Real-time anomaly detection in logs using rule mining and complex event processing at scale

DELFT UNIVERSITY OF TECHNOLOGY

MASTERS THESIS

Real-time anomaly detection in logs using rule mining and complex event processing at scale

Author:
Alexandros STAVROULAKIS

Supervisor:
Asterios KATSIFODIMOS

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

Web Information Systems Group
Software Technology

August 11, 2019

Declaration of Authorship

I, Alexandros STAVROULAKIS, declare that this thesis titled, "Real-time anomaly detection in logs using rule mining and complex event processing at scale" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: 

Date: 12/08/2019

An electronic version of this thesis is available at
http://repository.tudelft.nl/.

Real-time anomaly detection in logs using rule mining and complex event processing at scale

Abstract

Author:	Alexandros Stavroulakis
Student ID:	4747933
Email:	stavroulakisalexandros@gmail.com

Log data, produced from every computer system and program, are widely used as source of valuable information to monitor and understand their behavior and their health. However, as large-scale systems generate a massive amount of log data every minute, it is impossible to detect the cause of system failure by examining manually this huge size of data. Thus, there is a need for an automated tool for finding system's failure with little or none human effort.

Nowadays lots of methods exist that try to detect anomalies on system's logs by analyzing and applying various algorithms such as machine learning algorithms. However, experts argue that a system error can not be found by looking into a single event, but in multiple log event data are necessary to understand the root cause of a problem.

In this thesis work, we aim to detect patterns in sequential distributed system's logs that can capture effectively the abnormal behavior. Specifically as a first step, we will apply rule mining techniques to extract rules that represent an anomalous behavior, which potentially in the future may lead to a failure of a system. Except for that step, we implemented a real-time anomaly detection framework to detect problems before they actually occur.

Processing log data as streams is the only way to achieve a real-time detection concept. In that direction we will process streaming log data using a complex event processing technique. Specifically, we would like to combine rule mining algorithms with complex event processing engine to raise alerts on abnormal log data based on automatically generated patterns. The evaluation of the work is conducted on Hadoop's logs, a widely used system in the industry. The outcome of this thesis project gives really promising results, reaching a Recall of 98% in detecting anomalies. Finally, a scalable anomaly detection framework was build by integrating different systems into the cloud. The motivation behind this is the direct application of our framework to a real-life use case.

Thesis Committee:

Chair:	Prof. Dr. ir. Alessandro Bozzon, Faculty EEMCS, TU Delft
University Supervisor:	Dr. Asterios Katsifodimos, Faculty EEMCS, TU Delft
Company Supervisor:	Mr. Riccardo Vincelli, KPMG
Committee Member:	Dr. Mauricio Aniche, Faculty EEMCS, TU Delft

Acknowledgements

With the submission of this thesis I not only finishing my Master studies in Computer Science at Delft University of Technology, but I also conclude my student life. During this time I have been improved both as a scientist and as a person, by overcoming the challenges that I was facing. However the completion of this work would not have been possible without the help of many people that I want to thank.

First of all I want to thank my supervisor, Asterios Katsifodimos, for his endless help. His door was always open for me to listen to my problems and my crazy solutions. From the early stages of finding the thesis topic till the end of the work, his guidance was more than helpful and his expertise was crucial on lots of aspects.

Since this thesis was carried out as an internship in KPMG, I was really happy that I had a second supervisor from the company's side. Riccardo Vincelli thank you very much for your continuous support and guidance throughout the past nine months. With your experience and your enthusiasm we managed to overcome together lots of difficulties that I faced during this work. Thank you for spending much of your time with me, to listen to my issues and proposing interesting and helpful solutions.

I want to thank all my colleagues at KPMG for their help. Everyone helped a lot by listening and answering random questions that I had since they had experience in both engineering and research fields. Also I want to thank you all for welcoming me as a member of your team and making my time there very enjoyable.

Finally, I want to thank all my friends and my family for their unconditional support, not only during the past months, but during my whole student life. Without them it would be impossible to succeed.

Contents

List of Figures

List of Tables

List of Abbreviations

CEP	**C**omplex **E**vent **P**roccesing
ARM	**A**ssociation **R**ule **M**ining
SRM	**S**equential **R**ule **M**ining
HDFS	**H**adoop **D**istributed **F**ile **S**ystem
TP	**T**rue **P**ositive
FP	**F**alse **P**ositive
TN	**T**rue **N**egative
FN	**F**alse **N**egative
SMOTE	**S**ynthetic **M**inority **O**versampling **T**echnique
Gb	**G**iga **B**yte
CPU	**C**entral **P**rocessing **U**nit
TPE	**T**ree **P**arzen **E**stimator

Chapter 1

Introduction

Analyzing and monitoring a system's logs is not a new trend, even though it is one of the most popular research fields till now. Logs contain a large amounts of unstructured information about the operation of a system. However, due to their massive size and their non-consistent format, it is difficult to gain real value without extensive analysis. Especially nowadays, where more and more applications and services are deployed into the cloud, it is crucial to find a way to extract value from these system's log. Anomaly detection is probably the most common case to conduct an analysis in logs, because it detects things that went wrong in the execution. In that direction, this thesis aims to create an anomaly detection framework for applications and services hosted in the cloud. The research value of this work will be to detect anomalous patterns in logs via pattern mining techniques and then to integrate this machine learning technology with a complex event processing engine in streaming data, for real-time anomaly detection on system's logs.

1.1 Motivation

This thesis was carried out during an internship at KPMG. Thus, except for the motivation on scientific perspective, there is also direct motivation for the business itself. To begin with, the first motivation is to be able to create a real-time anomaly detection framework which can combine different scientific fields such as machine learning and streaming processing. Specifically, using complex event processing(CEP) in anomaly detection case on system's logs is quite interesting since no prior work has focused on this type of logs. The most researched logs that different works were focused on are web logs. However, in order to use a CEP engine, first some patterns have to be defined, which will be the patterns that have to be found in the streaming data. Generally, when using complex event processing technology, an expert is necessary, who will manually go through the logs and will determine the patterns that should be matched in the incoming streaming data. As one can imagine, due to the massive volume of the data it is almost impossible for an expert to go through all the log data and identify suspicious patterns for each case (it is a really time-consuming procedure which can also be very difficult). In order to reduce all this manual work, the pattern mining is introduced. Pattern mining is the procedure of detecting and extracting hidden patterns in the data. In this work, we will try to apply pattern mining algorithms in log data to extract patterns which can capture any abnormality that the log data contain. Another goal is to find a way to automate the pattern definition of the CEP engine by integrating it with machine learning technologies such as pattern mining techniques. Furthermore, we want to investigate what value can bring this work to a business like KPMG. Since most of the companies nowadays are moving their services into the cloud, it is crucial to adapt our anomaly detection framework in the current trends and deploy it into the cloud using state-of-the-art

tools. Finally, it is really important to find a general purpose of this work, such as the different possible applications that can be applied. For instance in can be exploited as a mean for finding the root cause of a system failure or for maintenance purposes of the systems in the cloud.

1.2 Research Questions

The aforementioned motivations lead us to investigate the following Research Questions and try to answer them through this thesis work.

1. Can we detect real-time anomalies in system logs in the cloud?
2. Can we use rule mining techniques to capture abnormalities in logs?
3. Can we learn rules that represent abnormal behavior ?
4. What technology can exploit patterns from logs?
5. What is the minimum number of abnormal data during training, in order to accurate capture future abnormalities?
6. On what use case for the industry this work can be applied?

The first question will be answered as the final outcome of this thesis work. Based on the final results we will be confident about the possibility of reaching the goal of this work by creating a new anomaly detection pipeline. The next two questions will be answered in Chapter 4, where we will investigate different machine learning approaches to extract patterns from log data. The fourth question will be answered as a next step of the previous questions.
One goal of this thesis is to generate patterns to be used in a complex event processing engine. Thus, the rules/patterns from the previous question will be used as patterns in the CEP. Question 5 is also a great challenge. As we know, under real world conditions, there is usually lack of abnormal data but plenty of normal, especially in cases such as the one that we investigate in this work. So it is important to examine what is the lowest bound for the number of abnormal data, from which we can generate patterns to capture efficiently future abnormal events. The last research question of this thesis is related both to the engineering part and to the dataset that we will use. Since the engineering part consists of quite a few different large-scale technologies and frameworks, and nowadays large companies such as KPMG are using them, it is quite relevant to create a log analysis framework by integrating these technologies. Also the dataset used is important, since it will be a real dataset from actual systems and thus the results will reflect a real case scenario.

1.3 Dataset

The dataset that we will use in this work contains logs from an instance of Hadoop Distributed File System(HDFS). Specifically this dataset was first introduced and created in [50] and Table 1.1 summarizes it.

System	Nodes	Messages	Size
Hadoop File System	203	11,197,692	2412 MB

TABLE 1.1: Hadoop File System Data

The log data were collected from the system running on Amazon's Elastic Compute Cloud. Every log message in the dataset corresponds to an event of a particular block in HDFS and the current project tries to detect the blocks which contain any abnormal behavior. An example of data points of the used dataset is presented below, where *blk∗* are the aforementioned blocks followed by the id. In Chapter 3 the motivation behind the choice of this dataset will be analyzed as well as the general pipeline of this work and all the required pre-processing steps that needed to be implemented.

```
081109 205613 929 INFO dfs.DataNode$PacketResponder: PacketResponder 0 for block blk_-3009784682784181114 terminating
081109 205613 929 INFO dfs.DataNode$PacketResponder: Received block blk_-3009784682784181114 of size 67108864 from /10.251.65.237
081109 205613 930 INFO dfs.DataNode$DataXceiver: Receiving block blk_373393615907828558 src: /10.251.35.1:38226 dest: /10.251.35.1:50010
081109 205613 930 INFO dfs.DataNode$PacketResponder: PacketResponder 1 for block blk_5157882435353849622 terminating
081109 205613 930 INFO dfs.DataNode$PacketResponder: Received block blk_5157882435353849622 of size 67108864 from /10.251.122.65
081109 205613 934 INFO dfs.DataNode$DataXceiver: Receiving block blk_-287809183401037259 src: /10.251.126.255:46911 dest: /10.251.126.255:50010
```

FIGURE 1.1: Raw HDFS data

1.4 Outline

This thesis will start with some information background about the different technologies and scientific fields that were necessary to implement the proposed methodology, as well as the relevant related work that is associated with the different fields of the thesis. In Chapter 3 we will present the case study of our work, along with the dataset and the metrics that we will use in order to evaluate this work. Chapter 4 contains the rule mining approach as well as the experiments of that field, to examine if we can generate and classify patterns from logs. Following we will present our anomaly detection tool using complex event processing engine with automate pattern construction. Chapter 6 contains the experiments and result of the thesis work, while the next Chapter presents the architecture of the framework and the theoretical background behind the used technologies. Finally, in Chapter 8 we evaluate our project based on previous similar works. Furthermore, we point the answers of the research questions that we have defined previously. Then we sum up our work, and we present its limitations as well as some further improvements.

Chapter 2

Literature Review

In this Chapter we will present some necessary background information that will be used through this project. A description of the different research fields, on which we will focus on, will be presented as well as, the necessary relevant related work.

2.1 Background

In this section, the most important definitions and terms for this thesis will be explained. Specifically we will present the theoretical background of what log analysis, complex event processing and rule mining are. This is necessary, since this project, which is a log analysis framework, is based on the integration of rule mining techniques with complex event processing technologies.

2.1.1 Log Analysis

All IT systems generate data, called logs, which represent their different activities and behavior. Log analysis is the field that evaluate these data in order to extract useful information for the system's functionality. However, since log data are quite difficult to be analyzed in their raw nature, a pre-processing is necessary to transform them into valuable information. Then it can be used in cases, such as, to detect patterns and anomalies, like system breaches. To conduct a log analysis, some complex processes have to be followed. For example the pre-processing that we mentioned before is necessary as well as some normalization and graph representation.

2.1.2 Complex Event Processing

Nowadays, due to the big data explosion and the daily use of sensors and smart devices, we collect continuously a massive amount of data that have to be analyzed [8]. A great challenge is to process the streaming data in real-time in order to detect event patterns on these streams. Complex event processing tries to address this challenge, to match these events with a number of predefined patterns. The outcome of this machine is some complex events which are a combination of simple events. Thus, CEP's advantage is that it can process an infinity number of data streams and detect desired complex event within them. This is the reason why CEP is quite popular today. It is mainly used in case where RFID-systems are used, to monitor and analyze the different devices. In order to detect complex events, an expert is required to define a set of rules-patterns that wanted to detect. For instance, a pattern can be: *if you receive a value of temperature above 40 degrees and detect smoke then make alert of fire*. In this case, data are collected from different sensors and if the above pattern is fulfilled within a time window, then an alert of fire is raising.

2.1.3 Rule Mining

Rule mining is one of the most popular and successful techniques of data mining. Its scope is to extract frequent patterns, interesting correlations and association structures among set of items in databases. The result of rule mining is some $IF..THEN$ statements that show the probability and relationship between items in a database.

Definition

The formal definition of rule mining firstly stated in [1]. Let I = I_1,I_2,..,I_m be a set of m distinct values, T be a transaction that contains a set of items such as T⊆I and D is the database that contains different transactions T_s. A rule is of a form $X \Rightarrow Y$ where X,Y⊂I are set of items that are called itemset and $X \bigcap Y \Rightarrow \emptyset$. These itemsets are the two most important parts of rule mining. X is called antecedent and Y is called consequent while the rule mean that X implies Y.

In rule mining we have to take into consideration two major values, the *support* and the *confidence* factors. These factors are assigned to a minimum value in order to remove rules and items that are not interesting for the user and to identify the most important relationships between the items.

- Support is an indicator of how frequently an item is exist in the data. Formally it is called the percentage/fractions of records that contain $X \bigcup Y$ to the total number of records in the database. This factor is calculated by : Support(XY) $= \frac{\text{Support count of XY}}{\text{Total number of items in database}}$

- Confidence indicates how many an $IF..THEN$ statements are found to be true in the data. Formally, confidence of association rule is defined as the percentage/fraction of the number of transactions that contain $X \bigcup Y$ to the total number of records that contain X, where if the percentage exceeds the threshold of confidence an interesting association rule $X \Rightarrow Y$ can be generated. The calculation of confidence is coming from $Confidence(X|Y) = \frac{\text{Support(XY)}}{\text{Support(X)}}$ formula.

As mentioned in [54], a problem in rule mining is split into two parts. The first step is to find the frequent itemsets whose occurrences exceed a predefined threshold in the database (support). The second part is to generate rules from those frequent itemsets with the constraints of minimal confidence. To give a better understanding of how rule mining technique works, we present the following example where we have a transaction database with different purchased items.

Transaction Id	Purchased items
1	A,D
2	A,C
3	A,B,C
4	B,E,F

TABLE 2.1: Rule mining example

Based on the above table we have for instance the itemset (A, B) with support value equal to one or the itemset (A, C) with support value equal to two. Thus, if we want to generate rules with 50% minimum support and 50% minimum confidence we have the following rules:

- A→ C with 50% support and 66% confidence
- C→ A with 50% support and 100% confidence

However in our case we will focus solely on the support factor since we will generate frequent itemset that we will then convert them to rule patterns.

