming languages, among others. Even though the systems had hundreds of direct

integrations to other systems, this analysis concentrates on the most general methods mentioned outside of the direct integrations. Figure 4.2 presents the distribution of how many unique systems supported a specific technology.

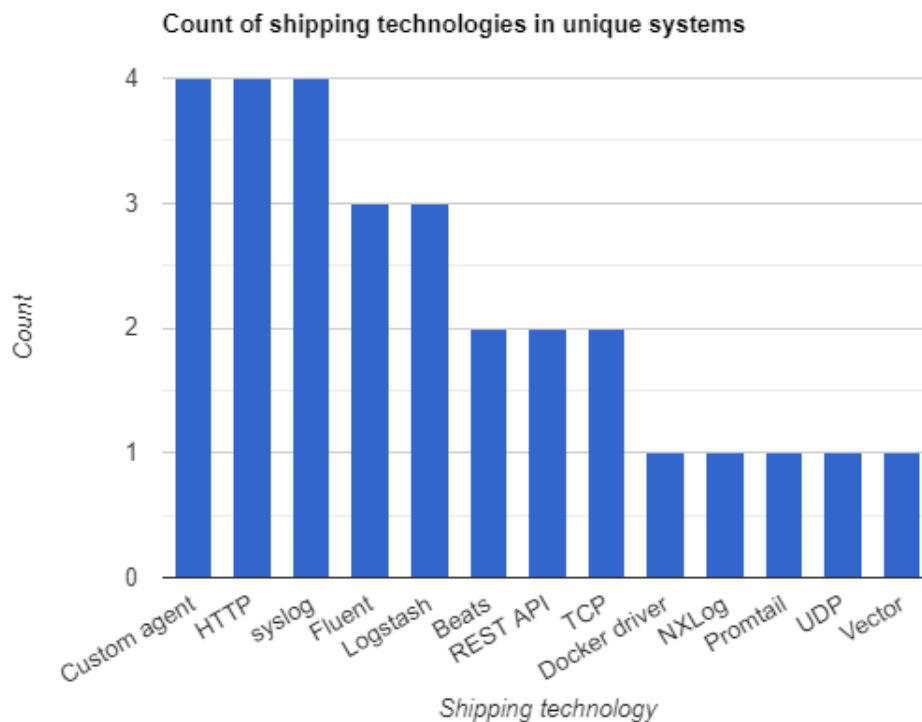


Figure 4.2: Count of shipping technologies in unique systems

From a log shipping transportation perspective, the systems assessed may be classified into two categories: those that employ or support premade shippers and those that create their own. The systems that do not use a custom agent supported at least one of the Fluent-based shippers (Fluentd[14] and Fluent Bit[13]), and Elastic-stack shipper (Beats[2] and Logstash[24]), or even both. Beats and FluentBit are open-source log shippers that are lightweight and function similarly to agents. They are set up to collect and ship logs forward to the collection system on the various components in its environment.

Custom agents, HTTP, syslog, Fluent-based, and Logstash, were the most widely supported technologies among the assessed systems. An intriguing observation is that plain TCP and UDP were not as prevalent as previously

stated by Chuvakin et al.'s (2013) book "Logging and Log management" [38]. UDP was once considered the most widely used method for log transportation. However, there seems to be a current shift in trend in supporting alternative transportation methods. Notably, HTTP transportation was also widely supported among the supported technologies.

## 4.2.2 Fault-tolerant shipping technologies

The *fault-tolerant shipping technologies* column in table 4.2 are the technologies used for log shipping that supports guaranteed delivery either by default or as an option. The distribution of how many unique systems supported a specific fault-tolerant technology is presented in figure 4.3

