rijksuniversiteit groningen

Web-scale outlier detection

A THESIS PRESENTED

BY

F. T. DRIESPRONG

TO

THE FACULTY OF MATHEMATICS AND NATURAL SCIENCES

FOR THE DEGREE OF

MASTER OF SCIENCE

IN THE SUBJECT OF

COMPUTING SCIENCE

UNIVERSITY OF GRONINGEN

GRONINGEN, NETHERLANDS

DECEMBER 2015

© 2015 - F. T. Driesprong
All rights reserved.

Thesis advisor: dr. A. Lazovik F. T. Driesprong

Web-scale outlier detection

Abstract

The growth of information in today's society is clearly exponential. To process these staggering amounts of data, the classical approaches are not up to the task. Instead we need highly parallel software running on tens, hundreds, or even thousands of servers to process the data. This research presents an introduction into outlier detection and its application. An outlier is one or multiple observations that deviates quantitatively from the majority and may be the subject of further investigation. After comparing different approaches to outlier detection, a scalable implementation of the unsupervised Stochastic Outlier Selection algorithm is given. The Docker-based microservice architecture allows dynamically scaling according to the current needs. The application stack consists of Apache Spark as the computational engine, Apache Kafka as data store and Apache Zookeeper to ensure high reliability. Based on this we empirically observe the quadratic time complexity of the algorithm as expected. We explore the importance of matching the number of worker nodes based on the underlying hardware. Finally the effect of the distributed data-shuffles is discussed which is sometimes necessary for synchronizing data between the different worker nodes.

Contents

List of Figures

Acknowledgments

I would like express my appreciation to Dr. Alexander Lazovik, my first research supervisor, for his patient guidance, substantive feedback and useful critiques of this research work. I would also like to thank Prof. dr. Alexandru C. Telea for his advice and support in the process of writing. My special thanks are extended to Quintor for providing a very pleasant place to work and valuable insights in the industry. Finally, I wish to thank my parents for their support and encouragement throughout my study.

Computer science is no more about computers than astronomy is about telescopes.

Edsger Dijkstra

1 Introduction

As said in the 17th century: "Whoever knows the ways of Nature will more easily notice her deviations; and, on the other hand, whoever knows her deviations will more accurately describe her ways" [87]. This statement illustrates how the uncovering of outliers is a concept that has interested people throughout history. The aim of this thesis is to explore modern web-scale techniques for detecting outliers in large and complex datasets produced in today's data-driven society.

First, Section 1.1 explains why this thesis subject is important and give an introduction to the topic of outlier detection. Second, Section 1.2 introduces the research questions and scope. The research is conducted at Quintor, introduced at Section 1.3.

1.1 Motivation

In today's world we have accumulated a truly staggering amount of data. Since the 1980s the total amount of data has doubled every 40 months [50]. This overwhelming growth of about 28% per annum is clearly exponential [64]. This increase of data is caused by a variety of factors, among which the increasing interconnectivity of machines through the use of (mobile) internet. Apart from these technical innovations, movements such as the Internet of Things (IoT) and social media have caused a continuous increase in the volume and detail of data [54].

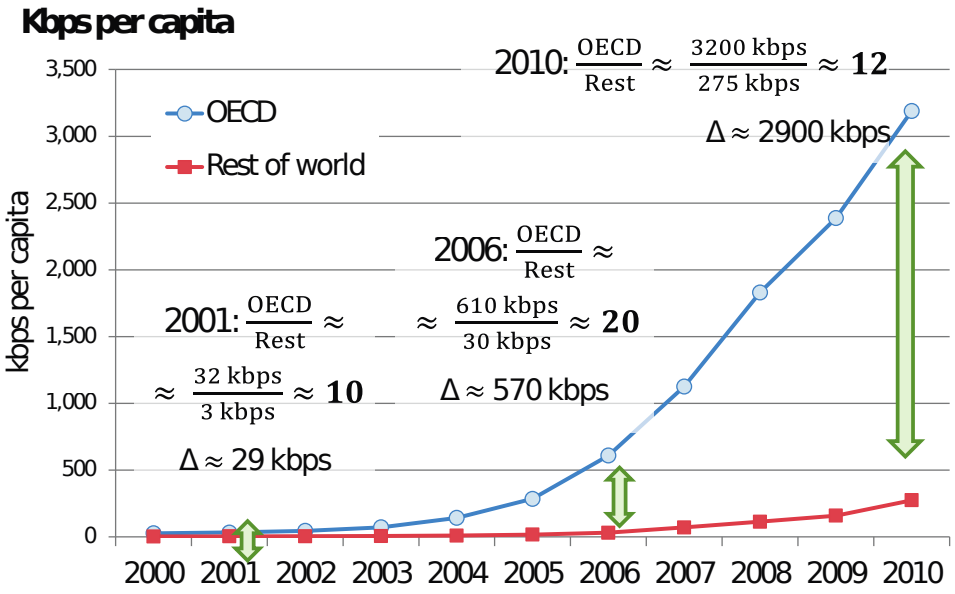

Figure 1.1.1: Kilobytes per second optimally compressed telecom capacity per capita [49]. Telecom includes fixed and mobile, telephony and internet.

Figure 1.1.1 illustrates the growth of telecom capacity over the years per capita. A distinction has been made by The Organization for Economic Co-operation and Development (OECD), an international economic organization of 34 countries founded in 1961 in order to stimulate economic progress and world trade. OECD countries are in general well-developed countries of which the GDP per capita is in the highest quartile. The figure shows that while in 2001 the average inhabitant

of the developed OECD countries had an installed telecommunication capacity of 32kbps, in 2010 the access capacity of an individual inhabitant had already multiplied by a factor of 100 to 3200kbps [49].

Besides the volume, data is shifting from a static to a continuously changing nature [46]. The handling of such large amounts of dynamic data is known as 'big data', a term which comprises datasets whose sizes are far beyond the ability of commonly used software tools to capture, curate, manage, and process within a tolerable time frame [88]. Many definitions of big data exist. However, we follow the definition [47]: 'Big data is a set of techniques and technologies that require new forms of integration to uncover large hidden values from large datasets that are diverse, complex and of a massive scale'. To obtain insights from large amounts of data, different tooling is required. For example, most relational database management systems and desktop statistics are not up to the task. Instead, we need 'highly parallel software running on tens, hundreds, or even thousands of servers' to process the data [58].

The process of extracting valuable knowledge from the large and complex amounts data is known as data mining, which is defined as 'the non-trivial extraction of implicit, formerly unidentified and potentially constructive information from data in databases' [66, 107]. The goal of data mining is to mine the so-called golden nuggets from the mountain of data [108]. The golden nuggets are the interesting observations which provide knowledge about the data. Outlier detection, a subdivision of Knowledge Discovery and Data mining (KDD) differs in the sense that it detects data which shows behavior that differs from the rest of the data, which is potentially a golden nugget [23].

Outlier detection has been studied by the statistical community as early as the 19th century to highlight noisy data from scientific datasets [74]. For example, for normally distributed data, the observations which lie outside three times the standard deviation are considered outliers [2]. Another commonly used method is based on a variety of statistical models, where a wide range of tests is used to find the observations which does not fit the model [11]. These parametric models are not always suitable for general purpose outlier detection, as it is not always

clear which distribution the data follows. Recent popular outlier detection algorithms sample the density of an observation by computing the average distance the k-nearest neighbours. This density is then compared with the density of neighbouring observations, based on the differences in density it can be determined if the observation is an outlier or not. Because of its popularity the algorithm is also discussed in Section 4.1.

The term 'anomaly' or 'outlier' is an ambiguous term, therefore a definition is in place. First, an outlier, sometimes referred to as an anomaly, exception, novelty, fault or error is defined in literature as:

- "An observation which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism." [31]
- "An outlying observation, or 'outlier,' is one that appears to deviate markedly from other members of the observation in which it occurs." [40]
- "An observation (or subset of observations) which appears to be inconsistent with the remainder of that set of data." [11]
- "An outlier is an observation that deviates quantitatively from the majority of the observations, according to an outlier-selection algorithm." [59]

We follow the last definition, as we believe it is of importance to prove quantitatively that the observation is different from the majority of the set. Outliers may be 'surprising veridical data', as belonging to class $\mathcal{A}$, but actually situated inside class $\mathcal{B}$, so the true classification of the observation is surprising to the observer [62]. Outlier detection analysis in big data may lead to new discoveries in databases, but an outlier can also be noise, a faulty sensor, for example. Finding aberrant or disturbing observations in large collections is valuable and a subject for further investigation. Outlier detection has a variety of applications, among which:

Log analysis Applying outlier-detection on log files helps to uncover issues with the hard- or software of a server. Applying it to network or router logs can unveil possible hacking attempts.

Financial transactions In recent years outlier detection has drawn considerable attention in research within the financial world [65]. By the use of outlier detection on credit-card transactions it is possible to indicate credit card fraud or identity theft [6].

Sensor monitoring As sensors are becoming cheaper and more ubiquitous in our world, outlier detection can be applied to monitor data streams and identify faulty sensors [37].

Noise removal Outlier detection is often used for the preprocessing of data in order to clear impurities and discard mislabeled instances [20]. For example, before training a supervised model, the outliers are removed from the training dataset, thus removing possible noisy observations or mislabeled data [92].

Novelty detection Novelty detection is related to outlier detection as a novelty is most likely different from the known data. An example application is the detection of new topics of discussion on social media [1, 77, 101].

Quality control "Outlier detection is a critical task in many safety critical environments as the outlier indicates abnormal running conditions from which significant performance degradation may well result, such as an aircraft engine rotation defect or a flow problem in a pipeline. An outlier can denote an anomalous observation in an image such as a land mine. An outlier may pinpoint an intruder inside a system with malicious intentions so rapid detection is essential. Outlier detection can detect a fault on a factory production line by constantly monitoring specific features of the products and comparing the real-time data with either the features of normal products or those for faults" [93].

Introducing outlier detection into the realm of web-scale computing introduces requirements on the architecture, data storage and deployment of the components. The data center has become a major computing platform, powering not only internet services, but also a growing number of scientific and enterprise applications

[105]. Deploying outlier detection systems on a cloud architecture allows the system to scale its capacity according to the current needs. For example, when an outlier detection algorithm processes to a stream of financial transactions, it needs to scale up at Christmas due the increase of workload, and can be scaled down every Sunday freeing up resources which can be used by other services such as reporting.

It is difficult to determine whether an observations is a true anomaly, but when it is marked as an outlier by the algorithm it is probably worth further investigation by a domain expert. Outlier detection is not new, but has not yet been widely implemented in a scalable architecture. From Quintor's perspective, a shift in requirements is becoming more evident each day. As the data changes faster, classical data warehousing or data mining is not up to the job and a shift has to made to the realm of real-time processing [8]. By providing real-time tactical support, one is able to drive actions that react to events as they occur. This requires shifting from batch-processing jobs to real-time processing. By real-time processing we refer to soft real-time whereby the added value of the results produced by a task decreases over time after the deadline expires [86]. The deadline is considered the expiration of the timespan in which the result is expected.

1.2 Research questions

The main focus of the thesis is outlier detection within a scalable architectures, as most standard outlier-detection algorithms do not consider an implementation on a web-scale level. Therefore the goal of this thesis is to provide a scalable implementation which can be used in the industry to apply outlier detection to very large datasets.

The research question and its subquestions are be supported, referred to and addressed throughout the thesis. By breaking the main question down into several sub-questions, the different concerns can be isolated and addressed separately.

Research question 1 *How to scale a general purpose outlier detection algorithms to a web-scale level.*

Outlier detection is part of the domain of Knowledge Discovery and Data mining (KDD). Most implementations are not developed with scalability in mind, but solely on the method to uncover outliers from the set of observations. Therefore the first question is to determine which algorithms can potentially benefit from a parallel implementation.

Research sub-question 1 *Which general-purpose outlier detection algorithms can potentially be scaled out.*

Scaling an algorithm out, sometimes referred as scaling horizontally, is distributing the workload of the algorithm across different machines which acts as a single logical unit. This enables to allocate more resources to a single job as more machines can be added if needed. The different methods of outlier detection need to be examined in order to determine whether it can be converted into a so-called 'embarrassingly parallel workload'. This requires the algorithm to be able to separate the problem into a number of parallel tasks which can be scheduled on a cluster of distributed machines.

Research sub-question 2 *Which computational engines are appropriate for outlier detection in a big-data setting.*

In recent years, a variety of computational engines have been developed for distributed applications, mostly based on the MapReduce model [28], such as Apache Hadoop[1], Apache Spark [104]. Other frameworks, such as Apache Mahout[2], provide additional algorithms which run on top of computational engines. An overview of the available computational engines and their characteristics will be presented.

Research sub-question 3 *How to adapt the algorithm to work with streams of data, rather than a static data set.*

[1]Apache Hadoop http://wiki.apache.org/hadoop/HadoopMapReduce
[2]Apache Mahout http://mahout.apache.org/

Rather than mining outliers from a static set of data, most of the time, data comes as a possible infinite stream of data. Example streams are sensor data or transactions which produce data over time. Both the algorithm and architecture need to cope with this way of processing data.

1.3 Quintor

Quintor is a leading player in the fields of Agile software development, enterprise Java / .Net technology and mobile development. Since its foundation in the year 2005, Quintor has been growing steadily. From their locations in Groningen and Amersfoort they provide support to their customers in facing the challenges that large-scale enterprise projects entail. Quintor has a software factory at its disposal, from where in-house projects are carried out.

To enterprise customers, Quintor provides services in the field of software development processes (Agile/Scrum), information analysis, software integration processes, automated testing, software development and enterprise architecture. Quintor provides full Agile development teams consisting of analysts, architects and developers.

From Quintors' perspective, the area of big-data is experiencing a tremendous growth and companies are generating more data in terms of volume and complexity every day. By processing these large volumes of data in a distributed fashion enables to extract the interesting parts. Based on the interesting observations companies obtain knowledge from their data to gain strategic insights.

There are only two hard things in Computer Science: cache invalidation, naming things and off-by-1 errors.

