Efficient Distributed Pipelines for Anomaly Detection on Massive Production Logs

XIAO CHEN
Abstract

The data volume of live corporate production logs is increasingly growing every day. On one hand, companies have to handle millions of data produced daily by their services which require high storage capacity. On the other hand, relevant information can be extracted from this massive amount of data and used for analysis according to different requirements, such as generating behavior patterns, detecting anomalies and making predictions. All of these can be achieved by machine learning and data mining techniques where the distributed platforms provide the computation ability and memory storage capacity for data intensive processing. Services such as payment monitoring in a company are very sensitive and require fast anomaly detection over streams of transactions. However, traditional anomaly detection techniques using distributed batch processing platforms such as Hadoop is very expensive to run and the anomalies cannot be detected in real time.

In order to overcome this drawback, Distributed Stream Processing (DSP) platforms such as Storm have proven to be a more flexible and powerful tool for dealing with such streams. Furthermore, since the anomaly patterns in data streams are not predefined and may change over time, unsupervised learning algorithms such as clustering should be used first to output significant anomalies which contribute to forming and updating anomaly patterns. The real-time anomaly detection on new data streams can be established by such patterns. This thesis project is aiming at providing a distributed system on top of Storm combining both batch-based unsupervised learning and streaming rule-based methods to detect anomalies in Spotify payment transactions in real time.

The anomaly detection system implements k-means and DBSCAN clustering algorithms as an unsupervised learning module to find out anomalous behaviors from payment transaction streams. Based on those anomalies, the frequent item set algorithm estDec is implemented to extract anomaly patterns. Stratified Complex Event Processing (CEP) engines based on Esper get reconfigured with such patterns to do rule-based anomaly detection in real time over absolute time sliding windows. Experimental results indicate that such a complex system over a unified data flow pipeline is feasible to detect anomalies in real time by rule-based
anomaly detection with CEP engine. Unsupervised learning methods can provide light weighted batch (nearly real time) based anomaly detection but different factors heavily influence the performance. The rule-based method shows that it performs better in the heavy anomaly density scenario in terms of sensitivity and lower detection latency.
Acknowledgements

First or foremost, I would like to thank my parents and my girlfriend for their continuously support which encourages me throughout my master study in Europe. Next, I would like to thank my supervisor Paris Carbone and Ying Liu for their smart ideas and patience to help me improve this master thesis. Many thanks goes to my industrial supervisor Mårten Sander in Spotify for his kind help in every aspect such as motivation of this work, discussion and relevant resources. I would also like to thank my examiner, Prof. Vladimir Vlassov, for his professional insight and advice about this thesis. Last but not the least, I would like to thank all my EMDC 2012 classmates especially, Pradeeban, Qi Qi, Orçun, Zen, Tomi, Dipesh, Anna, Roshan, Leo, Alex, Ale and Casey. Without you, I cannot have such a fantastic journey in Lisbon and Stockholm. Thank you.
# Contents

1 Introduction ................................. 1
   1.1 Motivations ............................... 2
   1.2 Challenges ............................... 2
   1.3 Contributions ............................ 3
   1.4 Structure of this thesis ................. 4

2 Background ................................. 5
   2.1 Anomaly detection ........................ 5
   2.1.1 Types of anomalies .................... 6
   2.1.2 Anomaly detection Methods .......... 9
       2.1.2.1 Supervised, unsupervised and semi-supervised 
            anomaly detection .................. 9
       2.1.2.2 Statistical, proximity-based and clustering-based 
            anomaly detection .................... 11
   2.2 Clustering ............................... 12
       2.2.1 Basic clustering methods .......... 13
       2.2.2 k-Means clustering algorithm ... 14
       2.2.3 Anomaly score ....................... 16
       2.2.4 DBscan clustering algorithm ...... 17
       2.2.5 Advantages and disadvantages of clustering based techniques 18
   2.3 Rule extraction .......................... 19
       2.3.1 Frequent itemset mining ............ 19
       2.3.2 Phases of the estDec algorithm .... 19
   2.4 Distributed Stream Processing Platforms .. 20
       2.4.1 Apache Storm ....................... 21
       2.4.2 Comparison with Yahoo! S4 ........ 25
   2.5 Apache Kafka ............................. 26
   2.6 Complex Event Processing ............... 27
       2.6.1 DSP vs. CEP ......................... 28
   2.7 Related work ............................ 28
       2.7.1 Clustering algorithms for stream data ..... 29
CONTENTS

2.7.2 Machine learning frameworks .................................. 29

3 System design ......................................................... 33
  3.1 Architecture Overview ........................................... 33
  3.2 Input data stream .................................................. 34
  3.3 Aggregation module ............................................... 35
  3.4 Unsupervised learning module ................................... 37
  3.5 Rule extraction module .......................................... 39
  3.6 Rule-based CEP module .......................................... 40
  3.7 Output module .................................................... 40

4 Implementation ....................................................... 43

5 Evaluation ............................................................. 45
  5.1 Goals ............................................................... 45
  5.2 Assumptions ....................................................... 45
  5.3 Evaluation Matrix ................................................ 46
  5.4 Evaluation settings ............................................... 46
  5.5 Rate anomaly scenarios ......................................... 47
  5.6 Evaluation result ................................................ 47
    5.6.1 K-means evaluation ........................................ 47
      5.6.1.1 Anomaly density ..................................... 47
      5.6.1.2 Anomaly score ...................................... 48
      5.6.1.3 Number of clusters ................................ 50
    5.6.2 DBSCAN evaluation .......................................... 52
    5.6.3 Rule-based anomaly detection with CEP .................. 53
    5.6.4 Fast detection by using CEP ............................... 54

6 Conclusions .......................................................... 57
  6.1 Future work ....................................................... 58

Bibliography ............................................................ 59

A JSON file used in implementation ................................ 65

B Stream definitions and subscriptions ............................. 67
# List of Figures

1.1 Key components associated with an anomaly detection technique. ....... 3
2.1 What are anomalies. .................................................. 6
2.2 Point anomaly. ....................................................... 7
2.3 Average temperature in Stockholm every month. ....................... 8
2.4 Collective anomaly. .................................................. 9
2.5 Cluster Example. ..................................................... 13
2.6 cluster transformation .............................................. 16
2.7 Anomaly example. .................................................... 17
2.8 Storm sample topology. ............................................. 22
2.9 Storm architecture. .................................................. 24
2.10 Main components in Storm. ....................................... 24
2.11 Kafka components. ................................................ 26
2.12 Kafka in Spotify. ................................................... 27
3.1 System Design. ....................................................... 34
3.2 Input data stream. ................................................... 35
3.3 Aggregation bolt. ..................................................... 36
5.1 Rates variation with different anomaly density. ....................... 48
5.2 Accuracy variation with different anomaly scores. ................... 49
5.3 FPR variation with different anomaly scores. ......................... 49
5.4 Sensitivity variation with different anomaly scores. ................. 50
5.5 Accuracy variation with different number of clusters. .............. 51
5.6 FPR variation with different number of clusters. .................... 51
5.7 Sensitivity variation with different number of clusters. .......... 52
5.8 Rates variation with different anomaly density for DBSCAN. .... 53
5.9 Rate variation (13.34%). .......................................... 53
5.10 Rate variation (3.94%). .......................................... 53
List of Tables

5.1 Performance matrix ........................................ 46
5.2 First detection time: Clustering-based vs. Rule-based CEP ........ 55
List of Acronyms and Abbreviations

DSP Distributed Stream Processing
CEP Complex Event Processing
HDFS Hadoop Distributed File System
ML Machine Learning
IFP Information Flow Processing
DBMS DataBase Management System
EPL Event Processing Language
WEKA Waikato Environment for Knowledge Analysis
MOA Massive Online Analysis
SAMOA Scalable Advanced Massive Online Analysis
SPE Streaming Processing Engines
DSL Domain Specific Language
FPR False Positive Rate
TP True Positive
TN True Negative
FP False Positive
FN False Negative
DBSCAN Density-Based Spatial Clustering of Application with Noise
Chapter 1

Introduction

Big data problems have attracted much attention from researchers in recent years. Social networks, sensor networks, health industry, e-commerce and other big data areas produce massive amounts of data. The large amount of information produced and stored everyday has pushed the limit of processing power and storage capacity. In order to process a large scale of data and analyze it accordingly, various distributed systems have been developed in both industry and academia. The most famous and influential one is Hadoop [1] with Hadoop Distributed File System (HDFS) [2], inspired by MapReduce [3] model from Google, providing reliable, scalable, distributed computing. However, it is a batch based system, and is relatively expensive to run multiple batches frequently. For some data models such as data streams, Hadoop cannot maintain its strength. In addition, some applications require real-time operations on data streams such as online analysis, decision making or trends predicting. Therefore, distributed stream processing platform is a better choice for those real-time applications.

Detecting patterns from data streams can be very useful in various domains. For instance, fire detection by sensor networks is very critical; the fire should be detected as soon as it happens. Fraud detection is also a very critical in finance market or online payment service. Traditional ways for pattern detection are to first store and index data before processing it [4], which cannot fulfill the real-time requirement. Detecting patterns is the first step of complex operations such as anomaly detection, containing methods from machine learning and data mining. Machine learning techniques mainly include two categories, classification and clustering, depending on whether a training set is given. For those big data streams without a training set in anomaly detection, clustering can be considered for identifying anomalous behaviors which contributes to extracting anomaly patterns in the real time analysis.

Furthermore, currently the streaming processing has been categorized into two emerging models that are competing: the data stream processing model and the
complex event processing (CEP) model [5]. The data stream processing model can be used for aggregation and clustering from data stream to find suspected anomalies. In addition, the CEP model aims at the pattern matching and filter out the true anomalies. Therefore, the scope of this real-time big data anomaly detection framework is to deal with unbounded streams of data by using the hybrid method of data stream processing model and complex event processing model. The combination of these two models could be advantageous over pure machine learning method on real-time anomaly detection.

1.1 Motivations

Anomaly detection has been a very popular and interesting problem in industries for many years, especially in those dealing with sensitive information such as traditional banking, e-commerce allowing payment and money transfer [4]. For instance, each day the Spotify payments team processes hundreds of thousands of payment transactions. It can be tough to sort through all of that data to extract information about how healthy the payment systems are. In order to make the systems optimized, it is necessary to identify the potential anomalies.

On the other hand, it is also very critical to identify the anomalies as soon as possible when they appear, and the large amount streaming data requires larger memory and a more scalable computational architecture. Traditional off-line anomaly detection methods may not adapt to these new challenges. Therefore, building a system for anomaly detection on top of Strom [6], a DSP platform, fulfils the requirement.

Since the anomalies types are unknown, unsupervised learning methods can help identifying the different potential anomalies without training data set labelled normal and/or anomalous. Meanwhile, the anomaly patterns may change over time, building a rule-extraction component to generate new anomaly pattern to detect new types of anomalies is very necessary. At last, by utilizing a CEP engine Esper [7], complex types of anomalies in the big data stream can be filtered out.

1.2 Challenges

Anomaly is an anomalous behavior that does not fit the expected normal patterns. We will introduce anomaly types and detection methods in detail in Chapter 2. It is straightforward that the behaviors that do not follow normal behavior patterns can be identified as anomalies. But how to build such normal behavior patterns can be a very challenging problem. Several main challenges are listed below [4].
1.3 Contributions

- Different domains contain different anomaly types. At times an anomaly in one domain may be the normal behavior in another, and vice versa.

- In some cases, the boundary between normal and abnormal behaviors is not very precise, which may produce a lot false positives and false negatives in the anomaly detection.

- Data with labels indicating whether it is normal or abnormal is not always possible. The data label should be decided by experienced experts.