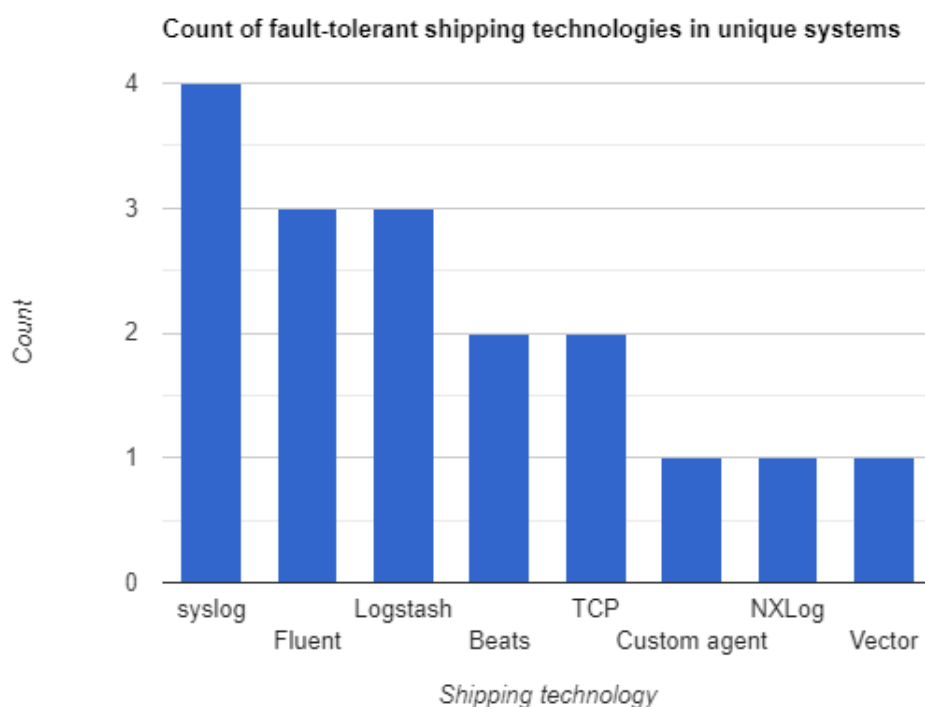


Figure 4.3: Count of fault-tolerant shipping technologies in unique systems

For near real-time fault analysis log management systems, delivery guarantees are essential. Some of the technologies offer several delivery guarantees, like Fluent [15]. Though many have guarantees, not all do. This can

be due to various reasons, such as limitations in the underlying protocols or variations in communication methods. For example, not all custom agents explicitly state that they provide a guaranteed delivery and may use different protocols based on the operational environment. HTTP and REST APIs do not natively provide guaranteed delivery but require workarounds to achieve it. Promtail and lambda Promtail stores failed deliveries for a certain number of time. This value, "backoff\_config" [26], can be changed, but it does have a theoretical limit of how long it can store them. Hence it cannot ensure a true delivery guarantee in all cases.

Based on the 18 log shipping technologies (excluding custom agents), it was discovered that 6 of them do not provide delivery guarantees for log transportation, which accounts for one-third of the total technologies evaluated. This suggests that providing delivery guarantees is becoming a default option for log transportation. Hence, it suggests a shift in the industry towards adopting more fault-tolerant log shipping methods that provide guaranteed delivery, contrary to Chuvakin et al.'s (2013) [38] statement of UDP being the standard practice.

### 4.2.3 Parsing methods

The *parsing methods* column in table 4.2 are the parsing methods supported by the systems. Figure 4.4 presents the distribution of how many systems supported a specific parsing method.

Since most log data is either semi-structured or unstructured, parsing methods can also be classified into these two groups. Semi-structured formats are easily parsed with their respective methods: CSV, KVP, JSON, and XML. JSON data has become commonly utilized, which can be seen from the support for its parsing method in figure 4.4. The popularity of JSON parsing can be attributed to its flexible and semi-standardized format and easy human readability. Moreover, JSON can support more complex data structures, such as nested objects, simplifying log generation and allowing all necessary data to be included.

On the unstructured side, Grok and Regex are useful for processing unstructured data, like logs created by syslog. Popular log shippers, e.g., Fluentd and Logstash, also provide premade regular expressions made specifically for well-known formats. Log data is often generated in plain text format and can contain large amounts of text. However, Regex is a powerful text-processing method that can be used to extract almost anything from any format. It is reasonable that Regex is the parsing method that is the most generally supported for unstructured data, given the number of patterns that are accessible online and its flexibility for modifying the patterns.