Phil Karlton

2 Background

This chapter provides insights from literature on both outlier detection in Section 2.1 and web-scale techniques in Section 2.2.

2.1 Outlier detection algorithms

Outlier detection algorithms aim to automatically identify valuable or disturbing observations in large collections of data. First, we identify the different classes of machine learning algorithms [34], which also applies for outlier detection algorithms:

Unsupervised algorithms try to find a hidden structure or pattern within a set of unlabeled observations. As illustrated in Figure 2.1.1, the observations given as the input of the algorithm are unlabeled, so no assumptions can be made. For ex-

ample, k-means clustering, which is a classical example of unsupervised clustering, can be applied on a set of observations to extracts k cluster heads. The set of observations is reduced to a set of cluster heads which can be reviewed or used as input for the next algorithm. In the case of clustering, it is not clear how many clusters are hidden in the observations. This makes it often hard to evaluate performance of unsupervised algorithms as in the case of the example the error function depends of the value of k. For each problem and dataset it is important to select the appropriate distance measure as the results are as good as the distance measure can differentiate the observations.

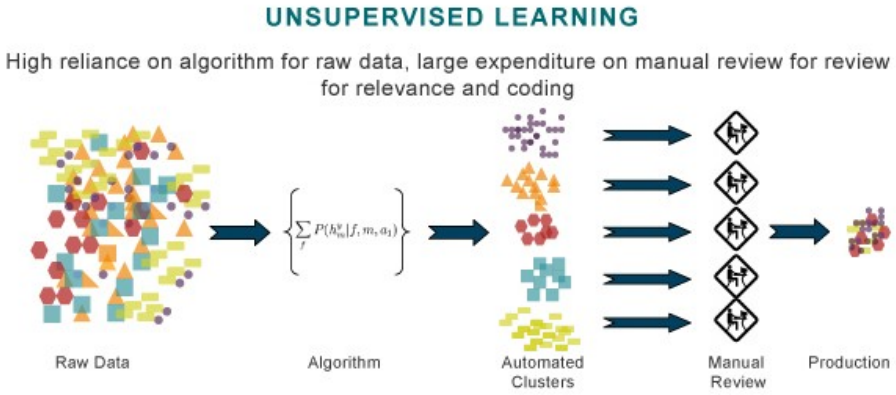

Figure 2.1.1: Unsupervised learning

Supervised is the machine learning task of inferring a function from labeled training data [78]. As depicted in Figure 2.1.2, supervised methods require as input set observations accompanied by a label which indicates to which class each observation belongs (for example, a label which is either legitimate or fraudulent) in order to assign the observation to a specific class. The training-set is used by the algorithm to learn how to separate the different classes. Based on the input the algorithm will learn a function to distinguish the different classes. Once the model is trained, it can be used to classify to an observation of which the label is

not known yet. A disadvantage of supervised learning with respect to outlier detection is that large amount of labeled data is needed to train the model, which is not always available.

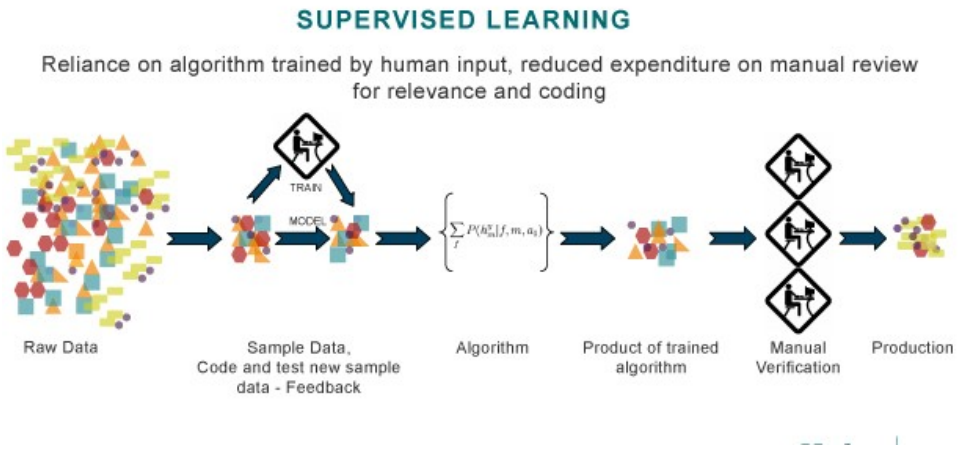

Figure 2.1.2: Supervised learning

Semi-supervised learning is a class between unsupervised and supervised learning. The algorithms consists of a supervised algorithm that makes use of typically a small amount of labeled data with a large amount of unlabeled data. First the model is trained using the labeled data, once done the model is further trained by bootstrapping the unlabeled data. The unlabeled data is presented to the algorithm and subsequently the algorithm is trained based on the prediction of the algorithm.

The next step in the process is to detect outliers based on the stream of observations. The outliers might tell something about the transaction in order to determine whether it is legit or possibly fraudulent. This information can be used when a financial transaction is made. Within a short amount of time it needs to be determined whether the transaction is trustworthy, if not the transaction may be canceled real time.

The goal of outlier detection is to identify the observations that, for some reason, do not fit well within the remainder of the model. A commonly used rule

of thumb is that observations deviating more than three times the standard deviation from the distribution are considered outliers [2]. Obviously this is not a very sophisticated method as it takes a global decision which is not locally aware. Mostly, such a global outlier model leads to a binary decision of whether or not a given observation is an outlier. Local methods typically compute unbound scores of 'outlierness'. These scores differ per algorithm in scale, range and meaning [38], but they can be used to discriminate different levels of outliers or to sort them and separate the top k-observations.

Within the literature, different classes of outlier detection algorithms exists:

Statistical based assumes that the given dataset has a distribution model which can be approximated using a function [43]. Outliers are those observations that satisfy a discordancy test in relation to the hypothesized distribution [11].

Distance based unifies the statistical distribution approach [67] by assuming that an observation is an inlier when the distance of the k-nearest observations is smaller than δ [68]. An updated definition which does not require the distance δ, but introduces n which defines whether point p is an outlier if $n-1$ other observations in the dataset have a higher value for D^k [81]. Where $D^k(p)$ is the sum of the distances to the nearest k observations with respect to p.

Density based algorithms identify an observation as an outlier if the neighbourhood of a given observation has a significantly lower density with respect to the density of the neighbouring observations [18, 19].

Besides the above-mentioned classes, there are many domain-specific outlier detection algorithms, for example for tumor detection within MRI data [89] or algorithms that detect interesting observations within engineering data [33].

Density-based approaches are sometimes seen as a variant of the distance-based approach as they share characteristics [71]. Distance based-algorithms consider an observation an outlier when there are fewer than k other observations within

δ distance [69, 70]. Another definition is the top n-observations which have the highest average distance to k nearest neighbours [7, 32]. Density-based algorithms take this one step further as they compute from the k-nearest observations their average distance to their k-nearest observations and compare them with their own distance to the k-nearest observations [84].

With respect to all available algorithm in literature, a subsection is presented in Table 2.1.1 based on the restrictions:

- The algorithm must be general-purpose and not only applicable within a specific domain.
- Many classical statistical algorithms, which are limited to one or only a few dimensions [53], have been left out as they are not applicable anymore.
- The observations that are the input data of the algorithm consist of a m-dimensional continues feature vector $\mathbf{x} = [x_1, \ldots, x_m] \in \mathbb{R}^m$.

Algorithm	**Type**	**Year**
HilOut [7]	Distance-based	2002
Local Outlier Factor (LOF) [4, 18]	Density-based	2000
Fast-MCD [82]	Statistical-based	1999
Blocked Adaptive Computationally Efficient Outlier Nominators [17]	Statistical-based	2000
Local Distance-based Outlier Factor (LDOF) [109]	Distance-based	2009
INFLuenced Outlierness (INFLO) [61]	Density-based	2006
No-name [81]	Distance-based	2000
No-name [12]	Distance-Based	2011
Connectivity-Based-Outlier-Factor (COF) [90]	Density-Based	2002
Local Outlier Probabilities (LoOP) [72]	Density-based	2009
Local Correlation Integral (LOCI) [80]	Distance-based	2003
Angle-Based Outlier Detection (ABOD) [71]	Angle-based	2008
Stochastic Outlier Selection (SOS) [60]	Distance-based	2012
Simplified Local Outlier Detection (Simplified-LOF) [84]	Density-based	2014

Table 2.1.1: An overview of outlier detection algorithms which match the above mentioned criteria.

All of the distance based and density based algorithms stated in Table 2.1.1 work with Euclidean distance, which is the most popular distance measure for continuous features. For discrete features, the Euclidean distance could be replaced by the Hamming distance. Also, other asymmetrical distance measures can be used.

A disadvantage of statistical-based approaches is that they are parametric since they try to model the data to a given distribution. The majority of the distance-based methods have a computational complexity of $\mathcal{O}(n^2)$ as the pair-wire distance between all observations is required which effectively yields an n by n distance matrix. This makes it difficult to apply the algorithm to very large datasets as the execution time grows quadratic which is not feasable for large datasets. A way to reduce the computational time is by using spatial indexing structures such as the KD-tree [13], R-tree [42], X-tree [14] or another variation. The problem using such optimized data structures is the difficulty to distribute the data structure across multiple machines.

Unfortunately the 'curse of dimensionality' also applies to ε-range queries and k-nearest neighbour search [76]. The effect of the dimensionality manifests itself as the number of dimensions increases and the distance to the nearest data point approaches the distance to the farthest observation [15]. When taking this to an extreme, as in Equation 2.1, where dist_{max} is the maximum distance to origin, and dist_{min} the minimum. When using an infinite number of vector, the distance between the vectors will approach zero [51]. The distance between them becomes less meaningful as the dimensionality increases and the difference between the nearest and the farthest point converges to zero [5, 16, 51].

$$\lim_{d \to \infty} \frac{\text{dist}_{\text{max}} - \text{dist}_{\text{min}}}{\text{dist}_{\text{min}}} \to 0 \tag{2.1}$$

Higher dimensionality not only hinders in discriminating the distance between the different observations, it also makes the outliers less intuitive to understand. For distance-based outlier algorithms, there are ways to evaluate the validity of the identified outliers which helps to improve the understanding of the data [69]. As the number of dimensions grows, this process becomes difficult and impossible in

extremes.

The use of high-dimensional data depends on the context and cannot be generalized. High-dimensional data can improve accuracy when all the dimensions are relevant and the noise is tolerable.

Another option is to reduce the number of dimensions, which means converting data of high dimensionality into data of much lower dimensionality such that each of the lower dimensions convey much information. Typical techniques that are used are Principal Component Analysis and Factor analysis [35]. There are more sophisticated techniques which focus on removing the noisy features which do not add any value to the result or even introduce noise [21], but this is outside the scope of the thesis.

2.2 Web-scale computing

In order to allow a system to scale, which is required to cope with the increasing workload, as well as to be able to process large amounts of data which does not fit on a single machine, web-scale technology is adopted. This typically involves the ability to seamlessly provision and add resources to the distributed computing environment. Web-scale is often associated with the infrastructure required to run large distributed applications such as Facebook, Google, Twitter, LinkedIn, etc.

Within literature, different definitions of cloud-computing have been proposed [83], there is diversity here, as cloud-computing does not comprise a new technology, but rather a new model that brings together a set of existing technologies in order to develop and execute applications in a way that differs from the traditional approach [110]. Web-scale architecture is an active field of research [25]. Web-scale applications rely on cloud computing which is one of the most significant shifts in modern IT for enterprise applications and has become a powerful architecture to perform large-scale and complex computing. Within literature, several definitions of cloud computing exists [3]. Clouds are used for different purposes and have numerous application areas. We define a cloud as; 'a large pool of easily accessible virtualized resources, which can be dynamically reconfigured to adjust

to a variable load, allowing for optimum resource utilization' [96].

On top of the cloud-computing environment big-data techniques are used. The world of big-data consists of a wide range of tools and techniques, which address and solve different problems. A mapping of the different components is given in Figure 2.2.1. Our focus is on the data-processing as our goal is to efficiently distribute the outlier detection algorithm onto a cluster of working nodes.

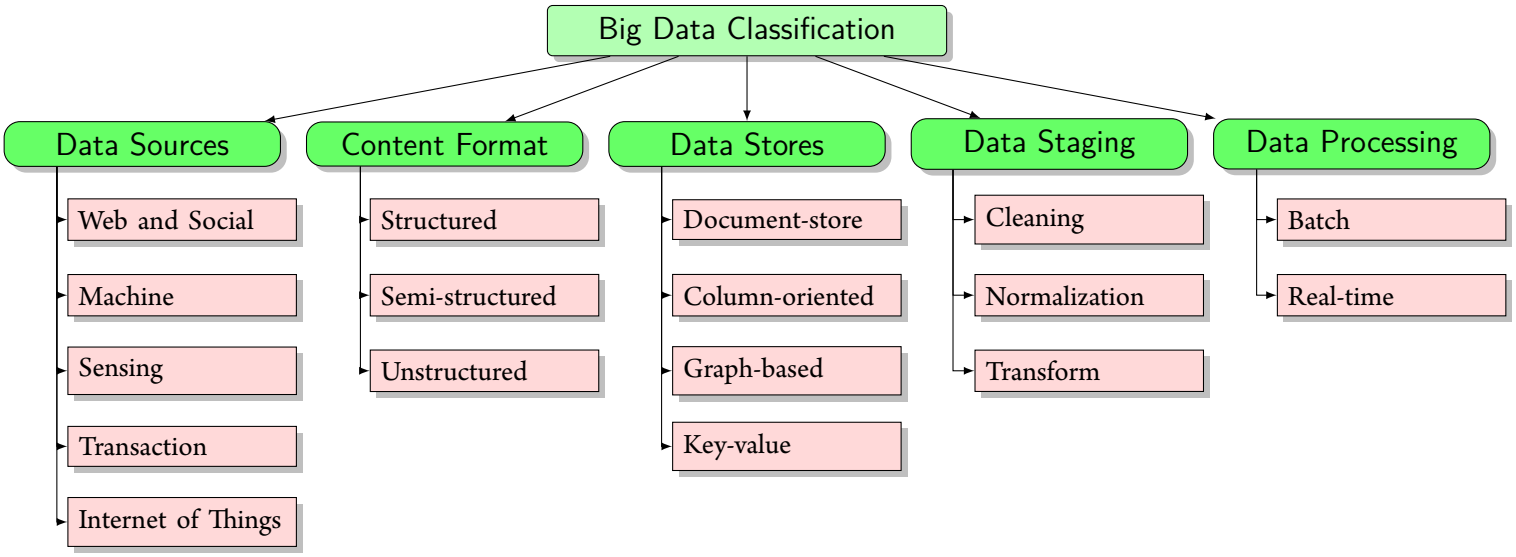

Figure 2.2.1: Overview Big-Data landscape [47]

It started in 2008 when Google introduced the MapReduce computing model which provides a model for processing large datasets in a parallel and distributed fashion on a cluster of machines [28]. Google used it to scale their PageRank algorithm to serve personalized results to the users of their search engine [9]. The MapReduce model is a simple yet powerful model for parallelizing data processing.