The rule mining methods are split in two distinct main categories: association rule mining[52] and sequential rule mining[53]. The main difference between these two methods is that association rule mining algorithms does not take into consideration the actual order of transaction. Thus, the generated frequent itemsets from the association rule mining method are usually more that the sequential method. On the other hand, in sequential mining, the order of the transaction is really important thus the generated itemsets contains the property of the correct order. However due to that restriction, the sequential algorithms are a bit slower than the association ones. In this work we will examine the effect of both methods in the anomaly detection case.

2.2 Related Work

After the explanation of the most important definitions and terms, it is crucial to present all the works that are related to the topics we will touch upon this thesis. Thus in the following sections, related literature that focuses on similar fields will be presented, and their weak and strong points will be analyzed.

2.2.1 Real-time log analysis

It is widely known that all IT systems generate records called *logs*, which capture the activities of the systems. Logs analysis is the work of examining these data in order to extract useful information about system activity and handling errors that occurred in the systems. Log analysis is a quite hot trend nowadays and as more and more IT infrastructures move to public clouds such as Amazon Web Services, Microsoft Azure and Google Cloud, it is crucial to conduct this analysis in real time aspect. In that direction, lots of production tools exist that trying to address that challenge. For instance, the state of the art tools are ELK [13] which is a stack of three tools (Elasticsearch, Logstash and Kibana) and Splunk[48] which collect, analyze and visualize logs from different sources. ELK is used for centralizing logging in IT environments. It includes ElasticSearch which is a NoSql database to store and query logs. Also it includes the Logstash which is the tool for gathering the logs from different sources and Kibana which is responsible for visualizing the logs. Although ELK's fame, it contains some pitfalls such as absence of anomaly detection capabilities.

All these tools use a supervised-based analysis method where they allow users to define log patterns or generate models based on domain knowledge. Despite their advantages, these methods have some shortcomings such as it is focused to what the user seeks, on known errors and also it is not adaptive to new data sources and formats. Except from the aforementioned state-of-the-art tools, also other tools such as *Logmatic*, *Sumo Logic*, *logz.io*, *Loggly*, etc exist, which are really famous in the field of log analysis and anomaly detection in logs. However almost all of these tools do not exploit the possibilities of applying machine learning techniques in order to analyze and detect insights in logs. Only *Sumo logic* use machine learning techniques, although they use only clustering or regression analysis which might not the best option to detect hidden pattern between logs from different sources. In our work we

want to investigate the integration of machine learning, and specifically some rule mining techniques, on the anomaly detection case.

Log analysis at scale

Another issue that has been emerged recently is the effectiveness of log analysis tools in large scale systems. Since we are in big data era, it is crucial to analyze big volumes of logs that emitted from different types of systems and applications. Except from some of the aforementioned production tools that have been successful in large scale systems(such as ELK), lots of research work has been done in that field of log analysis at scale. For instance in [44] they create a large scale monitoring system. Their system was build on top of Hadoop in order to inherit its scalability and robustness. They have also created a toolkit form displaying and analyzing the results of the collected log data. However their system do not aim at monitoring for failure detection but for a system that can process large volume of log data. Similarly, [49] have created a cloud-based aggregation and query platform for analyzing logs. In order to achieve storage and computation scalability they used OpenStack and MongoDb systems. Their scope was to store the logs, to aggregate them and to make queries on them to extract useful information. However, both methods that we mentioned before do not use any machine learning technique to get insights from log data. As it is obvious, the biggest goal of log analysis is to detect anomalies in the data to find root causes of any problem that occur. For that reason, the use of machine learning is more than necessary in that field.

For instance Debnath et al. [10] have created a real-time log analysis system using machine learning. Specifically they employ unsupervised machine learning to discover patterns in application logs and then to leverage these patterns with a real-time log parsing for an advanced log analytic tool. The above patterns are learned from "correct" logs and then, the learned models can identify anomalies. In order to operate in large scale, their system is deployed on top of Spark. Also they used Kafka for shipping logs and Elasticsearch to store them. Finally, they used also Kibana to visualize results and writing queries. Their model is consisted of :

1. The tokenization where logs are split into units
2. The identification of datatype based on RegEx rules
3. The pattern discovery using clustering
4. The incorporating domain knowledge where users can modify the extracted patterns

However their approach lacks of real-time service with zero-downtime since Spark lacks these features.

Another interesting work came from Xu et al. [50] where they first parse logs by combining source code schema analysis in order to create features of logs, and then they analyze these features using machine learning approached to detect anomalies. In their approach, they first convert logs in a structure format using the schema of the source code that produce the logs. Then, they create feature vectors by grouping related messages. After that they conduct anomaly detection using PCA and finally they visualize the results in a decision tree which explain how the problems have occurred and detected.

Furthermore, instead of machine learning, several works exists which are trying to tackle log analysis and anomaly detection using data mining techniques. For

instance, Jakub et al. [23] use data mining and Hadoop technique for log analysis. They used Hadoop MapReduce method in order to execute data mining in parallel for faster calculation. As the data mining part is concerned, they generate rules based on the logs data and then based on these rules they identify anomalies. The rule generation is made by grouping logs with some same features such as time, session and IP address.

2.2.2 Anomaly detection in logs

Log analysis with the aim to detect anomalies is not a new field, thus lots of works exist that trying to tackle this matter. For example, Min Du et al. [12] create an anomaly detection tool using Deep Learning techniques. They create a neural network using Long-Short term memory to model systems log as natural language sequences achieving really good score in Recall and Precision metrics. Their contribution is to capture normal behavior into patterns and then, any data deviates from these patterns are categorizes as abnormal. Furthermore, they argue that logs data can varies over time. Thus, updating the patterns collected online it is really crucial in order to be able to capture future abnormalities. To make this update they use users that provide their Deep Learning model with feedback about the current anomaly detection results.
Mariani et al. [34] trying to identify failure causes through system logs. They detect relationship between values and event in the logs of normal behavior and then they build a model that compares these dependencies with models generated by failure executions in order to detect which sequences brought the failure. Specifically it is a heuristic-based technique that capture problems not from a single event but from sequences of different system event. Their approach is split into three phases. In the first phase they collect the logs, by transforming the raw date into event and attributes. Then in the next phase they generate the model using FSA to depict the dependencies between the events. The last phase contains the failure analysis by comparing the different FSA models from the previous step.

A differently work come from Hayens et al. [21] where they present a contextual anomaly detection method in logs from streaming sensor network. The purpose of their work is that it is crucial to detect anomalies not only based on the content of the data but also from the context perspective. The proposed technique is composed of two modules: the content anomaly detector and the contextual anomaly detector. In the first module, an univariate Gaussian predictor detect the point anomalies. The other module creates sensors profiles and evaluate every sensor based on the value of the content data as anomalous or not. The profiles are defines using clustering algorithm. Thus when each sensor is assigned to a profiles and then is detected as anomalous by content, the context anomaly detector will determine the average expected value of the sensor group.

Lots of works are trying to detect anomalies in logs. However, since it is always crucial to find new methods to detect anomalies, we decided to create our own anomaly detection tool. Also our purpose in to deploy the system into the cloud, in contrast to the aforementioned related works.

2.2.3 Pattern mining in logs

Extracting pattern from log data is not a new trend, however it is very effective especially in case of finding patterns to capture abnormality in logs is crucial. For example Kimura et al. [26] describe a method to generate logs patterns for proactive

failure detection in large scale networks, because in such systems it is impossible to find the root of a problem just by examine the massive amount of log data. Their proposed system, which automatically learn relationships between log data and failures, consist of three sub-modules. The first one convert the log messages into log templates using a similarity score for each word to belong to a specific log template cluster. This module is applied in chunks of log messages because they argue that a failure stems from a combination of log massages, not by just one. Then, the second module collect the previous templates and tries to extract features that can characterize the generating patterns-templates. The final module contains a supervised machine learning approach to classify each pattern with respect of failure on the system or not. As ML technique they use the well-known SVM classifier to classify the future chunk log records. In their experiments, using Blue gen data, they achieve really good results in trying to proactively detect failures in network systems.

In the same direction, Dharmik et al. [6] used pattern mining techniques to identify security breaches in log records by dynamic rule generation. To create the rules, they used some specific attributes of the log data, such as IP address and port, since the searching field is the networks security, and they apply association rule mining algorithms such as Apriori. Then with these rules they train an anomaly detection classifier to detect future anomalies.

Another work comes from Hamooni et al. [20], where they have build a framework to extract patterns from log messages using unsupervised techniques. Specifically, they separate log messages into clusters and then they try to generate patterns that represent the log messages in each cluster. To group the messages they measure the maximum similarity distance between the messages. Then they use the Smith-Waterman algorithm to generate patterns for the cluster. As final step they try to combine the produced patterns in order to have a small number of of patterns that represent the whole space of log message.

Pattern mining in logs is not a new trend, thus lots of works exist on that field. However we can see based on the above works, that the case they investigate is to find pattern on network logs in order to capture patterns for intrusions. In our case we want to extract patterns to capture abnormality of the functionality of distributed systems.

2.2.4 Complex Event Processing in Logs

Since Complex Event Processing has gain a lots of interest the last years, some efforts exist on exploiting its capabilities in log analysis. For instance, Jayan et al. [24] describe a method to integrate Complex event processing with Network security. Specifically, they argue that it is quite difficult to detect a network security breach when an attack can occur in a lots of different devices. Thus, they describe a method of pre-processing all the sys log data in order to extract hidden patters on these data. Then they feed the CEP engine with the pre-processed data to create different alarms when an attack is occurred based on some specific features. Similarly, the same authors, make use of CEP and sys logs data by integrating also machine learning technology [25]. They argue that since the log data are massive, the reduction of their size is necessary. Thus they apply a SVM reduction on the pre-processed data before parsing them to CEP engine. By this reduction they improve their previous results [24] and they come up with the a solution to detect a network attack using both machine learning and Complex Event Processing.

In a complete different field Grell et al. [18] use Complex Event processing to monitor large scale internet services. Since these internet services are deployed in

large scale data centers, it is crucial to monitor their event. They conduct two different experiments, one using a watchdog inside the different services and one using business events and data such as CPU consumption, memory allocation, as well as user's action. Then applying CEP in these log data it is possible to detect future issues in the systems and to find the root cause of multiple problems based on the rules that can be generated from the data.

All related work that make use of complex event processing in logs are associated with networking logs. Thus, definitely we can argue that there is a absence of using complex event processing on system logs of distributed systems, which we will discuss in this thesis work.

2.2.5 Automated Complex Event Processing

Complex event processing (CEP) is a popular methodology which provides us with the possibility of analyzing real-time streams of data. The main purpose of CEP is to detect complex pattern in different types of data such as logs or RFID data. In order to detect these pattern it is necessary to provide to the system some pattern rules that can capture the events in the data. These rules are generally provided by experts of a specific field. However, nowadays, where we have a huge amount of data, it is difficult to manually determine these rules. Thus it is necessary to automate this process of defining rules for the CEP engine.

In that direction, Mehdiyev et al.[35] tried to automate that process of generating pattern rules in in complex event processing. The goal of their work is to use a machine learning model for identification and updating of pattern rules. For that reason they used Rule-based classifiers (supervised learning) on streaming data of RFID sensors. Specifically, they conduct experiments using six different rule-based classifiers (One-R ,JRip, PART, DTNB ,Ridor and Non-Nested Generalized Exemplars). The reasoning for choosing rule-based classifiers is the detection of rule patterns in an offline mode and then to provide the extracted rules to CEP engine, which can capture complex process events from streaming data. As the behavior of streaming data changes over time, in a specific period, the rule pattern identification has to be conducted offline and then submitted to the CEP engine. In their experiments, using accelerometer sensor data, they manage to achieve a really good accuracy using the above classifiers and thus they conclude that Rule-based classifiers can be use efficiently to generate rules for CEP system.

Another approach that use an online learning technique for rule generator is of Petersen et al.[40]. In their work, not only used machine learning algorithms to generate rules for the CEP engine but also these rules generated online based on the inputs and feedback in the system. Their method consist of three different stages. In the first stage, events are received and being pre-processed before performing some action due to possible failures in the data transfer, missing values or incorrect measurements. The second stage is the extraction of rule patterns from the data stream provided as a training data for the system. This stage begins with defining a time window (using an iterative method to define the size) for each incoming complex event in training phase. In parallel with this process, an online SVM with Gaussian radial basis function kernel (RBF) is created to classify complex events. The last step in this stage is to extract rules from the SVD model. This is achieved by finding the centroid for each Complex Event through a clustering algorithm. Then, support vectors, from the SVM model, and centroids are used to determinate the boundaries of a region of interest, which define an interval rule. The last stage of their work is to feed the rules into the CEP engine for processing the data streams. However,

using the same data as the previous work, they did not manage to achieve greater accuracy when using the ML technique alongside with CEP instead of just using Machine learning technique to classify the action of the person based on the sensor data.

Similarly Petersen et al. [41] proposed an automated method to generate Cep rules by mining event data. Their unsupervised approach has two stages. In the first stage an event clustering is proposed in order to detect complex event patterns from unlabeled data and then can be labeled with arbitrary classes. For that purpose they used the X-means clustering algorithm to define the clusters. Then labels to the clusters are assigned by experts.
The second step is to define rules from the labeled clusters. To accomplish that, they used the labeled clusters as a training set to build a SVM classifier for rule generation. Finally, their results were quite similar with those in their previous work [40].

Another method for automating CEP comes from Mousheimish et al. [37], where they propose a learning algorithm where complex patterns from multivariate time series to be learned. The purpose of this work is to produce CEP rules for proactive purposes. Their method is divided into two parts. The first one named USE and SEE employed at design-time, and the second named auto-CEP for the run-time processing. More specifically, the entire approach combines Early Classification on Time Series (ECTS) techniques with the technology of complex event processing. Their method contributes in both field of CEP and data mining. In the first step of their method (USE), they take classified instances and they extract a set of shapelets. Then, they extract sequences from the previous generated shapelets. The last step contains the parsing sequences to transform them into rule to deployed in the auto-CEP scheme.

An interesting work has been done from Margara et al.[33], where they proposed a framework called iCEP, for automated CEP rule generation, which find meaningful;l event patterns by analyzing historical traces.They extract event types and attributes and they apply an intersection approach on those patterns to detect hidden causalities between primitive and composite events. However their experimenting results were not the optimal, reaching a Recall and Precision values around 85-90%.

