Institute of Software Technology

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelor’s Thesis 148

Detecting Anomalies in System Log Files using Machine Learning Techniques

Tim Zwietasch

Course of Study: Computer Science

Examiner: Prof. Dr. Lars Grunske
Supervisor: M. Sc. Teerat Pitakrat

Commenced: 2014/05/28
Completed: 2014/10/02

CR-Classification: B.8.1

Abstract

Log files, which are produced in almost all larger computer systems today, contain highly valuable information about the health and behavior of the system and thus they are consulted very often in order to analyze behavioral aspects of the system. Because of the very high number of log entries produced in some systems, it is however extremely difficult to find relevant information in these files. Computer-based log analysis techniques are therefore indispensable for the process of finding relevant data in log files. However, a big problem in finding important events in log files is, that one single event without any context does not always provide enough information to detect the cause of the error, nor enough information to be detected by simple algorithms like the search with regular expressions. In this work, three different data representations for textual information are developed and evaluated, which focus on the contextual relationship between the data in the input. A new position-based anomaly detection algorithm is implemented and compared to various existing algorithms based on the three new representations. The algorithms are executed on a semantically filtered set of a labeled BlueGene/L log file and evaluated by analyzing the correlation between the labels contained in the log file and the anomalous events created by the algorithms. The results show, that the developed anomaly detection algorithm generates the most correlating set of anomalies by using one of the three representations.

Zusammenfassung

Logdateien werden heutzutage in nahezu allen größeren Computersystemen produziert. Diese enthalten wertvolle Informationen über den Zustand und das Verhalten des darunterliegenden Systems. Aus diesem Grund werden sie sehr häufig als erstes im Falle eines Fehlers oder sonstigem Fehlverhalten in einem System konsultiert. Wegen der enormen Anzahl an Ereignissen, die in Logdateien gespeichert werden, wird die Suche nach Einträgen, welche Informationen über einen Fehler im System besitzen, jedoch sehr erschwert. Computerbasierte Analysetechniken werden aus diesem Grund mehr und mehr unerlässlich für die Diagnose von Fehlverhalten aus Logdateien. In dieser Arbeit werden drei verschiedene Repräsentationsformen für textuelle Informationen vorgestellt und evaluiert, welche sich auf kontextuelle Abhängigkeiten zwischen den Daten in der Eingabe beschäftigen. Zusätzlich wird, basierend auf den drei entwickelten Repräsentationen, ein neuer positionsbasierter Algorithmus zum Erkennen von Anomalien vorgestellt, implementiert und mit verschiedenen anderen Algorithmen verglichen. Die Algorithmen werden auf einer semantisch gefilterten Instanz einer beschrifteten Logdatei ausgeführt, die von dem Supercomputer BlueGene/L stammt. Die Ergebnisse der Ausführung werden anschließend evaluiert, indem die Korrelation zwischen den vorhandenen Beschriftungen und den Ereignissen, die als anomal eingestuft wurden, analysiert wird. Die Ergebnisse zeigen, dass der entwickelte Algorithmus in zwei der drei Repräsentationen die insgesamt beste Korrelation zwischen Ereignissen, die als anomal eingestuft wurden und den beschrifteten Ereignissen, erzeugt.

Contents

Chapter 1

Introduction

1.1 Motivation

The increasing importance of analyzing log files in large computer systems makes it necessary to develop automated analysis techniques that are able to extract the relevant information of log files without the need of human input. Log files contain many information about the status of the corresponding system since basically all actions a system does are listed in them. Therefore, they are a great source of information for all kinds of system-related questions. Often, log files are even the only way to get information about errors that occurred in the system [Valdman, 2001]: distributed systems are sometimes impossible to debug. Even if they were, debugging long-running systems that are used for commercial purposes is often not viable. Also, systems that are installed by a customer itself cannot be maintained in real-time, which is why log files might be the only way for the software to communicate.

There are many ways to analyze a log file, starting with simple regular expression pattern search algorithms that find all events with a specified structure. More advanced techniques are based on pattern extraction and data mining that are often designed so that no human interaction with the analysis is necessary (unsupervised approaches). Those more advanced techniques not only search for structural equivalency of events, but also consider, for example, the contextual relation in which an event stands to other events. It is obvious, that this kind of analysis requires the algorithm to have a better understanding of the inner structure and semantic of the object to analyze. Therefore, these techniques have to deal with several problems like analyzing the semantic context without the help of training sets or other manually-generated information. One common approach for failure analysis in computer systems is the concept of anomaly detection. Anomaly detection refers to finding patterns in a set of data that do not correspond to the regular behavior within the set [Chandola et al., 2009]. It is particularly useful for applications that want to make sense of large amounts of information, but can also be used for other applications. Many approaches are using anomaly detection in order to monitor the health of the system and its components[Ray, 2004] or to detect error-prone structures in the developing phase [Hangal and Lam, 2002]. These applications of anomaly detection are crucial for large systems where manual debugging would not be possible or very expensive under normal

conditions. Another large area of application is the detection of unwanted actions by external entities, in particular preventing intruders from accessing important parts of the system [Lazarevic et al., 2003] or detecting different kinds of fraud approaches [Phua et al., 2010] or even suspicious actions of certain groups, like terrorists or robbers [Eberle and Holder, 2009]. Anomaly detection is therefore already researched quite well and there are many methodologies that have been developed to solve different problems in particular as well as generic approaches that try to provide a universal analysis technique in order to improve the general idea of anomaly detection itself. The algorithm that will be developed in this work will extend this knowledge with a new methodology for classifying anomalous data points in a set of data. It will be analyzed whether or not this algorithm is able to outclass existing anomaly detection methods by comparing the results with existing approaches.

The general design of the approach itself is as follows: First, the labeled BlueGene/L log file, which will be used in this study, is filtered by using the *Adaptive Semantic Filter* (Section 3.2) in order to reduce the amount of irrelevant and duplicate events in the data set. Based on that, the three new number representations (Section 3.3) are generated, which transform the event messages into a numeric vector. The number representations merge events, that are temporally close to each other, into one single numeric vector. This enables the analysis of contextual relations in the next steps. Based on this representation, the different anomaly detection algorithms, presented in Section 2.6, are executed. For the position-based anomaly detection algorithm developed in this work, the data set is first clustered by using the X-Means clustering technique and then processed by using the actual algorithm. The last step is the result evaluation (Chapter 5), in which the anomaly detection results for all three number representations will be compared with each other.

1.2 Document Structure

The remainder of this thesis is structured as follows. Chapter 2 presents relevant foundations and technologies that can be used in order to achieve the goals mentioned in the preceding section. Chapter 3 and Chapter 4 describes the envisioned approach, i.e., conceptual ideas and strategies for organizational purposes as well as the implementation details. Chapter 5 presents the results and findings of the study by evaluating and comparing the results of the anomaly detection algorithms. Chapter 6 discusses the related work. Finally, Chapter 7 draws the conclusions and outlines future work.

Chapter 2

Foundations and Technologies

This chapter describes different techniques, tools and technologies that will be used in this work. The listed tools and techniques are mostly used for the implementation and in the general approach explained in Chapter 3 and Chapter 4.

2.1 Types of Anomalies

As already mentioned in Chapter 1, anomaly detection has a wide range of use and is applied to many approaches that try to detect points of interest in a set of data. There are basically three different types of anomalies that have to be considered while analyzing the data [Chandola et al., 2009]:

1. *Point Anomalies* are single, outstanding data points that do not conform with the remainder of the data. As an example, consider Figure 2.1. The green and purple dots are considered point anomalies, since they differ from the normal data points. Detecting point anomalies in a data set is rather simple, since it is sufficient to define some kind of metric or static measurement and apply it to the data. There is no need to set the data points in relation to each other or to know about the structure of the data set as a whole.

2. *Contextual anomalies* on the other hand, are single data points or groups of data, that are outstanding only when seen in context to other, surrounding data points or data structures. These anomalies are much harder to detect, because they are only considered anomalous in a specific situation, like, for example, they are appearing at a position in time or space where they are actually not supposed to be. If these data points were looked at individually, without any contextual relation, they would not be anomalous. To detect these points, it is therefore necessary to have extended knowledge about the structure and behavior of the system. Figure 2.2 provides an example data set with a data point that is anomalous when seen in context to the structure of the remaining data (data point marked red in the diagram).

3. *Collective anomalies* at last, are groups of data that are considered anomalous compared to the remaining data set. The data points in the collection itself do not necessary have

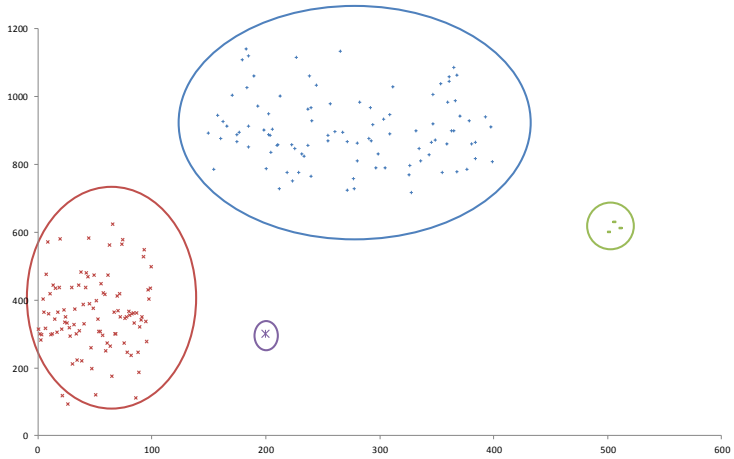


Figure 2.1. Example of anomalous single data points.

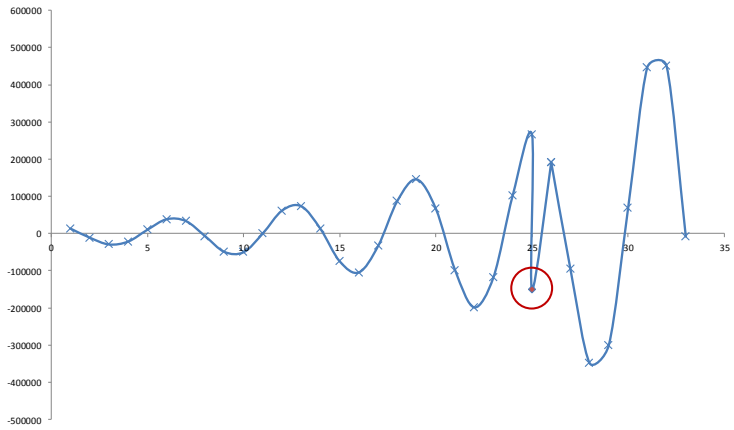


Figure 2.2. Contextual Anomaly - The red data point is anomalous in the context of the data structure.