- When the data stream is evolving (a.k.a concept drift), the old anomalous patterns should be also upgraded for detection efficiency.

Due to the aforementioned challenges, it is not easy to have a general perfect solution for anomaly detection. Therefore, the practical problems should be carefully studied according to their own contexts and characteristics.

Figure 1.1 described the key components when considering an anomaly detection problem.

![Figure 1.1: Key components associated with an anomaly detection technique.](image)

Our objectives are: 1) Since there is no training dataset, we will use unsupervised machine learning clustering algorithms for identifying particular anomalies; 2) We use the result of clustering algorithm to extract anomaly patterns; 3) We use CEP to identify anomalies by generated anomaly patterns in real time, the rules are reconfigurable, in order to make it adapt to concept drift.

### 1.3 Contributions

In this thesis, we present our solution to find anomalous transaction rates in payment system with the main focus on unsupervised method and rule-based method. We researched different anomaly detection methods and existing distributed platforms, trying to combine them for nearly real-time anomaly detection on big data streams. Our main contributions are listed below.
CHAPTER 1. INTRODUCTION

- Research on anomaly detection literatures and make an overview survey on machine learning and data mining;
- Present a hybrid anomaly detection system framework on a DSP platform Storm combining unsupervised learning and rule-based method;
- Implement unsupervised learning module containing k-means and DBSCAN clustering algorithms to perform light-weighted batch anomaly detection;
- Combine with a rule-extraction module to extract anomaly rules;
- Detect anomalies in real time with CEP engine by using anomaly patterns.

This work is part of payment team’s project at Spotify aiming at finding anomalies in the payment transactions to monitor the health of Spotify’s payment system.

1.4 Structure of this thesis

In Chapter 2, we introduced the background knowledge necessary to understand on anomaly detection definition and methods, basic clustering methods especially the k-means and DBSCAN clustering algorithms, a detailed description of distributed streaming processing platform Storm, introduction of frequent itemset algorithm estDec and some other concepts or tools that we will use in this thesis work. Furthermore, we reviewed the related work about anomaly detection and current research on distributed machine learning platforms.

Chapter 3 gives a detailed overview on the considerations we take for all design and implementation work as well as the general system architecture. In addition, we also discuss the limitation and assumptions within the problem domain.

In chapter 5, we design the experiments showing the anomaly detection efficiency of our system on Spotify payment transactions. The system evaluation is also provided.

At last, chapter 6 concludes this thesis work and gives further research direction as future work.
Chapter 2

Background

2.1 Anomaly detection

Anomaly detection (as known as outlier detection) is the process of finding data objects with behaviors that are very different from expectation. Such objects are called anomalies or outliers [8]. Anomaly detection has a wide applicability from different domains with documented uses in analyzing network traffic, monitoring complex computing systems such as data centers, as well as in analyzing credit card and retail transactions, manufacturing processes, and in surveillance and public safe applications [9]. Typical examples can be found in credit card fraud detection. That is, if the purchasing amount from a customer's credit card is much larger than the amount he spent in the past, this is probably an anomaly. If a customer has a Portuguese credit card and he pays bills in Portugal but in the next hour he pays bills in Sweden, this is probably another anomaly. In order to protect their customers, credit card companies usually have such common practice by using anomaly detection to detect such credit card fraud as quickly as possible. In fact, the ideal situation is that the anomalies can be detected immediately when such transaction happens. In Figure 2.1, Point A2 and the set of points A1 in the feature space can be regarded as anomalies since they are different and far from the other points.

Anomaly detection can be achieved by many techniques while the main task is to identify the abnormal behaviors from norm. Here, the assumption is made that the data objects with normal behaviors are the majority of the whole data set. In the credit card example, most credit card transactions are normal. If a credit card is stolen, the purchase amount or location could be very different from the authenticated owner’s previous purchasing record. The “difference” is what anomaly detection should detect.

However, it is not easy to identify the “real” anomalies. Some suspected
anomalies may turn out to be a false negative in different context and some “normal” behaviors may turn out to be a false positive if a few factors are not taken into consideration. In the previous examples, if there is a large amount purchase from a credit card, it may be regarded as an anomaly. However, the customer may buy an expensive product such as a TV while in the past purchase he only bought small stuff such as clothes, food, drinks. In this context, the purchase should not be regarded as an anomaly if prices are associated with goods. In the other example, the customer may pay bills from Sweden using a Portuguese credit card. The transaction will be regarded as an anomaly unless taking the payment IP address into consideration.

Therefore, before discussing the novel techniques used for anomaly detection, it should be defined the types of anomalies and their classification.

### 2.1.1 Types of anomalies

An anomaly is a data object that deviates significantly from the rest of the objects, as if it were generated by a different mechanism [8]. Here, we treat anomalies as “abnormal” behaviors in an object set while refer the other behaviors to normal (or expected) behaviors in the object set. Many different abnormal behaviors in a set of data objects can be regarded as anomalies. In general, anomalies can be classified into three categories: point anomaly (or global anomaly), contextual anomaly (or conditional anomaly) and collective anomaly [8] [9].

- **Point anomalies.** If a data object is significantly different from other data objects in a given object set, it is regarded as a point anomaly. For example, if an online retailer service suddenly finds the number of transactions made by who pay the bill by Paypal in Sweden drops to a very low rate, it should be regarded as a point anomaly. The reason probably is the Paypal APIs does not respond correctly at that time in Sweden. Since the point anomaly
2.1. Anomaly Detection

is the simplest one and quite common, it is very important to detect such anomalies as soon as possible in order to prevent the financial loss of the online retailer. In Figure 2.2, the point V1 in the red circle can be regarded as a point anomaly.

![Figure 2.2: Point anomaly.](image)

- Contextual anomalies. Sometimes a normal behavior is an anomaly if it is in a certain context. For example, the temperature is -20°C in Stockholm. It may be a normal value if the temperature is measured in winter, however, it can be a contextual anomaly if the temperature is measured in summer. Therefore a data object is a contextual anomaly if it deviates significantly with respect to a specific context of the object in a given data set. Contextual anomalies are also known as conditional anomalies because they are conditional on the selected context. Therefore, in contextual anomaly detection, the context has to be specified as part of the problem definition. For instance, in Figure 2.3, the point T1 in the red circle can be regarded as a point anomaly because the average temperature in summer normally cannot be only 3°C in Stockholm. Generally, in contextual outlier detection, the data objects are defined with the following two groups of attributes:

  - Contextual attributes: The contextual attributes of a data object define the object’s context (or neighborhood). In the example above, the contextual attributes may be date and location.
  - Behavioral attributes: These define the object’s non-contextual characteristics, and are used to evaluate whether the object is an outlier in the context to which it belongs. In the temperature example, the behavioral attributes may be the temperature, humidity, and pressure.
Collective anomalies. Given a data set, a subset of data objects forms a collective anomaly if the objects as a whole deviate significantly from the entire data set. Importantly, the individual data objects may not be anomalies. For example, in a car factory, every step on an assembly line should be within a fixed time but a bit delay is tolerable. Each step delay is not an anomaly from the point of view of single step. However, if there are 1000 steps, the accumulated delay will not be tolerable. That is to say, it is a collective anomaly. Unlike point or contextual anomalies detection, in collective anomalies detection, not only the behavior of individual objects should be taken into consideration, but also that of groups of objects. Therefore, to detect collective outliers, the background knowledge of the relationship among data objects such as distance or similarity measurements between objects is needed. For instance, in Figure 2.4, the set of data behavior C1 in red circle is different from the other set of data. It should be detected as a collective anomaly.

To summarize, a data set can have multiple types of anomalies. Moreover, an object may belong to more than one type of anomaly. Point anomaly detection is the simplest. Context anomaly detection requires background information to determine contextual attributes and contexts. Collective anomaly detection requires background information to model the relationship among objects to find groups of anomalies. However, a point anomaly or a collective anomaly could also be a contextual anomaly if analysed with respect to a context. Thus a point anomaly detection problem or collective anomaly detection problem can be transformed to a contextual anomaly detection problem by incorporating the context information. Furthermore, by transforming the data, for instance, by...
2.1. anomaly detection

aggregating it, it becomes possible to identify contextual and collective anomalies with point anomaly detection algorithms. In this case, the only difference is aggregation time granularity.

2.1.2 Anomaly detection Methods

Anomaly detection methods are various in the literature and practice [8] [9] [4]. Different categories can be divided from different perspectives. One way to categorize anomaly detection is whether a sample of data is given for anomaly detection model analysis and whether the given sample of data is labelled with a predefined “normal” or “anomalous”. According to this way, anomaly detection methods can be categories as supervised anomaly detection, semi-supervised anomaly detection and unsupervised anomaly detection. Another way to category anomaly detection is depending on how anomalies are separated from the rest of the data. They are statistical methods, proximity-based methods and clustering-based methods.

2.1.2.1 Supervised, unsupervised and semi-supervised anomaly detection

The main difference among the three methods is whether the given sample dataset is labelled with a predefined “normal” or “anomalous” for data training. In addition, Labelling is often done manually by a human expert and hence requires substantial effort to obtain the labelled training data set.

- Supervised anomaly detection. Supervised anomaly detection can be used in the condition that every data tuple in the training data set is provided expert-labelled “normal” or “anomalous”. Typical approach in such cases
is to build a predictive model for normal vs. anomaly classes. Any unseen data instance is compared against the model to determine which class it belongs to. In some applications, the experts may label only the normal objects, and any other objects not matching the model of normal objects are reported as outliers. Or in the other way around, experts may model the outliers and treat objects not matching the model of outliers as normal. However, two main challenges are addressed in this method. First, because of the heavy imbalance of normal objects and anomalies (anomalies are far less than normal objects); the sample data examined by domain experts and used in training set may not even have sufficient the anomaly types. The lack of outlier samples can limit the capability of classifiers built as such. Artificial anomalies may have to be made by experts in order to overcome these problems. Second, catching as many outliers as possible is far more important than not mislabelling normal objects as outliers in many anomaly detection applications. Consequently, the supervised anomaly detection algorithms have to recall the data set several times in order not to miss any anomalies. Therefore the key point of supervised methods of anomaly detection is they must be careful in how they train data and how they interpret data objects using the classifier due to the fact that anomalies are rare in comparison to the normal data samples.

• Semi-supervised anomaly detection. Different from all labelled data set in supervised method, this semi-supervised method only has a small set of the normal and/or outlier objects that are labelled, while most of the data are unlabelled. If some available labelled objects are normal, they can be used to train a model for normal objects together with unlabelled objects that are close by. The model of normal objects then can be used to detect outliers; those objects not fitting the model of normal objects are classified as outliers. However, if some labelled objects are anomalies, the semi-supervised method can be very challenging, because a small portion of anomalies cannot represent all kinds of anomalies in the training data set. Such techniques are not commonly used since it is not very effective, but getting assists from an unsupervised method which helps training the normal data can be an alternative to improve effectiveness.

• Unsupervised anomaly detection. Contrary to supervised anomaly detection, the unsupervised methods of anomaly detection do not have training data set labelled normal and/or anomalous. The techniques in this category make the implicit assumption that normal objects follow a pattern far more frequently than anomalies. Normal objects do not have to fall into one group sharing high similarity. Instead, they can form multiple groups, where each group
has distinct features. However, an anomaly is expected to occur far away in feature space from any of those groups of normal objects. The main drawback of this method is that if the assumption is not fulfilled, it will suffer from a large number of false negatives and false positives. The main advantage of this method is that the label is not needed. Therefore the result of unsupervised anomaly detection in an unlabelled data set can be the training data set for semi-supervised methods. Again, it assumes that the test data contains very few anomalies and the model learnt during training is robust to these few anomalies.