Subsequently in 2007 Microsoft launched a data-processing system to write efficient parallel and distributed applications more easily under the codename Dryad [57]. DryadLINQ provides a set of language extensions that enable a new programming model for large scale distributed computing. A DryadLINQ program is a sequential program composed of LINQ expressions performing arbitrary side-

effect-free transformations on datasets [103]. In November 2011 active development on Dryad had been discontinued, and Microsoft shifted their focus to the Apache Hadoop project [91].

The Apache Hadoop project is an implementation of the MapReduce model. It is the open-source implementation primarily developed by Yahoo, where it runs jobs that produce hundreds of terabytes of data on at least 10,000 cores [79]. Since then it has been adopted by a large variety of institutes and companies in educational or production uses, among which Facebook, Last.FM and IBM[1].

Although the name MapReduce originally referred to the proprietary Google technology, over the years it became the general term for the way of doing large scale computations. The open-source implementation that has support for distributed shuffles is part of Apache Hadoop[2]. A MapReduce job consists of three phases, namely Map, Combiner and Reduce [28]:

Map In the map phase operations on every individual record in the dataset can be performed. This phase is commonly used to transform fields, apply filters or join and grouping operations. There is no requirement that for every input record there should be one output record.

Combine For efficiency and optimization purposes it sometimes makes sense to supply a combiner class to perform a reduce-type function. If a combiner is used then the map key-value pairs are not immediately written to the output. Instead they are collected in lists, one list per each key value. When a certain number of key-value pairs have been written, this buffer is flushed by passing all the values of each key to the combiner method and outputting the key-value pairs of the combine operation as if they were created by the original map operation.

Reduce Before the reduce task it might be the case that distributed data needs to be copied to the local machine. When this is done, each key with its corresponding values is passed to the reduce operation.

[1]Hadoop: PoweredBy https://wiki.apache.org/hadoop/PoweredBy
[2]Apache Hadoop http://hadoop.apache.org/

First, input data is divided into parts and then passed to the mapper which executes in parallel. The result is partitioned by key and locally sorted. Results of the mapper-data with the same key are send to the same reducer and consolidated there. The merge sort happens at the reducer, so all keys arriving at the same reducer are sorted.

There is a variety of open-source frameworks based or inspired on the MapReduce model, each with their own characteristics, which are commonly used for big-data processing:

Apache Hadoop is the open-source implementation of the propriety MapReduce model. The Apache Hadoop consists of four modules: first, the Hadoop MapReduce framework, which consists of a YARN-based system for the parallel processing of large datasets. Second, the Hadoop Distributed File System (HDFS) in a distributed user-level file system which focuses on portability across heterogeneous hardware and software platforms [85] inspired by the Google File System [39]. Third, Hadoop YARN, which stands for Yet Another Resource Negotiator and is a framework for job scheduling and cluster resource management [97] and, last, Hadoop Common which provides the services for supporting the Hadoop modules.

Apache Spark is the implementation of the concept of the Resilient Distributed Datasets (RDDs) developed at UC Berkeley [106]. RDDs are fault-tolerant, parallel data structures that persist intermediate results in memory and enable the developer to explicitly control the partitioning in order to optimize data locality. Manipulating RDDs can be done using filters, actions and transformations by a rich set of primitives. The concept of RDD is inspired by MapReduce and Dryad.

MapReduce is deficient in iterative jobs because the data is loaded from disk on each iteration and interactive analysis, and significant latency occurs because all the data has to be read from the distributed file system [104]. This is where Apache Spark steps in by storing intermediate results in-memory.

Hadoop provides fault-tolerance by using the underlying HDFS which replicates the data over different nodes. Spark does not store the transformed information between each step, but when a block of data gets lost, the original data is loaded and Spark reapplies all the transformations, although it is possible to explicit save the state by enforcing a checkpoint. By the use of this strategy Spark's in-memory primitives provide performance up to 100 times faster than Hadoop for certain applications [102].

Software is a great combination between artistry and engineering.

Bill Gates

3 Architecture

This chapter describes the architecture on which the software is built and its influence on the results. Figure 3.0.1 illustrates the architecture using course-grained blocks. The arrows illustrate the dataflow through the system.

The emphasis of the architecture is on scalability. This means that it is simple to increase or decrease the number of worker nodes across a number of different physical machines as the machines are provisioned automatically, tending to implement or evolve to an 'elastic architecture' [22], which autonomously adapts its capacity to a varying workload over time [48], although the number of nodes is configured systematically to determine its performance for a set number of nodes. The underlying provisioning of the resources is done by Docker as described in section 3.1. The input data on which the algorithm performs the computations is kept on an Apache Kafka messaging system, as described in section 3.3. Finally,

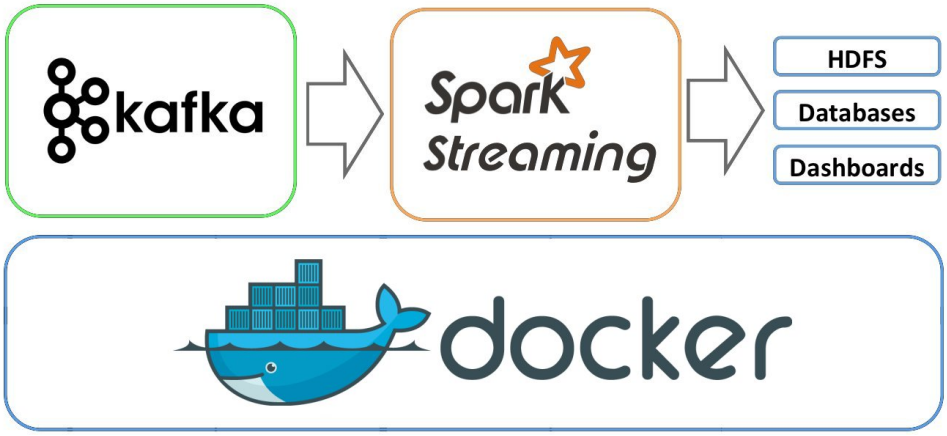

Figure 3.0.1: Architecture of the system

the computational framework itself, built upon Apache Spark, is discussed in Section 3.2.

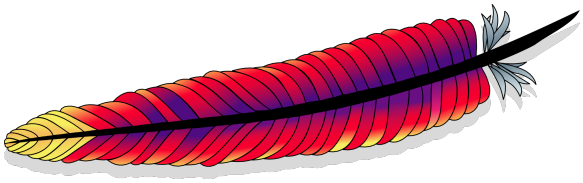

Figure 3.0.2: Logo Apache Software Foundation

The majority of the used software on which the architecture is built is part of the Apache Software Foundation[1], which is a decentralized community of developers across the world. The software produced is distributed under the terms of the Apache License and is therefore free and open source. The Apache projects are characterized by a collaborative, consensus-based development process and an open and pragmatic software license.

[1]Apache Software Foundation http://www.apache.org/

3.1 Docker

Docker is an open-source project that automates the deployment of applications inside software containers[2]. Docker uses resource isolation features of the Linux kernel to allow independent containers to run within a single Linux instance, avoiding the overhead of starting and maintaining virtual machines. A Docker container runs directly on top of the operating system, unlike a virtual machine, this does not require to run a separate operating system within the container. Docker also simplifies the creation and operation of task or workload queues and other distributed systems [26, 56].

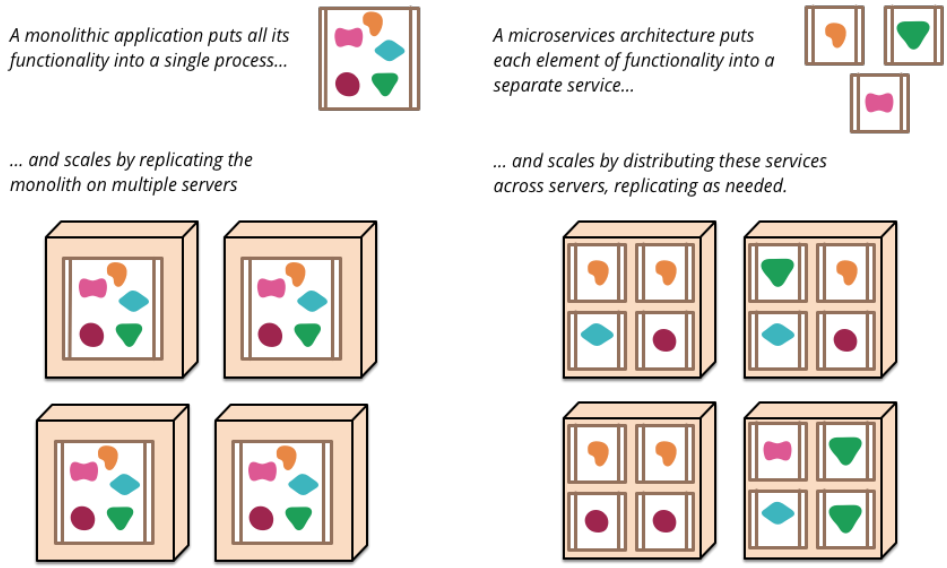

Figure 3.1.1: Microservice architecture [73]

Docker enables the encapsulation of different applications within lightweight-components which can easily be deployed on top of the docker daemon. This can be a single local docker-daemon or a distributed one across a pool of physical hosts. This enables deployment of the required components across a group of machines.

[2]Docker https://www.docker.com/

The Docker architecture is strongly inspired by the Microservice architecture as depicted in Figure 3.1.1, where each functionality is defined and encapsulated within its own container. Scaling such a system is done by adding or removing services (containers). For example, if there is a computationally intensive task scheduled on Spark, more Docker-containers can be spawned on the pool of hosts to devide the computational workload.

The stack is defined using Docker Compose[3], which is the successor of Fig[4]. Docker Compose is a tool for defining multi-container applications in a single file. The application, with all its dependent containers, is booted using a single command which does everything that needs to be done to get it running. At this moment Docker is used on a single machine, but it can transparently scale to multiple host to create a cluster using Docker Swarm[5].

3.2 Apache Spark

Apache Spark[6] is a computational platform that is designed to be distributed, fast and general-purpose. It is an extension of the popular MapReduce model [28] and it is more efficient when it comes to iterative algorithms as it is able to performs in-memory computations [102]. Figure 3.2.1 illustrates the Apache Spark running on top of Docker, but it can also run standalone. Spark comes with specialized libraries for machine learning or graph-processing.

Spark's computational model consists generally of the following steps:

Input-data Spark is able to fetch its input data from a variety of sources, including Hadoop Distributed File Systems, Amazon S3, Apache Flume, Apache Kafka, Apache HBase or any other Hadoop datasource.

Transform Transformations are defined based on the input dataset. Examples of

[3] Docker Compose https://www.docker.com/compose/
[4] Fig http://www.fig.sh/
[5] Docker Swarm https://docs.docker.com/swarm/
[6] Apache Spark http://spark.apache.org/

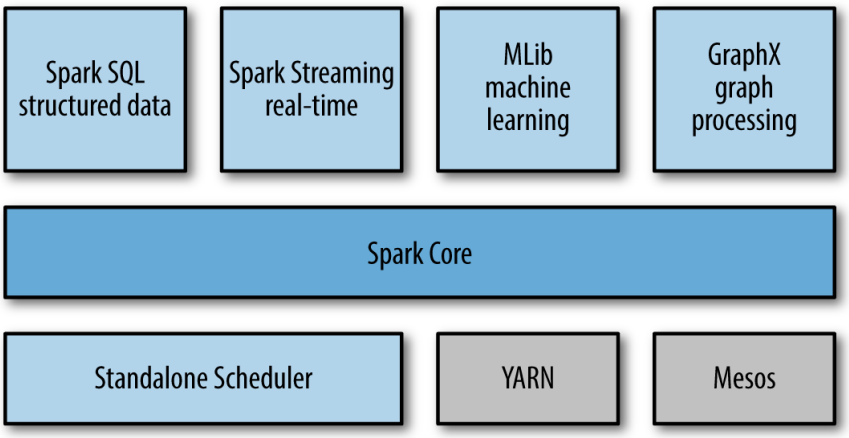

Figure 3.2.1: Apache Spark stack

transformations are: mapping the data to another format, joining different datasets or sorting the data in a specific order.

Aggregate After the distributed transform of the data, everything is aggregated and loaded into the driver's local memory or it is written to a persistent storage like HDFS.

Apache Spark works with Resilient Distributed Datasets (RDD), which is a read-only, partitioned collection of records. Each transformation within Spark requires a RDD as input and transforms the data into a new immutable RDD.

Apache Spark only writes to the file-system in a number of situations:

Checkpoint When setting a checkpoint to which it can recover in the event of data loss because one or more machines in the cluster becoming unresponsive as result of a crash or network failure.

Memory Every worker works with a subset of the RDD. When the subset grows to an extent in which is does not fit in the memory anymore, the data spills to disk.