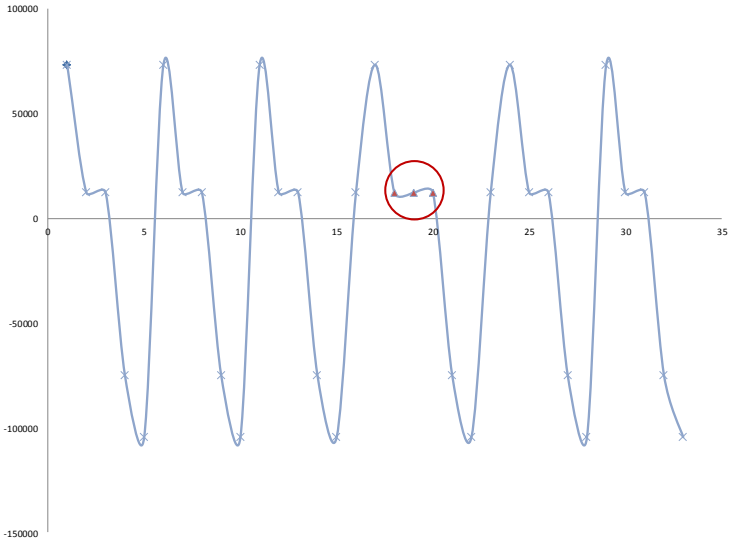


Figure 2.3. Collective Anomalies - The red data points form a collection that is anomalous when compared to the rest of the data structure

to be anomalous, the occurrence as a group at a specific position however leads to an anomalous structure. Figure 2.3 provides an example for this type of anomaly. As can be seen, the collection of data points in the center of the diagram (marked red) form a group of anomalies.

Especially when it comes to analyzing log files, contextual and collective anomalies might be of strong interest for the user. Detecting these structures is related to the concept of *Event Correlation* - according to [Sahoo et al., 2004], it is likely that periodically repeating events or long-term correlation effects (relation with events that happened a log time ago) are frequently occurring in log files. When applied to log files, events that occur within a similar time, close range or are occurring periodically might be related to each other. Especially when events do not contain important data individually but are important for the user because of the constellation they are in, these types of structures are hard to detect. *Example:* Consider the following events:

```
1: INFO Reading file A (MsgId: 1)
...
2: ERROR Bad file descriptor (File A)
...
3: ERROR - Could not process content of file A (MsgId: 3)
```

Each of the above events on their own do not contain enough information to let the user fully understand what is going on: The third event logs the occurrence of an error while processing file A. It is however unclear what error actually caused the system to fail - many other causes like an error in the processing routine could have been the reason for this event. The second event tells the user that a bad file descriptor caused an error regarding file A. Since there is however no ID coming with the error, it is not possible to relate the event to a location in the code. Lastly, the first event states, that the system is going to read file A. This message does whatsoever not contain any information about an error since it was generated before any error occurred. When all three events are related to each other, the source of the error (ID 1, file A), the cause (bad file descriptor) and the consequence (processing error in file A) can easily be found out. This information makes it easy to understand, locate and correct the error. The same applies for anomaly detection: Looking only on single events might not lead to the detection of these correlated events. Therefore, it is necessary to identify correlation structures in the file and group the anomalies according to these correlations.

2.2 Analysis Techniques

There are quite a lot analysis techniques for anomaly detection that have been developed in order to solve different problems in various fields of applications. The following presents two approaches that are related to this work.

▻ Clustering-based techniques

Clustering based approaches perform a categorization of data points in a set in order to retrieve different partitions which can then be analyzed separately [Leung and Leckie, 2005; Chandola et al., 2009]. There are different approaches for clustering-based anomaly detection, basically, they can be split up into four groups.

1. *Density-based techniques* cluster the data into groups according to the density of a certain region. Therefore, each data point in a cluster has to be near a certain amount of other points within the cluster [Leung and Leckie, 2005].
2. *Grid-based techniques* define several cells or hypercubes in the set of data, which then are used to cluster the space into groups of interest [Leung and Leckie, 2005].
3. *Centroid-based techniques* assume that anomalous data points are not within close range of cluster centroids [Chandola et al., 2009]. These techniques detect anomalies by evaluating how far away a single data point is from a cluster centroid. If the distance exceeds a certain threshold value, the data point is considered anomalous.
4. *Nearest Neighbor-based techniques*: Techniques that are based on *nearest neighbor* approaches assume that all normal data points are located close to each other while all

other data points are considered anomalous [Chandola et al., 2009]. Therefore, simple nearest neighbor based approaches are not capable of detecting contextual or even collective anomalies in data sets. However, there are approaches that use complex metrics in order to enhance the process of finding contextual anomalies [Boriah et al., 2008]. There are also approaches that first cluster the data set with an appropriate clustering algorithm, for example the k-nearest neighbor algorithm and prune all clusters that cannot be anomalous. The anomalies are then calculated based on this condensed set [Ramaswamy et al., 2000].

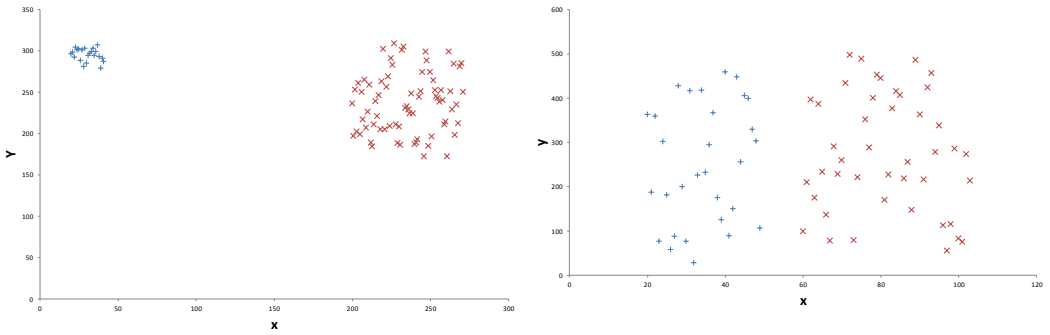
▻ Classification-based techniques

These techniques decide about anomalous data by evaluating whether or not the data instance belongs to a cluster [Chandola et al., 2009]. Therefore, only a special set of clustering algorithms that allow some data points not to be in any cluster can be used for this approach. Classification-based clustering techniques are commonly supervised, since the algorithm needs to learn how to differentiate between anomalous and normal data points.

2.3 Principal Component Analysis (PCA)

> *"In summary, PCA captures dominant patterns in data to construct a (low) k-dimensional normal subspace Sd in the original n-dimensional space. The remaining $(n - k)$ dimensions form the abnormal subspace Sa. By projecting the vector y on Sa (separating out its component on Sd), it is much easier to identify abnormal vectors. This forms the basis for anomaly detection."* Xu et al. [2009]

PCA can be used to structuralize and simplify large data sets with high dimensions. It is commonly used in image processing tasks but also in various other fields of application like information retrieval or text mining where it is important to reduce the raw amount of available data in order to come to a result in an acceptable time. In the context of detecting anomalies in log files, PCA can help to filter frequently repeating patterns in the data which makes it easier to detect anomalies. Basically, PCA finds dimensions that have the largest variances and sorts the dimensions in such a way that the first dimension contains the largest portion of the total variance and the last dimension the lowest portion [Ding and He, 2004]. In order to reduce the dimensionality, the last dimensions in the vector space can therefore be omitted after PCA has been applied, since those dimensions contain no valuable information. As a requirement for PCA, it is necessary, that all clusters have a lower variance within themselves compared to the variance between the clusters, since this would result in picking the wrong dimension (that contains little or no information at all) as illustrated in Figure 2.4



(a) Little variance within the cluster, high variance between the clusters. The algorithm would order the x-dimension before the y-dimension (which contains no information about the clusters).

(b) High variance within the cluster, little variance between the clusters. The algorithm would order the y-dimension before the x-dimension (which contains more information about the clusters).

Figure 2.4. PCA example for two dimensions.

Therefore, when the principal components analysis is applied to a numeric data set, the first dimensions that will be removed are those that contain little to no varaiance. Therefore, the information loss when removing those dimensions is as little as possible. In fact, the removal of dimensions that have a small variance might even lead to better results since these dimensions consist mostly of noise and would therefore reduce the weight of dimenions that contain more information.

2.4 Clustering

This section will present different clustering techniques that can be applied to a data set in order to detect certain points of interest. The goal is to find a clustering algorithm that is able to reliably cluster the data set without the need of a labeled training set.

2.4.1 K-Means

The k-means clustering algorithm is commonly used for cluster analysis since it is fast and rather easy to implement. The algorithm tries to find groups of data instances with similar size and low variance [Kanungo et al., 2002]. Given a number k and an initial set of cluster centroids, standard k-means first assigns each data instance in the set to the nearest

centroid using the euclidean metric. It then calculates a new centroid for each cluster by computing the means between all data points in the cluster. These two steps are repeated recursively until the centroid for all clusters does not change anymore. The K-Means algorithm is closely related to the k-Medoids algorithm that will be explained below and the *k-center* problem [Kanungo et al., 2002; Agarwal and Procopiuc, 2002] that tries to minimize the overall distance from the center to every other data point. According to [Mahajan et al., 2009], the K-Means problem is NP-hard and therefore no efficient solutions for solving this problem are known. The common procedure for the k-means problem is to iteratively execute the algorithm until a local minimum has been found [Kanungo et al., 2002]. This process is usually referred to as the *k-means algorithm* [Kanungo et al., 2002; MacQueen et al., 1967]. The algorithm is based on the assumption that the best center point for a cluster is the center of the cluster itself [Kanungo et al., 2002]. Of course, this assumption can lead to a high distortion of the center point when some data points in the cluster lie far away from the rest of the data points. Several approaches like the k-medoids try to reduce this effect. Because of the popularity of this algorithm, there are quite a few extensions to it that try to improve the precision and performance of the approach, like for example the k-means++ algorithm that tries to find better starting points or the k-medoids algorithm.