Another interesting work in association rule mining in combination with complex event processing presented by Jinglei Qu et al. [32] . Specifically they proposed a model of autoCep for online monitoring which automatically generated CEP rules using association rule mining methods. Their method is split in 3 phases. In the first phase, a pre-processing step on the data is occurred in order to keep only the most important attributes for their case. Then using association rule mining they generate rules using as input the output of the previous phase. The last phase was to convert the generated rules into CEP rules to be fed in the CEP engine. In order to keep only the most important attributes (factor in their case), in the first step, they used the gray entropy correlation analysis. In the second step they used the Apriori algorithm with fixed minimum support and confidence. A generated rule from that process had the format of $R =< condition, result >$. After the creation of rules an algorithm was applied to transform them into Cep rules. From the previous format, the result is extracted as the event type and the condition as the Cep pattern. Finally, also a timed window is defined in order to determine the pattern that fells in that window.

Another work that combines CEP and association rule mining is the system proposed by Moraru et al. [36]. In their work, they integrate CEP with data mining technique to be used on a smart cities scenario where data comes from multiple

sources. As the previous work, they use Apriori algorithm implemented in WEKA. Thus after collect the data, they generate rules to depict the most frequent itemsets. Then they come to some interesting correlations between the different data sources that can be used in a CEP engine for alerts on abnormal behavior.

Furthermore Mutschler et al. [38] propose a rule mining method using an extension of Hidden Markov Models, which is called Noise Hidden Markov Models. These models can be trained with existing low-level event data. The idea is to provide a method by which a domain expert can tag the occurrence of a important incident at a specific point in a period of time in a stream. The system then infers rules for automatically detecting such occurrences. Final, these rules are used then in a combination with Complex event processing to capture future occurrences on the rules.

As we can see from the above related works, lots of methods exist in order to automate the pattern generation of complex event processing. However only few works used association rule mining techniques for that purpose and none of them use sequential pattern mining. Also we saw that none of the aforementioned methods used their systems to detect anomalies on system logs, something that we want to investigate. Specifically if complex event processing can be used in anomaly detection in logs.

Chapter 3

Case study: Anomaly detection on distributed file-system logs

In this Chapter, the case of this thesis will be presented, as well as the dataset that was used. As mentioned in Chapter 1, the analyzed dataset contains log messages from the Hadoop File system [46]. However since this dataset contains raw log messages, it is crucial to explain the steps that were conducted in order to generate new, modified, in a structured way, datasets that can serve our purposes. Specifically based on the initial dataset, we have generated two distinct datasets, each used in different steps of our work. Nonetheless, initially have to be mentioned the reasons behind the decision to create these two versions of the initial dataset. This kind of data is really representative to examine the effectiveness of our framework since they are real log events from a widely used distributed system. Also recently, more and more companies are choosing to deploy a Hadoop cluster inside a Kubernetes cluster, thus our case will serve exact this purpose, to detect anomalies in a system inside Kubernetes. Furthermore, another reason for choosing HDFS logs is that each log message is connected with a specific entity, which in our case are the Blocks. Finally, in HDFS logs, a message solely can not depict any information or abnormality. A characteristic of these logs is that when a problem occurs, then you can examine it from a sequence of different message. Thus, since in our case we will try to extract patterns from sequential log messages, it is really useful to know prior if that sequence of messages can represent an abnormality or not.

3.1 General pipeline

In this section we will present a brief overview of the pipeline of the framework that we developed for anomaly detection. One purpose of this thesis is to combine the rule mining field with the complex event processing engine. Specifically we want to extract patterns from abnormal logs and the to feed these patterns in a complex event processing engine to detect future abnormal behavior in streaming log data. An overview of the whole general method is presented in 3.1.

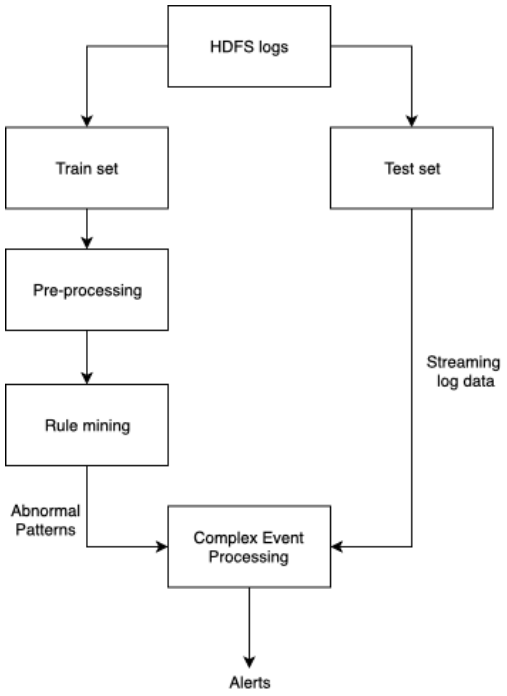

FIGURE 3.1: General Pipeline

After the graph representation of the framework, it is really important to analyze the necessary pre-processing steps conducted on the dataset.

3.2 Data pre-processing

To begin with, since the data are raw log files from the system, the first step is to pre-process them in order to handle them. Firstly we have to extract the block that each line is referred to, as well as the timestamp when the corresponding log line occurred. As an example we can examine the following log line and the extracted parts that we gathered.

```
081109 203518 143 INFO dfs.DataNode$DataXceiver: Receiving block blk_-1608999687919862906 src: /10.250.19.102:54106 dest: /10.250.19.102:50010
```

FIGURE 3.2: Raw log message

From the above log line we can extract:

- The Block id : *blk*_1608999687919862906
- The Time this log message occur : 081109 203518 143
- The actual message : *INFO dfs.DataNodeDataXceiver: Receiving block src:* /10.250.19.102 : 54106 *dest:* /10.250.19.102 : 50010

Then we have to encode the entire message of the log line to a specific number for simplicity. Table 3.1 presents that relationship between numbers and messages. Consequently instead of raw messages for each block, we have a sequence of numbers for each block which represents all the messages that each block contains. This

can also be found in Figure 3.3 where we present a graph of how raw messages are pre-processed into a structure form.

Messages	Number representation
Adding an already existing block (.*)	1
(.*)Verification succeeded for (.*)	2
(.*) Served block (.*) to (.*)	3
(.*):Got exception while serving (.*) to (.*):(.*)	4
Receiving block (.*) src: (.*) dest: (.*)	5
Received block (.*) src: (.*) dest: (.*) of size ([-]?[0-9]+)	6
writeBlock (.*) received exception (.*)	7
PacketResponder ([-]?[0-9]+) for block (.*) Interrupted	8
Received block (.*) of size ([-]?[0-9]+) from (.*)	9
PacketResponder (.*) ([-]?[0-9]+) Exception (.*)	10
PacketResponder ([-]?[0-9]+) for block (.*) terminating	11
(.*):Exception writing block (.*) to mirror (.*)(.*)	12
Receiving empty packet for block (.*)	13
Exception in receiveBlock for block (.*) (.*)	14
Changing block file offset of block (.*) from ([-]?[0-9]+) to ([-]?[0-9]+) meta file offset to ([-]?[0-9]+)	15
(.*):Transmitted block (.*) to (.*)	16
(.*):Failed to transfer (.*) to (.*) got (.*)	17
(.*) Starting thread to transfer block (.*) to (.*)	18
Reopen Block (.*)	19
Unexpected error trying to delete block (.*)ḂlockInfo not found in volumeMap	20
Deleting block (.*) file (.*)	21
BLOCK NameSystemȧllocateBlock: (.*)(.*)	22
BLOCK NameSystemḋelete: (.*) is added to invalidSet of (.*)	23
BLOCK Removing block (.*) from needed Replications as it does not belong to any file	24
BLOCK ask (.*) to replicate (.*) to (.*)	25
BLOCK* NameSystem ȧddStoredBlock : blockMap updated: (.*) is added to (.*) size ([-]?[0-9]+)	26
BLOCK NameSystem .addStoredBlock: Redundant addStoredBlock request received for (.*) on (.*) size ([-]?[0-9]+)	27
BLOCK* NameSystem .addStoredBlock: addStoredBlock : blockMap updated: (.*) is added to (.*) size ([-]?[0-9]+) But it does not belong to any file	28
PendingReplicationMonitor timed out block (.*)	29

TABLE 3.1: Message representation to numbers

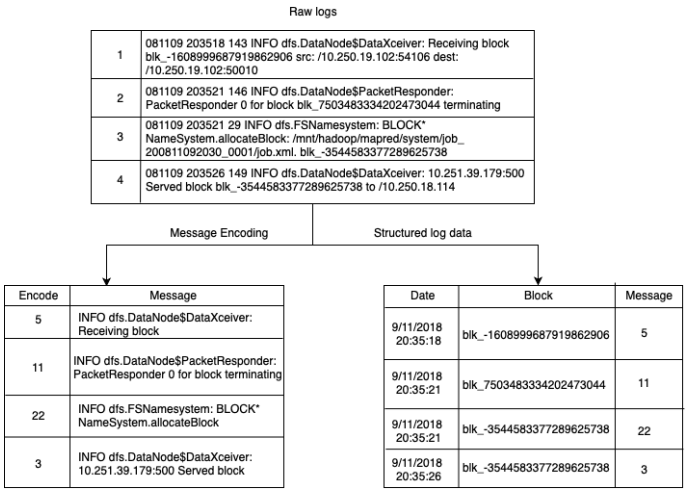

FIGURE 3.3: Data pre-processing pipeline

In total, the dataset contains 553.367 normal blocks and 16.839 abnormal blocks well- defined from the experts, in these 11 million log messages. Except for the number of normal and abnormal blocks, it is really important to present want kind of HDFS log messages we have. As we said, our dataset contains around 11 million messages consisting of 10.834.620 INFO, 362.814 WARNING and 258 ERROR messages. Since our goal is to detect an abnormality in patterns before it occurs we exclude the ERROR messages from our data. Thus, the goal is to detect an abnormal behavior only by examining "safe" messages such as INFO and WARNINGS instead of ERROR and FATAL which obviously depict an abnormal case. In the following section we will describe the generation of two new datasets that we used in our experiments.

3.3 Datasets generation

The first modification of the initial dataset, will be used in Chapter 4 for the generation and the examination of the different rule mining algorithms. For that purpose we had to group all the logs messages based on the Block id to create a dataset where each line will represent all the messages of a specific block in the correct time order. So for every line in the raw data, we use the pre-processing transformation, as presented in Table 3.1 and then we group the messages based on the Block name. An example of this modification is presented in Figure 3.4 where each line is a specific Block and all the sequence numbers are the different log messages that correspond to that Block in chronological order.

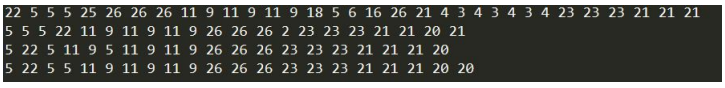

FIGURE 3.4: Grouped dataset

Using this dataset we managed to extract the frequent itemsets for each block and conduct the anomaly detection as we will in Chapter 4.

The second dataset that we created was necessary due to the limited resources that we have in order to test the complete framework of the log analysis. For that reason we used around 2 millions randomly picked event logs from the initial raw dataset, which represent 103.620 normal blocks and 2.792 abnormal blocks. This dataset will be used in Chapter 5, where the final framework is presented, as well as in Chapter 6, where the results of the different experiments will be presented.

To conclude, from the initial raw log data we had to create two different, modified, datasets, one by grouping all the messages based on the Block id, which is used in the experiments of Chapter 4, and a smaller than the initial one, which is used for the testing our log analysis framework in Chapters 5 and 6.

3.4 Metrics definition

To test the performance of our experiments and consequently the performance of the anomaly detection framework, we have to define what kind of metrics we will use to measure its effectiveness. To begin with, the entities, which will be characterized as positive and negative have to be defined. As we are focusing on detecting anomalies, we define as positive class the anomalous and as negative class the benign data. Thus, as *True Positives* we consider the abnormal data that we correctly categorized them as abnormal. As *False Positives* we define the normal data that we wrongly detect them as abnormal. Finally, *True Negative* and *False Negatives* are the righted categorized normal data and the wrongly characterization of abnormal data as normal respectively. Using the aforementioned definitions, the metrics will be presented.

As mentioned in [42], most anomaly detection approaches have abandoned the use of accuracy or the True Positive rates since they cannot evaluate correct the effectiveness of the approach. Since in that case, a False Negative error is most of the time more costly than False Positive ones, a common approach is to use metrics such as Recall and Precision. Recall measure how well the system can detect the anomalies, thus we will satisfy with a value of Recall as close to 100% as possible, which means that all anomalies are detected. Precision, on the other hand, measures the ratio of correctly predicted positive observations to the total amount of predicted positive observations. Since the goal is anomaly detection and usually in this field there is an issue with imbalance data (as we mentioned earlier, normal data are much more frequent than the abnormal ones), it is possible to end up with a small precision value. Finally, we measure also the F1 score, which is the harmonic mean of precision and recall. However in order to have a large F1 score we must have high value on both Precision and Recall. The formulas of all metrics that will be used are presented below.

$$Recall = \frac{TP}{TP + FN} \tag{3.1}$$

$$Precision = \frac{TP}{TP + FP} \tag{3.2}$$

$$F1\ score = 2 \times \frac{Recall \times Precision}{Recall + Precision} \tag{3.3}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.4}$$

In [42] is mentioned that the use of Accuracy metric is not appropriate for anomaly detection case. However in our work we make use of this metric, but not directly for the anomaly detection problem. The reason behind this decision is that in the first phase of our approach and experiments, we are trying to examine if we can find pattern in data that can efficiently capture abnormality. Thus, the goal is to generate frequent itemsets and classify them as normal or abnormal. So these experiments are a typical classification problem and thus accuracy is the most important measure. In the second phase of our work, where our actual framework exist, we make use of Recall, Precision and F1 score metric which are the most useful for the anomaly detection manner.

Chapter 4

Anomaly detection on patterns using rule mining techniques

The first phase of this work requires to extract hidden patterns from logs in order to examine if these patterns can accurately capture the abnormal behavior in the data. These hidden patterns will be the rules in the next phase which performs the actual logs analysis of the presented framework. Thus, from the data, the most frequent itemsets will be extracted and then we will try to classify these itemsets as normal or abnormal based on the data from their source. For that reason, in the experiments of this phase, we will focus on the *Accuracy* metric which is the most relevant for our classification aspect.

To give a better explanation about this procedure, we will use the data as presented in Figure 3.4 and all the required steps are depicted in Figure 4.1.

As mentioned before, the purpose of all the experiments in this chapter is to examine if the frequent itemset can play the role of rules in the next phase in order to capture the abnormal behavior in the log data. Since the state of the art mining methods of frequent itemset are the *association* and *sequential rule mining*, it is important to investigate both methods to have a clear view about the potential use of the frequent itemsets in anomaly detection of logs.