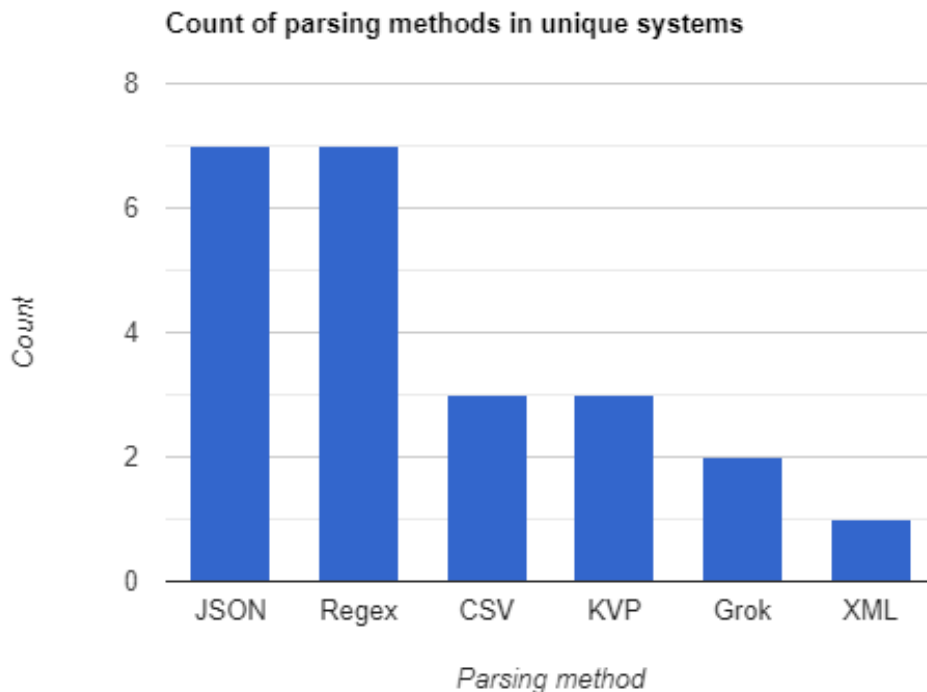


Figure 4.4: Count of parsing methods in unique systems

Based on the results from table 4.2, it is advised to support at least one structured and one unstructured parsing method, as most of the assessed systems have done so. In addition, JSON and Regex have been identified as excellent options as parsing methods due to their widespread popularity and versatility in handling different data structures.

## 4.3 Data storage methods and technologies

### 4.3.1 Storage systems and models

The *storage system* column in table 4.3 is the DBMS (Database Management System) or storage system used to store the log files, and the *storage model* column describes the data model.

A notable observation is that the log management systems examined either used a cloud object storage, predominantly Amazon S3, or a vendor-

Product	Storage system	Storage model	Storage consistency method	Replication support
Coralogix	AWS S3	Object storage	Strong**	Yes
Datadog	Unknown	Unknown	Unknown	Unknown
DataSet	Unknown	Event storage	Unknown	Yes
Elastic Stack	Elasticsearch	Document Storage, Search engine*	Eventual	Yes
Grafana	AWS S3 or GCS	Object storage	Strong**	Yes
InsightOps	AWS S3	Object storage	Strong**	Yes
Splunk	Splunk	File storage, Search engine*	Eventual	Yes
Sumo Logic	Unknown	Unknown	Unknown	Yes

\* Classification taken from DB-Engines [10]

\*\* AWS S3 is a strong read-after-write consistency, but bucket actions like bucket deletion are eventually consistent

Table 4.3: Data storage methods and technologies (DBMS data)

specific proprietary solution for log storage. As the name of the underlying DBMS for Datadog, DataSet, and SumoLogic was not disclosed in the documentation, we cannot conclude whether these systems utilize a proprietary system or rely on cloud object storage. Although the storage system name for DataSet was not clearly specified, the documentation did reveal that it uses a proprietary event storage. The distribution of the storage models is illustrated in figure 4.5, which emphasizes how object storage, followed by search engines and event stores, are the commonly used technologies of the assessed log management systems.

Since previous research by Afsaneh et al. (2011) [48] had shown that well-known technologies like Oracle, MS SQL, and Postgre SQL were often used, the initial prediction was that several log management systems would use a SQL-based storage system. However, the research was done in 2011, and technological advancements may have influenced the data storage systems utilized in modern log management systems, which would explain the results.

Modern log storage systems are built similarly to what is documented in Grafanas documentation [19]. Most of the systems assessed utilized an index