2.4.2 K-Medoids

The k-medoids algorithm [Arora et al., 1998] is a variation to the k-means algorithm mentioned above. The clustering algorithm chooses a set of k data points to represent the k clusters in the collection in such a way, that the data points that are not selected to be representative points can be clustered by evaluating the minimum distance between the data point and all the representative objects [Chu et al., 2002]. The data point then belongs to the cluster of the representative point with minimal distance. The general procedure of the algorithm is as follows (cf. [Chu et al., 2002]):

1. Arbitrarily choose an initial set M of k-medoids
2. Create a new set M_{new} by swapping one of the data points in the old set of M with a data set in the collection for which the total distance from the points in the collection to M_{new} is the lowest possible. Then set $M := M_{new}$
3. Repeat step 2 until no point in M changes anymore. Then save the data points and terminate the program.

The algorithm is considered to be more robust compared to the standard k-means approach, mainly because a medoid is less influenced by noise and outstanding data points than the mean score. The disadvantage with this approach is, that the calculation is less time-efficient than the calculation of standard k-means.

2.4.3 X-Means

The X-Means approach extends the standard K-Means by computationally determining the numbers of clusters to be created and additionally by speeding up the whole algorithm itself [Pelleg et al., 2000]. The main problems with the standard K-Means implementation is, that it does not scale well for larger data sets and that the number of clusters have to be specified manually [Pelleg et al., 2000]. There are several other approaches that simply rerun the standard algorithm for different values of k to solve the second problem. This however still takes a lot of time to execute and might result in finding bad, local optima. According to Pelleg et al. [2000], the X-Means algorithm produces better results, since the algorithm is able to dynamically change the number of k in the process. Basically, the algorithm optimizes the Bayesian Information Criterion (BIC) by choosing the number of clusters and the position of the cluster centroids in the data set. The K-Means algorithm is used for selecting the best subset of data points that is then refined in a further step (for more details, see [Pelleg et al., 2000]).

2.5 Tools

Since anomaly detection is a well researched and often used technology, there are a lot of frameworks and tools that can be used for data clustering, data representation, etc. The goal of this work is to create and evaluate different number representations and anomaly detection algorithms. The framework used in this work was chosen to be the Kieker framework (see Section 2.5.3). Weka is used for data visualization and clustering whereas ELKI provides several anomaly detection algorithms that are used for evaluating the different data representations and algorithms.

2.5.1 Weka

Weka[1] is a well known tool for machine learning and statistical analysis that is written in Java and licensed under the GNU General Public License. Weka provides both a java library that can be used within the code itself as well as a graphical user interface to access its various functionalities. It is mostly used for machine learning tasks such as data classification or for educational purposes. Amongst others, its features include data visualization, cross-validation, data clustering and data classification [Holmes et al., 1994]. The Weka Library workbench is completely open source and since April 2000 entirely rewritten in Java, including the implementations of all algorithms [Hall et al., 2009]. In this work, Weka will be used for data clustering as well as for creating the principal components in some approaches and visualizing the result sets.

[1] http://www.cs.waikato.ac.nz/ml/weka/

2.5.2 ELKI

ELKI[2] (Environment for DeveLoping KDD-Applications Supported by Index-Structures) is a framework for knowledge discovery in databases that is mostly used in research areas [Achtert et al., 2008]. Many data mining algorithms are quickly forgotten again after being presented. According to Achtert et al. [2008], the reason for that is, that the implementations for the algorithms are not available and that they cannot be fairly compared with each other. ELKI provides therefore a standardized schema for implementing these data mining algorithms so that they can be better compared with each other [Achtert et al., 2008]. The ELKI framework is open source and written in Java. It consists of a graphical user interface and several data mining algorithms, as well as a Java Library. The algorithms provided are mostly algorithms for data clustering, classification, item-set mining and outlier detection [Achtert et al., 2008]. In this work, the outlier detection algorithms of the ELKI framework will be used for evaluating the constructed data sets and anomaly detection algorithms created.

2.5.3 Kieker Framework

The Kieker framework is a powerful framework that can be used to monitor software runtime behavior and the inner structure of software systems [van Hoorn et al., 2009, 2012]. It was built in order to be modular, extensible and flexible [van Hoorn et al., 2012]. The framework is open source and can be used for Java applications. Basically, Kieker consists of three different parts:

1. The Monitoring components gather the input data that will be observed. By default, Kieker already provides several monitoring probes, e.g. for collecting memory usage or CPU data [van Hoorn et al., 2012].

2. The collected data will then be written into a data storage by using the Monitoring Writer component [van Hoorn et al., 2012].

3. The last part of the framework will then analyze the data by reading the monitoring data collected in the previous step.

The main idea of the structure in Kieker is the concept of analysis chains which can be created by connecting different plugins with each other. This is especially useful for classification purposes or machine learning algorithms since these techniques process a data collection sequentially. The framework is designed so that it produces a low overhead for each processing step and therefore allows many iterations to be executed in the program.

[2] http://elki.dbs.ifi.lmu.de/

2.6 Outlier Detection Algorithms

This section describes the anomaly detection algorithms that will be used for evaluation and comparison. The implementation of the following algorithms are taken from the *ELKI* Framework. The algorithms are applied to the three different number representations described in Section 3.3.

2.6.1 LOF

LOF (Local Outlier Factor) is an outlier detection algorithm that calculates the degree to which a data point is anomalous [Breunig et al., 2000]. The local outlier factor for a data point is calculated by analyzing the neighborhood around that data point which is done by measuring the density in this neighborhood. Basically, the algorithm consists of two steps: First, a local density measure for a data point is calculated to approximate the density of the neighborhood. This is done by computing the inverse average reachability distance between the current data point and all other data points in the neighborhood. Equation 2.1 shows the computation of the local reachability density for a specified parameter k.

$$lrd(a) = (\frac{\sum_{b \in N_k(a)} reachability\_dist_k(a,b)}{|N_k(a)|})^(- 1) \tag{2.1}$$

The *reachability distance* between two data points is defined as the maximum of either the real distance $d(a,b)$ between the two data points or the *k-distance* of b. $N_k(a)$ is a shortcut for $N_{k-distance(a)}(a)$. In principle, the *k-distance* is the distance between a data point and another point, so that a certain amount of neighbor data points have an equal or less distance to the data point than that distance. A more detailed description of the formula can be found in the work by Breunig et al. [2000].
The second step of the algorithm is to create the LOF-value for all data points based on the approximated local density for each point.

$$LOF_k(a) = \frac{\sum_{b \in N_k(a)} lrd(b)}{|N_k(a)|} \cdot \frac{1}{lrd(a)} \tag{2.2}$$

As can be seen in Equation 2.2, the *Local Outlier Factor* is the average reachability distance of all neighbors of *a* divided by the actual reachability distance of the data point itself. Henceforth, if for a data point a LOF is computed that is approximately 1, it means, that the density in the region of that data point is comparable to the average density of the data points in the neighborhood. A value greater than 1 indicates on the contrary, that the density around the data point is lower compared to the average density in the neighborhood. The lower the density gets, the more outstanding the position of this data point will become. Therefore, a relatively high LOF indicates local anomalous data points in that region.

The fact, that the LOF values are computed using only a local section of data points is very beneficial in data sets with varying density regions. Simple outlier detection algorithms would not be able to differentiate between these regions and would therefore misinterpret the regions with lower density as anomalous. Since there are no geometrical functions used in the algorithm, various different metrics that are not required to fulfill the triangle inequality can be applied to the approach easily.
The main problem of the LOF outlier detection algorithm is however the result interpretation. The only output of the algorithm are LOF-values for each data point. A value that is greater than 1 does however not necessarily mean, that the data point is anomalous. In data sets with a high density deviation between the data points, a value greater than 2 might be related to a normal data point. On the other hand, for data sets with almost equally distributed data points, a value greater than 1.2 might already indicate a significant anomaly.

2.6.2 KNNOutlier

The KNNOutlier algorithm is a *grid-based* anomaly detection approach for mining distance-based outliers [Ramaswamy et al., 2000]. The concept of the algorithm is to create partitions in the data sets which are then analyzed separately. As soon as the algorithm has determined, that a partition cannot contain any outliers, the whole partition is pruned. Other partitions are analyzed for outliers which are classified based on the distance to their neighbors.
The main idea for the algorithms is the (slightly changed) definition from Knox and Ng [1998]: *"A point in a data set is an outlier with respect to parameters k and d if no more than k points in the data set are at a distance of d or less from p."* (cf. [Ramaswamy et al., 2000]).
An algorithm that is implemented to exactly match this definition would however face three problems that are difficult to solve:

1. The definition of the parameter d is hard to determine. An approximate good number may only be found by trial-and-error approaches like iteratively repeating the algorithm with different values for d.
2. The algorithm would only be able to classify the data points into two categories (*anomalous* and *non-anomalous*). Determining the degree of outlierness of a data point would be impossible.
3. cell-based approaches that have linear complexity do not scale with a high number of dimensions. Determining the data points by distance d would therefore be complicated for high dimensional data.

To fix these problems, the KNNOutlier algorithm uses the *k-nearest neighbor* approach instead of the raw distances. The outlier ranking is done by sorting the data points based on the distance to its k^{th} nearest neighbor. That way, the user can clearly see in the end

results to what degree a data point is anomalous compared to its neighborhood. As a solution to the expensiveness of algorithms with similar implementations, Ramaswamy et al. [2000] developed a partition-based approach in order to analyze the data set faster. The main idea is, that, in order to retrieve the top n outliers, data points with a very small distance to their k^{th} nearest neighbor do not have to be included in the analysis since there is no way that these points can be outliers. According to Ramaswamy et al. [2000], these distances can be approximated without actually computing the precise values by partitioning the data set which may result in a significant performance increase due to I/O as well as computation savings. The steps for partitioning are as follows (for a detailed explanation, see [Ramaswamy et al., 2000]):

1. **Partition Computation**: A clustering algorithm determines the partitions used in the following steps.
2. **Approximation of the KNN bounds within each partition**: The lower and upper bounds of the k-nearest neighbor distances in the partitions are calculated.
3. **Relevant partition identification**: All partitions that may contain outliers are selected in this step.
4. **Outlier detection in candidate partitions**: In every relevant partition, the outlier detection algorithm is being executed.