Shuffle When data needs to be shared across different worker-nodes a shuffle occurs. By designing and implementing an algorithm, these actions should be

avoided or kept to an absolute minimum.

An important concept which needs to be taken into account is the number of partitions of an RDD, which is initially set by the source RDD, for example Kafka, Flume or HDFS. For example, for a `HadoopRDD` data source, which requests data blocks (64MB by default) from HDFS, the number of Spark-paritions set to the number of blocks [28]. This is a convenient way of managing the number of partitions, as this grows with the volume of the data. In the case of `KafkaRDD`, which reads from Kafka's distributed commit log, this is defined by the number of Kafka-partitions in the Kafka-cluster which may not always grow.

It is important to be aware of the number of partitions, as it controls the parallelism of the submitted Spark Job. The number of partitions caps the number of processes which can be executed in parallel. For example, if the RDD only has a single partition, there will be no parallel execution at all. Spark recommends two or three partitions per CPU core in the cluster[7]. This value is heuristically determined and in practice it needs to be tunes to obtain optimal performance as it differs per type of task.

The Spark Streaming library[8], introduced in version 1.2 of Apache Spark, makes it easy to build scalable, fault-tolerant streaming applications. Spark Streaming uses micro-batch semantics, as illustrated in Figure 3.2.2.

Figure 3.2.2: Spark streaming

The enabling micro-batched semantics on a continuous stream of observations is done by defining a window as illustrated in Figure 3.2.3[9]. The window is defined

[7]Apache Spark: Level of Parallelism http://spark.apache.org/docs/latest/tuning.html#level-of-parallelism

[8]Spark Streaming http://spark.apache.org/streaming/

[9]Spark Streaming Guide http://spark.apache.org/docs/latest/streaming-programming-guide.html

by two parameters:

Window length the duration of the window.

Sliding interval the interval at which the window operation is performed.

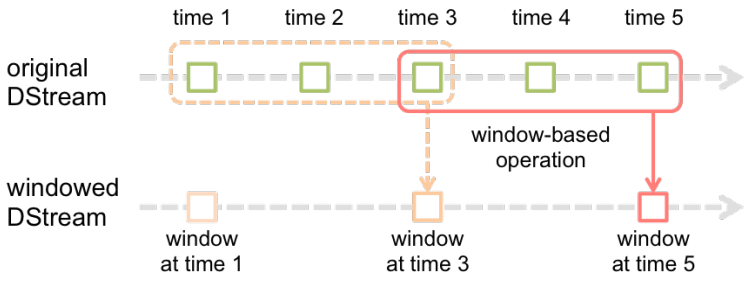

Figure 3.2.3: Applying windowing functions

This enables the window to have an overlap with earlier observations. Each time the requirement is full filled, a new job is dispatched with the window as input. This streaming concept is particularly well-suited using Apache Kafka as the data source, because it acts as a message queue.

3.3 Apache Kafka

Apache Kafka[10] provides functionality of a messaging queue in a distributed, partitioned and replicated way. Figure 3.3.1 illustrates the role of Kafka. First, the terminology is established, which is analogous to message queues in general:

Broker is a process running on a machine that together forms the Kafka cluster.

Producer is a process which publishes messages to the Kafka cluster.

Consumer is a process that is subscribed to topics and reads the feed of published messages.

[10] Apache Kafka http://kafka.apache.org/

Topics is a category identified by a string on which the messages are collected. Each producer and consumer can subscribe to one or more topics on which they write or read messages.

Partition Each topic is divided into a set of partitions in order to distribute the topic across a set of different machines.

Producers have the role of pushing messages into the log and the consumers read the log as messages are appended to the topics that are distributed across multiple machines in order to split the workload and volume of the data across a number of machines. The high-level topology of an Apache Kafka is given in Figure 3.3.1, where three producers, which publish messages to the cluster, and the three consumers, which receive the messages on the topics they are subscribed to.

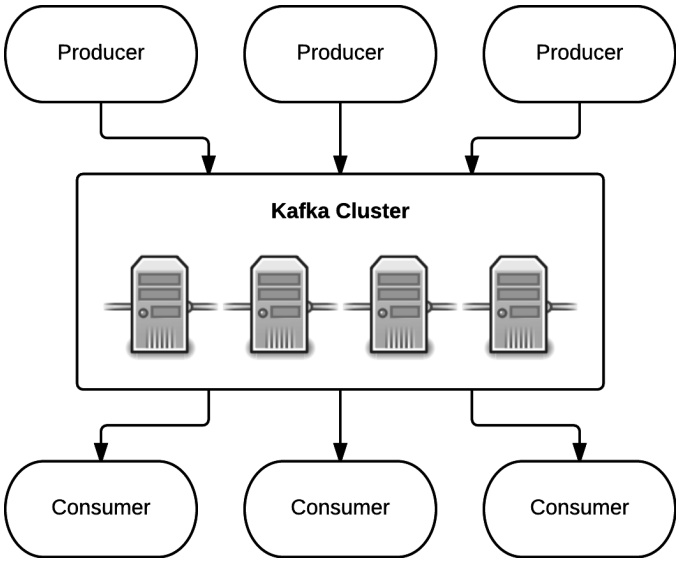

Figure 3.3.1: Apache Kafka high-level topology.

Apache Kafka is implemented as a distributed commit log, an immutable sequence of messages that is appended as new messages arrive, as illustrated in Figure 3.3.2. Each message in the partitions is assigned a sequential id-number called by the offset that uniquely identifies each message within the partition. They allow

the log to scale beyond a size that fits on a single server. By default, the partitioning is done in a round-robin fashion, but a custom partitioning algorithm can be implemented to enhance data-locality.

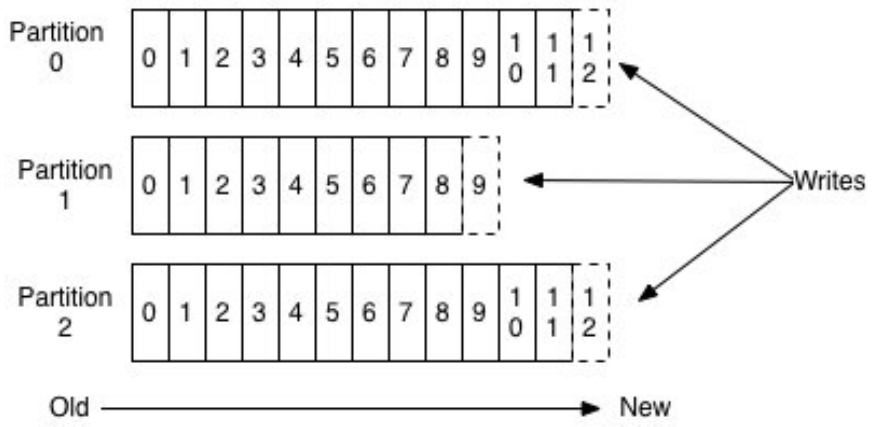

Figure 3.3.2: Anatomy of a topic, implemented as a distributed commit log.

The performance is effectively constant with respect to data size, so retaining lots of data is not a problem within the limitation of fitting a single partition on a single broker. Each partition can be replicated across a configurable number of brokers to ensure fault tolerance and availability of the Kafka cluster.

Apache Kafka allows producers to write arrays of bytes as a record. This means that serializing and deserializing the Scala data structures has to be done by the programmer. This is done in an easy and fast way using the Pickling[11] library which is fast and efficient.

3.4 Apache ZooKeeper

ZooKeeper[12] is a highly-available coordination service for maintaining configuration, naming, distributed synchronization and providing group services [55].

[11] Scala Pickling https://github.com/scala/pickling
[12] Apache ZooKeeper https://zookeeper.apache.org/

All of these kinds of services are used in some form or another by distributed applications, including Apache Kafka and Apache Spark. Because of the difficulty of implementing these kinds of distributed services, applications initially usually skimp on them, which make them brittle and difficult to manage in the presence of change. Even when done correctly, different implementations of these services lead to management complexity when the applications are deployed. This is where ZooKeeper steps in.

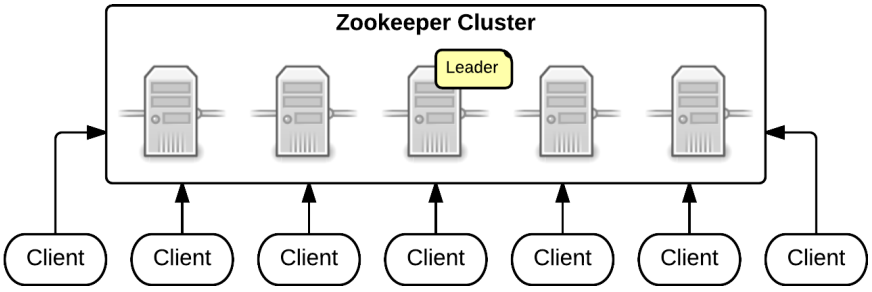

Figure 3.4.1: ZooKeeper topology.

As depicted in Figure 3.4.1, the ZooKeeper service comprises an ensemble of servers that use replication to achieve high availability and performance. ZooKeeper provides a shared hierarchical namespace which is consistent across all nodes. ZooKeeper runs in memory which ensures high throughput and low latency. ZooKeeper replicates the namespace across all nodes using a transaction log which ensures that all the mutations are performed by the majority of the ZooKeeper instances in the cluster. The operations on ZooKeeper are wait-free and do not use blocking operations such as locks. This is implemented in the leader-based atomic broadcast protocol named ZooKeeper Atomic Broadcast (ZAB) [63], which provides the following guarantees [55]:

Linearizable writes all requests that update the state of ZooKeeper are serializable and respect precedence.

FIFO client order all requests from a given client are executed in the order in which they were sent by the client.

As long as a majority of the servers are correct, the ZooKeeper service will be available. With a total of $2f + 1$ Zookeeper processes, it is able to tolerate f failures. When the cluster becomes partitioned, from, for example, a network failure, the partitions with a number of processes smaller than f becomes available and falls down to an auto-fencing mode, rendering the service unavailable [27]. An interesting and useful feature of ZooKeeper are ephemeral nodes. These are nodes in the hierarchical namespace of the ZooKeeper service that will be removed as the process which created them disconnects or becomes unavailable. This enables the ZooKeeper service to only show the list of nodes that are available.

In the software-stack used for running the experiment, ZooKeeper is used for:

Apache Kafka a master is elected for each partition within a topic. This is needed in case of a partition being replicated. One master is assigned to which the slaves follow. If the master dies, a new master gets elected out of the slaves.

Apache Spark Spark allows to have multiple masters of which only one is active. If the master dies, a standby-master can take over and resubmit the job.

3.5 Software

The software is written on top of the architecture as described above. The software uses the Spark Driver to communicate with the cluster. The way this works is illustrated in Figure 3.5.1. Instead of submitting a packed program to the cluster, the driver tells the worker nodes where the data is and the transformations are serialized and transferred to the worker nodes.

The Spark-driver is also deployed on the server as a Docker container. This ensures that the driver runs on the same cluster and that network communication between the cluster and the executors is minimal.

The software is built using Simple Build Tool[13] (SBT) which handles the flow of testing, running and packaging. Using the SBT Assembly plugin [14], a JAR file

[13]Simple Build Tool http://www.scala-sbt.org/
[14]SBT Assembly https://github.com/sbt/sbt-assembly

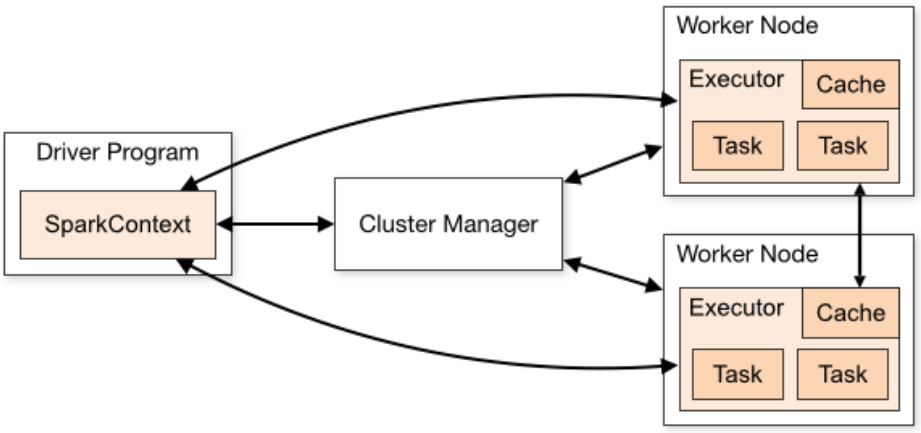

Figure 3.5.1: Spark context which sends the job to the worker nodes.

is generated which contains all the sources and the libraries it depends on. Packing specific versions of the dependent libraries into the JAR prevents the need of additional libraries having to be available at run-time.

The source of the algorithm is placed in a GIT[15] version control system called Github[16], which keeps track of changes to the source code. Github allows the developer to attach hoops to events, such as the push of new code to the repository. The code has been public available for whoever wants to use it.

The continuous-integration server Travis[17] is used to build a new version of the software each time a new push is done to the Github repository. The test-coverage of the software is tracked using Codecov[18]. Using Codecov, the test-coverage is visualized per line of code, based on that the test coverage can be increased.

Codacy[19] is used to keep track of the quality of code by performing static analysis. Each time changes are pushed to GIT, the code is analyzed for Scala specific code-smells which might incur errors, such as the creation of threads instead of using futures, the use of reserved key-words and high cyclomatic complexity or the

[15]GIT Version Control https://git-scm.com/
[16]Github https://github.com/rug-ds-lab/SparkOutlierDetection
[17]Travis CI https://travis-ci.org/rug-ds-lab/SparkOutlierDetection
[18]Codecov https://codecov.io/github/rug-ds-lab/SparkOutlierDetection
[19]Codacy https://www.codacy.com/app/fokko/SparkOutlierDetection

use of `var` instead of the immutable `val`.

Talk is cheap. Show me the code.