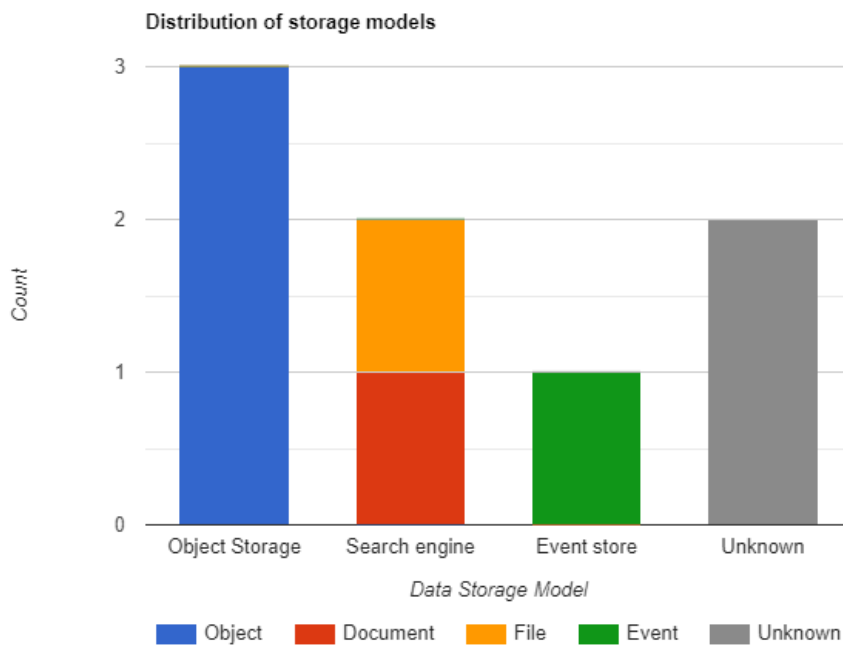


Figure 4.5: Distribution of storage models

storage and a log storage. The method provides query efficiency by storing the metadata of log files in a query-optimized system while storing the log files separately. This architecture enables faster and more efficient retrieval of log data from the storage system than classical file storage. Query-optimized index storages enable quick and easy retrieval because the system is built to handle metadata operations, such as indexing, searching and filtering based on different log attributes. For example, log retrieval becomes more effective by organizing the metadata in a query-optimized system instead of scanning each file, reducing the time and resources needed for data retrieval.

Based on the results, it can be inferred that utilizing a separate index and log storage is an appropriate method for log management systems. Object, document, or file storage are practical options for log storage, while columnar formats seem to be falling out of popularity despite DataSet's use of event storage. Object storages are a suitable choice due to their inherent scalability.

### 4.3.2 Storage consistency and replication

The *storage consistency method* column in table 4.3 describes the storage systems consistency concept, and the *replication support* column describes if the storage supports data replication.

High availability and eventual consistency are priorities in the BASE concept, which is the concept that S3 and GCS (Google Cloud Storage) were initially designed by. The more classical DBMS like Splunk and Elasticsearch are also built around eventual consistency. Both S3 and GCS have undergone updates that have resulted in a transition to a stricter consistency method [18] [1]. Both storage systems support strong read-after-write consistency, meaning newly created objects are instantly visible to all clients without delays. Despite a transition toward stricter consistency, not all operations utilize strong consistency. For instance, S3 bucket actions like list and delete are still eventually consistent. Nonetheless, both S3 and GCS still claim to provide high availability with better read consistency.

For a more fault-tolerant storage, replication support can be employed. Replication support increases resilience against data center outages when replication is done on a data center level. Using replication across multiple data centers with a replication factor of  $n$  ensures that the same data is stored in  $n$  number of data centers, thereby providing resilience in the event of  $n - 1$  data center failures. The fact that all log management storage systems assessed support replication attest to its significance. Replication eases recovery from disasters and enables highly available storage systems, both of which are essential features for systems requiring uninterrupted operation.

## 4.4 Data archival methods and technologies

The *retention method* column in table 4.4 is the default retention method for each product according to its documentation. It was found that all of the assessed log management systems showed support for time-based retention as the default option. In addition, many of the systems allow defining of the lifespan policy. Thus, the system allows for the customization of storage periods and tier amounts, allowing tiers to have storage times that suit the use cases' unique needs. This outcome is consistent with the idea that most data merely needs to be stored for as long as necessary for analysis and regulatory compliance.

The *archival storage model* column in table 4.4 is the storage model used for the archived data. Object storage is predominantly used, with the exception of Grafana and Splunk. Most systems use a "rolling" method as data

Product	Retention method	Archival storage model	Archival lifecycle structure
Coralogix	Time based	Object storage	Hot/warm/cold
Datadog	Time based	Object storage	Hot/cold
DataSet	Time based	Object storage	Hot/cold
Elastic Stack	Time based	Object storage	Hot/cold
Grafana	Time based	None	Hot
InsightOps	Time based	Object storage	Hot/cold
Splunk	Time based	File storage	Hot/warm/cold/ frozen/thawed
Sumo Logic	Time based	Object storage	Hot/warm/cold

Table 4.4: Data archival methods and technologies

moves from one tier to another as the data ages. The Grafana-stack’s Loki component only has a ”hot” layer that already stores its log files in S3 or GCS. It does not delete data from the storage directly but relies on the default object storage lifecycles in S3 and GCS. Out of the box, Grafana does not support archival and requires a custom workaround in AWS or Google Cloud to achieve it. Instead of archiving log data separately, InsightOps, and the Elastic-stack uses snapshotting. InsightOps does not use a ”rolling” method but provides a daily backup of logs ingested. Elastic-stack’s Elasticsearch also functions similarly to InsightOps by providing snapshots that may be retained for extended periods.