By following these steps, a big portion of data points can be filtered out from the data set before running the actual outlier detection routine.

2.6.3 SOF

The spatial outlier factor (SOF) is a spatial outlier detection algorithm that is designed to be efficient and able to distinguish spatial (like locations, etc.) and non-spatial (text, statistical data, ...) dimensions [Huang and Qin, 2004]. Similar to LOF, this algorithm will also produce a *factor*, i.e. a degree to which a data point is anomalous.
The computation of SOF consists of two steps: The first step is to calculate the local density ld for a data point. This is computed through Equation 2.3.

$$ld(a) = \frac{|impact\_neighborhood(a)|}{\sum_{b \in impact\_neighborhood(a)} attr_dist(a,b)} \tag{2.3}$$

where $impact\_neighborhood(a)$ is a list of neighbors for a determined by a reflexive and symmetric spatial relation and $attr\_dist(a,b)$ the distance between a and b in an n-dimensional euclidean space (for more information, see [Huang and Qin, 2004]). The local density of a data point a therefore is the inverse average distance of the neighbor data points to a.
The next step is to calculate the spacial outlier factor for the data point. Equation 2.4 shows

the computation of the SOF value.

$$SOF(a) = \sum_{b \in impact\_neighborhood(a)} \frac{ld(b)}{ld(a)} \cdot \frac{1}{|impact\_neighborhood(a)|} \tag{2.4}$$

If the SOF value for a data point is high, it means, that the local density around that point is low compared to the density of the neighbors of that data point. Therefore, the higher the SOF for a data point, the more likely it is, that this data point is an outlier.

2.6.4 ODIN

The *Outlier Detection using Indegree Number (ODIN)* algorithm uses the k-nearest neighbor graph in order to determine outlier data points [Hautamäki et al., 2004]. The main idea is to calculate a kNN graph with *unidirectional* edges and determine the *indegree* for all odes in the graph. If the indegree is lower than the threshold parameter T, the data point will be marked as an outlier. The kNN graph hereby is a weighted, directed graph that represents all data points as vertexes. Every vertex has exactly k edges to its k-nearest neighbors. The weight of an edge $e_{i,j}$ is the distance between the vertex v_i and v_j, which is provided by the distance function. the ODIN algorithm uses an extended version of this kNN graph where the graph has an edge between v_i and v_j if the original kNN graph has both an edge from v_i to v_j and en edge from v_j to v_i.

Chapter 3

General Approach

This chapter will give detailed insight in the solution finding process as well as the developed techniques used in this approach. The chapter is structured into four parts that explain each step of the general approach individually, starting with the description of the developed number representations, followed by the data filtering, data clustering and anomaly detection part.

3.1 Overview

The general idea of the algorithm is to group different data points into one single instance in order to find anomalous structures in the data set. Common approaches for anomaly detection mostly focus on the analysis of contextual relationships between single data points or, in the case of cluster-based approaches, between data points and clusters. The algorithm developed in this work especially focuses on the relation between different groups of data points and is therefore an approach that tries to detect outstanding *areas* in the data set. The general approach of this work consists of five different steps (see Figure 3.1):

1. The **filtering** step (Section 3.2) reduces the amount of events that are considered in the next steps. This step is crucial for the further analysis because it reduces the amount of redundant messages in the collection. This way, every message in the filtered data set gains more weight and importance and the created blocks of events become more significant.
2. The next step is to create the **number representation** (Section 3.3) of the events in the filtered collection. This is another crucial step in order to get decent results when detecting the anomalous data points. In this step, every event, respective every block of events, is assigned a vector that represents the content of the events. This work will evaluate three different number representation approaches in order to determine which of them is the most applicable for this type of data.
3. Based on the number representations created in the previous step, the data can then be processed by a **data clustering** algorithm (Section 3.4). This step may be skipped by the

external algorithms that will also be used for anomaly detection. The anomaly detection approaches developed in this work will however be executed on the clustered collection. The main reason to cluster the data set is to detect the different types of events.

4. The **anomaly detection** step (Section 3.5) then analyzes the preprocessed data set and detects potential outliers in the collection. For this purpose, several external outlier detection algorithms and one outlier algorithm developed in this work will be used together with the three developed number representations.

5. The final step is the **result evaluation** (Chapter 5), where the different number representation approaches and anomaly detection algorithms will be compared.

3.2 Data Filtering

Finding and removing duplicate or unrelated events that contain no information in a log file as a preprocessing step can be beneficial for further steps like the number representation generation. Merging or removing these events from the log file significantly reduces the number of event messages that need to be analyzed. When filtering a data set, it is very important to recognize possible correlation structures that might occur within the set. If certain events are in relation with each other, it might be crucial to analyze them as a whole. Therefore, deleting these events should be avoided when filtering the data set.
There are several approaches for filtering textual data like log files - the used techniques can basically be categorized into two groups, "*Temporal Reduction*" and "*Spatial Reduction*".

3.2.1 Temporal Reduction

The temporal reduction of events is based on the assumption, that events which were produced within a short time period, are likely to be coming from the same error [Tang et al., 1990]. Events of the same type, coming from the same source are merged into a single event within a short time span. Depending on the particular algorithm, different types of events within close range produce several different merged events. Choosing the time span in which these events are merged needs to be analyzed for each data set individual. Choosing a very long time span might result in merged events that have nothing particular in common while choosing the time span to short, duplicate messages will not be filtered out.

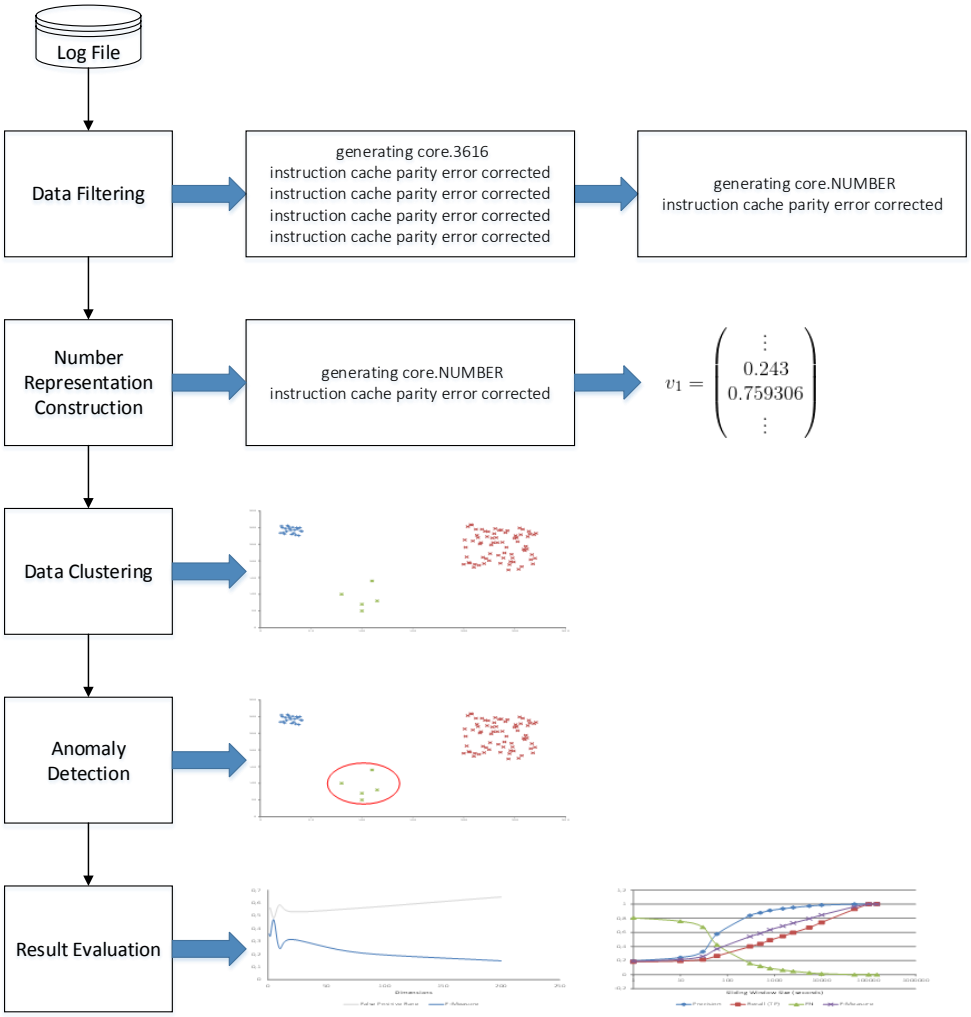


Figure 3.1. General approach.

3.2.2 Spatial Reduction

Distributed systems pose another challenge for filtering event logs. Logs may be gathered across a large number of different machines and centralized for a better analysis of the whole system [Liang et al., 2005]. The log events from these machines can be spatially correlated with each other in order to filter or merge messages that were caused by the same error but were distributed by several nodes in the system. If a failure is occurring on different machines in a distributed system, the error produced by these nodes can lead to multiple other errors in other nodes that were not the source of the error. With temporal filtering alone, these errors cannot be detected since they might be emitted from different nodes.

Spatial filtering can be done for example by defining a causal order of events and creating a graph that represents the event transmissions within the system nodes [Zheng et al., 2009]. By analyzing the relations between the vertexes in the graph, a spacial filtering algorithm might be able to associate the different events with each other and remove spacial duplicates.