Linus Torvalds

4 Algorithm

This chapter introduces and differentiates between two types of algorithms that have been evaluated from the literature in Section 2.1. The algorithms presented in Table 2.1.1 are distance based algorithms. For simplicity we use Euclidean distance, as in Equation 4.1, but all algorithms also work with other metrics. For example, metrics such as the Hamming distance, which can find the number of different symbols in strings of equal length and which is often used for comparing bit-sequences [45]. Another distance-function is the Levenshtein distance, which is used for comparing different strings of text. The Euclidean distance between observation a and b is the length of the line segment connecting them, and is therefore intuitive for people to grasp [29].

$$d(\mathbf{a}, \mathbf{b}) = \sqrt{\sum_{i=1}^{m} (a_i - b_i)^2} \tag{4.1}$$

The distance-based algorithm is related to the concept of clustering. An outlier can be seen as an observation which is not part of a cluster. Also, the algorithms take the locality into account. For example, for the Local Outlier Factor (LOF) algorithm, which compares the density of the surrounding neighbours, it is important to note that the global density may vary a lot across the feature space. Section 4.1 introduces the Local Outlier Factor algorithm, which introduces the concept of local densities and which has been extended and adapted by many papers. This algorithm gives a good idea what outlier detection is about. Next, the Stochastic Outlier Selection (SOS) algorithm is introduced in Section 4.2 and it elaborates on why it is a good fit for Apache Spark and LOF is not.

For outlier detection algorithms, there is no free lunch [100], which implies that there is no algorithm which outperforms all other algorithms [59]. Different algorithms excel under different conditions, depending on the characteristics of the data set.

4.1 Local Outlier Factor

Many distance-based outlier detection algorithms are based or strongly inspired by the Local Outlier Factor algorithm. The concept is depicted in Figure 4.1.1, where the black dots are the observations in the feature space and the dashed circles are the corresponding distances to the k^{th}-observation which is set to $k = 3$.

First, the distance to the k^{th}-nearest observations is computed. This is called the Local Reachability Distance (LDR), as defined in Equation 4.2, where k-distance(A) is the distance to the k^{th}-nearest observation of the neighbouring observation A.

$$\text{LDR}_k(A) = 1/\left(\frac{\sum_{B \in N_k(A)} max(k\text{-distance}(B), d(A,B))}{|N_k(A)|}\right) \tag{4.2}$$

After computing LDR for each observation, which requires the k-nearest neighbours and the k-nearest neighbors of the k-nearest neighbors, the Local Outlier Factor value can be obtained using Equation 4.3, which essentially compares the

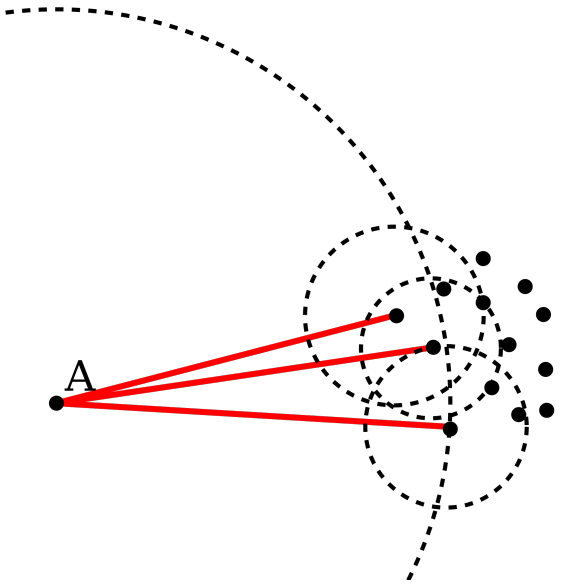

Figure 4.1.1: Local Outlier Factor with $k = 3$.

density of the observation itself with the density of the surrounding observations.

$$\mathrm{LOF}_k(A) = \left(\frac{\sum_{B \in N_k(A)} \mathrm{LDR}(B)}{|N_k(A)|} \right) / \mathrm{LDR}(A) \tag{4.3}$$

The algorithm relies heavily on nearest-neighbour searches which are hard to optimize on distributed computational platforms as they require searching through and sort all the data. Therefore the algorithm is not suitable to implement on top of Apache Spark.

4.2 Stochastic Outlier Selection

Instead of comparing local densities, as done by the Local Outlier Factor concept, the Stochastic Outlier Selection algorithm employs the concept of affinity which does comes from the field of [75, 95]:

Clustering employs affinity to quantify the relationships among observations [36].

For example, the concept of affinity been used for partitioning protein interactions [98] and clustering text [41].

Dimensionality reduction Stochastic Neighbor Embedding [52] or t-distributed Stochastic Neighbor Embedding [94, 95] is a nonlinear dimensionality reduction technique for embedding high-dimensional data into a space of two or three dimensions. The technique has been used among music analysis [44] and bio-informatics [99].

The implementation of the Stochastic Outlier Selection algorithm has been done in Scala[1] on top of Apache Spark. To determine the correct working of the algorithm, unit-tests have been written in order to ensure correct output in each stage in the algorithm. The output of the implementation is compared to the output generated by the Python implementation written by the author[2]. These unit-tests uncovered a bug in the authors' script, which has been patched by submitting a pull request[3].

The algorithm consists of a series of transformations on the data as described in the original paper [60]. These steps are elaborated in the following subsections, which describe how they are implemented.

4.2.1 Distance matrix

The distance matrix takes the n input vectors of m length and transforms it to a $n \times n$ matrix by taking the pairwise distance between each vector. We employ the symmetric Euclidean distance, as defined earlier in Equation 4.1. Being symmetric, the sets' distance $d(\mathbf{x}_i, \mathbf{x}_j)$ is equal to $d(\mathbf{x}_j, \mathbf{x}_i)$ and the distance to self is zero $d(\mathbf{x}_i, \mathbf{x}_i) = 0$. The distance matrix is denoted by $\mathbf{D}$, each row by $\mathbf{D}_i \in \mathbb{R}^m$ and each element by $D_{ij} \in \mathbb{R}$.

The implementation given in Listing 1 takes the Cartesian product of the input vectors to compute the distance between ever pair of observations. Next, the

[1] Scala programming language http://www.scala-lang.org/
[2] SOS https://github.com/jeroenjanssens/sos/blob/master/bin/sos/
[3] Github pull request https://github.com/jeroenjanssens/sos/pull/4/

```
def computeDistanceMatrixPair(data: RDD[(Long, Array[Double])]):
                                        RDD[(Long, Array[Double])] =
    data.cartesian(data).flatMap {
      case (a: (Long, Array[Double]), b: (Long, Array[Double])) =>
        if (a._1 != b._1)
          Some(a._1, euclDistance(a._2, b._2))
        else
          None
    }.combineByKey(
        (v1) => List(v1),
        (c1: List[Double], v1: Double) => c1 :+ v1,
        (c1: List[Double], c2: List[Double]) => c1 ++ c2
      ).map {
      case (a, b) => (a, b.toArray)
    }
```

Listing 1: Computing the distance matrix of a collection of feature vectors.

pairs are mapped to compute the Euclidean distance between every pair of vectors except to itself, as this does not carry any information and is not used by the algorithm. Finally, all the vectors are combined by the unique key of each vector returning the rows of the matrix.

4.2.2 Affinity matrix

The affinity matrix is obtained by transforming the distance matrix proportionally to the distance between two observations. The affinity quantifies the relationship from one observation to another. The affinity σ_i for each observation is found by performing a binary search, which makes the entropy of the distribution between overall neighbors equal to the logarithm of the perplexity parameter h.

$$a_{ij} = \begin{cases} \exp(-d_{ij}^2/2\sigma_i^2) & \text{if } i \neq j \\ 0 & \text{if } i = j \end{cases} \tag{4.4}$$

The perplexity parameter is the only configurable parameter of the algorithm denoted by h as in Equation 4.5. The influence of the h parameter is comparable to the k parameter in the k-nearest neighbours algorithm. It also alters the behaviour of the algorithm analogously: the higher the value h, the more it depends on the

surrounding neighbours. One important difference is that $h \in \mathbb{R}$ and $k \in \mathbb{N}$. The perplexity value has a deep foundation in information theory, but in practice should it be tuned by the domain expert to provide a good level of outlierness [60].

```
def computeAffinityMatrix(dMatrix: RDD[(Long, Array[Double])],
                          perplexity: Double = DEFAULT_PERPLEXITY):
                                  RDD[(Long, DenseVector[Double])] =
  dMatrix.map(r => (r._1, binarySearch(new DenseVector(r._2), Math.log(perplexity))))
```

Listing 2: Transforming the distance matrix to the affinity matrix.

Listing 2 applies the binary search on each row within the matrix which approximates the affinity. The maximum number of iterations in the binary search can be limited in order to interrupt the execution which will introduce some error, but will reduce the computational time. Similar, an accepted tolerance can be set to accept a small error in exchange for reduced computational time. The binary search iteratively bisects the interval and then selects the correct upper of lower half until the desired variances for each observation is found.

$$h = \{x \in \mathbb{R} | 1 \leq x \leq n - 1\} \tag{4.5}$$

The affinity matrix is obtained by applying Equation 4.4 on every element of the distance matrix. Perplexity is employed to adaptively set the variances which are computed using the introduced binary search, which approximates for every value the σ^2 so that every observation has effectively h neighbours [52]. This has to be done for every observation assuming that it has a unique place in space.

Listing 3 shows the recursive function used to find or approximate the perplexity of each row x_i in the distance matrix. The recursive function eliminates mutable variables which are impossible to avoid when using using loops. The use of immutable variables comes from the functional programming aspect of Scala and makes the code less error-prone.

```
def binarySearch(affinity: DenseVector[Double],
                 logPerplexity: Double,
                 iteration: Int = 0,
                 beta: Double = 1.0,
                 betaMin: Double = Double.NegativeInfinity,
                 betaMax: Double = Double.PositiveInfinity,
                 maxIterations: Int = DEFAULT_ITERATIONS,
                 tolerance: Double = DEFAULT_TOLERANCE): DenseVector[Double] = {

  val newAffinity = affinity.map(d => Math.exp(-d * beta))
  val sumA = sum(newAffinity)
  val hCurr = Math.log(sumA) + beta * sum(affinity :* newAffinity) / sumA
  val hDiff = hCurr - logPerplexity

  if (iteration < maxIterations && Math.abs(hDiff) > tolerance) {
    val search = if (hDiff > 0)
      (if (betaMax == Double.PositiveInfinity || betaMax == Double.NegativeInfinity)
        beta * 2.0
      else
        (beta + betaMax) / 2.0, beta, betaMax)
    else
      (if (betaMin == Double.PositiveInfinity || betaMin == Double.NegativeInfinity)
        beta / 2.0
      else
        (beta + betaMin) / 2.0, betaMin, beta)

    binarySearch(   affinity,
                    logPerplexity,
                    iteration + 1,
                    search._1,
                    search._2,
                    search._3,
                    maxIterations,
                    tolerance)
  }
  else
    newAffinity
}
```

Listing 3: Performing a binary search to approximate the affinity for each observation.

4.2.3 Binding probability matrix

The binding probability matrix defines the probability of observation x_i to x_j. The mathematical foundation, based on graph theory, computes the Stochastic Neighbour Graph and the subsequent generative process is out of scope and can be found

in the original paper [60].

$$b_{ij} = \frac{a_{ij}}{\sum_{k=1}^{n} a_{ik}} \tag{4.6}$$

The binding probability matrix can be applied by normalizing the affinity matrix such that $\sum_{k=1}^{n}$ sums up to 1. Equation 4.6 transforms the affinity matrix to the probability outlier matrix.

```
def computeBindingProbabilities(rows: RDD[(Long, DenseVector[Double])]):
                                        RDD[(Long, Array[Double])] =
  rows.map(r => (r._1, r._2 :/ sum(r._2)).toArray)
```

Listing 4: Transforming the affinity matrix into the binding probability matrix.

Listing 4 shows the implementation which divides each element by the sum of the vector.

4.2.4 Computing outlier probabilities

The last step is to compute the outlier probability of vector $\mathbf{x}_i$, which is given by the product of the column binding probability matrix as in Equation 4.7.

$$f_{\text{SOS}}(\mathbf{x}_i) \equiv \prod_{\substack{j=1 \\ j \neq i}}^{n} (1 - b_{ji}) \tag{4.7}$$

The implementation is given in Listing 5. The implementation uses a `flatmap` to add indices to the elements of the vector. The inline if-condition offsets the indices by one to skip the diagonal of the matrix. Important to note is the `zipWithIndex` operation is applied to the local collection and not on the RDD which would trigger a subsequent Spark job.

Finally the `foldByKey` groups all the keys and perform a fold which computes the product of each column. This last action invokes a data-shuffle, as the rows are

```
def computeOutlierProbability(rows: RDD[(Long, Array[Double])]):
                                                    RDD[(Long, Double)] =
  rows.flatMap(r => r._2.zipWithIndex.map(p =>
    (p._2 + (if (p._2 >= r._1) 1L else 0L), p._1))).foldByKey(1.0)((a, b) => a * (1.0 - b))
```

Listing 5: Computing the outlierness from the binding probability matrix.

distributed across the different worker nodes. The product represents the outlierness of each observation.

Computers are useless. They can only give you answers.

Pablo Picasso

5 Results

The purpose of this chapter is to empirically explore the scalability of the algorithm and architecture compared to different input sets. The quantitative results are presented using the algorithm as described in Section 4.2 and the architecture as presented in Chapter 3. First, Section 5.1 explains how the input data is generated. Second, the hardware is presented in Section 5.2 on which the tests are executed. Finally, in Section 5.3 the tests and the results are presented and evaluated.

5.1 Dataset

The dataset is generated beforehand. The input of the algorithm is a $m = 10$ dimensional vector as in Equation 5.1. The value of each feature vector is sampled in a pseudo-random fashion from a Gaussian distribution (also known as the normal distribution), shown in Figure 5.1.1. The main characteristic of this distribution is

that when given a theoretical infinite number of samples, approximately 68.2% of the sampled values are within the ± 1 standard deviation.

$$\mathbf{x} = [x_1, \ldots, x_m] \in \mathbb{R}^m \tag{5.1}$$

This results in a dataset where the number of outliers is limited as the mean of all the observations in the feature space is at the origin. This is not a problem as the goal is to determine the scalability of the algorithm and not its performance in determining if the observation is an outlier or not.