The *archival lifecycle structure* column in table 4.4 shows the amount of default storage tiers. A two-tiered strategy, with a hot query layer and a cold archival layer, was the default structure for most log management systems. Intriguingly, two systems provided a three-tiered approach. Incorporating more layers than two can increase the efficiency and performance of the system by storing only what is needed for single-day analysis in the hot layer, then moving it to a warm layer for weekly analysis before moving it to the cold archival layer. Grafana was distinctive because it only supported a hot query layer by default, which can quickly become more expensive than using several layers if the same amount of data is stored.

Given that the majority of the assessed systems use object storage, together with its scalability and affordability properties for long-term data storage, it is reasonable to conclude that object storage is a useful option for

log data archival. Additionally, creating daily snapshots of the entire data storage system consumes more storage space than storing data through the time-based "rolling" method. Hence it is reasonable to conclude that the "rolling" retention method is more suitable.

## Chapter 5

# Discussion

In this chapter, the implementation method, the results of the study and future work are discussed and reflected upon. Although many related works discuss challenges and solutions, not all cover the methods employed to solve them. This study fills some gaps by presenting the methods and technologies log management systems currently utilize for shipping, storage, and archival.

### 5.1 Reflection of implementation

The selection method used in this thesis is a straightforward yet effective approach for narrowing down the number of systems to research. As a result of the selection and elimination process, eight candidates were found. The implementation of the research methodology in this study was a suitable approach for limiting the in-depth review. A downside of this approach is any possible bias from the research. A selection team approach may be used in similar studies in the future to prevent any potential conscious or subconscious self-bias. Multiple experts would enable more comprehensive analysis and mitigate personal biases. Although a selection team approach was not available for this thesis, steps were taken to reduce any potential bias by including several sources for finding the log management systems as well as creating a generic use case.

While the analysis has uncovered some insights, it is important to note that, as with any selection-based research, it is possible that certain log management systems may have been overlooked during the search and elimination process.

During the initial selection step, it was observed that the log management system market appears to be primarily focused on SIEMs. However, considering the current market orientation towards SIEMs, the inclusion of SIEMs

that did not explicitly promote their suitability for IT operational purposes may have yielded unexpected results. This is because SIEMs might not have been optimally designed for the use case in this study.

Similarly, if the inclusion criteria had allowed log management systems to be built using the same underlying log management technology, it would have led to biased results favouring the methods and techniques employed by the Elastic stack.

## 5.2 Reflection of the results

Regarding RQ1 (*What are some of the more popular methods and technologies used for shipping and storage in log management systems?*), the in-depth review has uncovered what methods and technologies are utilised for log shipping, storage and archival.

Based on the results from the assessed log management systems, the log shipping component should ideally be able to support a diverse range of transportation methods. A decade ago, plain UDP was the predominant method. However, a significant shift in focus toward transportation methods that offer guaranteed delivery has occurred. As a result, custom agents, HTTP, Elastic-based, and Fluent-based shippers are the current most popular technologies. These shippers are widely used because they are versatile in processing and transporting log data in many different environments.

Regex and JSON have become the most used parsing techniques on the log parsing side. Based on the results from this thesis, the ability to handle various log formats is a key factor for parsing. Therefore, a log management system should ideally support structured and unstructured data parsing methods.

Most of the assessed log storage systems employed a separate storage for the log files and a separate index storage containing the metadata to enable more efficient querying. As most assessed storage systems utilise cloud storage, there has been a shift towards a stricter consistency method. Originally, S3 and GCS were built upon the eventual consistency concept but have become stricter by providing a strong read-after-write consistency. However, given that S3 and GCS were initially built with the eventual consistency in mind, and the two systems that did not employ cloud storage were also built with the eventual consistency concept, eventual consistency remains a viable approach. Replication support in log storage is essential to achieve higher fault tolerance, vital for operational systems requiring high uptimes.

Regarding log archival, it is advised to use time-based techniques. The usage of object storage for archival is the approach that was used by most

of the assessed systems. As for the lifecycle, a two-tiered approach is often considered an efficient way to boost performance by separating the querying and long-term storage layers. However, it should be noted that more than two layers can be added. For example, a three-layer structure can save costs by allocating the most critical data to the "hottest" layer and gradually moving it to "colder" layers as the data is less frequently accessed.

Regarding RQ2 (*Which of the discovered methods and technologies are suitable for near real-time fault analysis log management systems?*), the initial expectation was that almost all discovered methods and technologies would be suitable. As it turned out, all the shipping-related methods and technologies, except for the shipping technologies that did not support delivery guarantees, were suitable. Similarly, for storage, most of the assessed systems used object storage or search engines, which could scale horizontally and offer fast accessibility and fault tolerance through replication support. Regarding archival methods and technologies, all employed were considered to be suitable as they only improve performance and do not limit the system.