3.2.3 Adaptive Semantic Filter

The filtering technique used in this work in order to preprocess the BlueGene/L log file is called the *Adaptive Semantic Filter (ASF)*, which was developed by Liang [2007]. The implementation that is used in this thesis was developed by Pitakrat et al. [2014]. The disadvantage of most of the temporal and spacial filtering techniques is, that they require a deep understanding of the domain they are used in and that they often need a human operator in order to deal with new types of events [Liang, 2007]. Furthermore, simple techniques that use thresholds for identifying temporal or spatial relations are often not suitable to handle more complex situations and can produce wrong results [Liang, 2007].
The procedure of the ASF-filtering method basically consists of three steps: The first step is to create a keyword dictionary that contains all keywords contained in the input file. The keywords can be iteratively appended to the dictionary while filtering the data set. All encountered event descriptions will be normalized using several transformation rules like replacing hexadecimal numbers by a keyword or removing punctuations, prepositions or articles.
The second step is to create a number representation for every event, which is a binary vector that contains one dimension for each keyword. Every dimension is related to a particular keyword and the value of a vector at the respective position represents whether or not that keyword is contained in the event description.
The third and last step is to apply the ASF-filter by computing a correlation value that determines the relationship of an event to another event, known as the ϕ-correlation [Liang, 2007; Reynolds and Reynolds, 1977]. Based on this correlation coefficient, the adaptive

semantic filtering is applied. The main idea behind this filtering is, that events that are close to each other but differ more in their descriptions might still be related to each other, whereas events that have the same description but are temporarily far away might already be unrelated since they are likely to come from two different sources.

3.3 Data Representation

A very important element in finding outstanding structures in the data set is to compute a representation for the points within the collection. When gathering information out of textual data like the description of a particular log entry, there are many steps necessary in order to achieve a result that is able to represent the extracted information in an acceptable way. Numerical representations of a text should be able to determine which data points are similar to each other. Good representations therefore maximize the distance between dissimilar data points and minimize the distance for similar or equal text values. That way, classification algorithms are able to distinguish between the different kinds of data structures within a collection.

In this work, three different number representation algorithms are developed (described in the following). The main idea behind each of them is to group multiple events from the log file together and represent them as a single data point. This is done by sequentially reading the log file and storing a certain number of events into a collection (called the *sliding window*) while reading through the data set. When a new event e_{new} is read from the log file, the sliding window contains the new event and all previously read events that are "near" e_{new}. Both the *Tf.Idf number representation* (Section 3.3.1) as well as the *Feature Vector number representation* (Section 3.3.3) allow the sliding window size to vary. Therefore, for these two approaches, the sliding window contains all previously read events that are within a certain time span, the *sliding window size*. If the sliding window size is for example 300 seconds, the sliding window contains all previously read events where $|time\_stamp(e_{new}) - time\_stamp(e)| \leqslant 300$ holds. For the *Random Index representation* (Section 3.3.2), the sliding window size cannot be varied since the number of events grouped together represents the number of dimensions for this data point. Therefore, the sliding window for this approach contains always the same number of events.
Figure 3.2 shows the general process for creating numerical data points (*data instances*). Based on the filtered data set, the program is sequentially reading the data points. For every new event read, the sliding window is updated so that is always contains related events. Next, the data representation algorithms are executed which transform all the data points in the sliding window into one numerical data instance. These data instances are stored and used in the next steps.

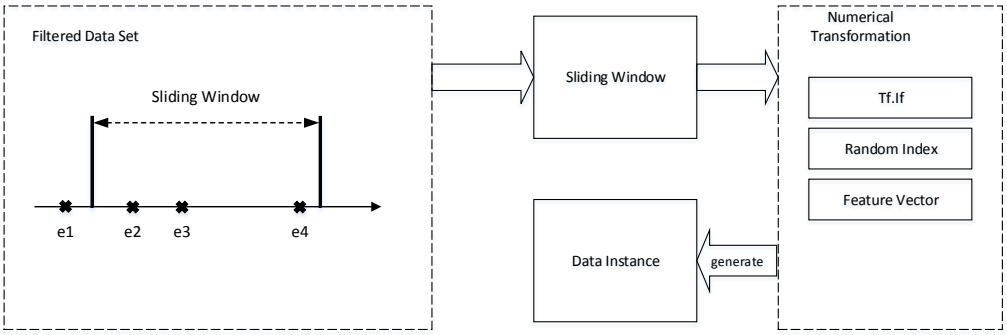


Figure 3.2. General process for creating data instances.

3.3.1 Tf.Idf Representation

A common approach for the numerical transformation of textual data is to represent the textual information as multidimensional Tf.Idf (term frequency, inverse document frequency) vectors. These values are relatively easy to extract, since it is only necessary to create a mapping from the term itself to the overall occurrence of the term in the data set. In a second preprocessing step, the Tf.Idf values can then be computed for each event. The extracted feature vector, which has one dimension for each relevant term, can then be used as a comparator to decide which events are similar and which are not. This is usually achieved by some kind of distance measure, like for example the euclidean distance. The approach of using Tf.Idf vectors for the retrieval of textual information is very common in information retrieval when it is necessary to retrieve textual information that is structured as chunks of data like documents in a large collection [Aizawa, 2003]. The idea of Tf.Idf is to determine the frequency (relative to the collection) of a word in a certain document while words that do not occur in very many documents have a higher Tf.Idf value than words that occur very frequently [Ramos, 2003]. The Tf.Idf value consists of two components: the first component, called term frequency, measures the number of occurrences within the document. The second component, called inverse document frequency, represents the rareness of a term relative to the collection. When multiplying these two values, the intuitive result is a measure of the relevance of a term in a specific document.

The mathematical definition of the Tf.Idf value varies from application to application, but a common definition would be like the following (c.f. [Ramos, 2003]). Given a collection of documents D, an arbitrary word w, and a particular document $d \in D$, the term frequency of a document is calculated as shown in Equation 3.1

$$df_d(term) = \sum_{t \in Terms(d)} equals(term, t) \tag{3.1}$$

where $Terms(d \in D)$ consists of all terms that are contained in the document and $equals(term, t)$ is defined as in Equation 3.2.

$$equals(a, b) = \begin{cases} 0 & ,if\ a <> b \\ 1 & ,if\ a = b \end{cases} \tag{3.2}$$

The document frequency computes how many documents contain a specific word. It is therefore defined as the folowing.

$$df_d(term) = |\{d : d \in D \cap term \in Term(d)\}| \tag{3.3}$$

Finally, the Tf.Idf value is computed as shown in Equation 3.4

$$tf.idf(term, d) = tf_d(term) \frac{|D|}{df_d(term)} \tag{3.4}$$

It is simple to apply the Tf.Idf values to log files by splitting up the file into blocks of events which are then considered documents. An enhanced approach would be to overlap the groups by a certain rate in order to better detect structural anomalies that are located between two blocks. This can be achieved by implementing a sliding window that, beginning with the first event in the log file, groups events that are temporally close to each other. By advancing the sliding window with the next event that has to be processed, it ensures, that all events in the current event collection of the window that have a timestamp difference of more than the threshold value are removed again from the collection. For every event in the log file, this changing collection can then be directly used as a document and the Tf.Idf values can be computed out of that group. Figure 3.3 visualizes the idea of the Tf.Idf computation.

Steps of the algorithm

- **Step 1**: Calculate the term frequency for each term.
- **Step 2**: The sliding window returns a group of temporally close events.
- **Step 3**: The Tf.Idf values are computed for each event in this group
- **Step 4**: The values are added component-wise. This is equivalent to calculating the Tf.Idf values from the group as a whole, but the advantage in adding the values component-wise and calculating them for each term separately is that the values can be computed in a separate preprocessing step.
- **Step 5**: In a post-processing step, the generated numerical data set will be compressed by using the principal components algorithm described in Section 2.3.

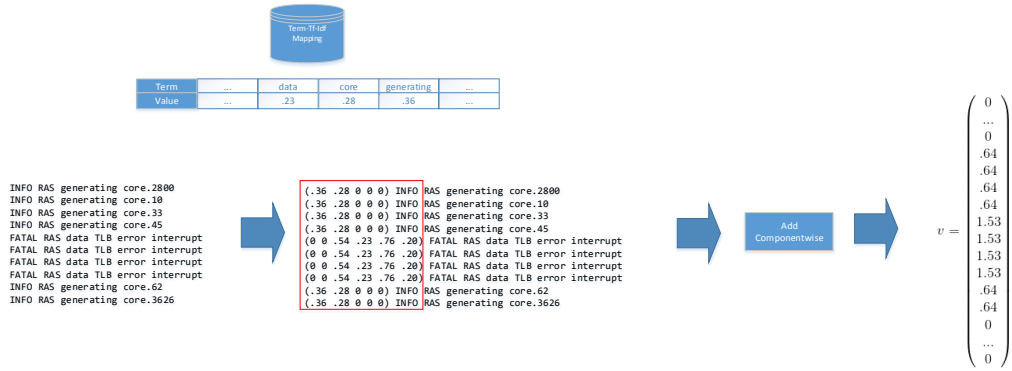


Figure 3.3. Basic idea of the Tf.Idf approach.

Advantages and Disadvantages of Tf.Idf

Advantages:

- events with high Tf.Idf scores impact the values the most.
- The more infrequent, rare terms in a group, the higher the value will get.
- The groups can vary in size - temporal correlations can be considered.

Disadvantages:

- Use for online analysis might be difficult since all vectors and clusters have to be updated when a new word appears.
- Dimensionality of the vector space depends on the number of terms in the data set. The dimensionality might have to be compressed by using techniques like principal component analysis.

Inverted Index Construction

An inverted index is a data structure that is used to map a term to a collection of documents that contain the term (so called *postings list*) [Cutting and Pedersen, 1989]. This data structure is used in the tf.idf approach to efficiently compute the inverse document frequency values. In the context of application in this work, a single event in a log file is here considered a document. Therefore, in the tf.idf algorithm, every event or block of event (depending on the number representation settings) in the log file is seen as a individual document and the values are computed individually for each event(-block).

3.3.2 Random Indexing Representation

The Tf.Idf representation above can only be used for larger data sets, if the number of dimensions is significantly reduced, for example by using the principal component analysis or by using special processing algorithms for sparse data. Reducing the dimension of the data can be very slow, especially when the data set does not fit into the memory anymore while sparse data always comes with the problem of processing high dimensional data (curse of dimensionality).

The random index representation solves the problem of high-dimensional vectors. The random indexing creates only one single value to represent an event in the log file (in contrast to the Tf.Idf approach that creates $|Terms|$ dimensions for one event). Therefore, for analyzing contextual anomalies, several events in the log can simply be combined by creating a vector where each dimension represents one event. The number of dimensions is then related to the block size for which the events are grouped into one instance.