2.1.2.2 Statistical, proximity-based and clustering-based anomaly detection

- Statistical anomaly detection. Statistical methods (a.k.a. model-based methods) rely on the assumption that the data objects are generated by a statistical model, and the data tuple not fitting the model are anomalies. These methods can be derived from unsupervised or semi-supervised learning, to train dataset with normal samples and use a statistical inference test to determine whether a new tuple is anomalous or not. The effectiveness of these methods is highly depending on how accurate the statistical model is fitting the given data set. Statistical methods can be divided into two parts according to how the models are learned: parametric methods and non-parametric methods. A parametric method is that the normal data objects are generated by a parametric distribution with a parameter $P$. The object $X$ will be generated with probability by a probability density function of the parametric distribution $f(X,P)$. The object $X$ is more likely an anomaly if the probability is smaller. A non-parametric method depends on the fact of input data rather than depends on a predefined statistical model. In this way, techniques such as histogram and kernel density estimation will be used for predict the value based on historical data.

- Proximity-based anomaly detection. Proximity-based methods assumes that the anomalous data objects are far away from their nearest neighbors. The effectiveness of these methods depends on the proximity or distance measure used. Proximity-based methods can be mainly categorized into distance-based methods and density-based methods. A distance-based method is relying on how far away a data object is from its neighbors. If the distance from its neighbors is above a certain threshold, it will be regarded as an anomaly. A density-based method is depending on the investigation on the density of the object and its neighborhood. If the density of it is much lower than that of its neighbors, it will be treated as an anomaly.

- Clustering-based anomaly detection. Clustering-based methods hold the
assumption that the normal data objects can be clustered into a dense and big group, while the anomalies are very far away from the big group centroid or grouped into small clusters or even do not belong to any clusters. The methods are highly depending on the relationship between data objects and clusters. Three conditions can be considered to detect anomalies. To be more detailed, first, if a data object is not belonging to any cluster, it should be regarded as an anomaly. Second, if a data object is far away from its cluster center, it should be identified as an anomaly. Third, if a data object is in a very low density cluster compared with the other big clusters, all the data objects in this cluster should be treated as anomalies. However, clustering is an expensive data mining operation [5]. Thus, a straightforward adaptation of a clustering method for anomaly detection can be very costly, and does not scale up well for large data sets.

Other techniques could also be utilized as anomaly detection methods from different disciplines. For instance, classification-based method is often used in the supervised anomaly detection, by classifying the training data labelled with “normal” and “anomalous” to build data set model. Techniques such as information-theoretic-based methods and spectral-based methods, are both depending on certain contexts. Different methods are not totally segregated, a hybrid method may be used for a particular anomaly detection task [4]. In conclusion, various anomaly detection methods have their own advantages and disadvantages. The methods should be chosen according to specific anomaly problems.

2.2 Clustering

Clustering as a data mining tool can automatically divide a data set into several groups or clusters according to the data characteristic of the data set [8]. The divided groups are very different from each other but the data objects within the same group are similar to each other. For instance, points are separated in three groups by distance in Figure 2.5 . Unlike classification, clustering does not need a label when dividing data objects into groups. It is very useful in a broad range of areas such as biology, security, business intelligence, web search.

In anomaly detection area, clustering analysis can be used for unsupervised learning methods. Without knowing anomalies in advance, clustering is a good choice for preprocessing the data set that can help researchers gain insights on the data distribution, observe the characteristics of each cluster, and focus on a particular set of clusters for further analysis. Though clustering can be utilized by anomaly detection, it is not specially designed for anomaly detection. Clustering
finds the majority patterns in a data set and organizes the data accordingly, whereas anomaly detection tries to capture those exceptional cases that deviate substantially from the majority patterns. At times the very different exceptional cases that are treated as anomalies in anomaly detection may be only noise from clustering analysis perspective. Anomaly detection and clustering analysis serve different purposes.

2.2.1 Basic clustering methods

Various clustering methods can be found in literature [4] [8] [10] [9]. It is difficult to have a clear categorization of all the clustering methods, because many categories are overlapping with each other. But a roughly organized categorization is good enough to help people get an insight of clustering algorithms. They can be classified into the following four methods: partitioning methods, hierarchical methods, density-based methods and grid-based methods.

- Partitioning methods. A partitioning method is a basic clustering method that divides a data set into several partitions. Given a data set of N data objects, it will be partitioned into $K(K \leq N)$ partitions representing clusters. Each data object will be exactly put into a cluster and each cluster must contain at least one data object. Most partitioning methods are distance-based, which means the general outcome of a good clustering algorithm is that the data objects are much closer in the same cluster whereas further away from the data objects in different clusters. In addition, other criteria are also considered for judging the quality of clustering algorithm such as local optimum.

- Hierarchical methods. A hierarchical method mainly has two ways to divide the data set into several clusters. One way is a top-down approach. It starts
2. BACKGROUND

from the whole data set in the same cluster, decompose the big cluster into small clusters in every successive iteration and finally every data object is in one cluster. The other way is a bottom-up approach. It starts from every data object forming its own group. One group can merge the nearby groups. Finally all the data objects will be merged into one or can be terminated by some criteria. These two ways both contain iterative steps, which is where the name comes from. The main drawback of these methods is once the merge or split is done, it cannot be undone. Therefore these methods can be used with small operation cost and different choices in clusters should not be considered.

• Density-based methods. As mentioned in partitioning methods, most of clustering algorithms are based on distance between objects, but other clustering methods can be based on the density among data objects. The whole idea is that a cluster starts from one point and grows bigger by putting the neighbor point in the cluster until it reaches the predefined threshold.

• Grid-based methods. A grid-based method is to categorize all the data objects into a finite number of cells to form a grid structure. All the clustering operations are performed on a grid structure. The main advantage of grid-based methods is the fast processing time because it only depends on the number of cells in each dimension in the quantize space not on the number of data objects.

2.2.2 k-Means clustering algorithm

As mentioned above, clustering is an expensive data mining operation. For its simplicity and speed, the k-means clustering algorithm is a good practice for unsupervised anomaly detection [8] [11]. K-Means clustering algorithm is a centroid-based partitioning method.

Definition: Given a data set containing n elements, \( D = \{x_1, x_2, x_3 \ldots x_n \} \), where each observation is a d-dimensional real vector (or in a Euclidean space). The k-means clustering algorithm aims at partitioning data set \( D \) into \( k \) clusters, \( C_1, C_2 \ldots C_k \), that is, \( C_i \subset D \) and \( C_i \cap C_j = \Phi \) for \( 1 \leq i, j \leq k \). The centroid of a cluster is defined as \( c_i \), to represent the cluster. In fact, it can be defined in different ways such as the mean of the cluster objects, the medoid of the cluster objects. Conceptually, centroid is the center point of the cluster. Distance \( \text{dist}(x, y) \) represents the Euclidean distance between point \( x \) and point \( y \). To measure the cluster quality, within-cluster sum of squares (WCSS) is defined. The WCSS of k-means cluster is the sum of squared error between all objects in
2.2. Clustering

$C_i$ and the centroid $c_i$, defined as:

$$E = \sum_{i=1}^{k} \sum_{p \in C_i} \text{dist}(p,C_i)^2$$

Where $E$ is the sum of the squared error for all objects in the data set $C_i$; $p$ is the point in space representing a given data object. This function is trying to minimize the square root within one cluster and make $k$ clusters as separated as possible.

The k-means clustering algorithm is an NP-hard problem, that is, in order to get the best result of clustering, the algorithm should run all the possible starting points and all the possible combinations. To overcome this tremendous computational cost, the common optimization is using the greedy approaches in practice. It is simple and commonly used.

The algorithm starts from random selected points as initial centroids of clusters. For the remaining points, they are assigned to the most similar clusters according to their distances from centroids. Several distance functions can be used such as Euclidean distance function. Then in each cluster, the newly centroid is calculated by all the points in it. Next, all the points are reassigned according to the distance between the new centroids and themselves. After the reassigning, the new centroids are recomputed. The iterative steps will continue until the clusters are stable. “Stable” means the clusters do not change any more since the last iteration. This is the termination condition.

The detailed procedure is described in Algorithm 1. Figure 2.6a, Figure 2.6b and Figure 2.6c show the cluster transformation.

### Algorithm 1 K-means algorithm

**Input:**
- $k$: The number of clusters;
- $D$: A data set containing $n$ objects;

**Output:**
- A set of $k$ clusters;

1: Arbitrarily choose $k$ objects from $D$ as the initial cluster centers;

2: while Cluster means change do

3: (Re) assign each object to the cluster to which the object is the most similar, based on the mean value of the objects in the cluster;

4: Calculate the mean value of the objects for each cluster to update their means;

5: end while;
The time complexity of the k-means algorithm is $O(nkt)$, where $n$ is the total number of objects, $k$ is the number of clusters, and $t$ is the number of iterations. Normally, $k \ll n$ and $t \ll n$. Therefore, the method is relatively scalable and efficient in processing large data sets.

The k-means method is only locally optimized and the algorithms often terminated when local optimum is achieved. Global optimum is not guaranteed in k-means cluster. The result of k-means clustering is highly depending on the initial data points selected which is the initialization cluster centroid. In order to get a good result of k-means clustering, the common way is to run the algorithm several times until the local optimum is reached and clusters do not change through different initial centroids.

2.2.3 Anomaly score

By using the k-means cluster method, the data set can be participated into $k$ clusters. As shown in Figure 2.7, the data set is divided into 3 clusters and in each cluster, the center of each cluster is marked with a red dot. For each object, $O$, an anomaly score is assigned to the object according to the distance between the object and the center that is closest to the object [8]. Suppose the closest center to $O$ is $C_O$; then the distance between $O$ and $C_O$ is $\text{dist}(O,C_O)$, and the average distance between $C_O$ and the objects assigned to $O$ is $L_{C_O}$. The ratio $\text{dist}(O,C_O)/L_{C_O}$ measures how $\text{dist}(O,C_O)$ stands out from the average. The larger the ratio, the farther away $O$ is relative from the center, and the more likely $O$ is an outlier. In Figure 2.7, points A, B, and C are relatively far away from their corresponding centers, and thus are suspected of being outliers.
2.2. Clustering

Figure 2.7: Anomaly example.

2.2.4 DBscan clustering algorithm

Other than the centroid-based partitioning clustering method k-means, the density-based clustering algorithms are good at finding clusters of arbitrary shape and have advantages in identifying anomalies. The most cited and representative one is DBSCAN [12], Density-Based Spatial Clustering of Application with Noise.

DBSCAN is a density-based clustering method, where the density of an object \( o \) can be measured by the number of objects close to \( o \). The basic concept of DBSCAN is to find neighborhoods from core objects to form dense regions as clusters. Two user-specified parameters are defined to quantify the neighborhood of an object: \( \varepsilon \) and MinPts. The \( \varepsilon \)-neighborhood of an object \( o \) is the space within a radius \( \varepsilon \) centered at \( o \). MinPts determines the density threshold of dense regions. The core objects are those containing at least MinPts objects in the \( \varepsilon \)-neighborhood. With these two parameters, the clustering task is converted to form dense regions as clusters by using core objects and their neighborhood.