Almost all methods and technologies discovered were suitable for near real-time fault analysis log management systems. This result could be partly due to the candidate selection method. To summarise the results of RQ2, all methods and technologies listed in chapter 4 are suitable, except for the shipping technologies that did not support delivery guarantees.

The results of this thesis provide information on the technologies and methods used in current log management systems. Researchers who wish to compare the effectiveness of the different methods and technologies, conduct further research on specific components, or practitioners who want to develop their log management system by utilising the currently most popular methods and technologies presented can all benefit from this information. Furthermore, by filling in knowledge gaps, the thesis offers practitioners a useful resource that helps develop more robust and effective log management shipping, storage, and archival.

### 5.3 Future work

As stated by Cândido et al. (2021) [36], the logging and log infrastructure research field is still in its early stages and therefore requires further research.

As a potential extension of this thesis, further evaluations of specific parts of log management systems can be carried out to validate their strengths and limitations. For instance, research comparing performance could be conducted to compare and evaluate the effectiveness of the different methods and technologies. Unfortunately, the lack of test data is a long-standing is-

sue in the log management system infrastructure field. However, Loghub's Github [23] offers a wide variety of log file formats in various sizes for this purpose. As explained by Zhu et al. (2019) [58], some parsing approaches may work well with smaller datasets but cannot perform well with larger quantities. Therefore, when comparing and evaluating parsing methods, it is crucial to run realistic tests using larger data sets that reflect the real-world environment because results from smaller tests might not be representative of the actual performance.

Another potential area for further research is exploring the effects of more complicated architectural designs in log management systems, such as adding a buffering queue like RabbitMQ or Kafka between the log generation and storage. Notably, the lack of available information about the assessed custom agents presents an opportunity to investigate whether they already utilize messaging queues. Additionally, for near real-time fault analysis log management systems, comparing the performance of different shipping architectures is an unanswered topic.

Further research can be conducted on the log archival side by comparing compression methods, storage methods, retrieval speeds, and archival formats. Although some log management systems already support compression techniques, evaluating the effectiveness of these methods and considering potential performance-enhancing alternatives is a possible research path.

## Chapter 6

# Conclusions

This thesis examines the methods and technologies utilised in log management systems suitable for near real-time fault analysis. The main contribution of this thesis is to offer insights into the methods and technologies employed for log shipping and storage, thereby bridging a gap in the existing literature.

The thesis explained the methodology used for selecting log management systems for an in-depth review. A generic use case for near real-time fault analysis log management systems was created as part of the methodology. Finally, the thesis presented the results of the in-depth documentation review of the selected log management system candidates. The results provided an overview of the methods and technologies used in log shipping, storage, and archival. Although some variations were observed among the COTS systems, the thesis identified several methods and technologies suitable for near real-time fault analysis log management systems.

The thesis results show that custom agents, Fluent- and Elastic-based shippers were the most commonly used transportation technologies. A shift in transportation methods towards guaranteed delivery from UDP has occurred. As for parsing, Regex and JSON were the most widely supported methods. Based on the results from this thesis, the ability to handle various log formats is a key factor for parsing. Therefore, log management systems should ideally support parsing methods for both structured and unstructured log data.

For log storage, the assessed log management systems utilised a separate storage for log files and index storage containing metadata of the log files for more efficient querying. With S3 and GCS being popular log storage systems, it has resulted in a stricter consistency concept now that both S3 and GCS provide strong read-after-write consistency. However, eventual consistency remains a suitable alternative considering that S3, GCS, and the two systems

that did not employ cloud storage were built on this concept. Furthermore, all assessed systems provided replication support, which provides higher fault tolerance, a vital requirement for systems that need to maintain high uptimes.

In log archival, all assessed systems employed time-based data retention methods as the default option. Object storages were the preferred storage system for archiving data. A two-tiered archiving strategy was the most common approach to separating the querying and long-term storage layers. However, more than two layers may be utilised.

Researchers and practitioners looking to compare, develop, and improve their log management systems can gain knowledge from the thesis results. The thesis fills knowledge gaps and offers practitioners a resource to create more robust and efficient log management systems for shipping, storing and archiving.

More research around the logging and log infrastructure field can still be done. For example, evaluations of specific components' methods and technologies can be performed to validate their strengths and limitations. Additionally, the effects of more complex log management infrastructures can be studied. Finally, the log archival field could be improved by evaluating compression techniques and considering performance-improving alternatives.