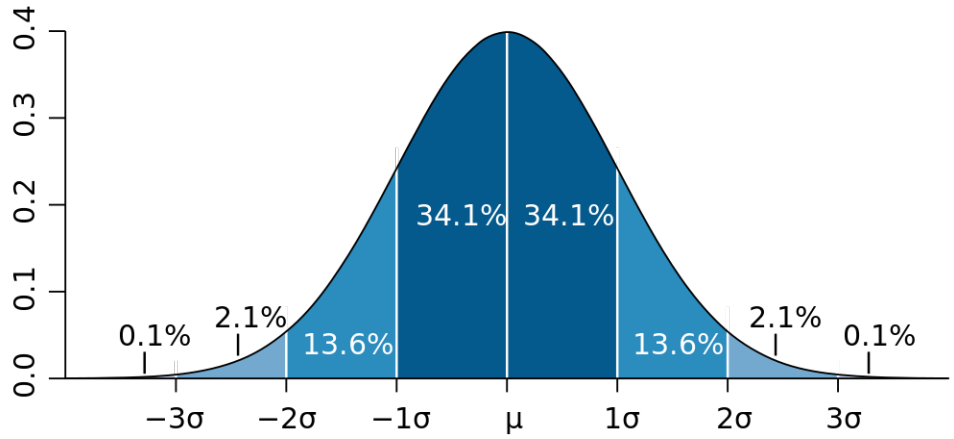

Figure 5.1.1: The curve from which the feature vectors are sampled. The vertical axis is the probability. The σ values depict the standard deviations.

5.2 Hardware

This section describes the hardware which is used to run the tests. Docker provides an abstract interface which allows us to run the stack on a set of homogeneous resources. Nevertheless the hardware impacts the actual performance, therefore is it important to describe the hardware which is essential to repreduce the same numbers. To create an isolated environment which is not affected by other processes

running on the physical hardware, a dedicated server has been assigned. Quintors' local Dell PowerEdge T620 development-server 'Big-Willem' has been assigned for running the algorithm and for producing the results. Big-Willem has two Intel Xeon cpu's with six cores each supporting Hyper-threading, thus providing 24 logical cores in total. The main memory consists of 128 gigabytes in total. The virtual machine runs on 16 cores, has 64 gigabytes of memory assigned to it and is fully dedicated to running the software stack.

5.3 Results

First, the general execution time as a function of the input size is analyzed in Subsection 5.3.1. Second, the impact on the execution time by adding additional workers to the Spark cluster is explored in Subsection 5.3.2. This gives a good insight in the parallelizability of the algorithm. Finally, the stages of the algorithm are defined and insights are given regarding the execution time per stage in Subsection 5.3.3.

5.3.1 Execution time

The total execution time provides insight of the execution time of the algorithm with a fixed number of resources. The number of observations is doubled for each iteration starting from 500 up to 64000. Every iteration is performed five times to obtain stable results. The execution time is given in seconds and is measured after the initialization of the Spark-cluster until the results of the distributed computation are returned to the driver.

In Figure 5.3.1 we observe a quadratic increase in execution time, which is expected to be equal to the dominant computational complexity. In the case of the algorithm, the step of taking the Cartesian product for computing the distance matrix in Subsection 4.2.1 is in order $\mathcal{O}(n^2)$ and therefore the algorithm will perform analogous.

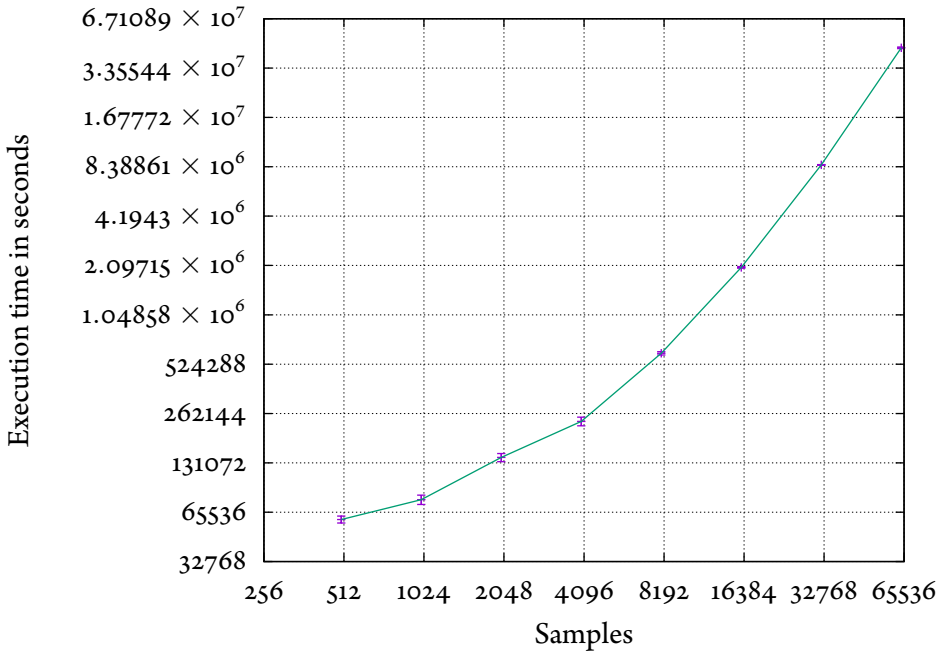

Figure 5.3.1: Execution time as a function of the input size.

5.3.2 Parallelization

The purpose of this experiment is to determine the reduction of execution time by adding additional Spark workers to the cluster. Figure 5.3.2 illustrates the execution time as a function of the number of workers.

A single Spark master and a single Apache Zookeeper instance is used. The number of Kafka nodes is equal to the number of Spark workers since when the number of workers increases, the number of partitions also needs to grow to parallelize the work and share the workload. Therefore more brokers are desirable to divide the partitions over different brokers. Each worker node within Apache Spark has 2 cores and 6 gigabytes of memory assigned.

We observe that the execution time decreases until workers $= 5$, where the cluster works with 10 CPU's and 30 gigabytes of memory. For the relative small dataset

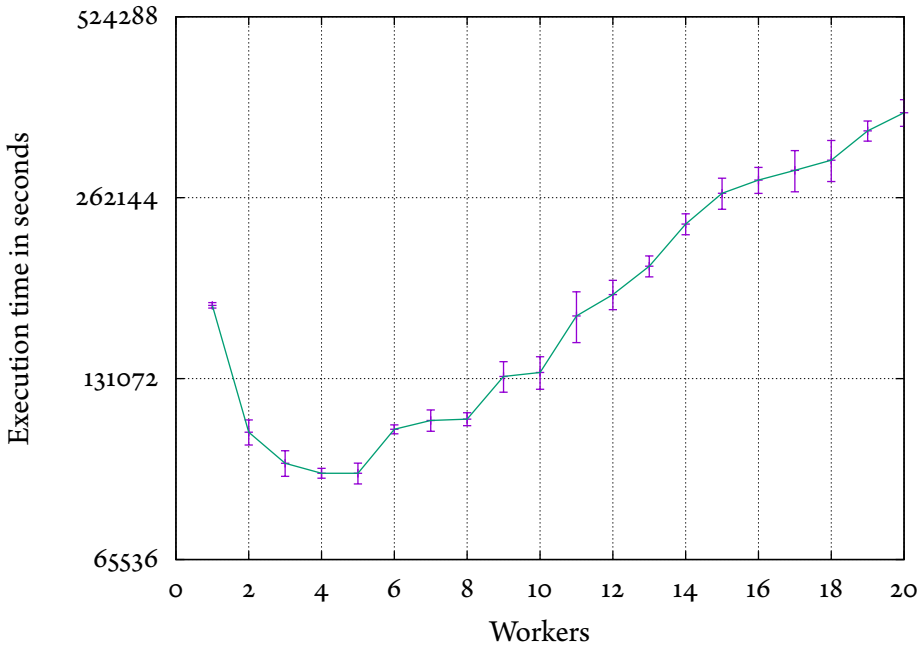

Figure 5.3.2: Execution time as a function of the number of workers where each worker gets two cores and six gigabytes of RAM assigned, input size $n = 2000$, and each worker has three partitions.

of $n = 2000$, the memory of each worker is not a resource constraint. The bottleneck is the processing power, which increases significantly when adding more nodes to the cluster. As the machine features 12 physical cores and 24 logical cores, the number of 10 CPU cores assigned to Spark is optimal, leaving some power for the Spark Master, Zookeeper and Apache Kafka instances. When increasing the number of nodes the context-switches are becoming dominant, therefore the number of nodes has to be chosen precisely according to the available hardware for obtaining the best resource utilization.

5.3.3 Shuffle-behaviour

The shuffle-stages of Apache Spark are introduced in Subsection 3.2 and are known for having a major impact on the performance. Especially when the job is data-

intensive in particular [24]. At a data-shuffle the RDD, which is scattered across different machines, re-distribution of the data from many to many machines is required.

Even though moving data is expensive, sometimes it is necessary. For example, certain stages need to consolidate data on a single machine so that it can be co-located in memory to perform computations for the next stage. The algorithm contains two shuffle stages at:

Distance matrix At the first step, described in Subsection 4.2.1, the Cartesian product of the vector input is taken to construct the distance matrix between all pairs.

Outlier probabilities The last step takes the product of the matrix, as described in Subsection 4.2.4. As all the rows of the matrix are distributed across the different nodes. This requires a shuffle to transfer each row to a single machine to compute the final result. This stage does also includes the binary search to find the affinity for each row, as described in Subsection 4.2.2.

The Cartesian product is first computed using the `cartesian` primitive, which returns all the pairs of vectors. Then the `flatmap` primitive is used to compute the distances and leaves out the diagonal as it does not hold any information. Last, using `combineByKey` is used to reduce the pairs to a RDD of vectors which contain the distance from one to another observations. The intermediate steps, transforming the distance matrix to the affinity matrix in Subsection 4.2.2 and computing the binding probability matrix as described in Subsection 4.2.3 can be done locally on each worker for each vector in the RDD. Lastly, the position of the value in the vector is assigned a key and the final result is computed using `foldLeft` which requires a shuffle as the rows of the matrix are local, and the columns are distributed across multiple machines. Using the Spark Event Log[1], the time taken for each stage can be monitored. This information is helpful to obtain insights about the bottlenecks of the algorithm.

[1]Spark monitoring `http://spark.apache.org/docs/latest/monitoring.html`

	$N = 2400$				$N = 4800$				$N = 9600$			
Stage	Distance	Product	Collect	Sum	Distance	Product	Collect	Sum	Distance	Product	Collect	Sum
Run 1	114	90	6	210	258	330	6	594	900	1380	9	2289
Run 2	114	90	6	210	252	330	6	588	780	1380	12	2172
Run 3	114	90	6	210	258	330	6	594	900	1380	10	2290
Run 4	114	90	6	210	282	330	6	618	1080	1380	8	2468
Run 5	108	102	6	216	264	330	6	660	1260	1380	9	2649
Run 6	114	90	6	210	270	402	5	677	1080	1380	11	2471
Run 7	126	84	6	216	252	330	6	588	1320	1380	12	2712
Run 8	144	90	6	240	276	402	5	683	960	1380	12	2352
Run 9	138	102	6	246	270	330	5	605	900	1380	11	2291
Run 10	114	102	5	221	258	342	6	606	960	1380	11	2351
Average	120.00	93.00	5.9	218.9	5.7	345.6	264	615.3	1014	1380	10.5	2404.5
Std.dev	12.00	6.48	0.32	13.32	0.48	29.96	10.2	35.31	170.76	0	1.43	177.56
Total	42.49%	54.82%	2.70%		42.91%	56.17%	0.92%		42.17%	57.39%	0.43%	

Table 5.3.1: Time taken for each stage of the algorithm execution. The distance column refers to the Cartesian product of the vectors to compute the distance-matrix. The product column refers to the shuffle required to consolidate each column of the matrix on a single worker. The collect column is the time taken to bring the results back to the driver. All numbers are in seconds.

Table 5.3.1 provides insights in the execution time per stage. Each stage is delimited by a shuffle at which the different worker nodes in the cluster have to coordinate and transfer the data which is required for the next stage. We observe that the last stage, which entails the binary search and the shuffling of the columns, covers the majority of the execution time. At first glance the Cartesian product is likely to be the dominant factor, but the share of this stage decreases as the number of observations grows.

This research is concluded by the Research Question defined in Section 1.2. Subsequently, further research is presented which provides further uncovered research questions which have been exposed by the presented work.

The introduction given in Section 1 illustrates the growth of information in today's society. The main goal of outlier detection is to provide automated extraction of possible valuable observations within a dataset which is too big or complex to be analyzed by traditional statical software. Instead, extracting potential outliers within these staggering amounts of data, a scalable approach is required. The background of outlier detection and a variety of useful applications is presented in Section 1.1, among which its use in the fields of financial transactions, sensor monitoring, quality control and more.

After evaluating different definitions, we acquire the definition of outlier detection as: 'An outlier is an observation that deviates quantitatively from the majority of the observations, according to an outlier-selection algorithm'. Important is the notion of quantifying the outlierness of a particular outlier. Chapter 2 provides a solid background on outlier detection in Section 2.1. The different types of outlier detection are introduced, and a set of potential algorithms is presented. Subsequently, as outlier detection is part of the field of Knowledge Discovery and Data Mining and the computational platform to scale and distribute the algorithm is rarely taken into account. Therefore the Web-scale paradigm is first introduced in Section 2.2 and describes the computational models and tools which can be used to parallelize and scale the process of outlier detection.