In addition, for a core object \( q \) and an object \( p \), \( p \) is directly density-reachable from \( q \) if \( p \) is within the \( \varepsilon \)-neighborhood of \( q \). Thus, in DBSCAN, \( p \) is density-reachable from \( q \) (with respect to \( \varepsilon \) and MinPts in object set \( D \)) if there is a chain of objects \( p_1, \ldots, p_n \), such that \( p_1 = q, p_n = p \), and \( p_{i+1} \) is directly density-reachable from \( p_i \) with respect to \( \varepsilon \) and MinPts, for \( 1 \leq i \leq n, p_i \in D \). Furthermore, two objects \( p_1, p_2 \in D \) are density-connected with respect to \( \varepsilon \) and MinPts if there is an object \( q \in D \) such that both \( p_1 \) and \( p_2 \) are density-reachable from \( q \) with respect to \( \varepsilon \) and MinPts. It is clear to see that if \( o_1 \) and \( o_2 \) are density-connected and \( o_2 \) and \( o_3 \) are density-connected, then \( o_1 \) and \( o_3 \) are density-connected as well [8].
The procedure of DBSCAN forming clusters is that all objects in data set \( D \) are marked as “unvisited” initially. DBSCAN randomly selects unvisited object \( p \), marked \( p \) as “visited”, and check whether the \( \varepsilon \)-neighborhood of \( p \) has at least \( \text{MinPts} \) objects. If not, \( p \) marked as noise point which is an anomaly in the data set \( D \). If yes, a new cluster \( C \) is created for \( p \), and check the \( \varepsilon \)-neighborhood of \( p \) to add those objects which do not belong to any cluster into cluster \( C \). DBSCAN iteratively checks the neighborhood of core objects which are in cluster \( C \) until no more objects can be added. The objects that are added to clusters is marked as “visited”. To find next cluster, DBSCAN randomly choose an “unvisited” object and start the iterative process again until all the objects are visited. The pseudo code of DBSCAN will be shown in 3.4. The algorithm is effective in finding arbitrary-shaped clusters with appropriate settings of the user-defined parameters, \( \varepsilon \) and \( \text{MinPts} \).

### 2.2.5 Advantages and disadvantages of clustering based techniques

In this section we will discuss the advantages and disadvantages of clustering based techniques.

- **Advantages.** The biggest advantage is that clustering based techniques can run in an unsupervised mode. They will identify the anomalies without predefined labels. They can also identify the potential anomalies rather than manually identify by experts. In addition, such techniques can handle complex data type by simply applying a clustering algorithm on a particular data type. The running time of clustering based techniques is short compared with other techniques because the cluster number is relatively smaller than the number of data objects.

- **Disadvantages.** On the one hand, because of the nature of the algorithms, the performance of such techniques is highly depending on the effectiveness of clustering normal data objects. On the other hand, clustering and anomaly detection serve different purpose, hence such techniques are not optimized for anomaly detection. In clustering based techniques, the data objects have to be assigned to one cluster even though it may be an anomaly if more clusters are defined, other criteria such as anomaly score are needed for identifying the anomalies. The computational complexity is a bottleneck for clustering based techniques, especially if \( O(N^2d) \) clustering algorithms are used.
2.3 Rule extraction

2.3.1 Frequent itemset mining

Since the anomaly patterns embedded in the data streams may change as time goes by, it can be valuable to identify the recent change in the online data streams. Frequent itemset mining is one of the areas that focus on mining data and discover patterns from data [13]. Frequent itemsets means a set of items that appears in many baskets is said to be “frequent”. The definition of closed itemset is that an itemset is closed in a data set if there exists no superset that has the same support count as this original itemset. With these two definitions, the problem that frequent itemset mining solves can be formulate like this: given a set of transactions, find the top K inferred item combinations with support larger than the predefined minimal support. Support means the ratio of transactions in which an itemset appears to the total number of transactions.

Most of the traditional frequent itemset mining techniques aim at transaction databases which is offline batch-based data mining. Several frequent itemset mining researches on online streaming data can be found in literature, such as the estDec algorithm for mining recent frequent itemsets from a data stream [14], the CloStream algorithm for mining frequent closed itemsets from a data stream [15].

The CloStream algorithm is a reasonably efficient algorithm. A limitation of this algorithm is that it is not possible to set a minimum support threshold. Therefore, if the number of closed itemsets is large, this algorithm may use too much memory. Other than focusing on closed frequent itemsets, the estDec algorithm is to find recent frequent itemsets adaptively over an online data stream. It can be noticed that not all of itemsets that appear in a data stream are significant for finding frequent itemsets. Because an itemset which has much less support than a predefined minimum support is not necessarily monitored since it has less chances to be a frequent itemset in the near future. Therefore, by decaying the old occurrences of each itemset as time goes by, the effect of old transactions on the mining result of the data stream is diminished. In this way, the processing time and memory requirement can be decreased by sacrificing the accuracy which has only small influence on results.

2.3.2 Phases of the estDec algorithm

The estDec algorithm mainly contains four phases: parameter updating, count updating, delayed-insertion and frequent itemset selection. The detailed algorithm is described in Section 3.5.
• Parameter updating phase. When a new transaction is generated in the data stream, the total number of transactions set adds this transaction and updates its content.

• Count updating phase. If previous itemsets in a monitoring lattice appear in this transaction, then the count of itemsets is updated to current state. The monitoring lattice is a prefix-tree lattice structure which maintains the different combination of items that appear in each transaction. After the updating, if the update support of an itemset in a monitoring lattice becomes less than a predefined threshold, it will be pruned from the monitoring lattice because it is no longer considered significant. If the number of itemset is 1, the itemset will not be pruned.

• Delayed-insertion phase. The main goal is to find the most possible itemset which becomes frequent. On the one hand, if the a new itemset appears in the newly generated transaction, it will be insert into a monitoring lattice. On the other hand, for the itemset which is not in the monitoring lattice, if the number of it is more than 1 and its estimated support is large enough, it will be inserted into the monitoring lattice.

• Frequent itemset selection phase. The mining result of all current frequent itemsets in a monitoring lattice is produced but only when it is necessary.

Since our aim is to mine frequent itemset on big data streams, the memory usage must be taken into consideration. Even thought closed frequent itemset mining is more accurate, the frequent itemset mining is accurate enough for anomaly pattern extraction. For more efficient memory usage and processing time, estDec is a better choice for our aim.

2.4 Distributed Stream Processing Platforms

Many distributed platforms and computing systems are available for different distributed computing problems. The most famous and widely used one is Hadoop, which is a batch based distributed system for recommendation system and processing massive amounts of data in industry. Inspired by Hadoop, Spark [16], an open-source distributed data analytics cluster computing framework on top of HDFS (Hadoop Distributed File System), is created for high speed large-scale data processing. It incorporates in-memory capabilities for data sets with the ability to rebuild data that has been lost. However, those big data solutions focus on batch processing of large data set and are not flexible for the stream processing paradigm. Therefore, new distributed systems handling stream processing are
2.4. DISTRIBUTED STREAM PROCESSING PLATFORMS

developed, such as Apache Storm [6]. Storm supports the construction of topologies processing continuous data as streams. Different form batch processing systems, Storm will process data endlessly until it manually terminated. In this section, we will introduce Apache Storm’s concepts and architecture in detail, as well as make comparisons with Apache S4 [17], another streaming processing platform.

2.4.1 Apache Storm

Apache Storm is a distributed and fault-tolerant real time computation system. It can reliably process unbounded data streams in real time [6]. It is originally acquired and open sourced by Twitter, later it maintained by Apache Software Foundation in 2013. The most recent version is 0.9.1. It is written in Conjure but supports multiple languages.

As a distributed streaming platform, what Storm is mainly different from Hadoop is the core abstraction. Hadoop is designed for processing data batches by using the MapReduce method, whereas Storm is designed for streaming data, the core abstraction is stream. A Storm application creates a topology representing the structure of stream data interfaces. Sample topology can be seen in Figure 2.8. It provides similar functionality as a MapReduce job but the topology will conceptually run indefinitely until it is manually terminated. The main concepts in Storm are described below.

- **Topology.** All the information of a Storm application is based on topology. Topology defines the stream partitioning though spouts and bolts. In short, a topology is a graph of spouts and bolts that are connected with stream groupings.

- **Stream.** It is the Storm’s core abstraction. In Storm, unbounded data tuples continuously flowing in a distributed fashion forms the stream. It can be processed in parallel, according to the predefined fields in tuples.

- **Spouts.** A spout in storm is the source of topology input. Normally spouts will read from external sources such as Kestrel, Kafka [18] or Twitter streaming API. Two modes are defined in spouts, reliable and unreliable. In the reliable mode, the spout will resend the tuples which are failed previously, whereas the spout will just send the tuples without caring about whether they reach the destination or not in the unreliable mode.

- **Bolts.** All processing functions in Storm are done by bolts, such as filtering, aggregation, join and communication with external sources. Bolts consume
the tuples from previous components (either spouts or bolts), process data from tuple and emit tuples to next components.

- **Stream groupings.** Storm utilizes stream grouping mechanism to determine the partitioning of streams among the corresponding bolt’s task. Storm also provides different default grouping methods which are described below.
  
  - **Shuffle grouping.** Tuples are distributed in a random round-robin fashion. It randomizes the order of the task ids each time when it goes through them all. Thus the bolt tasks are guaranteed to have equal tuples from the stream.
  
  - **Fields grouping.** The stream is divided by fields and goes to the bolt’s tasks according to the field’s name.
2.4. Distributed Stream Processing Platforms

- All grouping. Every subscribed bolt has the exactly same replicated stream in all grouping method.
- Global grouping. The whole stream goes directly into the bolt’s task with the lowest id.
- None grouping. Currently it is the same as shuffle grouping, but the original idea is that different bolts with none grouping method will execute in the same thread without caring about grouping.
- Direct grouping. The emitting bolt will decide which subscribed bolt it will send stream to.
- Local or shuffle grouping. If one or more subscribed tasks are executed in the same worker process, this method will distribute the tuples within those tasks like a shuffle grouping.

- Tasks and workers. A task is a thread of execution within spouts or bolts. Spouts or bolts can have many tasks according to the requirement. Tasks are the basic components receiving sub-stream by fields. A worker is a process within a topology. Each worker process is a physical JVM and executes a subset of all the tasks for the topology. Naturally, Storm tries to distribute tasks to workers evenly.

Storm architecture is described in Figure 2.9. The above concepts give a general picture of Storm, however, it is important to understand the Storm architecture in order to have a clear idea how Storm works. As shown in Figure 2.10, the three main components of a Storm cluster are Nimbus, Zookeeper and Supervisors.

- Nimbus and Supervisors. Both Nimbus and Supervisors are daemons for managing and coordinating resources allocation in Storm through Zookeeper. Nimbus is the master node in Storm that acts as entry point to submit topologies and code for execution on the cluster. It is similar to Jobtracker in Hadoop. It is responsible for distributing code around the cluster, assigning tasks to machines and monitoring for node failures. Whereas the supervisors are the worker nodes to listen for work assigned to relevant machines and start and stop worker processes as necessary based on what Nimbus has assigned to them. Each worker process corresponds to a JVM process that executes a subset of a topology; a running topology consists of many worker processes spread across many machines.

- Zookeeper. A Zookeeper cluster coordinates all the resource reallocation by storing the configurations of Storm workers between Nimbus and
Supervisors. In addition, Storm cluster is very stable because of the Zookeeper design. Nimbus daemons and Supervisor daemons are stateless.
and fail-fast. Zookeeper keeps all the state in local disk. When Nimbus or Supervisors are failed, they can restart and come back to work with Zookeeper’s backup like nothing happened.

Furthermore, Storm implements a set of characteristics to ensure its performance and reliability. Storm uses ZeroMQ for message passing, which removes intermediate queueing and allows messages to flow directly between the tasks themselves. Under the covers of messaging is an automated and efficient mechanism for serialization and deserialization to Storm’s primitive types. Storm is also focus on fault tolerance and management perspectives. Storm implements guaranteed message processing such that each tuple is fully processed through the topology; if a tuple is discovered not to have been processed, it is automatically retransmitted from the previous component. Storm also implements fault detection at the task level, where upon failure of a task, messages are automatically reassigned to quickly restart processing, especially with a fail-fast design. Storm does better process management than Hadoop, where processes are managed by supervisors to ensure that resources are fully and evenly used.