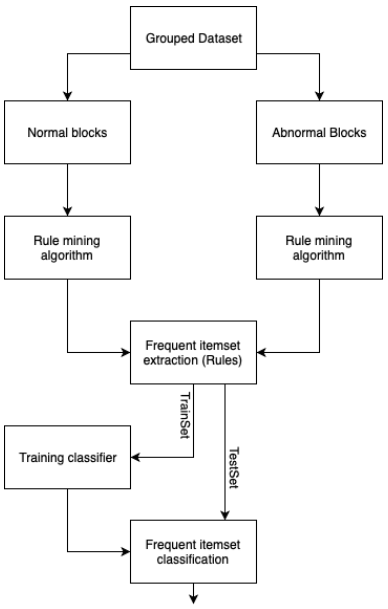

FIGURE 4.1: Classification of frequent itemsets Pipeline

4.1 Dataset

For the experiments of this Chapter, the first modified dataset, as presented in Chapter 3, will be used, where each line represent all the event messages in chronological order of every block. Since we have the labels for each line of the dataset we can conduct our experiments to examine if we can capture efficiently the abnormal behavior in logs and correctly classify the frequent itemset as normal or abnormal.The number of normal data points that we have is 553.367 while the abnormal ones are only 16.839 entries. However this is a common case when working with anomaly detection in logs since the normal data are more common and more frequent that the abnormal. Thus, in the following experiments also the problem of imbalanced data has to be addressed.

4.2 Method 1: Association rule mining

In order to examine whether *association rule mining*(ARM) is the appropriate technique to analyze the logs and capture the abnormality in the frequent itemsets, we have to make experiments using the state-of-the-art ARM algorithms. The goal of this experiment is to classify the rules (the most frequent itemsets) either as normal or abnormal, based on the given dataset. To generate the most frequent itemsets we used the *Apriori algorithm*[3]. Furthermore, to handle the imbalanced data (since we have more normal than abnormal data), we apply SMOTE[9] on the training dataset. Last we use the Random forest classifier to conduct the classification task, using also a 10-fold cross validation [27].

An important parameter of the algorithms is the support factor. More details can be found on Chapter 2. Various experiments were conducted in order to identify the most appropriate value of that parameter. More specifically we tried the generation of the frequent itemsets with support factor values in range of 1% - 60% of the number of data entries, and different number of data fields, ranging for 5 to 20 messages per entry, in order to have more clear results about the efficiency of using association rule mining to detect anomalies in logs. Also we decided to keep those itemsets that include at least 2 items, because the sets that contain only one item cannot be considered as a pattern worth looking for. Lastly experiments with another restriction were conducted. In one case we kept all frequent itemsets from both normal and abnormal data. In the other case, we make a comparison in the frequent itemsets and we remove from the abnormal frequent itemset dataset those that exist also in the normal one.

Actually the following results of these experiments confirm our initial thought that ARM is not the appropriate technique for our case, since it does not take into consideration the sequences of the log messages, something which is really crucial in our case to detect the anomalies in real time and predict the errors.

The following matrices and graphs present the results of the experimenting using ARM and the different values of the adjustable parameters as well as the restriction of the comparison or not, of the common frequent itemsets of normal and abnormal.

Num.of.Fields	Min_Support	Accuracy	TN	FN	FP	TP
5	0.03	0.49	2317	2121	3216	2929
	0.05	0.5	2662	2423	2871	2627
	0.08	**0.5**	**2483**	**2243**	**3050**	**2807**
	0.1	0.49	2568	2422	2965	2628
	0.2	0.48	1106	1010	4427	4040
	0.3	0.5	2767	2525	2767	2525
	0.4	0.49	1660	1515	3873	3535
10	0.01	0.45	1674	1957	3859	3093
	0.03	0.47	2499	2522	3034	2528
	0.05	0.49	2691	2494	2842	2556
	0.08	0.52	3037	2551	2496	2499
	0.1	0.48	2579	2460	2954	2590
	0.2	0.51	2332	1899	3201	3151
	0.4	0.48	2624	2571	2909	2479
	0.5	**0.52**	**2988**	**2531**	**2545**	**2519**
	0.6	0.49	2267	2087	3266	2963
15	0.01	0.47	3072	3178	2461	1872
	0.03	0.49	3261	3161	2272	1889
	0.05	0.39	1833	2706	3700	2344
	0.08	0.48	3616	3568	1917	1482
	0.1	0.49	3562	3372	1972	1678
	0.2	0.48	2067	2064	3466	2986
	0.3	0.55	4565	3739	968	1311
	0.4	0.57	5101	4074	433	976
	0.5	**0.57**	**4976**	**4023**	**557**	**1027**
	0.6	0.52	2943	2445	2590	2605
20	0.01	0.30	132	1947	5402	3103
	0.03	0.35	715	2034	4818	3016
	0.05	0.37	297	1468	5236	3583
	0.08	0.41	1468	2083	4065	2967
	0.1	0.38	772	1731	4762	3319
	0.2	0.38	1642	2567	3891	2483
	0.3	0.58	3632	2527	1901	2523
	0.4	0.59	5313	4093	220	957
	0.5	0.58	5240	4063	293	987
	0.6	**0.59**	**5210**	**4029**	**323**	**1021**

TABLE 4.1: ARM results, without itemset comparison

Fields	Min_Support	Accuracy	TN	FN	FP	TP
10	0.01	0.44	1191	1553	4342	3497
15	**0.01**	**0.52**	**5202**	**4736**	**331**	**314**
	0.03	0.52	5236	4735	297	315
	0.05	0.46	4058	4148	1475	902
	0.08	0.43	3233	3643	2301	1407
	0.1	0.5	4564	4358	969	692
20	0.01	0.53	5533	4881	0	170
	0.03	**0.54**	**5534**	**4863**	**0**	**187**
	0.05	0.53	5534	4938	0	112
	0.08	0.53	5534	4956	0	94

TABLE 4.2: ARM results, with itemset comparison

Based on the above results we can conclude that we have some interesting outcomes. To begin with we can clearly see that with the *association rule mining* we cannot achieve a high accuracy on the anomaly detection, since the average Accuracy is almost 50% in any different case. Furthermore, we can see that as the minimum support factor becomes larger, in some cases the Accuracy is being decreased and in other cases we can see that it is increased. Thus we can argue that this approach is unsteady and consequently not appropriate for our case. Also the comparison of normal and abnormal frequent itemset brought a slight improvement but not a significant one. Another interesting result that verify that *ARM* is not a good technique is that in almost all experiments we had a low number of True Positives. However since our aim is to perform anomaly detection we should have as large number of True Positives as possible.

To conclude, in ARM we generate frequent itemsets but without keeping respect to the order of the messages. Thus, the same itemsets exist both on normal and abnormal data, and it is impossible to detect efficiently the anomalies using the frequent itemsets with no respect to the sequence order. As an example of this theory, we present a sample of data on which we apply the ARM technique and the resulted frequent itemsets.

Input data	Frequent Itemsets
22 5 5 5 26 26 26 13 11 9 13 11 9 13 11 9 23 23 23 21 21 21 22 5 5 5 26 26 11 9 11 9 11 9 26 23 23 23 21 21 21 20 22 5 5 5 26 26 11 9 11 9 11 9 25 26 18 5 26 6 16 21 4 3 4 23 23 23	22 11 13 26 11 13 5 11 13 9 11 13 18 11 22 25 18 11 26 18 11 18 11 5 18 11 9 26 22 11 5 3 2 22 11 9 26 5 23 22 11 9 26 18 5 25 22 11 9 26

TABLE 4.3: Association rule frequent itemsets

For instance, we can see that the first itemset ("22 11 13") does not appear in

any row of the data in that specific order. So this "rule" can be generated from both normal and abnormal data and so the classification process will not have the desire results. For all the aforementioned reasons, the next step is to experiment with sequential rule mining techniques in order to examine if we can improve our results and eventually to generate patterns that can accurate capture an anomaly behavior in logs.

4.3 Method 2: Sequential rule mining

In the previous section we saw that the application of association rule mining in our case does not bring great results since this technique does not take into consideration the order of the log messages. Thus, a solution to improve our detection model might be to use sequential rule mining(SRM) techniques, which are generating frequent itemsets with respect to the original order of the messages. For that reason we choose to use the PrefixSpan algorithm [39] to generate the sequential frequent itemsets. The different experiments and the parameters that we used in SRM experiments were the same as in ARM experiments. The following results present the findings of the application of SRM in pattern finding for anomaly detection in logs.

Fields	Min_Support	Accuracy	TN	FN	FP	TP
5	0.01	0.49	2869	2750	2664	2301
	0.03	0.56	3542	2697	1992	2353
	0.05	0.54	3190	2483	2343	2568
	0.08	0.51	3176	2822	2357	2228
	0.1	0.44	2444	2782	3089	2269
	0.2	**0.65**	**5532**	**3673**	**1**	**1377**
	0.3	0.54	4691	3982	843	1068
10	**0.01**	**0.71**	**5395**	**2957**	**139**	**2094**
	0.03	0.65	4685	2886	848	2164
	0.05	0.54	3622	2948	1911	2103
	0.08	0.68	5533	3321	1	1730
	0.1	0.66	4956	2936	577	2115
	0.2	0.63	5504	3827	29	1223
15	**0.01**	**0.71**	**5255**	**2764**	**278**	**2287**
	0.03	0.64	4095	2345	1439	2705
	0.05	0.69	5379	3126	154	1925
	0.08	0.63	5531	3934	2	1117
	0.1	0.70	5534	3193	0	1857

TABLE 4.4: SRM results, without itemset comparison

Fields	Min_Support	Accuracy	TN	FN	FP	TP
5	0.01	0.52	5529	5040	5	10
10	**0.01**	**0.70**	**5531**	**3173**	**2**	**1878**
	0.03	0.46	3647	3787	1886	1263
	0.05	0.52	5494	5071	40	19
15	0.01	0.48	1903	1809	3630	3241
	0.03	0.52	5270	4854	263	197
	0.05	**0.70**	**5527**	**3162**	**7**	**1889**
	0.08	0.56	5534	4600	0	450
	0.1	0.57	5495	4465	38	585

TABLE 4.5: SRM results, with itemset comparison

From the aforementioned results of this second approach we can argue that we have a slight improvement on the accuracy of the itemset classification. However the degree of the improvement is not the desired one since best achieved accuracy is around 70%. For that reason it is inevitable to come up with another approach that will drastically increase our classification results.

4.4 Method 3: Sliding window and Sequential rule mining

As we saw in the previous experiments, despite the fact that SRM improves the results in contrast to ARM, we have not yet achieved results that can accurately detect anomalies in logs. One reason can be that in SRM we use a maximum of 15 messages for every block (and specially the first 15 messages). Thus, it is possible to lose really important information and messages that comes after the first 15 messages. So there is a need for an approach where we will use sliding windows in order to include the whole range of messages for each block but also not to include a large number of messages in a row which is computational costly. In order to use sliding windows, first the data have to be split (both normal and abnormal) into bins where each bin will contains messages of a block in a specific time duration (eg.30 seconds).

To begin with, as we can see in the following graph, we split normal and abnormal data into bins of 30 seconds in order to examine how many messages exist in this time window.

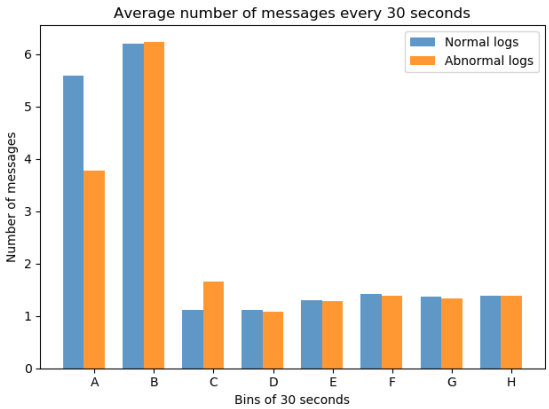

FIGURE 4.2: Messsages per bins of 30 seconds

As we can see, in the first 2 bins, the number of messages of a normal block is almost six in both cases. On the other hand in abnormal blocks we have almost four and six messages respectively. Then in both normal and abnormal we can see that in a time duration of 30 seconds we have just one message in any block. Thus, a sliding window of 5-6 can be the right one. However we have to make experiments in order to define the best size of sliding window. Furthermore, except for this parameter, we also have to estimate the best support factor for the SRM with respect to the sliding window size.

In order to determine the right pair of sliding window size and support factor, to maximize our accuracy score, we implement a Bayesian optimization technique[47]. To formulate this optimization problem, we need to define 4 major parts:

1. **Objective function:** It is the function that we want to minimize its loss or, in our case, to maximize the accuracy score. The function that we test is the sequential rule mining algorithm in combination with the sliding window method that we have described earlier.

2. **Domain space**: The domain space is the range of input values that we want to evaluate. In our case, we have two parameters that we want to test, the support factor and the size of sliding window. Each of these parameters has its own range. For the support factor we used a uniform distribution [4] in range $0.01 - 0.2$. For the sliding window size we used again a uniform distribution with range $1 - 10$. The reason why we did not test a higher value of sliding window size was explained in the section with the memory issues of ARM. These memory issues hold in SRM algorithms as well.

3. **Optimization algorithm**: It is the method used to create the probability model and choose the next values from the space to evaluate. In our optimization experiment we used the Tree-structured Parzen Estimator(TPE) model [5]. The TPE build a model by applying the Bayes rule. It uses

$$p(y|x) = \frac{p(x|y) * p(y)}{p(x)} \tag{4.1}$$

   where $p(x|y)$ is the probability of the hyper-parameters given the score(accuracy) of the objective function.

4. **Number of evaluations**: In order to have accurately estimated results we set the optimization algorithm to evaluate our method at least 100 times.

The result of the implementation of the Bayesian optimization are presented in the following heat map.

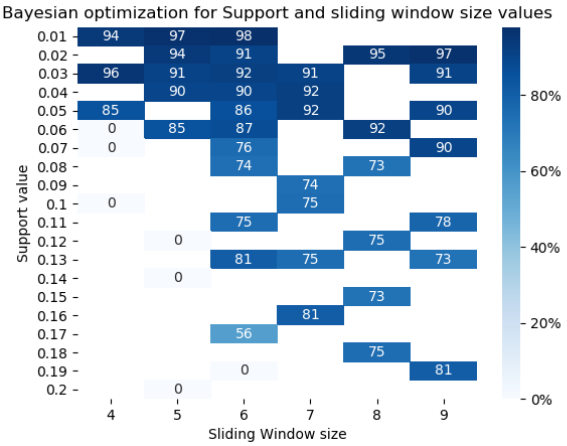

FIGURE 4.3: Bayesian optimization for sliding window size and support factor

As we can see, the best sliding window size is 6 and the best support factor is 1% , since we have achieved 98% accuracy in anomaly detection. To understand better why we used the sliding window, we present an example of a data row and the transformed data after the sliding window application.

Input data	Sliding windowed data
5 5 22 5 11 9 11 9 11 9 26 26 26 23 23 23 21 21	5 5 22 5 11 9 5 22 5 11 9 11 22 5 11 9 11 9 5 11 9 11 9 11 11 9 11 9 11 9 9 11 9 11 9 26 11 9 11 9 26 26 9 11 9 26 26 26 11 9 26 26 26 23 9 26 26 26 23 23 26 26 26 23 23 23 26 26 23 23 23 21 26 23 23 23 21 21

TABLE 4.6: Sliding Window data

In that part, since we have a really good accuracy, it is crucial to explain our approach clearly and present the final results. Our implementation has the following steps:

- We split the dataset into train and test set, both including normal and abnormal blocks.
- For the training set:
  - Separately for normal and abnormal blocks, we apply the sliding windows of length 6 and step 1. Thus we end up with a new dataset that every row has 6 number of messages as a sequence.