# Bibliography

- [1] Amazon s3 strong consistency. <https://aws.amazon.com/s3/consistency/>. Accessed: 2023-04-20.
- [2] Beats: Data shippers for elasticsearch — elastic. <https://www.elastic.co/beats/>. Accessed: 2023-04-24.
- [3] Best security information and event management (siem) software. <https://www.g2.com/categories/security-information-and-event-management-siem#grid>. Accessed: 2023-03-25.
- [4] Capterra: Find evaluate top software business services. <https://www.capterra.com/>. Accessed: 2023-03-25.
- [5] Common logfile format. <https://www.w3.org/Daemon/User/Config/Logging.html#common-logfile-format>. Accessed 5.3.2023.
- [6] Coralogix documentation. <https://coralogix.com/docs/>. Accessed: 2023-04-12.
- [7] Database of databases. <https://dbdb.io/>. Accessed: 2023-04-18.
- [8] Datadog documentation. <https://docs.datadoghq.com/>. Accessed: 2023-04-14.
- [9] Datset documentation. <https://app.scalyr.com/help/welcome>. Accessed: 2023-04-15.
- [10] Db-engines - knowledge base of relational and nosql database management systems. <https://db-engines.com/en/>. Accessed: 2023-04-18.
- [11] Elastic stack. <https://www.elastic.co/elastic-stack/>. Accessed: 2023-04-12.

- [12] Elastick stack documentation. <https://www.elastic.co/guide/index.html>. Accessed: 2023-04-14.
- [13] Fluent bit. <https://fluentbit.io/>. Accessed: 2023-04-24.
- [14] Fluentd — open source data collector — unified logging layer. <https://www.fluentd.org/>. Accessed: 2023-04-24.
- [15] Fluentd: High availability config. <https://docs.fluentd.org/deployment/high-availability>. Accessed: 2023-04-24.
- [16] G2 - business software and services reviews. <https://www.g2.com/>. Accessed: 2023-03-25.
- [17] Gartner peer insights - choose enterprise technology software and services with confidence. <https://www.gartner.com/peer-insights/home>. Accessed: 2023-03-25.
- [18] Google cloud storage consistency documentation. <https://cloud.google.com/storage/docs/consistency>. Accessed: 2023-04-20.
- [19] Grafana loki storage documentation. <https://grafana.com/docs/loki/latest/storage/>. Accessed: 2023-04-20.
- [20] Grafana stack documentation. <https://grafana.com/docs/grafana/latest/>. Accessed: 2023-04-14.
- [21] Graylog: Industry leading log management siem. <https://www.graylog.org/>. Accessed: 2023-04-12.
- [22] Insightops documentation. <https://docs.rapid7.com/insightops/>. Accessed: 2023-04-14.
- [23] Loghub: A large collection of system log datasets for log analysis research. <https://github.com/logpai/loghub>. Accessed: 2023-04-28.
- [24] Logstash: Collect, parse, transform logs - elastic. <https://www.elastic.co/logstash/>. Accessed: 2023-04-24.
- [25] Logz.io. <https://logz.io/>. Accessed: 2023-04-12.
- [26] Promtail: Configuration. <https://grafana.com/docs/loki/latest/clients/promtail/configuration/>. Accessed: 2023-04-24.
- [27] Sematext - it system monitoring tools for devops. <https://sematext.com/>. Accessed: 2023-04-12.

- [28] Splunk documentation. <https://docs.splunk.com/Documentation>. Accessed: 2023-04-15.
- [29] Sumo logic docs. <https://help.sumologic.com/>. Accessed: 2023-04-15.
- [30] Trustradius: Software reviews, software comparisons and more. <https://www.trustradius.com/>. Accessed: 2023-03-25.
- [31] The 5 v's of big data, Mar 2021. <https://www.techtarget.com/searchdatamanagement/definition/5-Vs-of-big-data>. Accessed 15.3.2023.
- [32] 2022 state of observability and log management, 2022. <https://go.era.co/state-of-observability/>. Accessed 10.3.2023.
- [33] ANASTOPOULOS, V., AND KATSIKAS, S. K. A methodology for building a log management infrastructure. In *2014 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)* (2014), pp. 000301–000306.
- [34] BANDOR, M. S. Quantitative Methods for Software Selection and Evaluation.
- [35] BRAY, T. The javascript object notation (json) data interchange format. <https://doi.org/10.17487/RFC8259>. Accessed 5.3.2023.
- [36] CÂNDIDO, J., ANICHE, M., AND VAN DEURSEN, A. Log-based software monitoring: a systematic mapping study. *PeerJ Comput. Sci.* 7 (May 2021), e489.
- [37] CHEN, B., AND JIANG, Z. M. J. Characterizing and detecting anti-patterns in the logging code. In *Proceedings of the 39th International Conference on Software Engineering* (2017), ICSE '17, IEEE Press, pp. 71–81.
- [38] CHUVAKIN, A., SCHMIDT, K. J., AND PHILLIPS, C. *Logging and log management: The Authoritative Guide to understanding the concepts surrounding logging and log management*. Syngress, 2013.
- [39] DU, X., YU, Y., WANG, P., HUANG, Z., AND WU, B. Unstructured log oriented fault diagnosis for operation and maintenance management. In *Proceedings of the 3rd International Conference on Computer Science and Application Engineering* (New York, NY, USA, 2019), CSAE '19, Association for Computing Machinery.