In summary, Apache Storm’s concepts and structure make it well designed for streaming processing. It hides the low level resource allocation and flow control from users and provides abstract interfaces for users to build specific distributed stream processing applications.

2.4.2 Comparison with Yahoo! S4

Other than Storm, several early implementations of distributed streaming platform concept are also very interesting, such as Yahoo! S4. Yahoo! S4 was initially developed and released by Yahoo! Inc. and later became an Apache Incubator project. S4, Simple Scalable Streaming System, is a distributed stream processing engine inspired by the MapReduce model. It is a general-purpose, distributed, scalable, partially fault-tolerant, pluggable platform that allows programmers to easily develop applications for processing continuous unbounded streams of data [17].

The main advantage of Storm compared with S4 is that Storm provides guaranteed processing which is at-least-once-delivery property whereas S4 does not have such a property and may potentially lose data during processing. Storm also provides transparent task distribution but S4 needs a complex configuration using XML-like file. The active community of Storm users is another advantage for Storm. However, S4 provides an automatic load balancing mechanism. In Storm, the Zookeepers only distribute resources and tasks evenly among Supervisors; it does not provide complex load balancing mechanisms.
2.5 Apache Kafka

Apache Kafka is public-subscribe messaging rethought as a distributed commit log [18]. It is an open-source message broker project written in Scala originally developed by LinkedIn and later maintained by the Apache Software Foundation. The most recent version is 0.8.1.1. It aims at providing a persistent, distributed, high-throughput, low-latency platform for handling real-time log data. The high level topology is shown in Figure 2.11. That is, producers publish messages through Kafka cluster and the consumers that subscribe to a topic receive the messages in it. Brief introduction is provided in this thesis because Kafka will be used in our implementation. It provides the big data streams of payment transaction logs which serve as the input of the whole Storm-based anomaly detection system.

![Figure 2.11: Kafka components.](image)

- **Topics.** A topic is a certain feed in which published messages are categorized. Consumers can subscribe to this topic to get relevant messages while producers will publish these messages for a configurable period of time no matter whether they are consumed or not.

- **Distributions.** Since the log file may grow larger and exceed the fixed storage size, it is essential that the partitions of the log are distributed over the servers in the Kafka cluster and each server handles data and requests for a share of the partitions. The partitions are replicated and make fault tolerance available.

- **Producers.** Producers publish messages to different topics. The main role of producers is to decide how to assign messages to partitions within topics. By using techniques such as round-robin method, the producers can fulfill the load balancing requirement.
2.6  Complex Event Processing

- Consumers. By subscribing to a topic, the consumers can get massages within this topic. The tricky thing in Kafka consumer is that Kafka has only provided a total order over messages within a partition, not among different partitions in a topic.

- Guarantees. From high-level perspective, Kafka provides the following guarantees: first, if a message m1 is sent earlier than m2, m1 will appear before m2 in the log; second, when a consumer wants to have the logs, messages are stored in order in them; third, if there are N replicas, Kafka can tolerate up to N-1 server failures without losing any committed messages in the log.

Since Spotify use the stable Kafka 0.7.1 in production, some custom extension has been provided to strengthen the service. Because of the crossing site partitions, end-to-end reliable delivery is ensured for the quality of service. A compression and encryption service is also provided to ensure the network transmission quality and security. The structure is in Figure 2.12.

![Figure 2.12: Kafka in Spotify](https://www.jfokus.se/jfokus14/preso/Reliable-real-time-processing-with-Kafka-and-Storm.pdf)

2.6  Complex Event Processing

In the Information Flow Processing (IFP) domain, the traditional system for processing information is DataBase Management System (DBMS) which will firstly store and index data in the database, then process the data according to user’s requirement. However, some areas such as intrusion detection or fire
detection do not require the storage of all informations. On the contrary, those areas need fast respond from the real-time information. This requirement inspires the development of complex event processing (CEP), which treats information items as event flows [5]. By predefined processing rules, the CEP can filter and detect occurrence of certain patterns from complex events by its powerful expressiveness. If detected, CEP will notify related parties [19]. For its high throughput, high availability and low latency, the real-time detection can be achieved on CEP. The event processing language (EPL) that CEP used is a SQL-like language, which makes it easy to provide complex processing logic on data streams. All the processes are carried out in the memory, the external storage is not needed.

2.6.1 DSP vs. CEP

There are many differences between DSP and CEP. The most significant difference is the model they view for information flow. DSP model describes the information flow as streams of data from various sources while CEP model treats the information flow as notifications of continuous events. Another difference is that DSP addresses scalability by using distributed systems while CEP does not. Therefore, a combination of DSP and CEP can achieve both scalability and high expressiveness.

2.7 Related work

Anomaly detection has been researched for a wide range of application domains and diverse research areas such as statistics, machine learning and data mining. Literatures on those subjects can be found in a very large volume. Normally many anomaly detection techniques are designed for specific problems while others are more generic. Machine learning (ML) is closely related to anomaly detection for which many ML algorithms can be used. Traditional anomaly detection approaches focus on the database applications and apply ML algorithms on them by passing data set multiple times. Recent and more interesting challenges for anomaly detection have been to detect anomalies on a large scale of data in real time. Traditional database management systems (DBMS), which need to store and index data before processing it, can hardly fulfil the requirements of timeliness coming from such domains [5]. Mining patterns in data stream has been an essential role because patterns that do not conform to expected behaviors are regarded as anomalies [20]. Various approaches to the problem of anomaly detection (or novelty detection) have been described in [21]. Concept drift has also been very actively researched [22]. Eduara J. Sponasa et al proposed a learning
technique, based on the k-means clustering algorithm, aiming at treating anomaly detection and concept drift by single strategy. By updating the model of normal concept, their algorithm is able to work in an online fashion [23].

2.7.1 Clustering algorithms for stream data

Traditional ML algorithms are not optimized for a large scale of stream data, because the efficiency of algorithms requires hardware support which cannot be satisfied with a single machine, such as the computation ability and the memory capability. One economy way is to use distributed systems to assign tasks among a group of machines. Despite of hardware support, the ML algorithm scalability is also a very important issue. Not all the ML algorithm can be implemented in a distributed manner; the design natural of the algorithms heavily influences the scalability. Furthermore, the efficiency of the distributed ML algorithms is usually not as good as their efficiency on the single machine because of the cost of network communications among machines in the distributed system.

To find patterns from data streams without training data, unsupervised anomaly detection utilizing distributed clustering algorithms should be used. Clustering is an unsupervised ML task which builds a model to cluster data according to similarity. Many clustering algorithms optimized for data stream have been studied with the improvement in performance, memory usage, computational complexity, clustering quality and scalability [10] [24]. For instance, BIRCH [25], Scalable k-means [26], Single pass k-means [27], Stream LSearch [28], CluStream [29], D-Stream [30], ODAC [31], SWClustering [32], ClusTree [33], DGClust [34], StreamKM++ [35].

K-means is a centroid-based partitioning method and widely used for stream clustering algorithm. Aforementioned algorithms, scalable k-means, single pass k-means, CluStream, are all utilizing the basic k-means idea. Therefore, massive amount of data stream can be handled by implementing k-means algorithm on top of DSP platforms.

2.7.2 Machine learning frameworks

Machine learning and data mining are two terms that are commonly confused. Both of them employ the same methods and overlap significantly. The difference between them is their purposes: machine learning is focusing on the prediction, based on the known properties learned from training set while data mining is focusing on the discovery of unknown properties in the data. In the following parts, we will not strictly distinguish these two terms since they are really close.

Existing ML frameworks can help researchers run ML tasks and understand the data behaviors, which is a good practice for building anomaly detection
applications on a large scale of stream data. What’s more, such ML frameworks represent the different problem domains when using ML algorithms. Several open source ML frameworks are introduced below.

The most famous one is the Waikato Environment for Knowledge Analysis (WEKA) [36] [37], developed by the University of Waikato, New Zealand which is regarded as a batch based workbench for ML algorithms. WEKA contains a collection of ML algorithms for data mining tasks [12] and tools for preprocessing. WEKA has a graphic interface for underlying functionalities, making it easier for user to prototype ML algorithms. It also provides the functionalities for visualization. The characteristic of this framework is to store all the data in memory and run sequentially on a single machine. In some algorithms, the dataset requires multiple passes.

Related to WEKA and developed by the University of Waikato, Massive Online Analysis (MOA) [38] is a framework for data stream mining. It contains evaluation tools and a collection of online ML algorithms for stream clustering, classification and recommender system. The most important difference between stream processing and batch processing is that the streaming data set is infinite. Stream processing cannot store all the incoming data otherwise the memory will be easily filled. Instead, it should use one-pass approach which means the arriving data is only to be processed once. Therefore a model or summery should be frequently updated whenever data arrives. Though the MOA is designed for streaming data, it still runs on a single machine.

In order to make stream processing ML framework more distributed, Scalable Advanced Massive Online Analysis (SAMOA) [39], is developed by Yahoo! Research. SAMOA is a distributed streaming ML framework that contains a programing abstraction for distributed streaming ML algorithms [40] [41]. It provides an interface for different streaming processing engines (SPEs) such as Apache S4 and Apache Storm. New ML algorithms can be added to SAMOA without dealing with the details of SPEs. The main contribution made by SAMOA is the abstraction adapter layer for various SPEs. It does not provide many ML algorithms in the original implementation.

Taking advantage of distributed processing platforms, Mahout is a framework aiming at building a scalable machine learning library [42]. The early Mahout implementation runs on top of Hadoop and provides batch machine learning processing on a MapReduce model. Since Apr.25th 2014, the community stops receiving new MapReduce algorithm implementations and the future Mahout implementation will be built on top of a Domain Specific Languages (DSL) for linear algebraic operations developed recently. Programs written in this DSL are automatically optimized and executed in parallel on Apache Spark [16]. However, no matter Apache Spark or Hadoop is used, the distributed processing platforms characteristic makes Mahout a batch based machine learning framework, which is
not optimized for stream processing.

More ML frameworks for data stream are also available but not as popular as those mentioned above. For instance, Vowpal Wabbit is another example of streaming ML framework based on the perceptron algorithm. Debellor is an open source stream-oriented platform for machine learning. Storm.pattern project is an effort to adapt existing ML model into Storm using Trident abstraction. Similar with Storm.pattern, Trident-ML project is also a real-time online ML library built on top of Storm.

Those ML frameworks are designed for generic purpose on building a collection of ML algorithms, not aiming at specific anomaly detection problems. However, such ML frameworks offer inspirations and make the anomaly detection problem easier to solve.
Chapter 3

System design

In order to detect anomalies on Spotify payment transactions, we have designed a distributed system on top of Storm to the best of our effort. The anomaly detection components are mainly three parts. Firstly we have developed an unsupervised learning module based on k-means clustering algorithm. In addition, we employ rule association algorithms to build a rule-extraction module by analyzing anomalies’ context as well as the output of clustering algorithm. Furthermore, with the generated rules, we utilize CEP module integrated with Storm to perform rule-based detection, for the purpose of reducing false positives from unsupervised learning. We have offered a modular approach in our system; different algorithms can be implemented on the related modules for future replacement without changing the system structure. In this chapter, we will present the general system structure, detailed components and module functionalities, respectively.

3.1 Architecture Overview

In order to detect anomalies in big data streams, our system is built on top of a distributed streaming processing platform, Storm, utilizing the its functionalities and features. Storm provides high level data abstractions such as spout and bolt, which makes it easy to handle the data streams. The general architecture of our system is composed of five main modules: aggregation module, unsupervised learning module, rule extraction module, rule-based CEP module and output module. We also provide a client application called PaymentMonitorBuilder that constructs a Storm topology and submits it into the Storm Nimbus for deployment. The main tasks of topology will be configured during the Storm preparation once the topology is submitted. The whole structure of the system is presented in Figure 3.1.