  - In these new datasets (normal, abnormal), we apply the SRM algorithms with min support factor 1%, resulting the most frequent itemsets in both cases. Then we conduct the comparison similar to the previous experiments, resulting to unique abnormal and normal frequent itemsets.

- On the test set, we iterate over the rows and if a sequence of frequent itemset exist in the test sequence, the we label it as abnormal or normal respectively.

However with this approach we achieve 80% of accuracy. Another feature that we don't have taken into consideration is the length of a sequence in a normal and abnormal case. For instance, it is important to check the minimum sequence of our data.

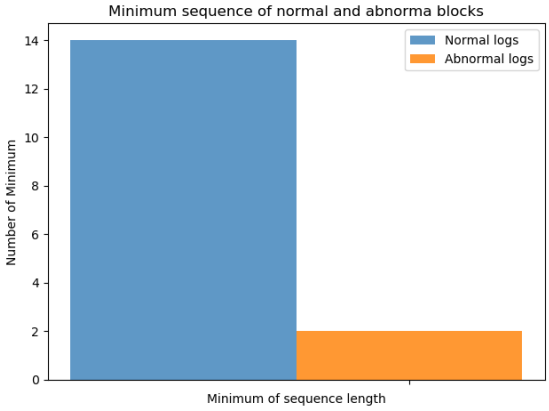

FIGURE 4.4: Minimum number of message in normal and abnormal blocks

We can see that there is no normal sequence that has less than 14 messages. Thus, another restriction will be that sequences of less than 4 messages will be labeled as abnormal. The final results of our approach are the following.

Sliding window size	Min_Support	Accuracy	TN	FN	FP	TP
6	0.01	98%	5298	10	235	5040

TABLE 4.7: SRM results, with sliding window

From all the above experiments, we can conclude that the frequent itemsets can effectively capture the abnormality from the raw data that we have. Thus, we will use sequential rule mining technique to generate rules from the raw HDFS data to be used as patterns in the complex event processing engine for the early alerts for abnormal behavior in the logs.

Chapter 5

Rule transformation to complex event processing

After experimenting with the effectiveness of anomaly detection through itemsets, we can build our main framework for automating the pattern creation in complex event processing engine, by transforming the previous rules into CEP rules, to generate alerts on abnormal log data.

5.1 Rules generator

In the first step we use the training raw dataset and we apply our sequential rule mining technique as presented in Chapter 4 to generate the most frequent itemsets of the abnormal data. Specifically we conduct the following steps to generate the final rules:

- Separately for normal and abnormal data, we apply a sliding windows of length 6 and step 1. Thus, we end up with a new dataset that every row has 6 number of messages as a sequence.
- In these new datasets (normal, abnormal), we apply the SRM algorithm with min support factor 1%, resulting the most frequent itemsets in both cases. Then we conduct the comparison similar to the previous experiments, resulting to unique abnormal and normal frequent itemsets.
- Then we keep only the final abnormal frequent itemsets to be used as rules.

In order to apply the SRM algorithm, we have to pre-process the raw data as we mentioned in previous chapter to make grouped data for having all the messages of a block in the same row. Then these itemsets serve the purpose of rules from which we can generate the patterns in the complex event processing engine.

5.2 Automate complex event processing engine

After the rules generation, we have to create the process of automating the complex event processing engine. Particularly we have created a Flink Job that read the above rules and generates automatically patterns, using the most frequent itemsets, in order to detect anomalies in streaming log data.

To begin with, we start our Flink job by defining some configurations, that we will analyze in the next section, and we create a Kafka consumer to read the streaming data from the Kafka topic. Next we read the previously generated rules from

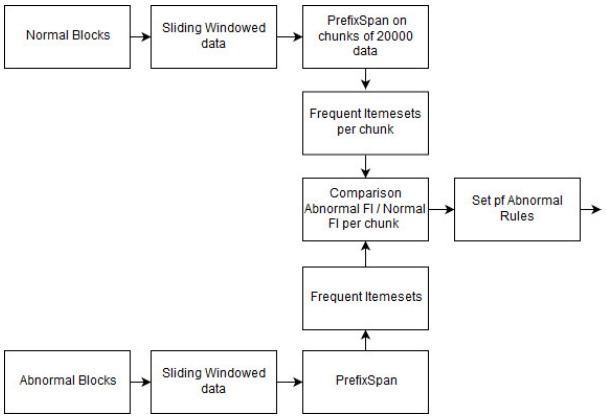

FIGURE 5.1: Rules generator Pipeline

the Persistent Volume space [30] that have been stored and for every rule it creates a pattern using the Pattern API of Flink. A pattern definition in Flink Cep is represented by the start of the pattern and then by a sequence of "next" occurrences that have to be presented in the same exact order in the streaming data. Since we wanted to have patterns with strict restrictions to sequence, we used the command 'next' to construct the sequence of the pattern. In the example of a pattern in Algorithm 1, we are trying to detect the sequence of messages 13 9 11. Suppose the streaming data of a specific block arrive to the system with the messages 5 6 8 13 9 11 in that exact order. Then the aforementioned pattern will make a match with the sequential data of the blocks and an alert of abnormality will be raised.

Algorithm 1: Scala code of Pattern definition

```
var pattern= Pattern.begin[Logs]("start").where(message ==13 );
.next("middle").where(message == 9);
.next("end").where(message == 11);
```

In parallel, we process the streaming data by mapping them to our pre-processor function to transform the raw message into an object of Block id, message and timestamp. This timestamp will be used to chronologically order properly the incoming streams based on their Block id. Then, for every pattern that we have created, we use the CEP API of Flink by passing the patterns and the processed streaming data. The result of this API is the matches between the patterns and the streaming log data. These results are transformed into alert which indicates in which blocks a match have been occurred and thus it raises an alarm of abnormality on that block. These alerts then are published to another Kafka topic. In this part we have to mention that for every rule, the Flink job is executed in parallel checking all the streaming data for matching in all rules. An example of an execution of the job using 5 rules can be depicted in Figure 5.2.

We can see in the above image that after the mapping and watermarking of the streaming input, the result is passing into all 5 rules for finding a matching pattern. Finally, the pseudo code of our implementation is presented below as well as an example of the execution of our system by presenting the streaming log data that enter in Flink in Figure 5.3 and the alerted result in Figure 5.4 that we have as an outcome.

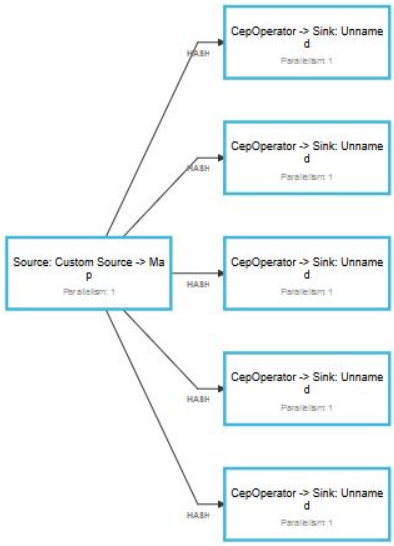

FIGURE 5.2: Flink job execution graph

Algorithm 2: Pseudo code of Automated Complex Event Processing Job

```
Input: Rules
Input: Streaming-log-messages
Output: alert(Anomaly Blocks)
initialization;
logs = Streaming-log-messages.map => Preprocessing=>Logs(Block,
 message,timestamp);
kafka-consumer= Watermarks(timestamp);
logs = logs.keyBy(Block);
for rule in Rules do
    pattern = rule;
    patternStream = Cep(logs,pattern);
    FinalPatterns= List(patternStream);
end
for f in FinalPatterns do
    alert = FindMatches(f);
    alert.sink(Kafka-producer);
end
```

5.3 Flink job configuration

In this section we will analyze all the configurations and decisions chosen for Flink in order to have a fully working framework. To begin with, the entire Flink job, which is our program, is written in Scala [45], a Functional programming language, since Flink works more efficiently using Scala instead of Java. The next parameter that we have to adjust was the level of parallelism. Actually we choose to use this level equal to 1 since the Kafka topic that we used to send the streaming data contains just one partition and one replication. A common approach in the integration of Kafka and Flink is to use a level of parallelism in Flink equal to the number of partitions

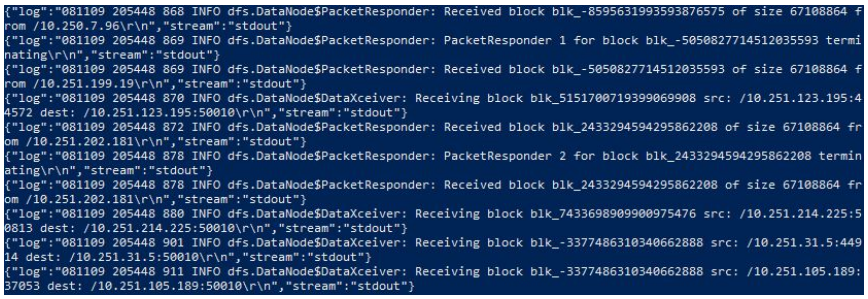

FIGURE 5.3: Input log data

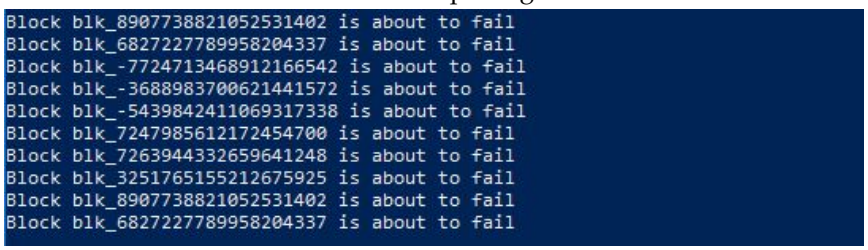

FIGURE 5.4: Final alerts of Flink CEP job

of the topic where it reads the data from. As far as the reading of rules from Flink concerns, we choose to use the default IO package of Scala. The reason behind this decision was that in the beginning we tried to read the rules and save them to a list using Flink's Dataset API. However there is a limitation in Flink which does not allow a job to use both Dataset and Datasteam API simultaneously. Thus, since Datastream API was definitely needed, we manage to read the rules from the file using the default Scala package.

After the determination of the initial parameters, the most important configuration of our Job have to be tuned, which is the time characteristic of processing the streaming data. To begin with, Flink support three different notions of time [15], Processing time, Event time and Ingestion time. In this point we will describe the characteristic of each of these time in order to understand which notion suits best our problem.

- Processing Time: It refers to the system time of the machine that conduct an operation. When a Job runs of processing time, then all its operations will use the system clock of the machine
- Event Time: It refers to the time that a specific event occurred on its producing device. Usually this time is included within the records before even processed by Flink and it has to be extracted from the record. Thus, the process of time depends on the data alone.
- Ingestion Time: It refers to the time an event enters the Flink. Actually this notion is something between Processing time and Event time.

In Figure 5.5 the difference between these three notions are clearly depicted.

Since we have a fully understanding of the different notions of time now, we can say that the notion that we need in our case is the Event time. The first reason behind this decision is that the log data that we use are produced from another system, the Hadoop system. Thus, in order to detect anomalies we have to analyze the data

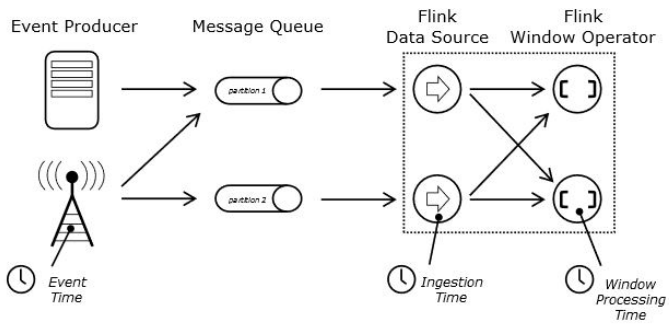

FIGURE 5.5: Different notions of time in Flink streaming processing

using the time they were produced and not the time that they entered to another system. This is because the right timed order is really crucial in our approach when we need real time alerts on anomaly data. Another advantage for our case is that we have the timestamp of each log within the message so we can easily extract it. Finally, since our whole system is composed off several different tools, the transfer of data from one to another will induce a kind of delay and thus by using processing time in Flink, the results will be inaccurate.

The final crucial configuration that we have to adjust is related to the previous choice of the Event time. Specifically, after the decision to use Event time when processing the streaming data, we have to configure a program to specify how to generate Event Time Watermarks, which is the mechanism that signals progress in event time. To use the Event time, Flink needs to know the timestamp that needs to assign to a specific event. Thus, Flink needs to assign Watermark to measure the event time. To generate these Watermarks, Flink has some different methods [17]. The simplest way is to use the default *.assigntimestamp* operation when processing the streaming data. However, this action requires some specific characteristics on the streaming data. The most important one is that the data, and consequently the timestamps must be in a monotony timed order. However, in our case, by using this default operation we have a monotony violation. This comes from the fact that we have an integration between Kafka and Flink where Kafka sends the streaming data to Flink. But, despite the fact that initially Kafka reads the data in the right order from FluentD, it does not send them in the right order to Flink, due to some inner operations. So the solution is to use Periodic or Punctuated Watermarks. The difference between these two is that in Periodic Watermarks, the system assigns timestamps and generates watermarks periodically. On the other hand, Punctuated generate watermarks whenever a certain event indicates that a new watermark might be generated. In our case both options are working and the produce the same results. However, since Punctuated generate a watermark on every single event and each watermark causes some computation downstream, we have a degrade performance of our program. Thus, we decide to use the Periodic watermark generation to assign timestamp to the streaming data.

Chapter 6

Experiments and Results

In this Chapter, we present the experiments to examine the effectiveness of the anomaly detection framework that we have proposed in the previous Chapter. Firstly, we have to indicate what dataset will be used in these experiments. As presented in Chapter 3, we will use the second generated dataset consisting of 103.620 Normal blocks and 2.792 Abnormal Blocks. Furthermore, in contrast to the experiments conducted in Chapter 4, we will evaluate the results using mainly the Recall ,the Precision and the F1 score. Recall is the most important factor since it can measure how many of the existing anomalies we actually found. In anomaly detection cases it is important to capture all abnormalities without considering a lot about miss-capturing normal data as abnormal. The following sections present the main experiments and results of this thesis.