Given a set of events E, the algorithm first creates a random number R_t for all terms in the collection. After removing irrelevant terms (e.g. frequent terms or stop words) from this mapping, the algorithm then computes for each event a representative number by adding all the random numbers in the event into one value.

$$R(e \in E) = \sum_{t \in Terms_e} R_t \tag{3.5}$$

After computing these values for all events in the collection, the next step is to group every block of events created by the sliding window into a vector of the size d to enable contextual analysis of the data. The resulting vector v_i can then be used for further steps (clustering, anomaly detection). Equation 3.6 shows the computation of the vector $v_i \in \mathbb{N}^d$ for the block $c_i \in \mathbb{N}$ in the collection.

$$v_i = \begin{pmatrix} R((c_i * d) + 1) \\ \vdots \\ R((c_i * d) + j) \\ \vdots \\ R((c_i * d) + d) \end{pmatrix} \tag{3.6}$$

Figure 3.4 illustrates the process of creating the random index representation.

Advantages and Disadvantages of Random Indexing

Advantages:

- The algorithm produces only one dimension for one event. The dimensionality is therefore independent of the grammar in the data set.

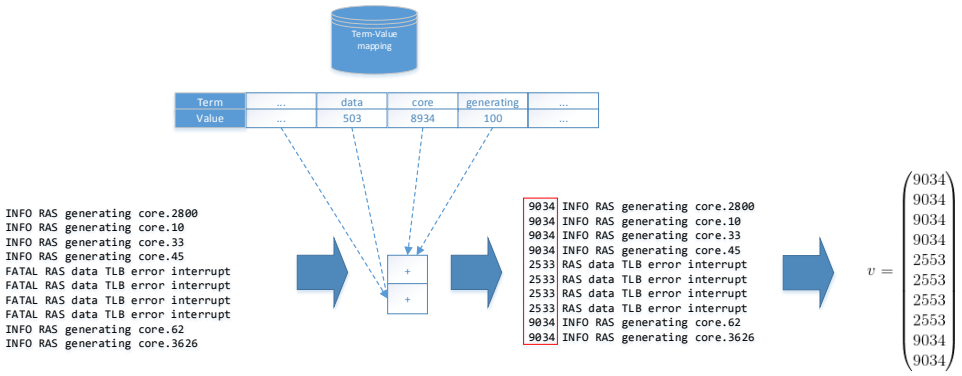


Figure 3.4. Basic idea of random indexing.

- The dimension of a data point in the resulting collection can be adjusted by the user. A higher dimension correlates to the detection radius for contextual anomalies.
- The temporal ordering of events can be considered - it is possible to detect chronologically outstanding anomalies.
- New words do not lead to new dimensions. An online approach can be realized with little effort.
- Equally rare terms are differentiated - Tf.Idf representations thread equally frequent words as the exact same word.
- No sparse data, no need for dimensionality reduction algorithms.
- Other dimensions that are added to the data set have greater impact (e.g. a separate dimension for the severity of an event, or even user-defined new dimensions for any kind of data)

Disadvantages:

- Rare terms have no effect on the score. All terms are equally rare.
- Frequent terms therefore have to be filtered out in a preprocessing step.
- The number of events in one representation has to be static. This is an important disadvantage because events in a filtered log file may vary much in their temporal distance to their neighbors.

3.3.3 Feature Vector Representation

Another approach for representing the events in the log files as numerical values is to choose the dimensionality of the vector space so that it matches the number of event types in the log file. A number in a particular dimension then represents the occurrence of that type of event in the current block. Figure 3.5 visualizes this approach. The red box represents the current position of the sliding window in the log file. The figure demonstrates how the dimensions are retrieved in the log file and how the resulting vectors are generated. In the example, the log file consists of 4 different event types. Therefore, generated vectors for each event has four dimensions. The sliding window contains five events of the third event type and two events of the fourth event type. The resulting numerical representation is therefore the vector $v = (0, 0, 5, 2)^T$. By filtering out event

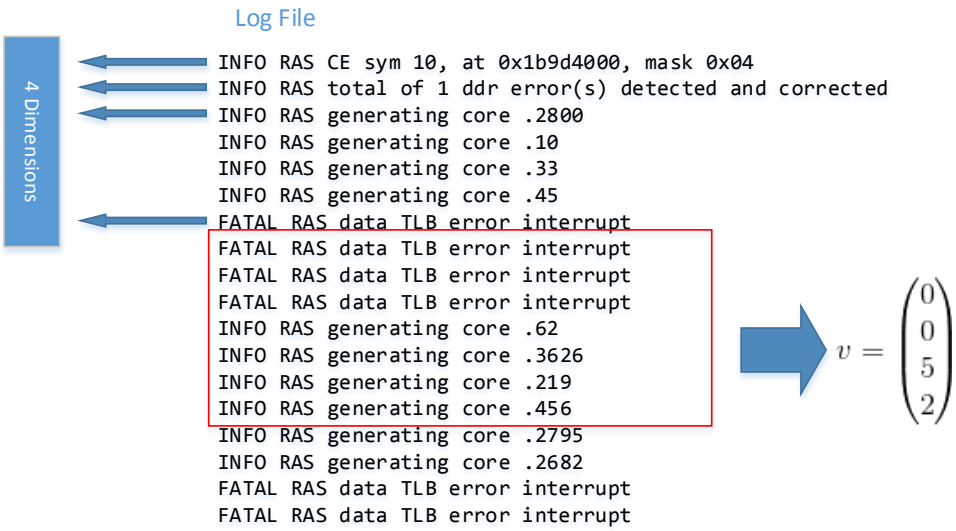


Figure 3.5. General idea of the feature vector representaation. The log file consists of 4 different event types, therefore 4 dimensions are generated. The vector representation is computed for each block of events (sliding window).

types that do not occur a certain number of times in the log file, the number of dimensions can be reduced. The Evaluation in Chapter 5 will filter out event types for which there are less than five occurrences in the log file. However, with only this possibility of dimension reduction, the approach still needs to be post-processed by using the principal components to reduce the dimensions even more.

Advantages and Disadvantages of the Feature Vector Representation

Advantages:

- The groups can vary in size - temporal correlations can be considered.
- The number of dimensions can be reduced by filtering out event types with low occurrence.
- New dimensions can simply be appended to the vector - if a new type of event occurs in the data set, the representation does not have to be rebuilt again.

Disadvantages:

- Dimensionality of the vector space depends on the number of event types in the data set. The dimensionality might have to be compressed by using techniques like principal component analysis.
- The rareness of events cannot be represented with this approach.

3.4 Data Clustering

Data clustering will be used in some of the algorithms analyzed in this work. There are many clustering algorithms available that can be used for the approach in this work, however, not all are equally suited for the task. The following criteria should be fulfilled:

1. The number of clusters are unknown. The clustering algorithm should therefore be able to automatically determine the number of clusters on its own.
2. There are many data points in the data set. Therefore the clustering algorithm has to be fast and scalable. The overhead produced should be as low as possible.
3. The clustering must be able to create a decent categorization of the items contained in the data set.

As already explained in Section 2.4, the X-Means algorithm is able to determine the number of clusters and quickly cluster the data set and will therefore be used for the clustering task.

3.4.1 Advantages of data clustering

The main reason to cluster the data set is to split the events into groups of different event types. A log file usually contains many different kinds of messages that might occur at different frequencies in the collection. This however does not necessarily mean that those

data points are more anomalous than the more frequent occurring ones. By clustering the data points, the goal is to split these messages up so that they can be analyzed individually. This procedure might force the anomaly detection algorithm to value the constellation of the events more than the different types of messages and therefore could consider contextual relationships better.
Example: Consider a system that is iteratively executing three steps:

1. Read input
2. Compute data
3. Generate result

In either of these steps, there are most likely different event messages that are occurring only in this step. If however the computation of the data is the most complex step, there will presumably be much more events that are related to the data computation part in the log. Since the anomaly detection algorithm will especially mark rare events or event constellations as anomalies, it is possible that the events that are emitted in the other two steps will be considered anomalous. By clustering the data points, these three steps can be separated from each other under the right conditions. The clustered data can then be analyzed separately for the three different steps and errors within them can be detected better.

Another reason for data clustering is, that the amount of data that will be compared with each other gets reduced. This could be especially important for log files with a very large amount of events.

3.5 Anomaly Detection

This section describes the approach for the developed position-based anomaly detection algorithm. The algorithm presented in the following is developed in order to determine how important the clusters in log file-like data sets are. The idea behind the concept is, that each cluster defines an area of similar data relations. As explained in Section 3.3, the numerical representations group temporal related events into one data instance. These data instances are meant to represent the contextual difference between every instance, i.e., they represent the *context* of a group of events at a given position in the log file. The context c_n of a data instance di_n is defined by the types of events that are contained in the instance as well as by the temporal relations between these events. Therefore, a cluster contains data instances with similar event type constellations (i.e. instances with similar events), that is, with similar contexts. If a data instance is far away from the cluster center, the hypothesized reason is, that the context of the area defined by the data instance is in some way outstanding. This might be due to rare events, but can also be caused by an outstanding constellation of events within this area. The more unusual the constellation

or the events within the data instance, the farther away it gets from the majority of events with similar contexts. Equally, clusters that consist of only a few data points define areas of outstanding context. Even if the distances to the center is very small, data instances within a tiny cluster have to be considered anomalous too.

Most of the existing outlier detection algorithms analyzed in this work use k-nearest neighbor distances in order to evaluate how outstanding a data instance is. This approach however utilizes the cluster centers as "idealized" area contexts that represent the approximated standard-context for the data instances contained in the cluster. The distance $d(di_n) := di_{ideal} - di_n$ represents how ordinary the data instance di_n in contrast to the "ideal" data instance is. Using the k-nearest neighbors, the calculated distance between a data instance di_n and its neighbors would only represent the density around di_n. If the same context occurs more than once, the local density will get higher in the respective position of di_n. This however does not necessarily mean, that di_n is not anomalous, since reports of faulty processes are usually not reported just once in the whole log file but are reported multiple times. The kNN distance would therefore not represent the outlierness of the data instance but only determine, whether similar context structures have ever happened in the log file. As mentioned before, determining the "ideal" data point will be done by running a data clustering algorithm. Since the K-Means algorithm clusters the data by approximating the center point as an average of all data points in the cluster, this approach might be the best choice for determining those data points.