Here the topology is a graph where each node is a spout or a bolt. Edges are
indicating which bolt subscribes to which streams. Values must be serializable in the stream and each stream has its own id. Streams are also defined with a schema that names the fields in the tuple.

As we can see from the system structure, two paths of the data stream can be found after the aggregation module. One path goes to unsupervised learning module. By unsupervised learning, the data tuples in the stream are put into several clusters in order to find the suspected anomalies. The rule-extraction module utilizes the suspected anomalies to train the general anomaly’s rule for CEP. The other path with the same data stream goes directly to CEP module for rule-based detection. The reason we have designed such a structure is that although unsupervised learning can find anomalies individually, a lot of false positive exist in the unsupervised learning result. To reduce the false positives, a rule-extraction module and a rule-based CEP module are very necessary.

### 3.2 Input data stream

In Storm, the basic data abstraction is stream, which is an unbounded sequence of tuples. The entry point of a topology is a spout consuming tuples from an external source and emitting them into topology as the source of streams. We use Kafka to provide the source of streams and a spout to subscribe to the Kafka topics to pull the corresponding data tuples. The Kafka topics are similar with tweets’ hashtags.
in Twitter. The topics are log messages from various Spotify backend services published in Kafka with special names. The example is given in Figure 3.2.

![Figure 3.2: Input data stream.](image)

We also provide an offline log consumer as the source of streams. By reading every line from the log file continuously, it is possible to transform the offline log into a sequence of tuples, though it is bounded.

### 3.3 Aggregation module

Due to the different payment transaction service latency or API failures, the tuples of payment log data from Kafka are not strictly sorted. Therefore we build the aggregation module to make the raw data tuples formalized for unsupervised learning. We make an assumption the received data tuples from last five minutes are tolerable and defined as real-time, to be more accurate, nearly real-time. In reality, we have found that the payment tuple’s timestamp from Kafka shows transactions happened about half a minute ago.

A bolt is implemented in the aggregation module with filtering and parsing functions. The tuples containing useful data is collected by filtering the key words in the stream. We build a JSON (Appendix A) file to define the related attributes in the tuple and get the corresponding value from the tuples according to it. We also define a timeout value called timeoutMillis to aggregate the filtered tuples. For instance, the default timeout is 60 seconds, defined as DEFAULT_TIMEOUT_MILLIS. Later the new tuples are created every 60 seconds and labeled with different fields such as “timestamp”, “country”, “payment provider”, and “success”. By aggregating with different fields, the stream can be divided into several sub-streams. For instance, if the stream is aggregated by “country” and “payment provider”, the sub-stream may contain the number of payment transactions paid from Sweden and by Paypal in last 60 seconds. Next, the bolt will emit such value wrapped by those fields to unsupervised learning module. The detailed description is given in Figure 3.3.

We also implement aggregation bolt in a sliding window mechanism. Two parameters are set for such a mechanism, window size and granularity. Granularity
defines the time interval for aggregation and window size defines the period for bolt functionality. For instance, by default the granularity is 1 second, window size is 60 seconds and starting time is 0, the aggregation bolt will output the rate aggregation every 1 second. The output timestamp will be [0,1, ...,59], [1,2, ...,60], [2,3, ...,61], ... This mechanism is designed for CEP bolt in order to identify anomalies faster. In the other bolts, the default time granularity is 1 minute, therefore the unsupervised bolt firstly buffer the data from aggregation bolt and accumulate locally to be compatible with the time granularity 1 minute. The output of aggregation bolt is the aggregated rate result according to time granularity. For instance, 2014-02-27T00:00:01.003+00:00, 1, 1, 0, IE, netgiro, where the fields are timestamp, total rate, success rate, failure rate, country and payment provider, respectively.

As mentioned in Section 2.1.1, the contextual anomalies and collective anomalies can be transformed into point anomalies. For instance, the payment failure rates per second are very different, varying in a relatively large scale. It is hard to set standard criteria to identify such collective anomalies. However, if the failure rates are calculated per minute, they are steady and anomalies are easy to be picked out. By applying different time granularity in the aggregation module, such collective anomalies can be transformed into point anomalies. Therefore, we mainly focus on the point anomalies in our implementation.
3.4 Unsupervised learning module

Unsupervised learning module can be implemented by different clustering algorithms. We define ClusteringBolt as the implementation of the unsupervised learning module. Our approach is mainly built on k-means clustering algorithm for its speed and simplicity and DBscan clustering algorithm for efficiency.

The anomaly score 2.2.3 is imported for identifying the anomalies in k-means clustering algorithm. Both distances from the cluster centroid and cluster density are used to determine the anomalies. Furthermore, a sliding window mechanism in clustering is defined when clustering those values from aggregation module. That is, if the default window is 60 minutes, every minute the clustering bolt receives a tuple from aggregation bolt, it will run cluster algorithm on the tuple with the latest 59 tuples. If the instance in the tuple is far away from its cluster centroid or belonging to a small cluster, it will be treated as an anomaly and output to next module. When initializing, if the number of instances is less than the sliding window, the clustering bolt will not run clustering algorithm. On the other hand, when the number of instances is equal or higher than the sliding window, the oldest instance in the tuple will be automatically deleted to maintain the constant number of instances. We implement a data structure called BoundedList to achieve this goal. The algorithm is described in Figure 2.

---

**Algorithm 2** K-means based anomaly detection algorithm

**Input:**
- $k$: The number of clusters;
- $D$: A data set containing $n$ objects;
- $t$: the anomaly score to define anomalies

**Output:**
- A set of identified anomalies;

1: Arbitrarily choose $k$ objects from $D$ as the initial cluster centers;
2: while Cluster means change do
3: (Re) assign each object to the cluster to which the object is the most similar, based on the mean value of the objects in the cluster;
4: Calculate the mean value of the objects for each cluster to update their means;
5: end while;
6: Retrieve the average distance of all the objects from their centroid in each cluster;
7: Calculate the anomaly score, that is, every object’s distance from centroid divided by the average distance in this cluster;
8: If the anomaly score is higher than $t$, output this object.
We have noticed that several factors have an influential impact on the clustering result such as the anomaly density, the anomaly score threshold and the number of clusters. Therefore we make those arguments configurable and run experiments on those factors. The result is shown in Chapter 5.

**Algorithm 3 DBSCAN based anomaly detection algorithm**

**Input:**
- $D$: A data set containing $n$ objects;
- $\varepsilon$: a radius of an object which determines the its neighborhood;
- $MinPts$: the minimum number of points required to form a dense region

**Output:**
- A set of identified anomalies;

1. Cluster $C = 0$
2. for each unvisited point $p$ in data set $D$ do
3. mark $p$ as visited;
4. NeighborPts = all points within $p$’s $\varepsilon$-neighborhood (including $p$);
5. if $\text{sizeof}(\text{NeighborPts}) < MinPts$ then
6. mark $p$ as noise and output $p$;
7. else
8. $C = \text{nextcluster}$;
9. add $p$ to cluster $C$;
10. for each point $p’$ in NeighborPts do
11. if $p’$ is not visited then
12. mark $p’$ as visited;
13. NeighborPts’ = all points within $\varepsilon$-neighborhood of $p’$ (including $p’$);
14. if $\text{sizeof}(\text{NeighborPts}) \geq MinPts$ then
15. NeighborPts = NeighborPts joined with NeighborPts’;
16. end if
17. end if
18. if $p’$ is not yet member of any cluster then
19. add $p’$ to cluster $C$;
20. end if
21. end for
22. end if
23. end for

At the same time, we also use density-based clustering algorithm DBSCAN. We have studied the distribution of our data set, which fulfils Gaussian Distribution. In order to avoid the disadvantages of k-mean clustering algorithm, we decide to implement an alternative clustering algorithm DBSCAN to see the anomaly
3.5 Rule extraction module

By monitoring the anomaly trend, the rule extraction module utilizing frequent itemset mining algorithm can generate rules for CEP module to filter out those anomalous tuples which do not match the anomaly pattern. Consequently, the false positives can be reduced by such techniques. In our implementation, the estDec algorithm described in 2.3 is used for rule extraction module. The detailed algorithm is described in Algorithm 4 [14].

Before frequent itemset mining, the attributes are nominalized from the aggregation module in order to get an itemset. We nominalize the data tuples in several fields. For instance, we categorize field timestamp into 24 intervals according to its real hour. We also maintain the fields of country and payment and nominalize field rate in predefined intervals such as [0,10). After such nominalization, the frequent itemset mining algorithm calculates the each nominal tuple’s occurrence and maintain it in a tree structure. When new tuples comes, the algorithm will automatically add or prune the tree structure. The more same nominalized itemset comes, the more frequent this itemset is. As we introduced in 2.3.1, the estDec algorithm will output the top K frequent itemsets as the CEP rules. Since the input of rule extraction bolt is the output of clustering bolt, the different output will result in different frequent itemsets. When concept drift happens in data streams, that is, the anomaly patterns have changed over time, the rules can be updated from the new streams.

Some parameters used in the algorithm is introduced below. A dacay rate $d$ is the reducing rate of a weight for a fixed decay-unit which determines the chunk of information to be decayed together. $|D|_K$ represents the total number of transactions in the current data stream and the $T_k$ represents the transaction. The count of the itemset is denoted by $cnt$. The maximum error count of the itemset is denoted by $err$. The transaction identifier of the most recent transaction that contains the itemset is denoted by $MRtid$. $S_{prn}$ is defined as a threshold for pruning, and should be less than a minimum support $S_{min}$. In a data set $D$, the count of an n-itemset $e$ can be estimated by the individual counts of its subsets. Its maximum and minimum count are defined as $C^{max}(e)$ and $C^{min}(e)$, respectively.
$C^{upper}(e)$ is the upper bound of its actual count.

### 3.6 Rule-based CEP module

More than a traditional big-data analytic system, Storm is also an example of a CEP system. CEP systems are typically categorized as computation and detection oriented. By integrating the open-source CEP provider, Esper [7], with a Storm bolt, this CEP component can be used to identify meaningful events from a flood of events, and then take actions on those events in real time through user-defined algorithms. The generated rule example query from rule extraction bolt output can be “select * from PaymentEvt(timestamp = 0, country = ’FI’, payment = ’paypal’, rate in [0,10))”.

As we mentioned in 3.3, the default setting of the time granularity is 1 second and the sliding window size is 60 seconds. One of the same data streams after aggregation bolt goes directly to CEP module for rule-based detection. Thus the CEP will detect anomalies from this data stream according to the sliding window size since the anomalous rate is also generated by minutes. The advantage of the sliding window mechanism is that the CEP can detect anomaly every second, where the anomaly can be detected as soon as the possible when it matches the generated anomaly pattern. Because of this characteristic, our whole system can detect anomalies in real-time. As mentioned above, the rules are coming from the rule extraction module. The rates are changing over time so that the rules for anomaly detection can be dynamic. CEP can detect anomalies by adapting dynamic rule update.