6.1 Baseline Experiment

As we saw in Section 4.4, the number of messages of a Block plays an important role on the classification of that Block as normal or abnormal. We saw that a great amount of abnormal blocks hold the property that they have only two or three messages. However in the rule generator part in Section 5.1, it is not possible to create a rule that can capture this property by using the SRM algorithm. Thus, in this first experiment we decided to exclude from the testing dataset all the abnormal blocks that holds this property. Also we decided not to exclude them from the training set, from which the rules are generating, since they do not affect the outcome of the rule generation. With this configuration we conduct three experiments, each with a different Train/Test split and with a 5-cross validation approach [27]. The results of these experiments are presented in Table 6.1 and Table 6.2 .

Experiment	Training set	Test set	Log messages	Rules	TP	FP	TN	FN
1	60%	40%	1.013.248	65	598	666	40782	16
2	80%	20%	506.074	64	301	559	20165	22
3	40%	60%	1.520.838	73	937	6624	55548	4

TABLE 6.1: Results from Baseline experiment

Experiment	Recall	Precision	Accuracy	F1 score
1	97%	47%	98%	63%
2	93%	35%	97%	51%
3	99%	12%	89%	21%

TABLE 6.2: Metrics of Baseline experiment

From the above results there are lots of interesting outcomes that are worthy analyzing. To begin with the results in Table 6.1, we can see that the number of rules in all cases are almost the same. Another important point is the number of False Negatives, which we can see they are just a few, which means that we have managed to accurately capture the majority of abnormal blocks. Furthermore, despite the fact that we have larger number of False Positives than True Positives, the results are really promising. This stem from the fact that the False Positives are the normal blocks that have been alerted as abnormal. However the total number of normal blocks are way more larger than the number of False Positive, in a scale that it is almost insignificant.

As far as the results in Table 6.2 is concerned, we can clearly see that our system achieved its main purpose. The Recall range is almost perfect which means that we managed to accurately detect all the abnormalities. However, the values in Precision and F1 score are significantly lower. This is totally expected since we are facing the problem of the imbalanced data. We have a far greater number of normal data than abnormal, something that affects the values of these metrics. Also, since the most appropriate dataset split is the 60-40, we can argue that the most representative results of our tool include a Recall of 97%. Finally, one can argue that we have a strange results in Table 6.2, since we have better Recall value with less training data. However this is quite ordinary in our framework, since it can accurately capture abnormality in produced patterns even with few abnormal training data. Thus, in a 40-60 split, we have more abnormal data to test and consequently a larger number of True Positives. This characteristic of our framework will be verified in Section 6.3

6.2 Effect of number of messages per block

In the second experiment we decided to try to include the property of the number of messages per block in the testing phase. In order to do that, we make some alteration in the implementation of the Flink job as presented in Chapter 5. Specifically the alteration was to include another function that will count the number of messages per block in a fixed timed window. To do that we had also to handle the streaming data a bit differently. Thus, we end up with two different mapping variables of these data, the initial one as described in Chapter 5 and the new one based on the new timed window. Then we create a pattern that will match every block that has less than four messages in a specific time range. By this implementation we were able to actually capture all these abnormal blocks that hold this property of the number of messages and alert them. Thus, the following Table present actually the final results of our framework. Furthermore, in this Table we have included the number of abnormal blocks that were tested in order to depict how many abnormal blocks we have with this specific property in contrast to the previous experiments. For instance, in first experiment in Table 6.1, the total number of abnormal blocks were 614(598 + 16) while in this experiment we have 1117 abnormal blocks. Thus, we can see that we had eventually 503 abnormal blocks with less than four messages, which is a significant number.

In this experiment we can see that our final results have been improved a bit by capturing also the abnormal blocks with the small number of messages. Firstly, we can see that we reach a Recall rate in 99% in the 60-40 split which depict that our systems works almost perfectly. Also we have a quite big improvement on both Precision and F1 score, in a range between 15-20 %. Thus, we can conclude that our work served its purpose and answered the research question about making a proper

Experiment	Testing abnormal blocks	Recall	Precision	Accuracy	F1 score
1	1117	99%	62%	98%	76%
2	559	96%	48%	97%	64%
3	1676	99%	20%	89%	33%

TABLE 6.3: Metrics of the effect of number of messages per block

log analysis and anomaly detection tool by integrating machine learning and big data technologies.

6.3 Necessary number of abnormal data

Another interesting research question that we have to address, is the examination of the minimum number of abnormal blocks that we need to have in prior, in training phase, to generate accurately rules that can efficiently capture the abnormality in the blocks of testing phase. Generally, this concept is called sample efficiency, and measures the number of learning samples an algorithm needs in order to reach a certain accuracy score. It is true that with a large number of samples, every algorithm can achieve a high accuracy level. However in real life, having a massive number of training samples is, except for almost imaginary, extreme costly and time-consuming (to train the model). Thus, we have to test our implementation in terms of sample efficiency, to check how many abnormal samples we must have to capture accurately every abnormal behavior.

For that reason, we conduct experiments using a constant number of normal blocks and a variable number of abnormal blocks in the rule generation phase. Particularly, we used 60% of normal blocks in training phase and a range of 5% till 80% of abnormal blocks. The goal was to examine if we have accurate alerts of abnormal blocks using the rules from the training phase but also test the number of False positive blocks that we capture. In Figure 6.1 we can see the results of this experiment, using the different size of abnormal blocks.

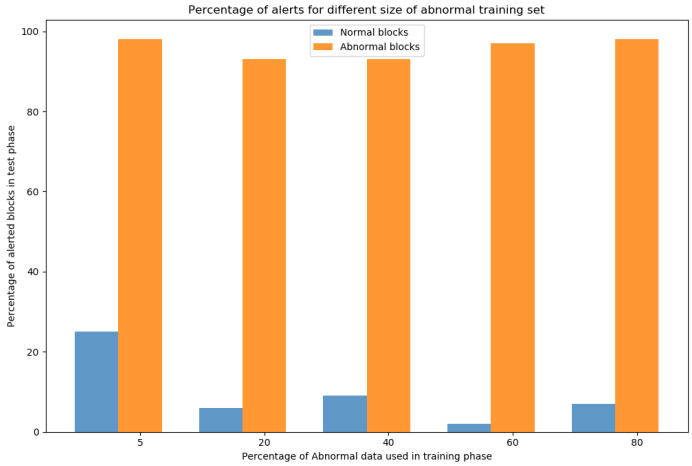

FIGURE 6.1: Experiments with different size of abnormal dataset in training phase

From Figure 6.1 we can see that despite the size of training set of abnormal blocks, 5%, 20%, 40%, 60% and 80%, we manage to have alerts on around 96% - 98% of all abnormal blocks in the testing phase. Thus, with this result we can propose that using even a small number of abnormal block to generate rules, we can efficiently capture all the necessary abnormality for finding future abnormal data. Another useful insight from the above experiment is the number of incorrect alerts of normal blocks. Specifically we can see that using 5% of abnormal blocks, we got alerts on 25% of all normal blocks in test phase. As the size of abnormal blocks in training is increasing, the percentage of normal alerted blocks is decreasing, which means less false positive results. However, since a common case in real life is to have lots of normal data but really few abnormal ones, we can conclude that even with small number of abnormal data, we can efficiently use our framework to detect future abnormality. However, the size of the available abnormal data will affect the false positive rate, which is the faulty alerted blocks in our case, but not in an extent to be restrictive for the use of our framework

6.4 Experiment to improve Precision

In the previous section we saw that our tool has a great performance in detecting anomalies, which is proven by the achieved Recall. However there is a clear issue with the levels of Precision and F1 score which are quite low. The reason for these low values is that, despite the detection of almost all abnormal blocks, we categorize also a significant number of normal blocks as abnormal. These results are associated with the rules generation. More specifically, in the rule generation phase, we end up with some rules that represent the normal behavior. The reason is that we have a really large number of normal data and probably the support factor that we used in generating normal rules and comparing with the abnormal is not the most appropriate. Thus despite the fact the we have ended up with a support value of 0.01 based on the Bayesian optimization, we will conduct experiments of using a lower support value during the normal rule generation. The results of these experiments are presented in Tables 6.4 and 6.5.

Support value	TP	FP	TN	FN
0.009	1057	546	40902	60
0.007	1058	305	41143	59
0.005	1084	25	41423	33
0.001	1082	5	41443	35

TABLE 6.4: Results using different support factor on normal rule generation

Support value	Recall	Precision	F1 score
0.009	94.6%	66%	77.7%
0.007	94.7%	77.6%	85.3%
0.005	97%	97.7%	97.3%
0.001	96.8%	99.5%	98.1%

TABLE 6.5: Metrics using different support factor on normal rule generation

As we can see, by lowering the support value in normal rule generation, we have less False Positive results and consequently we have a great improvement on the Precision and F1 score value. Furthermore, we can also argue that there is a small degradation in the performance of the tool in terms of Recall. However this degradation is not that significant, thus we can conclude that using a support factor of 0.01 in abnormal rules generation and a value of 0.001 in the normal rules, we have the best results for our tool. Although the good performance in these experiments, there is a possibility of overfitting the system since with such a small support value, we actually capture all the possible normal patterns, which however need further investigation to determine it's impact in the generalization of the algorithm.

6.5 Algorithm computational complexity analysis

In this last experiment we will try to measure the computational complexity of our technique and to examine which factors have the greatest impact. Based on our implementation, each message (or sequence of messages) are tested with every rule in order to find a match. Thus, theoretically the complexity of our algorithm is $\mathcal{O}(message * rules)$. However, we want to examine in what extend each of these two factors affect the performance of our system.

For that reason we will measure the runtime of the algorithm by tuning these two factors. Specifically we will measure the time when the first data enters the Flink job until the last data passing through the Flink operation for the pattern matching. The two factors that we want to investigate is the number of rules that we will test for matching and the size of the testing data that we have. For that reason we use different number of rules from 5 to 50. For the size of the testing data we use different split percentage for train and test. The result of this experiment is presented in Figure 6.2 as well as the number of log messages in each testing percentage in Table 6.6.

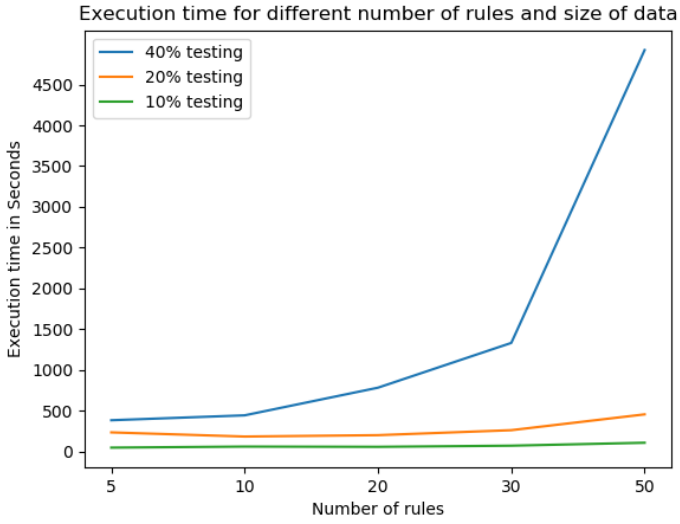

FIGURE 6.2: Runtime experiments

To begin with interpreting the above results, we have to present the number of log data in each of the three cases.

Testing Percentage	Number of log messages
40%	1.024.294
20%	576.853
10%	243.075

TABLE 6.6: Number of log messages per test set

In the first case, when using the 40% of the dataset for testing, we can see that, especially after using 30 rules, the runtime is really high. More importantly, in that case, after using more than 40 rules, the Flink job was crashing due to memory issues. On the other hand, using a smaller number of log data, we can see that the runtime is quite low, independently of the number of rules that we examine. Thus, we can conclude that our framework is being influenced, in term of runtime, mainly from the size of the input data and not from the number of rules. In real life, probably we will not have that high number of testing data in a short period of time, but also not limited resources, so our anomaly detection framework is working quite efficiently also it terms of runtime execution.

6.6 Comparison with similar works

From all the above experiments we can see that our work has really promising results in term of anomaly detection in distributed system's log. However, it is crucial to evaluate our results by comparing with the results of other works that were focused on the same data. To begin with, a master thesis work exists [19], where they conduct experiments with unsupervised techniques as well as a deep learning approach, such as LSTM, to detect anomalies in same HDFS logs. The best results they achieve was a Recall and Precision score of 88% and 89% respectively. Another work that used the same data comes from [12], in which they used again deep learning techniques. Their Precision and Recall score was 95% and 96% respectively. Finally, Xu et al. [51], who are the people that generated HDFS dataset, achieve a Recall rate of 100% and a Precision of around 87%, using data mining and statistical learning methods.

Thus, we can conclude that our system outperforms the majority of the already existing method of detecting anomalies on the same dataset, opening a totally new field of using rule mine and complex event processing for anomaly detection in logs. Furthermore, we can argue that our approach is also simpler that the aforementioned, such as Deep learning techniques. This is a great advantage since it needs less time to train the model that conduct the anomaly detection.

Chapter 7

Scaling out anomaly detection in the cloud

7.1 Pipeline

One great challenge of this thesis is to create a framework that will be entirely hosted into the cloud. Thus, the decision was to build the anomaly detection framework and all the necessary tools inside a cluster in the cloud. After the examination of all the available tools that exist and can support our work, we decide to build our tool using 4 well-known tools, Kubernetes, FluentD, Kafka and Flink.

Beginning with Kubernetes, the idea was to create a Kubernetes cluster and host all the remaining tools there as pods. Also since we are in the era of Micro-services and lots of applications are split in smaller services, one possibility is to host all the services inside the cluster. As a results, it will be easier to gather the logs of the different services and conduct an anomaly detection in logs using our tool. FluentD is a really important tool since it can collect all the logs of the different application to a centralize space. In our case however, since we will have one application that generate logs, FluentD will be configured to read data only from that source. Also another advantage of FluentD is that it has a well integration with Kafka, which is our next tool that we used.

Using Kafka, we have a great tool for passing all the necessary data and spreading them to our desire directions. Thus, during Kafka installation, we have created two Kafka topics, one to publish the log data and one to publish the anomaly detection results. Last but not least, we choose to use Flink as a way to handling the streaming log data. Also, since one research question of this thesis is to automate the complex event processing engine and Flink contains a really powerful CEP engine, it was inevitable that we will use this tool. Combining Flink's power in streaming processing and the complex event processing engine, we can unlock a whole new field with lots of possibilities in the anomaly detection of any system's logs.

As we said before, one goal is to package all the tools into one framework inside a cluster. In Figure7.1 we can see how the architecture of the tool looks like. Also it is crucial to explain the data flow in the system. From Figure 7.1 we can understand how the data flow, but in the next paragraph we provide a brief explanation of that flow and the decision behind that.