The proposed algorithm in this section determines the distances to the cluster center for each data instance as described above. Similar to the LOF outlier detection, the algorithm outputs a *degree* to which a data instance is anomalous. The following presents two approaches for an anomaly detection algorithm that uses the cluster center to determine the outlierness of data points.

3.5.1 A Position-Based Anomaly Detection Algorithm

The first approach is to order the data instances in each cluster based on their distance to the cluster center. This order is then used to determine the level to which the data instance is anomalous. The farther away the data point is from the center, the higher the position in the order gets. In order to identify data points in a tiny cluster, the algorithm first computes the *global outlier level (gol)* that indicates how large the cluster is in which the current data point is contained in.
The global outlier level for a cluster C in a data set D is defined as

$$gol(C) = P(x \notin C) = 1 - P(x \in C) = 1 - \frac{|C|}{|D|} \tag{3.7}$$

The outlier level of a data point d in its cluster C_d is then computed as

$$outlierLevel(d) = \frac{position(d)}{|C_d|} gol(C_d) \tag{3.8}$$

The global outlier level returns a higher value when the current cluster C contains few data points and a lower value if the cluster contains many points. It therefore can be seen as a global weight factor for all clusters in the collection. The term $\frac{position(d)}{|C_d|}$ computes the local outlierness of a data point within cluster C_d. The farther away the data point is from the center of the cluster, the higher the *outlierLevel* becomes. Therefore, the closer the *outlierLevel* is to 1, the more anomalous the data point is. Similar to LOF, there is however no constant threshold for *outlierLevel* that determines between anomalous and non-anomalous data points. It is therefore required to specify the threshold or the number of anomalies manually.

3.5.2 Extension: Including the Rareness of Events using Tf.Idf

In extension to the algorithm above, the following approach describes another way of detecting outliers in log files. Since the random index approach as well as the feature vector approach do not consider the rareness of the events, it might be a good idea to include the term rareness in the process of detecting anomalies.
Let $\mathcal{S}$ be the set of data points and $\mathcal{A}$ be the set of anomalous data points with $A \subseteq S$. In order to determine the degree to which a data point is anomalous, two factors have to be considered:

1. The level of outlierness of a specific data point (see Equation 3.8)
2. The textual rareness of an event in the context of all other events

Therefore, the level to which a data instance $v \in S$ is anomalous could be defined as follows:

$$anomalyLevel(v) = rareness(v) \cdot outlierLevel(v) \tag{3.9}$$

where $rareness(v)$ determines the rareness of all terms contained in v and $outlierLevel(v)$ represents a measure of how much the data instance is outstanding compared to the rest of the data.

The rareness of a data point $v \in S$ is defined as

$$rareness = \frac{\sum_{i=0}^{|v|} \sum_{j=0}^{|Terms(v)|} tf.idf(v_{i,j})}{\max_{x \in S} \left(\sum_{i=0}^{|v|} \sum_{j=0}^{|Terms(v)|} tf.idf(v_{i,j}) \right)} \tag{3.10}$$

where $Terms(x)$ represents all relevant terms of a single event and $tf.idf(x)$ represents the term/document frequency as shown in Equation 3.11.

$$tf.idf(term, v) = (\sum_{t \in Terms(v)} equals(term, t)) \cdot (\log \frac{|S|}{|\{x : x \in S \wedge term \in Term(x)\}| + 1}) \tag{3.11}$$

$$equals = \begin{cases} 0 & , if\ a <> b \\ 1 & , if\ a = b \end{cases} \tag{3.12}$$

Since the rareness of events is already included in the Tf.Idf data representation, the extension might be viable only for the two other number representations. The rareness extension adds a third weight to the *outlierLevel* factor and is computed by considering the global outlierness of a cluster, the local position of a data point within that cluster and the rareness of the events contained in the data point.

Chapter 4

Implementation

This chapter describes the general structure of the implementation that was used in this work. The implementation includes every step mentioned in Chapter 3, i.e. data filtering, creating the number representations, clustering the data set, detecting outliers and the evaluation of the results. The code is written in Java and uses the Weka library for the clustering part as well as the Kieker framework as a general structure. The design of the project is mostly determined by the Kieker framework: Each step that is executed on the data set is broken down into an *execution chain*, which consists of different *plugins* that are sequentially executed and connected by the chain. The first plugin in each execution chain is a log reader plugin, which reads the events of the specified log file sequentially into the chain. In this project, there are several different execution chains that each are assigned different tasks. The following presents the code structure and explains several aspects of the implementation.

4.1 Filtering and Preprocessing

As already mentioned in the last Chapter, the first step is to use the *Adaptive Semantic Filter (ASF)* to reduce the number of the log file. For the *Feature Vector* approach, another preprocessing step is to count the different categories contained in the log file while the Tf.Idf approach requires the inverted index data structure, that also has to be created before the number representation construction. Figure 4.1 shows a class diagram of the first preprocessing step. A similar preprocessing chain for the Tf.Idf approach is not illustrated in this diagram but can be seen in Figure 4.2 (*InvertedIndexConstructionPlugin*). As can be seen in the diagram, the filtering step is initialized with the *RegexLogReader* plugin, which uses one or multiple regular expression definitions that can be defined by the user in a separate configuration file to extract the events contained in the log file. To extract the BlueGene/L events that are used in this work, the configuration file defines the following regex command:

```
Regex.Pattern.DefaultBlueGene = (?<label>.).*(?<date>\\ d4-\\ d2-\\ d2)-
(?<time>\\ d2.\\ d2.\\ d2.\\ d*) \\ s(?<node>\\ S*)\\ s(?<ras>\\ w+)\\ s(?<facility>
\\ w*)\\ s(?<severity>\\ w*)\\ s*(?<description>.*)
```

(the keyword *DefaultBlueGene* can be freely adjusted by the user.) By adding more regular expressions to the file, more types of events can be processed. The log file reader reads one event at a time and then passes the event to the next plugin in the execution chain, which is the *BlueGeneAdaptiveSemanticFilter* plugin. this filter plugin applies the ASF filtering and occasionally passes filtered events to the next plugin in the chain, the *LogFileWriterPlugin* which then writes the filtered events into a separate file.

In order to compute the feature vector representation, the next execution chain that is applied to the data is the *CategoryCountExecutionChain*. As the name suggests, this chain counts all existing categories in the log file which is done by sequentially reading through the filtered log file. After receiving a new message from the log file reader, the *CategoryCountPlugin* normalizes the event description and stores the resulting event into a hash-map that consists of $(event, count)$ pairs. Once the reading is done, all stored events that have a higher occurrence than a specified value are saved into a separate file that is used in the feature vector number representation routine.

Similarly, the first preprocessing step of the Tf.Idf approach is computed on the filtered data set by sequentially processing the events that are emitted by the log file reader. In order to compute the Tf.Idf scores, an inverted index that contains of $(term, event)$ pairs is computed in the class *InvertedIndexConstructionPlugin*.

4.2 Creating the Number Representations

After the filtering and pre-preprocessing step is done, the next step is to create the number representations. As illustrated in Figure 4.2, the main execution chain for creating numeric representations is the *NumericTransformationExecutionChain* class. Depending on the *RepresentationType* value, this chain creates any one of the three developed number representations (feature vector, Tf.Idf and random index). First, the chain uses the log file reader *RegexLogReader* to read the events contained in the filtered log file. Newly read events are redirected to the specific number representation where they are further processed.

The random index representation (class *RandomIndexRepresentationPlugin*) creates a static time window with a specified number of events by receiving events from the log file reader. Based on that, the algorithm described in the last Chapter is applied. The feature vector and Tf.Idf approaches use a similar concept, except that they use the *TimeWindow* class for a dynamic number of events based on the time-spans between received events. As a post-procesing step for these two algorithms, the principal components algorithms is connected between the number representation algorithm and the next plugin. The PCA algorithm is contained in the Weka library, therefore the *WekaPrincipalComponents* plugin serves only as an adapter for the Weka algorithm. After creating the numerical representations, the