Chapter 3 introduces the architecture which allows the application to run on a cluster of machines distributed across a data center. The underlying pattern is the Microservice pattern which consists of suites of independently deployable services. Apache Spark is used as the computational platform which enables large-scale processing. Spark also takes care of failing nodes in the cluster by restarting the task onto another worker. Apache Kafka is used as the data-source because of its ability to scale and divide load. Kafka acts as a message queue which integrates nicely with Spark Streaming to apply the algorithm as new messages are appended. All the services are deployed using Docker which enables fast and easy deployment

on possibly heterogeneous hardware using abstraction.

Starting with Research sub-question 1; *'Which general-purpose outlier detection algorithms can potentially be scaled out'*, a variety of the types and important outlier detection algorithms are given in Table 2.1.1. This boils down to two suitable algorithms in Section 4, i.e. Local Outlier Factor (LOF) based algorithms and the Stochastic Outlier Selection (SOS) algorithm. The LOF algorithm is a widely used algorithm and extensions are proposed in a variety of papers. The SOS algorithm is chosen over the LOF algorithm because the latter requires many k-nearest neighbour queries which are difficult to implement efficiently in a map-reduce model. Beside that the ability to extract outliers is comparable.

Research sub-question 2 focuses on the field of big-data and distributed computing to explore the possibilities of scaling; *'Which computational engines are appropriate for outlier-detection in a big-data setting'*. As 'big-data' is an ambiguous term, therefore a definition is in place, in Section 2.2 we give the definition: 'massively parallel software running on tens, hundreds, or even thousands of servers'. Many outlier detection algorithm do not take scalability into account and solely focus on the ability to detect outliers. The present goal is to scale outlier detection on a large pool of easily accessible virtualized resources, which can be dynamically reconfigured in order to adjust to a variable load. The infrastructure given in Chapter 3 enables scaling the algorithm by adding or removing workers based on load or requirements. Packing the services inside Docker containers enables fast deployment onto a large number of machines. As the number of machines increases, it becomes more probable that one fails, therefore availability and coordination is provided by Apache Zookeeper, which provides reliable distributed coordination.

The last Research sub-question is 3; *'How to adapt the algorithm to work with streams of data, rather than a static set'*. Data is generated faster and real-time data processing is required to act upon what is happening real time. By using the Spark Streaming library and the Apache Kafka data source this can be done in an efficient micro-batching characteristic. The number of observations which are taken and the interval between them is defined by a window which also allows windows to overlap.

Based on the results in Chapter 5 a number of conclusions can be drawn. We observe that:

- The distributed implementation of outlier-detection is feasible and follows the computational complexity of the algorithm $\mathcal{O}(n^2)$ as the function of the input size.
- The number of partitions of the RDD needs to be tuned to the size of the input and the number of worker nodes to utilize the resources in the cluster most optimal.
- The configuration of Apache Spark has to be tuned to the specifications of the systems to avoid unnecessary swapping, which induces significant overhead and slows down the computations.

All the computations within Spark are executed in stages. After each stage, synchronization between the workers is done and data is exchanged which is required for the next stage. Based on these observations, the algorithm can be further optimized as explained in the next section.

This is the first public available implementation on top of Apache Spark. The problem lies in the computational complexity of the computation of the distance matrix. By taking the Cartesian product, where the computational complexity is quadratic in respect to the input size, this cannot always be feasible for humongous datasets. In production a squared number of additional machines is required when increasing the input size increases linearly. This can obviously be retained by limiting the window size of the micro batching semantics, but making the windows too small might lead to unstable results as the set of data becomes too small. The architecture has proven to scale across different machines, but it can be improved by answering the following questions:

- A more powerful primitive for the Cartesian product. The computation of the distance matrix is a special case as it takes the Cartesian product of itself. This edge case might be susceptible to optimization.

- Explore the possibility of an optimized data-structure such as the Barnes-Hut tree [10] in a distributed fashion. This will replace the Cartesian product. Instead of computing the exact distance it takes an approximation and runs in $\mathcal{O}(n \log n)$ time. This has been done in a shared memory environment [30].

To iterate is human, to recurse divine.

L. Peter Deutsch

Bibliography

[1] Novelty detection: a review—part 1: statistical approaches. *Signal Processing*, 83(12):2481 – 2497, 2003. ISSN 0165-1684. doi: http://dx.doi.org/10.1016/j.sigpro.2003.07.018.

[2] *Intelligent Data Mining: Techniques and Applications (Studies in Computational Intelligence)*. Springer, 2005. ISBN 3540262563.

[3] Survey on cloud computing. *International Journal of Computer Trends and Technology*, 4(9), 9 2013.

[4] C.C. Aggarwal, J. Han, J. Wang, and P.S. Yu. A framework for clustering evolving data streams. In *Proceedings of the 29th International Conference on Very Large Data Bases*, volume 29 of *VLDB '03*, pages 81–92. VLDB Endowment, 2003. ISBN 0127224424.

[5] Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. On the surprising behavior of distance metrics in high dimensional space. In *Lecture Notes in Computer Science*, pages 420–434. Springer, 2001.

[6] E. Aleskerov, B. Freisleben, and B. Rao. Cardwatch: a neural network based database mining system for credit card fraud detection. In *Computational Intelligence for Financial Engineering (CIFEr), 1997., Proceedings of the IEEE/IAFE 1997*, pages 220–226, Mar 1997. doi: 10.1109/CIFER.1997.618940.

[7] Fabrizio Angiulli and Clara Pizzuti. Fast outlier detection in high dimensional spaces. In Tapio Elomaa, Heikki Mannila, and Hannu Toivonen, editors, *Principles of Data Mining and Knowledge Discovery*, volume 2431 of *Lecture Notes in Computer Science*, pages 15–27. Springer Berlin Heidelberg, 2002. ISBN 978-3-540-44037-6.

[8] B. Azvine, Z. Cui, D.D. Nauck, and B. Majeed. Real time business intelligence for the adaptive enterprise. In *E-Commerce Technology, 2006. The 8th IEEE International Conference on and Enterprise Computing, E-Commerce, and E-Services, The 3rd IEEE International Conference on*, pages 29–29, June 2006. doi: 10.1109/CEC-EEE.2006.73.

[9] Bahman Bahmani, Kaushik Chakrabarti, and Dong Xin. Fast personalized pagerank on mapreduce. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 973–984, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0661-4. doi: 10.1145/1989323.1989425.

[10] Josh Barnes and Piet Hut. A hierarchical o(n log n) force-calculation algorithm. *Nature*, 324(6096):446–449, dec 1986. doi: 10.1038/324446a0. URL http://dx.doi.org/10.1038/324446a0.

[11] V. Barnett and T. Lewis. *Outliers in Statistical Data*. Wiley, Chichester New York, 1994. ISBN 9780471930945.

[12] Stephen D. Bay and Mark Schwabacher. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 29–38, New York, NY, USA, 2003. ACM. ISBN 1-58113-737-0.

[13] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975. ISSN 0001-0782. doi: 10.1145/361002.361007.

[14] Stefan Berchtold, Daniel A. Keim, and Hans-Peter Kriegel. The x-tree: An index structure for high-dimensional data. In *Proceedings of the 22th International Conference on Very Large Data Bases*, VLDB '96, pages 28–39, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc. ISBN 1-55860-382-4.

[15] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is "nearest neighbor" meaningful? In *In Int. Conf. on Database Theory*, pages 217–235, 1999.

[16] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is "nearest neighbor" meaningful? In Catriel Beeri and Peter Buneman, editors, *Database Theory — ICDT'99*, volume 1540 of *Lecture Notes in*

Computer Science, pages 217–235. Springer Berlin Heidelberg, 1999. ISBN 978-3-540-65452-0. doi: 10.1007/3-540-49257-7_15.

[17] Nedret Billor, Ali S. Hadi, and Paul F. Velleman. Bacon: blocked adaptive computationally efficient outlier nominators. *Computational Statistics and Data Analysis*, 34(3):279 – 298, 2000. ISSN 0167-9473. doi: http://dx.doi.org/10.1016/S0167-9473(99)00101-2.

[18] M.M. Breunig, H.P Kriegel, R.T. Ng, and J. Sander. Lof: Identifying density-based local outliers. *SIGMOD Rec.*, 29(2):93–104, may 2000. ISSN 0163-5808.

[19] M.M. Breunig, H.P Kriegel, R.T. Ng, and J. Sander. Lof: Identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD '00, pages 93–104, New York, NY, USA, 2000. ACM. ISBN 1581132174.

[20] Carla E. Brodley and Mark A. Friedl. Identifying and eliminating mislabeled training instances. In *In AAAI/IAAI*, pages 799–805. AAAI Press, 1996.

[21] Kerstin Bunte, Barbara Hammer, Axel Wismueller, and Michael Biehl. Adaptive local dissimilarity measures for discriminative dimension reduction of labeled data. *Neurocomputing*, 73(7-9):1074–1092, 3 2010. ISSN 0925-2312. doi: 10.1016/j.neucom.2009.11.017.

[22] R. Buyya, J. Broberg, and A. Goscinski. *Cloud Computing: Principles and Paradigms*. Wiley, 2011. ISBN 0470887990.

[23] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):15:1–15:58, jul 2009. ISSN 0360-0300.

[24] Yanpei Chen, Rean Griffith, Junda Liu, Randy H. Katz, and Anthony D. Joseph. Understanding tcp incast throughput collapse in datacenter networks. In *Proceedings of the 1st ACM Workshop on Research on Enterprise Networking*, WREN '09, pages 73–82, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-443-0. doi: 10.1145/1592681.1592693.

[25] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, Y. Xing, and S. Zdonik. Scalable distributed stream processing. In *CIDR 2003 - First Biennial Conference on Innovative Data Systems Research*, Asilomar, CA, 1 2003.

[26] CloudAve. Oscon: Conversations, deployments, architecture, docker and the future?, 7 2013. URL https://www.cloudave.com/30655/oscon-conversations-deployments-architecture-docker-and-the-future/.

[27] G Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed Systems: Concepts and Design (3rd Edition)*. Addison Wesley, 2000. ISBN 0201619180.

[28] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 1 2008. ISSN 0001-0782.

[29] Michel Marie Deza and Elena Deza. *Encyclopedia of Distances*. Springer Berlin Heidelberg, 2009. doi: 10.1007/978-3-642-00234-2_1.

[30] John Dubinski. A parallel tree code. *New Astronomy*, 1(2):133 – 147, 1996. ISSN 1384-1076. doi: http://dx.doi.org/10.1016/S1384-1076(96)00009-7. URL http://www.sciencedirect.com/science/article/pii/S1384107696000097.

[31] G. Enderlein. Hawkins, d. m.: Identification of outliers. chapman and hall, london – new york 1980, 188 s., £ 14, 50. *Biometrical Journal*, 29(2):198–198, 1987. doi: 10.1002/bimj.4710290215.

[32] Eleazar Eskin, Andrew Arnold, Michael Prerau, Leonid Portnoy, and Sal Stolfo. A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data. In *Applications of Data Mining in Computer Security*. Kluwer, 2002.

[33] Hongqin Fan, OsmarR. Zaïane, Andrew Foss, and Junfeng Wu. A nonparametric outlier detection for effectively discovering top-n outliers from engineering data. In Wee-Keong Ng, Masaru Kitsuregawa, Jianzhong Li, and Kuiyu Chang, editors, *Advances in Knowledge Discovery and Data Mining*, volume 3918 of *Lecture Notes in Computer Science*, pages 557–566. Springer Berlin Heidelberg, 2006. ISBN 978-3-540-33206-0. doi: 10.1007/11731139_66.

[34] Usama M. Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. Advances in knowledge discovery and data mining. chapter From Data Mining to Knowledge Discovery: An Overview, pages 1–34. American Association for Artificial Intelligence, Menlo Park, CA, USA, 1996. ISBN 0-262-56097-6.

[35] I K Fodor. A survey of dimension reduction techniques. Technical report, Lawrence Livermore National Lab., CA (US), 2002.

[36] B. J. Frey and D. Dueck. Clustering by passing messages between data points. *Science*, 315(5814):972–976, feb 2007. doi: 10.1126/science.1136800.

[37] Ryohei Fujimaki, Takehisa Yairi, and Kazuo Machida. An approach to spacecraft anomaly detection problem using kernel feature space. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, KDD '05, pages 401–410, New York, NY, USA, 2005. ACM. ISBN 1-59593-135-X. doi: 10.1145/1081870.1081917.

[38] Jing Gao and Pang-Ning Tan. Converting output scores from outlier detection algorithms into probability estimates. In *Data Mining, 2006. ICDM '06. Sixth International Conference on*, pages 212–221, Dec 2006. doi: 10.1109/ICDM.2006.43.

[39] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. *SIGOPS Oper. Syst. Rev.*, 37(5):29–43, October 2003. ISSN 0163-5980. doi: 10.1145/1165389.945450.

[40] Frank E. Grubbs. Procedures for detecting outlying observations in samples. *Technometrics*, 11(1):1–21, feb 1969. doi: 10.1080/00401706.1969.10490657.

[41] Renchu Guan, Xiaohu Shi, Maurizio Marchese, Chen Yang, and Yanchun Liang. Text clustering with seeds affinity propagation. *IEEE Transactions on Knowledge and Data Engineering*, 23(4):627–637, 2011. ISSN 1041-4347. doi: http://doi.ieeecomputersociety.org/10.1109/TKDE.2010.144.

[42] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. *SIGMOD Rec.*, 14(2):47–57, June 1984. ISSN 0163-5808. doi: 10.1145/971697.602266.

[43] Ali S. Hadi, A. H. M. Rahmatullah Imon, and Mark Werner. Detection of outliers. *Wiley Interdisciplinary Reviews: Computational Statistics*, 1(1):57–70, jul 2009.

[44] Philippe Hamel and Douglas Eck. Learning features from music audio with deep belief networks. In *Proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR)*, pages 339–344, August 2010.

[45] Rw Hamming. Error Detecting and Error Correcting Codes. *Bell System Technical Journal*, 26(2):147–160, 1950.

[46] J. Han, M. Kamber, and J. Pei. *Data Mining: Concepts and Techniques, Second Edition (The Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann, 2006. ISBN 1558609016.