7.1.1 Data flow

As presented in Figure 7.1, the entire framework is hosted in a Kubernetes cluster. The application that is responsive for the rules generation is deployed as a pod and its results are saved to a file into the Persistent Volume [30] of the cluster. The script that plays the role of log generator, which is also deployed as a pod in the cluster,

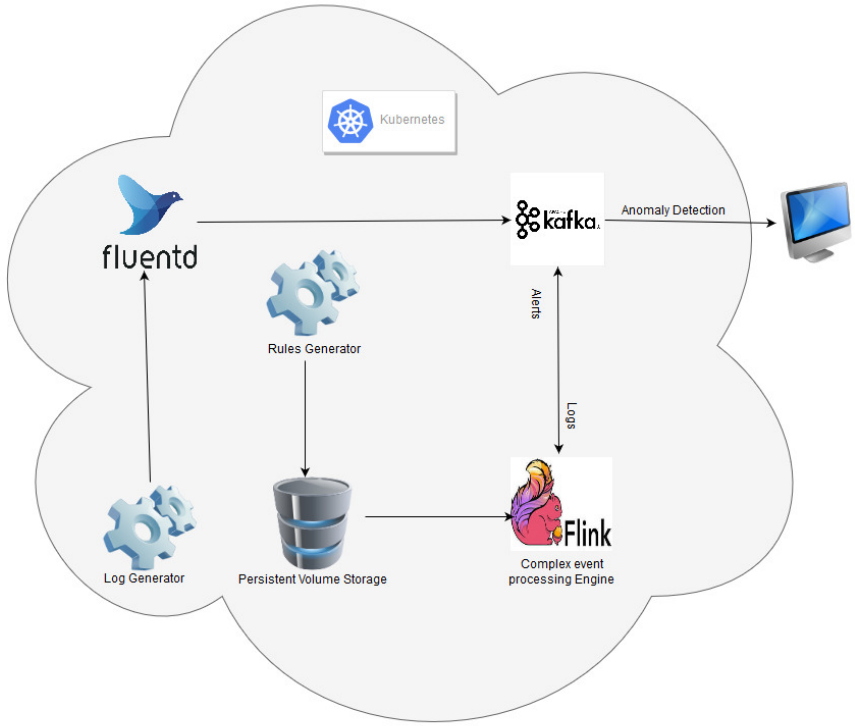

FIGURE 7.1: Framework Structure

read the entire raw file of HDFS logs and it prints the log messages line by line as real logs. Fluentd, which is our log collector in the cluster, read the logs of the aforementioned pod and publish each line to a Kafka topic. Then Flink is connected with Kafka in order to read the logs from the Kafka topic as streaming data. In parallel, Flink reads the file with the rules that was saved in the storage of the cluster in order to create the necessary patterns for the complex event processing engine. Finally, the results, which in our case are the blocks that have been predicted as abnormal, are published back to another Kafka topic.

7.2 Large scale configuration

In this section we will describe the main technologies that we use in the implementation of the thesis project, including their purpose and why we chosen them instead of other similar frameworks. Furthermore, we will describe the configuration parameters of each tool in order to have a fully working anomaly detection framework.

7.2.1 Scaling system

In order to scale out our system we have decided to host every different service and tool into a cluster. For that reason we choose Kubernetes to be the host system for our anomaly detection tool.

Definition

Kubernetes is a portable, extensible open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available. Google open-sourced the Kubernetes project in 2014. Kubernetes builds upon a decade and a half of experience that Google has with running production workloads at scale, combined with best-of-breed ideas and practices from the community [22].

FIGURE 7.2: Kubernetes

Kubernetes will serve us as the main tool. All other tools that will be used, will be deployed inside the Kubernetes cluster. In such way it is easier to monitor all the tools that we will use and also we can be sure that we will achieve the concept of Micro-service. Also, it has to be mentioned that the Kubernetes cluster was hosted on Microsoft Azure cloud. The reason for using the Azure Kubernetes Services was that it is a really easy and efficiently way to create a Kubernetes cluster in the cloud, without considering updating and maintenance of all cluster components such as the nodes.

Configuration

As we have mentioned before, our entire framework will be hosted in a Kubernetes cluster running on Azure. The most important configuration of the cluster is that we have to define a Persistent Storage [30] space for our framework. Thus we need to define a *Persistent Volume* and a *Persistent volume claim*. The value of the former is to provide to the cluster a solid storage space on which our tool can save different necessary files, such as the file that contains the rules, as we will describe later. The latter gives access to a user to use this Persistent Volume storage. Also since we are working with streaming and maybe infinite data, it is really important to have a solid storage to store these data instead to spread them to all Kubernetes cluster with the risk of facing memory and storage issues.

7.2.2 Containers

In order to deploy application to Kubernetes cluster, we have to create containers of each different tool that we want to deploy. Containers are really famous nowadays because you can have an application and all necessary dependencies and libraries together into one image. For that reason, we decide to containerize all our tools with the help of Docker. Docker [11] is a tool that was designed to deploy and run application using containers. Using containers, as we mentioned above, you can combine your application and the necessary libraries and dependencies into one package in

order to run it in almost any environment. Thus, with Docker you can dockerize every kind of application into image and deploy it to Kubernetes cluster, which is our goal.

FIGURE 7.3: Docker

Docker contains some really useful concepts that have to be mentioned here.

- Docker Images: Docker is similar to a virtual machine image, however the main difference is that images in docker share the Linux kernel with the host machine. Thus Docker has better performance and it is more lightweight than a general virtual machine.
- Dockerfile: Dockerfiles are scripts that define how to build the docker image, just like Makefiles. In a Dockerfile you can specify the necessary dependencies and library that your image needs and environmental variables in order to install your application properly.

In this work, for the Python applications that we created, we had to build our own images and thus we created our specific Dockerfiles. For the tools that we used, such as Flink and Kafka, we used already build images provided by different contributors.

7.2.3 Log collector

Since all services will be hosted into a Kubernetes cluster, it is necessary to find an appropriate tool that can collect all the important logs that we want. Thus we choose to use FluentD as log collector.

Definition

Fluentd is an open source data collector, which lets you unify the data collection and consumption for a better use and understanding of data. Fluentd tries to structure data in JSON format, as much as possible. This allows Fluentd to unify all facets of processing log data: collecting, filtering, buffering and outputting logs across multiple sources and destinations [43].

Thus we used Fluentd in order to collect all logs from the Kubernetes cluster and ship them to our storage tool (Kafka). The reason for choosing Fluentd instead of other log shippers like Logstash, was that it is really fast and it needs less resources in contrast to other tools. Also it pre-process the logs in order to transform them into JSON format which help us to analyze them easier. Furthermore it has lots of input and output plugins so it can be connected easy with other tools. Finally it can deployed as Daemonset(will be analyzed later) on Kubernetes to collect the logs of all application and services.

FIGURE 7.4: FluentD

Configuration

The reason we choose FluentD to be part of the framework is that it can provide us with the important mechanism of collecting all the logs from the systems and transmit them to the desire sources. To deploy FluentD to the Kubernetes cluster, we have to take into consideration different configuration parameters and some different deployments.

To begin with, the first action is to deploy a Service Account [31] for the FluentD. The purpose of this is to to give access to all FluentD pods to the Kubernetes API. Next we have to define a *ClusterRole* for the tool to have grant access to all resources (pods and nodes) and a *ClusterRoleBinding* [28] which will bind the *ClusterRole* of Fluentd as well as the Service Account that we defined in the previous step. Following that, the next step is to deploy the FluentD to Kubernetes as a *DaemonSet*[29]. The purpose of deploying the tool as a *Daemonset* is to ensure that when a new node is added to the cluster, then automatically a new FluentD pod will be also created in order to collect all the logs for the new added Node. Finally the last action is to create a configuration file for the FluentD with all the necessary configurations. For instance we have to restrict from which pod the tool will read the logs. Since we only have one specific pod that will produce logs in our case, it is crucial to define the parameter for FluentD to read data only from that pod. In any other case, since we have deployed the tool as a *Daemonset*, we will have logs from all the pods in the cluster, something that in our case will make our framework a way more complex. Another important parameter is to define the Kafka Broker on which it will publish the logs that it collects and the memory usage in order to collect and publish the logs to the desired Kafka topic.

7.2.4 Message sink

Then it is crucial to find a tool which will collect the logs, as well as to send them to other tools. Thus we need a tool to play the role of mediator on a data flow. For that reason we decided to use the well known tool, called Kafka, which is also easily integrated with other tools such as FluentD.

Definition

Apache Kafka is an open source project for handling real-time data streams. Initially is was developed by LinkedIn who then open sourced it as an Apache project in 2011. It is a distributed messaging system providing fast, highly scalable and redundant messaging through a pub-sub model. Kafka's distributed design gives it

several advantages. First, Kafka allows a large number of permanent or ad-hoc consumers. Second, Kafka is highly available and resilient to node failures supporting automatic recovery. In real world data systems, these characteristics make Kafka an ideal fit for communication and integration between components of large scale data systems [2].

The basic architecture of Kafka consists of topics, producers, consumers and brokers. All Kafka messages are organized into topics. If you wish to send a message you send it to a specific topic and if you wish to read a message you read it from a specific topic. A consumer pulls messages off of a Kafka topic while producers push messages into a Kafka topic. Lastly, Kafka, as a distributed system, runs in a cluster. Each node in the cluster is called a Kafka broker.

FIGURE 7.5: Apache Kafka

In our project, Kafka serves the role of a storage tool in which we store all the streams of logs that we want to analyze, publishing them to a specific topic, before being processed by other tools like Flink.

Configuration

Kafka is one of the most important tools in our framework since it is actually the channel where the data actual flows. Thus, it is important to have an actual working and stable Kafka cluster inside Kubernetes cluster. So except for Kafka, we have to deploy Zookeeper as well, to maintain configuration information, naming, providing distributed synchronization and group services to coordinate Kafka nodes.

Zookeeper

Zookeeper [55] is a top-level software developed by Apache that acts as a centralized service and is used to maintain naming and configuration data and to provide flexible and robust synchronization within distributed systems. Zookeeper keeps track on status of the Kafka cluster nodes and it also keeps track of Kafka topics and partitions.

FIGURE 7.6: Zookeeper

Configuration

Initial we have to create a Service to resolve the domain name to an internal Cluster IP. This IP uses Kubernetes internal proxy to load balance calls to any Pods found from the configured selector. Also we create a Headless service to return all the IPs of Kafka pods based on the above Service. The last step is to deploy Zookeeper as a *Statefulset*. The value of *Statefulsets* on Kubernetes is that it offer stable and unique network identifiers, persistent storage, ordered deployments, scaling, deletion, termination and automated rolling updates. Alongside with the *Statefulset*, we deployed also a *PodDisruptionBudget* to keep the Zookeeper service stable during Kubernetes administrative events such as draining a node or updating Pods. The number of zookeeper pods that we used is three in order to have a stable deployment without memory issues for our case.

Kafka

The Deployment of Kafka contains the same steps as Zookeeper's. Thus, in the end we have a cluster of three Kafka brokers. However, in the configuration file of Kafka we set also the two topics that we will use. In first topic we will publish the log data that we will send then to Flink and in the other topic we will publish the alerts (abnormal blocks) as being resulted from the Flink's complex event processing Job.

7.2.5 Streaming processing

The final tool that is needed is a streaming processing tool to handle the streaming logs. Also we want a tool that contains a complex event processing engine since this is one of our main functions that our anomaly detection tool need. Thus, we use Flink, which is the best tool for processing streaming data and also contains a powerful complex event processing API.

Definition

Apache Flink [14],[7] is an open source framework for processing streaming data. Its streaming engine, which provide the user with low-latency and high-throughput is written in both Java and Scala. It is the number one tool nowadays for streaming processing, taking the place of Spark in the streaming field. Flink contains a variate

of API with Datastream, Dataset and Complex event processing APIs being among the most important ones. Except for the APIs, a Flink cluster consist of three types of processes. The first one is the client which is responsible transform the source code of a program into a dataflow graph. Then, a client submit this graph to the second process, which is the Job manager. The role of Job manager is to coordinates the distributed execution of the graph and defines the Task managers, which will actually process the data. Task managers are the third crucial component in a Flink cluster. They are responsible for the actual processing and execution of the necessary operation of the program and they report their status and results back to the Job manager. Finally, it has to be mentioned that the aforementioned dataflow graph is a directed acyclic graph (DAG) which consist of data streams and stateful operators.

FIGURE 7.7: Flink

Configuration

Flink is our main stream processing framework and the tool that we want to automate. For the deployment of Flink into the Kubernetes cluster we used the general deployment as mention in the tutorial [16]. Specifically we deployed one Job Manager and two Task managers, each with 1Gb of Java Heap memory, 8 CPU cores and 32Gb physical disk. Also we deployed a service to resolve the domain name to an internal Cluster IP. Except for these default configurations, lots of parameters have to be tuned inside the Flink job that we created, which are analyzed in Chapter 5.

7.2.6 Rule generator and log publisher

The program that is responsible for rules generation was Dockerized to an image with the help of Docker and then is was deployed as a pod to the cluster. Its configuration file just gives access to the storage of the cluster to store the resulted rules to a file. Furthermore, since we do not have a real Hadoop cluster running into the cloud, but only the log data, we had to make a "fake" application to simulate the operation of Hadoop that produce the logs. Thus we make a simple application that read the log data from a file and then it just prints them line by line, as a real application logs. The log publishing program was also Dockerized , without the need for having access to the storage volume. However the singularity of this application come to the function that read and prints the logs. Since the reading and printing of a file is a very simple function, due to the high speed of execution, Kafka is unable to read properly all the logs lines, but instead it reads only a part of it. The actual problem was the way that Kubernetes prints the logs and consequently what logs FluentD collect to transit to Kafka. Due to log file rotation of inner system of Kubernetes we have to reduce the speed of printing the data from the python function. Thus we used a time sleep of some millisecond in order to have properly all the data published to Kafka topic. Except for that, another configuration of that program was

to print endlessly data after the file is finished. The reason behind that, was a limitation between the cooperation of Kafka and Flink. When Flink read streaming data from Kafka, is needed, in a periodical time, from Kafka to send a heartbeat to the task managers of Flink, otherwise the entire Flink job fails. Thus after the end of the log file we just send every twenty seconds data from the responsible pod to Kafka and then to Flink. Both programs were implemented using Python programming language. Both programs were implemented using Python programming language.

Chapter 8

Discussion and Conclusion

8.1 Discussion

In this work, we managed to create a real-time anomaly detection tool by integrating data mining technologies along with complex event processing that is deployed in the cloud. We automate the difficult and time-consuming process of defining rules in a complex event processing engine with the purpose of receiving alerts on future anomaly data. Furthermore, the entire framework was build using large scale technologies resulting to a concrete and scalable anomaly detection system. As we have presented in Chapter 6, we have really promising results. Specifically, we achieve a value of Recall around 98% and a Precision around 99%. Furthermore, another success is the scalability of the system which is hosted inside a Kubernetes cluster and the integration multiple popular big data frameworks. Finally, a strong point of this work is the ease deployment of the whole framework into the cloud, something really important for any industry that wants to exploit the capabilities of this work.

Except for our work's success, at this point, it is crucial to examine if we have managed to answer the research questions that we have defined in Chapter 1.