### 3.7 Output module

In output module, several output bolts can be implemented for different destinations. The simplest way is to output the result in a file. The file content is very straightforward. Another way is to publish the result in Kafka or in a distributed memory cache system Memcached. Applications made by PHP and JavaScript can subscribe to the Kafka topic or read from Memcached to make the result visualized. For instance, the stream from aggregation bolt can go directly to the output bolt and visualize the rate trend. Another implementation of output bolt is to alert when anomaly is detected, such as sending the payment administrator an email or printing a message on private IRC channel.
Algorithm 4 the estDec algorithm

Input:
- $D$: A data stream;
- $d$: A given decay rate
- $ML$: A monitoring lattice;

Output:
A complete set of recent frequent itemsets $L_k$;

1: $ML = \emptyset$;
2: for each new transaction in $D$ do
3: read current transaction $T_k$
4: // Parameter updating phase
5: $|D|_k = |D|_{k-1} \times d + 1$
6: // Count updating phase
7: for all itemset $e$ s.t. $e \in (2^{T_k} - \{\phi\})$ and $e \in ML$ do
8: $cnt = cnt \times d^{(k-MRtid)} + 1$; $err = err \times d^{(k-MRtid)}$; $MRtid = k$
9: // Pruning
10: if $(cnt/|D|_k) < S_{pr} \text{ and } |e| > 1$ then
11: Eliminate $e$ and its child node from $ML$;
12: end if
13: end for
14: // Delayed-insertion phase
15: $T'_k = ItemFiltering(T_k)$;
16: for all itemset $e$ s.t. $e \in (2^{T'_k} - \{\phi\})$ and $e \not\in ML$ do
17: if $|e| = 1$ then
18: Insert $e$ into $ML$; $cnt = 1$; $err = 0$; $MRtid = k$
19: else
20: Estimate $C^{max}(e)$ and $C^{min}(e)$;
21: if $C^{max}(e) > C^{upper}(e)$ then
22: $C^{max}(e) = C^{upper}(e)$;
23: end if
24: if $C^{max}(e)/|D|_k \geq S_{ins}$ then
25: Insert $e$ into $ML$;
26: $cnt = C^{max}(e)$; $err = C^{max}(e) - C^{min}(e)$; $MRtid = k$
27: end if
28: end if
29: end for
30: // Frequent itemset selection phase
31: $L_k = \emptyset$;
32: for all itemset $e \in ML$ do
33: $cnt = cnt \times d^{(k-MRtid)}$; $err = err \times d^{(k-MRtid)}$; $MRtid = k$
34: if $(cnt/|D|_k) \geq S_{min}$ then
35: $L_k = L_k \cup \{e\}$;
36: end if
37: end for
38: end for
Chapter 4

Implementation

The entry point of our system is based on class PaymentMonitoring. It is an effort to provide a high level API that combines known and experimental techniques that can aid anomaly detection applications. There is support for both a real-time production environment but also local execution of anomaly detection pipelines for evaluation purposes. This thesis project is built on Apache Storm. The main source of stream is coming from Kafka. CEP bolt is built on Storm-Esper and the clustering algorithms are using Weka machine learning libraries. Frequent itemset mining bolt is using the estDec algorithm. Execution can be partitioned (sharded) into different component groups based on field hashing and it works in a stream fashion.

In addition to data mining algorithms and CEP operability a StatefulBolt abstraction has been implemented on-top of Storm to define all stateful PEs that are being used in a continuous anomaly detection pipeline that operates over:

- **Absolute Time Sliding Windows** (eg. when dealing with production logs that are being updated on the fly in Kafka): In this case each stateful PE invokes user defined operations over a sliding window when timeouts occur according to a local clock and the parameters set during the initialization of the execution.

- **External Time Sliding Windows** (eg. when having historical logs with explicit timestamps defined per event and read from a local file). In this case time progressed artificially based on the timestamps read from each tuple in the pipeline and each stateful operator invokes user defined functions when certain timeouts occur as in the first case over a sliding window. External time sliding windows are especially useful during the evaluation of an anomaly detection pipeline. WARNING: It should be carefully noted that historical events are ordered incrementally (per shard-group) in order to have a correct and consistent execution with external timestamps.
The main builder for anomaly detection Storm topologies is provided within PaymentMonitoring class. An typical Topology that uses clustering combined with frequent pattern mining and complex event processing is defined in Algorithm 5.

**Algorithm 5 Topology example**

1: String sampleFile = “~/anomalous-data/samplelog.txt”;
2: PaymentMonitoring.create().fromFile(sampleFile).toAggr()
   .ratesFor("country","payment").every(60000l)
   .toClustering().window(300).clusterCount(2).thresholdModifier(1.5)
   .toRuleExtraction().discretize(10).minSupport(0.5)
   .minimalFields("country","payment","timestamp")
   .toComplexEventProcessing().toOutput().fileWriter()
   .toFinalize().startDebugAndKillAfter(100000L);

This translates into several distributed processing elements (PEs) that route streams and form an execution DAG. The statements above for example yield the following PEs along with their stream definitions and subscriptions in Appendix B.

In total, we have 174 commits, 47 files and approximately 32k lines of code.
Chapter 5

Evaluation

5.1 Goals

The main goal of this thesis work is to test the accuracy rate by different methods of anomaly detection. We mainly focus on the unsupervised methods of anomaly detection accuracy rate and rule-based anomaly detection accuracy rate using rule extraction and CEP. We will also discuss the assumptions and limits in both anomaly detection methods as well as the evaluation result of our experiments.

5.2 Assumptions

Since the unsupervised methods of anomaly detection do not have training data set labelled normal and/or anomalous, the techniques in this category make the implicit assumption that normal objects follow a pattern far more frequently than anomalies. Therefore we make several assumptions below:

- We assume that the percentage of anomalies is far small in the whole data set comparing with that of normal data.

- We assume that the anomalies are the ones that are very different from others.

- We assume that the original data set does not contain any anomalies unless manually added. Because even a purely normal data set can be detected anomalies if according to different criteria.
5.3 Evaluation Matrix

We use accuracy and false positive rate (FPR) as the performance criteria based on the following metric shown in Table 5.1.

<table>
<thead>
<tr>
<th>Predicted Result</th>
<th>Actual Result</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Anomalous</td>
<td>Normal</td>
<td></td>
</tr>
<tr>
<td>Anomalous</td>
<td>True Positive (TP)</td>
<td>False Positive (FP)</td>
<td></td>
</tr>
<tr>
<td>Normal</td>
<td>False Negative (FN)</td>
<td>True Negative (TN)</td>
<td></td>
</tr>
</tbody>
</table>

The explanation is listed below:

- True Positive (TP) is a condition when a manually set anomaly is successfully detected by the anomaly detection system.
- True Negative (TN) is a condition when normal data is not detected as an anomaly. In other words the anomaly detection system does not treat it as an anomaly and makes the right choice.
- False Positive (FP) is an alert that indicates an anomaly is detected when in fact there is only normal data.
- False Negative (FN) is a failure of the anomaly detection system to detect an actual anomaly.

The accuracy and false positive rate (FPR) are measured using the following formula:

\[
Accuracy = \frac{TP + TN}{TP + TN + FP + FN}
\]

\[
FPR = \frac{FP}{TP + FP}
\]

In addition, we also provide sensitivity (a.k.a. recall) to show the percentage of detected anomalies from total anomalies:

\[
Sensitivity = \frac{TP}{TP + FN}
\]

5.4 Evaluation settings

At the time of writing this thesis, the Storm cluster in Spotify AB has 20 nodes: one node is used for Nimbus, UI and it also acts as a Supervisor; the other nineteen nodes are all used for Supervisors. We used Storm UI to monitor the status of our deployed topology in Storm cluster.
5.5 Rate anomaly scenarios

The straightforward way to evaluate the health of Spotify payment system is to detect anomalies on the payment transactions. Many reasons can lead to anomalous behaviors, for instance, the payment provider APIs are not responding, which may increase the transaction failure rates. Or due to the Internet traffic congestion, payments rates decreased dramatically from only one country. Those payment transaction behaviors are very suspicious. Therefore it is very important to detect anomalies on payment transaction rates in order to prevent business loss. The action should be taken as soon as possible when an anomaly happens.

We use the real payment transaction data of Feb. 27th 2014 as our basic experimental data. We firstly remove as much anomalies as possible by manually checking the data set and replace them with normal ones. After such preprocessing procedures, we can assume there is no anomaly in the data set. Next, we randomly put anomalies which are very different from neighbor data. For instance, if the data value is very high in a period of time, we randomly put several anomalies with low value, and vice versa. They are obviously anomalies in the data set. The detected sub-stream is aggregated by fields “country” and “payment provider” in the aggregation bolt.

5.6 Evaluation result

We first evaluate the unsupervised method to detect anomalies. We have chosen several main factors which may influence the accuracy, false positive rate (FPR) and sensitivity in k-means clustering algorithm. They are anomaly density, number of clusters and anomaly score. We also evaluate DBSCAN clustering algorithm by comparing with k-means. The experiment series are repeated several times and get the average of repetitions in order to reduce the experimental error.

5.6.1 K-means evaluation

5.6.1.1 Anomaly density

Anomaly density means the percentage of the anomalies in the total dataset. As mentioned above, the clustering method can only detect anomalies only if the anomaly percentage is small. The experiments are comparing the three rates with different anomaly density to verify this statement. In order to exclude the effect of other variants, the default setting of unsupervised method is used where the sliding window is 60, the cluster number is 2 and the anomaly score is 16. The experimental anomaly density is set to 0.67%, 1.32%, 2.04%, 3.94%, 6.71% and 13.34%, respectively.
In Figure 5.1, the accuracy variation maintains constant when the anomaly density is below 2%, after that, it drops when the density increases. When there are 13.34% anomalies in the dataset, the accuracy is only around 86%. However, with anomalies density increasing from 0.67% to 13.34%, the sensitivity decreases from 70% to 2.86%. That is to say, if anomaly density is high, only a few anomalies can be detected. The reason is related to clustering algorithm nature, which separates data according to their similarity. If too many anomalies exist, the anomalies are put into the same cluster. Instead, those normal data which are different from anomalies will be treated as “anomalies”. It is very interesting to see the FPR first decreases until 3.94% density but later increases when anomaly density is increasing. The FPR should also decrease like accuracy and sensitivity in our concept. However, too many anomalies in a dataset make it hard to identify the real anomalies by clustering algorithm, therefore the FPR increases after a certain threshold of anomaly density. In this scenario, the certain threshold is 3.94%.

5.6.1.2 Anomaly score

The experiments are comparing the three rates with different anomaly scores. In order to exclude the effect of other variants, the default setting of unsupervised method is that the sliding window is 60, the cluster number is 2. Since the anomaly density influences the accuracy rate from the last experiment, three sets of experiments are designed to explore the rates behavior under different levels
of anomaly density. The anomaly density is set to 0.67% (low), 3.94% (medium) and 13.34% (high), respectively. The experimental number of anomaly score is set to 2, 3, 4 . . . 16, respectively.

Figure 5.2: Accuracy variation with different anomaly scores.

Figure 5.3: FPR variation with different anomaly scores.
In Figure 5.2, it is very clear to see anomaly density has a much stronger influence on accuracy than anomaly score, but we still can see that as the anomaly score increases and approaches infinite, then the accuracy approaches to a certain level which is 100% in the low anomaly density (0.67%). It can be seen in Figure 5.3 that anomaly score has a strong effect on FPR. When anomaly score increases, the FPR is decreasing. Both high (13.34%) and low (0.67%) density could result in a high FPR while the medium (3.94%) density results in a relatively low FPR. It is easy to understand that as the anomaly score increases, fewer normal data is mislabeled as anomalous. In Figure 5.4, as the anomaly score increases, the sensitivity decreases, because some anomalies are also mislabeled if the score is too high at times. Also for high anomaly density, the rate is decreasing but nearly maintains constant because it has not detect many anomalies.