- [40] FEKETE, A. *CAP Theorem*. Springer New York, New York, NY, 2017, pp. 1–1.
- [41] GERHARDS, R. The syslog protocol. <https://doi.org/10.17487/rfc5424>. Accessed 5.3.2023.
- [42] GILLESPIE, M., AND GIVRE, C. *Understanding Log Analytics at Scale, 2nd Edition*. O’Reilly Media, Inc., 2021.
- [43] JADHAV, A. S., AND SONAR, R. M. Evaluating and selecting software packages: A review. *Information and Software Technology* 51, 3 (2009), 555–563.
- [44] JI, W., DUAN, S., CHEN, R., WANG, S., AND LING, Q. A cnn-based network failure prediction method with logs. In *2018 Chinese Control And Decision Conference (CCDC)* (2018), pp. 4087–4090.
- [45] KAREN KENT, M. S. Guide to computer security log management, 2006.
- [46] KREPS, J. *I Heart Logs*. O’Reilly Associates Incorporated, Oct 2014.
- [47] LIN, X., WANG, P., AND WU, B. Log analysis in cloud computing environment with hadoop and spark. In *2013 5th IEEE International Conference on Broadband Network Multimedia Technology* (2013), pp. 273–276.
- [48] MADANI, A., REZAYI, S., AND GHARAEI, H. Log management comprehensive architecture in security operation center (soc). In *2011 International Conference on Computational Aspects of Social Networks (CASoN)* (2011), pp. 284–289.
- [49] OKUMURA, M., AND FUJIMURA, S. Constructing a log collecting system using splunk and its application for service support. In *Proceedings of the 2016 ACM SIGUCCS Annual Conference* (New York, NY, USA, 2016), SIGUCCS ’16, Association for Computing Machinery, pp. 103–106.
- [50] OLINER, A., GANAPATHI, A., AND XU, W. Advances and challenges in log analysis. *Commun. ACM* 55, 2 (feb 2012), 55–61.
- [51] PETE SHOARD, ANDREW DAVIES, M. S. Magic quadrant for security information and event management, Oct 2022. <https://www.gartner.com/doc/reprints?id=1-2BDC4CEU&ct=221010&st=sb>. Accessed: 2023-03-25.

- [52] PHILLIP M. HALLAM-BAKE, B. B. Extended log file format. <https://www.w3.org/TR/WD-logfile.html>. Accessed 5.3.2023.
- [53] ROCHIM, A. F., AZIZ, M. A., AND FAUZI, A. Design log management system of computer network devices infrastructures based on elk stack. In *2019 International Conference on Electrical Engineering and Computer Science (ICECOS)* (2019), pp. 338–342.
- [54] SUGAYA, M., IGARASHI, K., GOSHIMA, M., NAKATA, S., AND KURAMITSU, K. Extensible online log analysis system. In *Proceedings of the 13th European Workshop on Dependable Computing* (New York, NY, USA, 2011), EWDC '11, Association for Computing Machinery, pp. 79–84.
- [55] SUKMA, N., SRISAWAT, W., SA-NGA NGAM, P., AND LEELASANTITHAM, A. An analysis of log management practices to reduce it operational costs using big data analytics. In *2019 4th Technology Innovation Management and Engineering Science International Conference (TIMES-iCON)* (2019), pp. 1–5.
- [56] VOSSEN, G. *ACID Properties*. Springer New York, New York, NY, 2016, pp. 1–3.
- [57] YACHING, D. The pivotal role of log analytics in modern it infrastructure, May 2019.
- [58] ZHU, J., HE, S., LIU, J., HE, P., XIE, Q., ZHENG, Z., AND LYU, M. R. Tools and benchmarks for automated log parsing. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice* (2019), ICSE-SEIP '19, IEEE Press, pp. 121–130.