- *Can we detect real-time anomalies in system logs in the cloud?*: This question was answered as the overall outcome of this thesis work. We managed to create a real-time anomaly detection tool and deployed in the cloud using big data tools.
- *Can we use rule mining techniques to capture abnormalities in logs?*: In Chapter 4 we apply rule mining techniques to examine if we can capture abnormality. From the experiments we can verify that we manage to capture abnormal behavior in logs with 98% accuracy.
- *Can we learn rules that represent abnormal behavior?*: As an outcome of the previous research question we manage to learn which patterns represent the abnormality and thus can be used in next steps of this thesis work.
- *What technology can exploit patterns from logs?*: The answer to this question is the Complex Event Processing engine that we used in our tool. We translate the rules into CEP patterns to rise alerts on abnormal streaming data.
- *What is the minimum number of abnormal data during training, in order to accurate capture future abnormalities?*: This question, as we mentioned in Chapter 6, is related to the concept of sample efficiency. Specifically it measures the necessary training samples a model needs in order to reach a desired accuracy level. Since in real life, but also in our case, it is costly and impossible to have a massive number of abnormal training samples, it is crucial to test the efficiency of

our implementation in terms of the required number of data points. In the experiments in Chapter 6 we tried to answer this question. Eventually we have proven with even the 5% of the abnormal data that we have we managed to detect almost all abnormal blocks in test phase. Thus even with small number of available data, our tool is quite effective.

- *On what use case for the industry this work can be applied?*: The last question will be answered in a more abstract way. Since our tool was implemented in the cloud, it is really useful from companies that host their services in the cloud as well. Furthermore, we used data from a real distributed system that is widely used by all modern and decent companies. Thus, the system can be applied as well to data of other distributed systems that companies use. Finally, more and more software architectures make use of the concept of microservices. So collected logs form the different services and conduct anomaly detection to find root cause in case of failure is possible using our system.

8.2 Limitations

Despite its success, this thesis work shares some limitations that have to be mentioned. Each of the following limitations is related to a different part of the work, such as the implementation part or to a research perspective area.

- *Rule generation*: For our system to be working, we have first to generate the rules and then to feed the Flink job with the streaming data. Thus, in case of updating the rules, we have to stop the job and re-run it after the completion of the rule generator. That means that we cannot generate rules on streaming data, but only on static.

- *Memory management*: This limitation has been revealed in the runtime experiment in Chapter 6. Despite the fact that memory optimization is a subject out of scope of this thesis, with the experiment in Section 6.5, we saw that there is an issue in memory management. When we have a massive number of log messages(eg. 1.5 million messages), our framework fails to process all the data in a small period of time. This issue is highly related to an internal issue of complex event processing of Flink. Since the streaming data pass through every rule in the complex event processing engine, without any deletion of old and examined data, after some time the memory becomes full resulting some memory and runtime problems.

- *Number of Rules*: A limitation related to the number of rules also exists. For instance, from experiments, we figured out that some generated rules are really similar to each other, resulting to an increased number of rules. For example, two possible generated rules are "5 22 5" and "5 22 5 5 5". We can see that the first rule is also part of the second. However our system will generate both these rules, making the system a bit slower since it will examine the streaming data on both these almost same rules. Thus it is efficient enforce a minimum prefix property on the space of the generated rules in order to solve the issue of similar generated rules.

- *Generalization*: Lastly, a drawback of our anomaly detection system is that it has not been tested with another dataset. Since finding a real-time log dataset is quite challenging, we managed to test our system only with the HDFS logs.

However with the right pre-processor, the framework can efficiently generalize to other log data.

8.3 Conclusion and further improvements

To conclude, in this thesis work we investigate the possibility of creating an anomaly detection system for distributed log files. Additionally, we wanted to integrate rule mining technique with complex event processing and deploy the entire system to the cloud for scalability purposes. In a great extent we managed to achieve all the goals of this work, developing to a tool with really promising results in the filed of anomaly detection in system logs. Thus, this system can be applied in different cases such as root cause analysis of systems or even for predicting future failures before they actually happen. Since we can detect a problem before it occurs, it is possible to do the right actions to prevent the systems from failing. However, as in every project, there are some further improvements that can be done in the future. In the previous section we saw that there are some limitations of this work. It is clear that improvement have to be done in order to tackle these aforementioned limitations. All these limitations and the further improvements can fall in two different categories. Specifically, in this work, since it is consisted of algorithmic and engineering part, improvements exist in both these fields.

Algorithmic improvements

From algorithmic perspective, one improvement is to examine the effect of online rule mining techniques for pattern generation on streaming data instead of the classic static approach that was followed in this work. In that way, it will be possible to create rules on the fly and having a system that will continuously learn from new abnormal data without the need of model re-train. Another improvement has to be done in terms of generalization. First an examination of the effectiveness of the tool in another dataset has to be conducted. Then, if it is possible, to create general pre-processor for all kind of distributed system log data.

Engineering improvements

Since our system was deployed in the cloud and it consists of different tools that work together, there are some improvements that could help anyone who would like to use it in production. For instance, an improvement concern the way the tool is deployed into the cloud. In this work we had to deploy every tool independently and the to connect them by creating configuration files. An improvement will be to deploy everything at once using for example the Helm technology of Kubernetes. Finally, an improvement can be in terms of memory management in order to solve the limitation that we have presented above. Since the memory issue stems from the internal operation of complex event processing of Flink, the alteration of the engine falls in the engineering part of the work because the entire code of Flink process has to be changed.

Bibliography

[1] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. "Mining Association Rules Between Sets of Items in Large Databases, SIGMOD Conference". In: vol. 22. June 1993, pp. 207–. DOI: 10.1145/170036.170072.

[2] *Apache Kafka*. URL: https://kafka.apache.org/intro.

[3] Zachary K Baker and Viktor K Prasanna. "Efficient hardware data mining with the Apriori algorithm on FPGAs". In: *13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'05)*. IEEE. 2005, pp. 3–12.

[4] B Ross Barmish and Constantino M Lagoa. "The uniform distribution: A rigorous justification for its use in robustness analysis". In: *Mathematics of Control, Signals and Systems* 10.3 (1997), pp. 203–222.

[5] James S. Bergstra et al. "Algorithms for Hyper-Parameter Optimization". In: *Advances in Neural Information Processing Systems 24*. Ed. by J. Shawe-Taylor et al. Curran Associates, Inc., 2011, pp. 2546–2554. URL: http://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf.

[6] D Bhayani. "Identification of security breaches in log records using data mining techniques". In: *International Journal of Pure and Applied Mathematics* 119 (Jan. 2018), pp. 743–756.

[7] Paris Carbone et al. "Apache Flink™: Stream and Batch Processing in a Single Engine". In: *IEEE Data Eng. Bull.* 38 (2015), pp. 28–38.

[8] *CEP*. URL: https://flink.apache.org/news/2016/04/06/cep-monitoring.html.

[9] Nitesh V. Chawla et al. "SMOTE: Synthetic Minority Over-sampling Technique". In: *J. Artif. Int. Res.* 16.1 (June 2002), pp. 321–357. ISSN: 1076-9757. URL: http://dl.acm.org/citation.cfm?id=1622407.1622416.

[10] B. Debnath et al. "LogLens: A Real-Time Log Analysis System". In: *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. 2018, pp. 1052–1062. DOI: 10.1109/ICDCS.2018.00105.

[11] *Docker*. URL: https://www.docker.com/.

[12] Min Du et al. "DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning". In: Oct. 2017, pp. 1285–1298. DOI: 10.1145/3133956.3134015.

[13] "ElasticSearch, "Open-Source Log Storage,"" in: (Aug. 2017). URL: https://www.elastic.co/products/elasticsearch.

[14] *Flink*. URL: https://flink.apache.org/.

[15] *Flink event time*. URL: https://ci.apache.org/projects/flink/flink-docs-stable/dev/event_time.html.

[16] *Flink k8s*. URL: https://ci.apache.org/projects/flink/flink-docs-stable/ops/deployment/kubernetes.html.

[17] *Flink watermarks*. URL: https://ci.apache.org/projects/flink/flink-docs-stable/dev/event_timestamps_watermarks.html.

[18] Stephan Grell and Olivier Nano. "Experimenting with Complex Event Processing for Large Scale Internet Services Monitoring". In: (Jan. 2008).

[19] Aarish Grover. "Anomaly Detection for Application Log Data" (2018). Master's Projects. 635". In: DOI: https://doi.org/10.31979/etd.znsb-bw4d. URL: https://scholarworks.sjsu.edu/etd_projects/635.

[20] Hossein Hamooni et al. "LogMine: Fast Pattern Recognition for Log Analytics". In: *CIKM*. 2016.

[21] M. A. Hayes and M. A. M. Capretz. "Contextual Anomaly Detection in Big Sensor Data". In: *2014 IEEE International Congress on Big Data*. 2014, pp. 64–71. DOI: 10.1109/BigData.Congress.2014.19.

[22] Kelsey Hightower, Brendan Burns, and Joe Beda. *Kubernetes: Up and Running Dive into the Future of Infrastructure*. 1st. O'Reilly Media, Inc., 2017. ISBN: 1491935677, 9781491935675.

[23] Breier Jakub and Jana Branišová. "Anomaly Detection from Log Files Using Data Mining Techniques". In: *Lecture Notes in Electrical Engineering* 339 (Jan. 2015), pp. 449–457. DOI: 10.1007/978-3-662-46578-3_53.

[24] K. Jayan and A. K. Rajan. "Preprocessor for Complex Event Processing System in Network Security". In: *2014 Fourth International Conference on Advances in Computing and Communications*. 2014, pp. 187–189. DOI: 10.1109/ICACC.2014.52.

[25] K. Jayan and A. K. Rajan. "Sys-log classifier for Complex Event Processing system in network security". In: *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. 2014, pp. 2031–2035. DOI: 10.1109/ICACCI.2014.6968471.

[26] T. Kimura et al. "Proactive failure detection learning generation patterns of large-scale network logs". In: *2015 11th International Conference on Network and Service Management (CNSM)*. 2015, pp. 8–14. DOI: 10.1109/CNSM.2015.7367332.

[27] Ron Kohavi et al. "A study of cross-validation and bootstrap for accuracy estimation and model selection". In: *Ijcai*. Vol. 14. 2. Montreal, Canada. 1995, pp. 1137–1145.

[28] *Kubernetes Cluster Role*. URL: https://kubernetes.io/docs/reference/access-authn-authz/rbac/.

[29] *Kubernetes Daemonset*. URL: https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/.

[30] *Kubernetes Persistent Volume*. URL: https://kubernetes.io/docs/concepts/storage/persistent-volumes/.

[31] *Kubernetes Service account*. URL: https://kubernetes.io/docs/tasks/configure-pod-container/configure-service-account.

[32] Q. J. Lei. "Online monitoring of manufacturing process based on autocep". In: *International Journal of Online Engineering*. 2017.

[33] Alessandro Margara, Gianpaolo Cugola, and Giordano Tamburrelli. *Towards automated rule learning for complex event processing*. Tech. rep. 2013.

[34] L. Mariani and F. Pastore. "Automated Identification of Failure Causes in System Logs". In: *2008 19th International Symposium on Software Reliability Engineering (ISSRE)*. 2008, pp. 117–126. DOI: 10.1109/ISSRE.2008.48.

[35] Nijat Mehdiyev et al. "Determination of Rule Patterns in Complex Event Processing Using Machine Learning Techniques". In: *Procedia Computer Science* 61 (Dec. 2015), pp. 395–401. DOI: 10.1016/j.procs.2015.09.168.

[36] Alexandra Moraru. "COMPLEX EVENT PROCESSING AND DATA MINING FOR SMART CITIES". In: 2012.

[37] Raef Mousheimish, Yehia Taher, and Karine Zeitouni. "Automatic Learning of Predictive CEP Rules: Bridging the Gap Between Data Mining and Complex Event Processing". In: *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems*. DEBS '17. Barcelona, Spain: ACM, 2017, pp. 158–169. ISBN: 978-1-4503-5065-5. DOI: 10.1145/3093742.3093917. URL: http://doi.acm.org/10.1145/3093742.3093917.

[38] C. Mutschler and M. Philippsen. "Learning event detection rules with noise hidden Markov models". In: *2012 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*. 2012, pp. 159–166. DOI: 10.1109/AHS.2012.6268645.

[39] Jian Pei et al. "Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth". In: *Proceedings 17th international conference on data engineering*. IEEE. 2001, pp. 215–224.

[40] Erick Petersen, Marco To De Leon, and Stephane Maag. "An online learning based approach for CEP rule generation". In: Nov. 2016, pp. 1–6. DOI: 10.1109/LATINCOM.2016.7811563.

[41] Erick Petersen et al. "An Unsupervised Rule Generation Approach for Online Complex Event Processing". In: Nov. 2018, pp. 1–8. DOI: 10.1109/NCA.2018.8548210.

[42] Clifton Phua et al. *A Comprehensive Survey of Data Mining-based Fraud Detection Research (Bibliography)*. May 2013.

[43] Fluentd Project. *Fluentd | Open Source Data Collector*. URL: https://www.fluentd.org/.

[44] Ariel Rabkin and Randy Katz. "Chukwa: A system for reliable large-scale log collection". In: *Proceedings of the 24th International Conference on Large Installation System Administration* (Jan. 2010).

[45] *Scala*. URL: https://www.scala-lang.org/.

[46] Konstantin Shvachko et al. "The hadoop distributed file system." In: *MSST*. Vol. 10. 2010, pp. 1–10.

[47] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. "Practical Bayesian Optimization of Machine Learning Algorithms". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 2951–2959. URL: http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf.

[48] "Splunk, "Turn Machine Data Into Answers". In: (Aug. 2017). URL: https://www.splunk.com.

[49] J. Wei et al. "Analysis farm: A cloud-based scalable aggregation and query platform for network log analysis". In: *2011 International Conference on Cloud and Service Computing*. 2011, pp. 354–359. DOI: 10.1109/CSC.2011.6138547.

[50] Wei Xu et al. "Detecting Large-scale System Problems by Mining Console Logs". In: *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*. SOSP '09. Big Sky, Montana, USA: ACM, 2009, pp. 117–132. ISBN: 978-1-60558-752-3. DOI: 10.1145/1629575.1629587. URL: http://doi.acm.org/10.1145/1629575.1629587.

[51] Wei Xu et al. "Online System Problem Detection by Mining Patterns of Console Logs". In: *Proceedings of the 2009 Ninth IEEE International Conference on Data Mining*. ICDM '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 588–597. ISBN: 978-0-7695-3895-2. DOI: 10.1109/ICDM.2009.19. URL: https://doi.org/10.1109/ICDM.2009.19.

[52] Chengqi Zhang and Shichao Zhang. *Association rule mining: models and algorithms*. Springer-Verlag, 2002.

[53] Qiankun Zhao and Sourav S Bhowmick. "Sequential pattern mining: A survey". In: *ITechnical Report CAIS Nayang Technological University Singapore* 1 (2003), p. 26.

[54] Qiankun Zhao and Sourav S Bhowmick. "Association Rule Mining: A Survey". In: Jan. 2003.

[55] *Zookeeper*. URL: https://zookeeper.apache.org/.