[47] I.A.T. Hashem, I. Yaqoob, N.B. Anuar, S. Mokhtar, A. Gani, and S.U. Khan. The rise of "big data" on cloud computing: Review and open research issues. *Information Systems*, 47:98–115, 2015. ISSN 0306-4379.

[48] Nikolas Roman Herbst, Samuel Kounev, and Ralf Reussner. Elasticity in cloud computing: What it is, and what it is not. In *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13)*, pages 23–27, San Jose, CA, 2013. USENIX. ISBN 978-1-931971-02-7.

[49] Martin Hilbert. How much information is there in the "information society"? *Significance*, 9(4):8–12, 2012. ISSN 1740-9713. doi: 10.1111/j.1740-9713.2012.00584.x.

[50] Martin Hilbert and Priscila López. The world's technological capacity to store, communicate, and compute information. *Science*, 332(6025):60–65, 2011. doi: 10.1126/science.1200970.

[51] Alexander Hinneburg, Charu C. Aggarwal, and Daniel A. Keim. What is the nearest neighbor in high dimensional spaces? In *Proceedings of the 26th International Conference on Very Large Data Bases*, VLDB '00, pages 506–515, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. ISBN 1-55860-715-3.

[52] Geoffrey E. Hinton and Sam T. Roweis. Stochastic neighbor embedding. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 857–864. MIT Press, 2003.

[53] VictoriaJ. Hodge and Jim Austin. A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22(2):85–126, 2004. ISSN 0269-2821. doi: 10.1007/s10462-004-4304-y.

[54] J. Holler. *From machine-to-machine to the internet of things introduction to a new age of intelligence*. Academic Press, Amsterdam, 2014. ISBN 9780124076846.

[55] P. Hunt, M. Konar, F.P. Junqueira, and B. Reed. Zookeeper: wait-free coordination for internet-scale systems. In *Proc. ATC'10, USENIX Annual Technical Conference*, pages 145–158. USENIX, 2010.

[56] Iron.io. How docker helped us achieve the (near) impossible, 4 2014. URL http://blog.iron.io/2014/04/how-docker-helped-us-achieve-near.html.

[57] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. In *Proceedings of the 2007 Eurosys Conference*, Lisbon, Portugal, March 2007. Association for Computing Machinery, Inc.

[58] A. Jacobs. The pathologies of big data. *Commun. ACM*, 52(8):36–44, 8 2009. ISSN 0001-0782.

[59] Jeroen Janssens. *Outlier Selection and One-Class Classification*. PhD thesis, june 2013.

[60] E.O. Postma J.H.M. Janssens, F. Huszar and H.J. van den Herik. Stochastic outlier selection. (TiCC TR 2012-001), 2012.

[61] Wen Jin, AnthonyK.H. Tung, Jiawei Han, and Wei Wang. Ranking outliers using symmetric neighborhood relationship. In Wee-Keong Ng, Masaru Kitsuregawa, Jianzhong Li, and Kuiyu Chang, editors, *Advances in Knowledge Discovery and Data Mining*, volume 3918 of *Lecture Notes in Computer Science*, pages 577–593. Springer Berlin Heidelberg, 2006. ISBN 978-3-540-33206-0. doi: 10.1007/11731139_68.

[62] George H. John. Robust decision trees: Removing outliers from databases. In *Knowledge Discovery and Data Mining*, pages 174–179. AAAI Press, 1995.

[63] F.P. Junqueira, B.C. Reed, and M. Serafini. Zab: High-performance broadcast for primary-backup systems. *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 0:245–256, 2011.

[64] S. Kaisler, F. Armour, J.A. Espinosa, and W. Money. Big data: Issues and challenges moving forward. In *System Sciences (HICSS), 2013 46th Hawaii International Conference on*, pages 995–1004, 1 2013.

[65] Pradnya Kanhere and H. K. Khanuja. A survey on outlier detection in financial transactions. *International Journal of Computer Applications*, 108(17): 23–25, dec 2014. doi: 10.5120/19004-0502.

[66] Mehmed Kantardzic. *Data Mining: Concepts, Models, Methods and Algorithms*. John Wiley & Sons, Inc., New York, NY, USA, 2002. ISBN 0471228524.

[67] Edwin M. Knorr and Raymond T. Ng. A unified approach for mining outliers. In *Proceedings of the 1997 Conference of the Centre for Advanced Studies on Collaborative Research*, CASCON '97. IBM Press, 1997.

[68] Edwin M. Knorr and Raymond T. Ng. Algorithms for mining distance-based outliers in large datasets. pages 392–403, 1998.

[69] Edwin M. Knorr and Raymond T. Ng. Finding intensional knowledge of distance-based outliers. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, pages 211–222, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. ISBN 1-55860-615-7.

[70] Edwin M. Knorr, Raymond T. Ng, and Vladimir Tucakov. Distance-based outliers: Algorithms and applications. *The VLDB Journal*, 8(3-4):237–253, February 2000. ISSN 1066-8888. doi: 10.1007/s007780050006.

[71] Hans-Peter Kriegel, Matthias S hubert, and Arthur Zimek. Angle-based outlier detection in high-dimensional data. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, pages 444–452, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-193-4. doi: 10.1145/1401890.1401946.

[72] Hans-Peter Kriegel, Peer Kröger, Erich Schubert, and Arthur Zimek. Loop: Local outlier probabilities. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, CIKM '09, pages 1649–1652, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-512-3. doi: 10.1145/1645953.1646195.

[73] J Lewis and M Fowler. Microservices, 3 2014. URL http://martinfowler.com/articles/microservices.html.

[74] F.Y. Edgeworth M.A. Xli. on discordant observations. *Philosophical Magazine Series 5*, 23(143):364–375, 1887. doi: 10.1080/14786448708628471.

[75] Laurens Maaten. Learning a parametric embedding by preserving local structure. In David V. Dyk and Max Welling, editors, *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS-09)*, volume 5, pages 384–391. Journal of Machine Learning Research - Proceedings Track, 2009. URL http://jmlr.csail.mit.edu/proceedings/papers/v5/maaten09a/maaten09a.pdf.

[76] R. B. Marimont and M. B. Shapiro. Nearest Neighbour Searches and the Curse of Dimensionality. *IMA Journal of Applied Mathematics*, 24(1):59–70, August 1979. ISSN 1464-3634. doi: 10.1093/imamat/24.1.59.

[77] Markos Markou and Sameer Singh. Novelty detection: a review—part 2:: neural network based approaches. *Signal Processing*, 83(12):2499 – 2521, 2003. ISSN 0165-1684. doi: http://dx.doi.org/10.1016/j.sigpro.2003.07.019.

[78] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning (Adaptive Computation and Machine Learning series)*. The MIT Press, 2012. ISBN 026201825X.

[79] Yahoo! Developer Network. Yahoo! launches world's largest hadoop production application, 2 2008. URL https://developer.yahoo.com/blogs/hadoop/yahoo-launches-world-largest-hadoop-production-application-398.html.

[80] S. Papadimitriou, H. Kitagawa, P. Gibbons, and C. Faloutsos. LOCI: Fast outlier detection using the local correlation integral, 2003.

[81] S. Ramaswamy, R. Rastogi, and K. Shim. Efficient algorithms for mining outliers from large data sets. *SIGMOD Rec.*, 29(2):427–438, may 2000. ISSN 0163-5808.

[82] Peter J. Rousseeuw and Katrien Van Driessen. A fast algorithm for the minimum covariance determinant estimator. *Technometrics*, 41(3):212–223, August 1999. ISSN 0040-1706. doi: 10.2307/1270566.

[83] S. Sathyavani, T.P. Senthilkumar, and M. Phul. Survey on cloud computing. *International Journal of Computer Trends and Technology*, 9, 2013.

[84] Erich Schubert, Arthur Zimek, and Hans-Peter Kriegel. Local outlier detection reconsidered: A generalized view on locality with applications to spatial, video, and network outlier detection. *Data Min. Knowl. Discov.*, 28(1): 190–237, January 2014. ISSN 1384-5810. doi: 10.1007/s10618-012-0300-z.

[85] J. Shafer, S. Rixner, and A.L. Cox. The hadoop distributed filesystem: Balancing portability and performance. In *Performance Analysis of Systems Software (ISPASS), 2010 IEEE International Symposium on*, pages 122–133, March 2010. doi: 10.1109/ISPASS.2010.5452045.

[86] K.G. Shin and P. Ramanathan. Real-time computing: a new discipline of computer science and engineering. *Proceedings of the IEEE*, 82(1):6–24, Jan 1994. ISSN 0018-9219. doi: 10.1109/5.259423.

[87] Sir Francis Bacon. *Novum Organum*. Mobi Classics. Chicago, IL: Open Court Publishing, 1620.

[88] C. Snijders, U. Matzat, and U.D. Reips. Big data: Big gaps of knowledge in the field of internet science. *International Journal of Internet Science*, 1, 2012.

[89] C. Spence, L. Parra, and P. Sajda. Detection, synthesis and compression in mammographic image analysis with a hierarchical image probability model. In *Mathematical Methods in Biomedical Image Analysis, 2001. MMBIA 2001. IEEE Workshop on*, pages 3–10, 2001. doi: 10.1109/MMBIA.2001.991693.

[90] Jian Tang, Zhixiang Chen, Ada Wai-Chee Fu, and David Wai-Lok Cheung. Enhancing effectiveness of outlier detections for low density patterns. In *Proceedings of the 6th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, PAKDD '02, pages 535–548, London, UK, UK, 2002. Springer-Verlag. ISBN 3-540-43704-5.

[91] Microsoft Technet. Announcing the windows azure hpc scheduler and hpc pack 2008 r2 service pack 3 releases, 11 2011. URL http://blogs.technet.com/b/windowshpc/archive/2011/11/11/hpc-pack-2008-r2-sp3-and-windows-azure-hpc-scheduler-released.aspx.

[92] H.S. Teng, K. Chen, and S.C. Lu. Adaptive real-time anomaly detection using inductively generated sequential patterns. In *Research in Security*

and Privacy, 1990. Proceedings., 1990 IEEE Computer Society Symposium on, pages 278–284, May 1990. doi: 10.1109/RISP.1990.63857.

[93] Y. Thakran and D. Toshniwal. Unsupervised outlier detection in streaming data using weighted clustering. In *Intelligent Systems Design and Applications (ISDA), 2012 12th International Conference on*, pages 947–952, Nov 2012. doi: 10.1109/ISDA.2012.6416666.

[94] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *The Journal of Machine Learning Research*, 9(2579-2605):85, 2008.

[95] Laurens van der Maaten and Geoffrey E. Hinton. Visualizing high-dimensional data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[96] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: Towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55, dec 2008. ISSN 0146-4833.

[97] Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, pages 5:1–5:16, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2428-1. doi: 10.1145/2523616.2523633.

[98] J. Vlasblom and S. J. Wodak. Markov clustering versus affinity propagation for the partitioning of protein interaction graphs. *BMC Bioinformatics*, 10: 99, 2009.

[99] Izhar Wallach and Ryan H. Lilien. The protein-small-molecule database, a non-redundant structural resource for the analysis of protein-ligand binding. *Bioinformatics*, 25(5):615–620, 2009.

[100] David H. Wolpert and William G. Macready. No free lunch theorems for search, 1995.

[101] Chantelle Wood and Russell R. C. Hutter. The importance of being emergent: A theoretical exploration of impression formation in novel social category conjunctions. *Social and Personality Psychology Compass*, 5(6):321–332, 2011. ISSN 1751-9004. doi: 10.1111/j.1751-9004.2011.00353.x.

[102] Reynold S. Xin, Josh Rosen, Matei Zaharia, Michael J. Franklin, Scott Shenker, and Ion Stoica. Shark: Sql and rich analytics at scale. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, pages 13–24, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2037-5. doi: 10.1145/2463676.2465288.

[103] Yuan Yu, Michael Isard, Dennis Fetterly, Mihai Budiu, Úlfar Erlingsson, Pradeep Kumar Gunda, and Jon Currey. Dryadlinq: A system for general-purpose distributed data-parallel computing using a high-level language. In *OSDI'08: Eighth Symposium on Operating System Design and Implementation*. USENIX, December 2008.

[104] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pages 10–10, Berkeley, CA, USA, 2010. USENIX Association.

[105] Matei Zaharia, Benjamin Hindman, Andy Konwinski, Ali Ghodsi, Anthony D. Joesph, Randy Katz, Scott Shenker, and Ion Stoica. The datacenter needs an operating system. In *Proceedings of the 3rd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'11, pages 17–17, Berkeley, CA, USA, 2011. USENIX Association.

[106] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 15–28, San Jose, CA, 2012. USENIX. ISBN 978-931971-92-8.

[107] Osmar R. Zaiane. Introduction to data mining, 1999. URL http://webdocs.cs.ualberta.ca/~zaiane/courses/cmput690/notes/Chapter1/.

[108] Dongsong Zhang and Lina Zhou. Discovering golden nuggets: data mining in financial application. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 34(4):513–522, Nov 2004. ISSN 1094-6977. doi: 10.1109/TSMCC.2004.829279.

[109] Ke Zhang, Marcus Hutter, and Huidong Jin. A new local distance-based outlier detection approach for scattered real-world data. In *Lecture Notes*

in Computer Science, pages 813–822. Springer Science and Business Media, 2009. doi: 10.1007/978-3-642-01307-2_84.

[110] Q. Zhang, L. Cheng, and R. Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1): 7–18, 4 2010. ISSN 1867-4828.

Colophon

This thesis was typeset using LaTeX, originally developed by Leslie Lamport and based on Donald Knuth's TeX. The body text is set in 11 point Egenolff-Berner Garamond, a revival of Claude Garamont's humanist typeface. The body text is set in 11 point Arno Pro, designed by Robert Slimbach in the style of book types from the Aldine Press in Venice, and issued by Adobe in 2007. A template that can be used to format a PhD dissertation with this look & feel has been released under the permissive AGPL license, and can be found online at github.com/asm-products/Dissertate or from its lead author, Jordan Suchow, at suchow@post.harvard.edu.