5.6.1.3 Number of clusters

Number of clusters means a set of clusters generated by the clustering algorithm in the output. Though [8] suggests A simple method is to set the number of clusters to about $\sqrt{\frac{n}{2}}$ for a data set of $n$ points. In expectation, each cluster has $\sqrt{2n}$ points. The experiments are comparing the three rates with different numbers of clusters. In order to exclude the effect of other variants, the default setting of unsupervised method is that the sliding window size is 60 and the anomaly score is 16. The experimental number of clusters is set to 2 to 6 and three different levels of anomaly density are set to 0.67%, 3.94% and 13.34%, respectively.
In Figure 5.5, 5.6 and 5.7, accuracy, FPR and sensitivity are heavily influenced by different levels of anomaly density, though the number of clusters has very slight influence on those rates. In other words, similar to the set of anomaly score experiments, the cluster number is not sensitive in this point anomaly problem. The reason is that our anomaly detection criteria are based on distance. No matter
how many clusters are created, both the distance from centroid and the average distance have changed due to the formation of new clusters. Thus, if the anomaly score remains the same, the accuracy and sensitivity will keep stable yet FPR will increase when cluster number increases.

5.6.2 DBSCAN evaluation

Since we have made the assumption the anomalies are the ones that are very different from others, we use the default value of $\varepsilon$ where $\varepsilon = 0.9$. In addition, the default value of MinPnt is 6 which means if an core object has less than 6 neighbors, it will be labelled as “noise” (anomaly). It is easy to understand that anomalies are the ones in minority of the data set, if not, the anomalies are not anomalies any more. So the we will use the default MinPnt. We also use different anomaly density to see the performance of DBSCAN.

Similar with k-means clustering, we also use anomaly density for this experiment. As mentioned above, the clustering method can only detect anomalies only if the anomaly percentage is small. The experimental anomaly density is set to 1.43%, 2.81%, 4.22%, 8.32% and 13.71%, respectively.

As we can see from the figure 5.8 and compare with the figure 5.1, the tendency of accuracy, FPR and sensitivity of DBSCAN is similar with that of k-means clustering algorithm with different anomaly density. However, DBSCAN has a higher sensitivity (88.55% at 2.81% anomaly density) and lower FPR (8.65% at 2.81% anomaly density) than k-means when the anomaly density is
5.6. E

**5.6. Evaluation Result**

- **Figure 5.8:** Rates variation with different anomaly density for DBSCAN.

- **Figure 5.9:** Rate variation (13.34%).

- **Figure 5.10:** Rate variation (3.94%).

Low (below 4.22% anomaly density). Moreover, the FPR of DBSCAN clustering algorithm remains low and constant when anomalies become denser while the FPR of k-means clustering algorithm increases when anomalies become denser. Both sensitivities decrease heavily with higher anomaly density, so we can see from results that clustering algorithms only perform good and acceptable with low anomaly density. Also we can see that DBSCAN performs better than k-means in such a low anomaly density scenario, which has relatively higher sensitivity and lower FPR but both of them have high accuracy.

### 5.6.3 Rule-based anomaly detection with CEP

In this section, we evaluate the rule-based anomaly detection with CEP. After extracting rules from clustering result, the CEP engine can detect anomalies by
applying those rules on new coming streams. Being generated from clustering result, the rules may be created for normal objects pattern if the anomaly density is very low. It will result in very high false positive rate and low sensitivity which is not what we want. The reason comes from the frequent item set natural: the rules can only be generated from most frequent item. Therefore, our rule-based anomaly detection with CEP is good for relatively high anomaly density scenario. We have done experiments under two anomaly density: 3.94% and 13.34%, and compare the result with DBSCAN.

As we can see from the figure 5.9 and 5.10, DBSCAN has a much better performance than CEP rule-based method in the relatively low anomaly density. DBSCAN method has higher accuracy, much lower FPR and higher sensitivity. However, in the relatively higher anomaly density scenario, rule-based CEP method has same level of accuracy with DBSCAN method, relatively higher FPR and much higher sensitivity. The reason behind this result is that even though the sensitivity of clustering is not very high, the ones being detected are true positives, which have a clear pattern on anomalies. By adapting such a pattern, CEP can detect anomalies more and quicker. Also the FPR rate is tolerable in this scenario though it is higher than that from clustering method. Therefore we can draw the conclusion that rule-based CEP anomaly detection performs better than clustering methods in the high anomaly density scenario.

5.6.4 Fast detection by using CEP

In the previous sections, we introduce clustering based anomaly detection which is nearly real-time because we aggregate the payment transactions every minute. Only after every minute, the anomalies can be detected. However, the transactions happen every second, if aggregation granularity is minute, it will delay the detection speed, making the real-time anomaly detection as nearly real time detection. In order to improve our system’s detection speed and make a truly real-time anomaly detection, we have used a sliding window mechanism with CEP which can provide the ability to detect anomalies in real time. We have made the aggregation granularity 1 second and sliding window 60 seconds. In figure 3.1, the clustering bolt collects such aggregated streams to aggregate every minute for compatibility with the algorithm design. In addition, the transactions number in sub-streams is not very large, it is better to have 1 minute aggregation for better anomaly detection performance. The CEP bolt which receives sub-streams directly from aggregation bolt applies this sliding window mechanism to detect anomaly in real time by the rules generated from rule-extraction bolt.

To prove our fast detection idea, we have designed an experiment by manually put anomalies in the data set. In table 5.2, we can see the first time of CEP detecting anomalies is the timestamp 2014-02-28T00:00:38.003+00:00 while
5.6. Evaluation Result

Table 5.2: First detection time: Clustering-based vs. Rule-based CEP

<table>
<thead>
<tr>
<th>Anomaly detection method</th>
<th>First detection time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clustering-based</td>
<td>2014-02-28T00:01:00.003+00:00</td>
</tr>
<tr>
<td>Rule-based CEP</td>
<td>2014-02-28T00:00:38.003+00:00</td>
</tr>
</tbody>
</table>

The first time of clustering based anomaly detection is the timestamp 2014-02-28T00:01:00.003+00:00. The rule-based CEP method has faster detection rate than clustering-based methods. The clustering bolt and rule-extraction bolt should have trained rules for CEP in the initial stage. For instance, our anomaly detection system spends the whole day transactions of 2014-02-27 to generate an 24-hourly rule for CEP and later on the rule extraction bolt will update rules by new streams.
Chapter 6

Conclusions

Anomaly detection has been an active research area for many years. Applying anomaly detection on a large scale of data is a big challenge because of the practical hardware constraint such as storage capacity limit and processing power limit. In addition, the traditional methods of anomaly detection focus on analyzing off-line data, building the data model through training sets, and applying the trained data model on newly arrived data to find anomalies. However, it is common to have a large scale of data and new data is coming continuously in the current world. Obviously, new anomalies patterns may exist. The old fashion is challenged by the new requirement which is to detect potential anomalies patterns on a large scale of continuous data automatically. With the development of distributed computing technology in recent years, distributed systems can be utilized to process a large scale of data by overcoming the hardware constraint. Furthermore, Distributed streaming platforms make it possible to deal with new paradigm stream processing. Therefore, it is very interesting to apply anomaly detection techniques on top of a distributed system to discover potential anomalies in streaming data and alert when detect anomalous data.

In this thesis, we proposed an anomaly detection system framework on top of a DSP platform, Storm. We used unsupervised machine learning methods, k-means and DBSCAN clustering algorithms, to detect suspected anomalies in the clustering module. In order to adapt to the concept drift, the suspected anomalies are sent to a rule extraction module to generate anomaly patterns. Frequent itemset mining algorithm estDec is used for get most frequent patterns. By applying these patterns on a CEP engine Esper, the new stream data can be detected for anomalies in real time.

We have evaluated both unsupervised learning method and CEP rule-based method on anomaly detection by experiments. The experiments indicate that it is feasible to detect anomalies by both clustering-based and CEP rule-based method. Different factors heavily influence unsupervised learning anomaly
detection performance. Comparison between the DBSCAN and CEP rule-based method shows that the latter one has a higher sensitivity than the other in heavy anomaly density scenario though there is trade-off with accuracy and false positive rate. We also show our system can detect anomaly in real time.

6.1 Future work

This work is only the initial attempt for anomaly detection based on distributed streaming platform. Though the evaluation result shows our system can find out anomalies efficiently and the new anomaly detection solution based on DSP platform is promising, the natural of detecting potential anomalies still needs much researching work and should be put into practice by different implementations. To continue this initial work, possible future research directions that could be very interesting to study and be carried out are summarized below:

1. Try different clustering algorithms such as OPTICS or other clustering algorithms for higher efficiency.

2. The rule extraction module can be replaced to statistical methods such as logistic regression for a more accurate data model.

3. To increase the accuracy of current anomaly detection solution, classification methods should be employed after carefully analysing the huge amount of received data.

4. In order to make the detection more general, the distributed anomaly detection system should support different types of data and involve more relevant attributes.

5. Add API support for other anomaly detection methods for wider usage.

6. Improve the system performance by researching the issues such as scalability, fault tolerance.
Bibliography


Appendix A

JSON file used in implementation

```json
[
    {
        "attrName": "timestamp",
        "fileIndx": 0,
        "attrType": "ordinal",
        "fieldType": "dt"
    },
    {
        "attrName": "country",
        "fileIndx": 1,
        "attrType": "nominal",
        "fieldType": "str"
    },
    {
        "attrName": "payment",
        "fileIndx": 2,
        "attrType": "nominal",
        "fieldType": "str"
    },
    {
        "attrName": "success",
        "fileIndx": 3,
        "attrType": "binary",
        "fieldType": "bool"
    },
    {
        "attrName": "subscription",
```
"fileIndx": 4,
"attrType": "nominal",
"fieldType": "str"
},

{
"attrName": "rate",
"attrType": "numeric",
"fieldType": "intg"
},

{
"attrName": "success_rate",
"attrType": "numeric",
"fieldType": "intg"
},

{
"attrName": "failure_rate",
"attrType": "numeric",
"fieldType": "intg"
}
Appendix B

Stream definitions and subscriptions
<table>
<thead>
<tr>
<th>Component Name</th>
<th>Component Type</th>
<th>Description</th>
<th>Input Stream(s)</th>
<th>Output Stream(s)</th>
<th>Grouping</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>FileLogSpout</td>
<td>A spout reading from a local file</td>
<td>-</td>
<td>default</td>
<td>-</td>
</tr>
<tr>
<td>aggregate</td>
<td>RateAggregateBolt</td>
<td>A bolt aggregating rates per grouping periodically</td>
<td>source:default</td>
<td>1)aggregates</td>
<td>fieldsGrouping</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2)paggrates</td>
<td></td>
</tr>
<tr>
<td>clustering</td>
<td>ClusterBolt</td>
<td>A bolt that models aggregates and outputs outliers periodically</td>
<td>aggregate:aggregates</td>
<td>default</td>
<td>fieldsGrouping</td>
</tr>
<tr>
<td>nominalBuilder</td>
<td>NominalBuilder</td>
<td>A bolt that gathers initial statistics and outputs nominalizers per numeric field</td>
<td>aggregates:pагgregates</td>
<td>nominalizers</td>
<td>fieldsGrouping</td>
</tr>
<tr>
<td>rule-extraction</td>
<td>PatternExtractor</td>
<td>A bolt that nominalizes outlier tuples and extracts CEP query statements</td>
<td>clustering:default</td>
<td>cepin</td>
<td>fieldsGrouping</td>
</tr>
<tr>
<td>esper</td>
<td>RCEsperBolt</td>
<td>A bolt that runs a CEP engine and configures it with queries, operating on aggregate events</td>
<td>1)rule-extraction:cepin</td>
<td>2)aggregates:pагgregates</td>
<td>fieldsGrouping</td>
</tr>
<tr>
<td>outputAggr</td>
<td>PayLogFileWriter</td>
<td>A bolt that logs intermediate emitted events from target components</td>
<td>any</td>
<td>-</td>
<td>fieldsGrouping</td>
</tr>
</tbody>
</table>