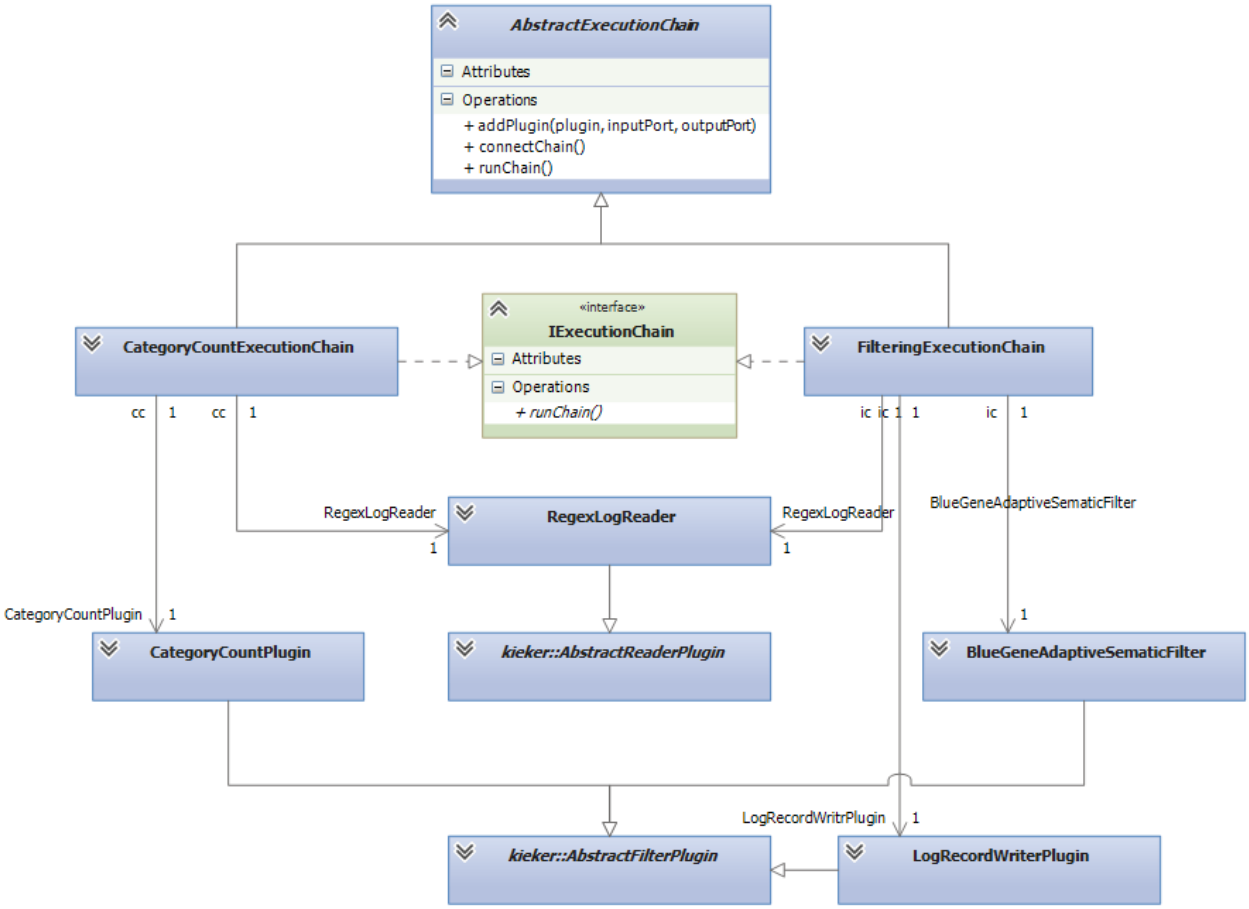


Figure 4.1. Class diagram of the data filtering execution chain and the preprocessing step of the feature vector approach.

chain uses a *NumericalStorePlugin* to pick up all processed events created by the previous algorithm. These events are stored in a local list which is then used in the next step to cluster the data set. For the ELKI algorithms used in this study, this number representation is saved separately and then further processed by using the ELKI interface.

4.3 Clustering and Anomaly Detection

For the anomaly detection approach developed in this work, the next step is to cluster the data. This is done in the *KMeansAnomaly* class (Figure 4.3) before the anomaly detection step. The class uses two enumerations to specify the clustering algorithm and the used distance function. Notice, that not all clustering types and distance functions have been used for the result evaluation. Another anomaly detection method has been developed, which uses the ELKI library for EM-clustering. This algorithm is defined in *ElkiEMAnomaly* but was never used for the evaluation either because the results were simply not very good. The *PerformanceMonitor* class records the memory usage and the time used for the clustering and anomaly detection. After detecting anomalous points with the proposed algorithm developed in the previous chapter, the anomalous data points are then merged with the filtered log file. This is done by the *ResultMerge* class for the developed algorithm and by the *ElkiResultMerge* class for the externally created results from ELKI.

4.4 Result Evaluation

Th last step is to create result representations based on the generated set of anomalous data points. As can be seen in Figure 4.4, there are two execution chains defined for evaluating the result. The *EvaluationExecutionChain* creates label distance, distribution and time window representations that can afterwards be used to create a graph or diagram. The *ROCEvaluationExecutionChain* is used to create the true positive, false positive, true negative and false negative rates, which are then used in various diagrams.

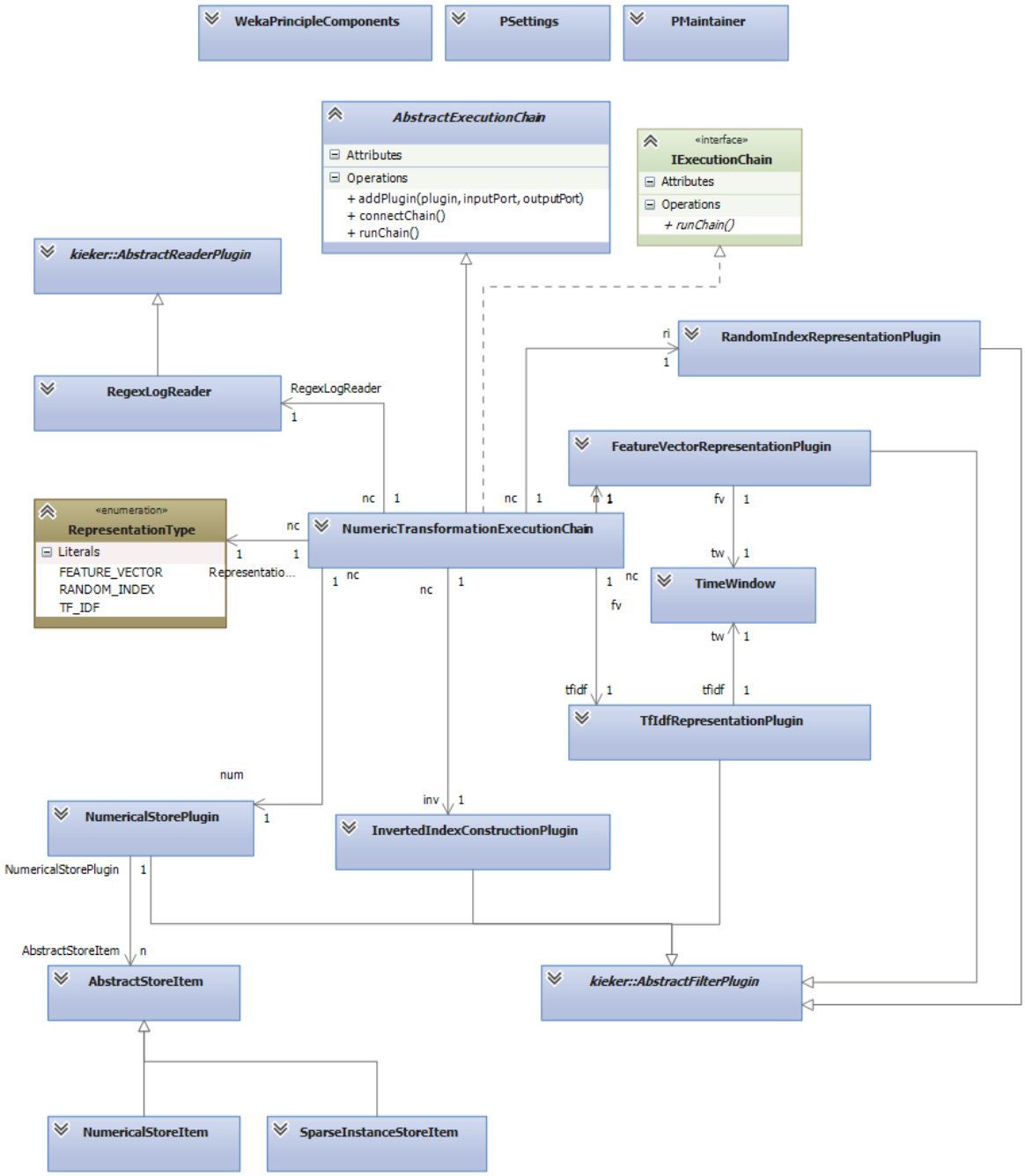


Figure 4.2. Class diagram for the number representation implementation part.

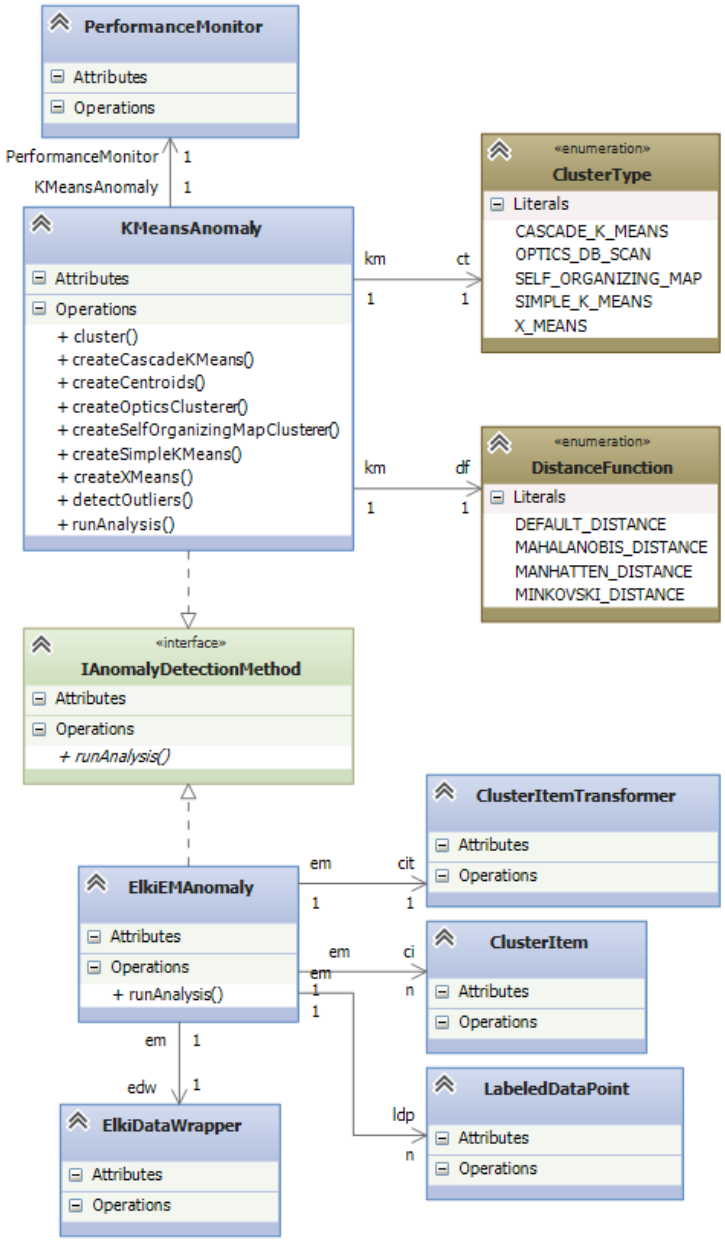


Figure 4.3. Class diagram for the clustering and anomaly detection part.

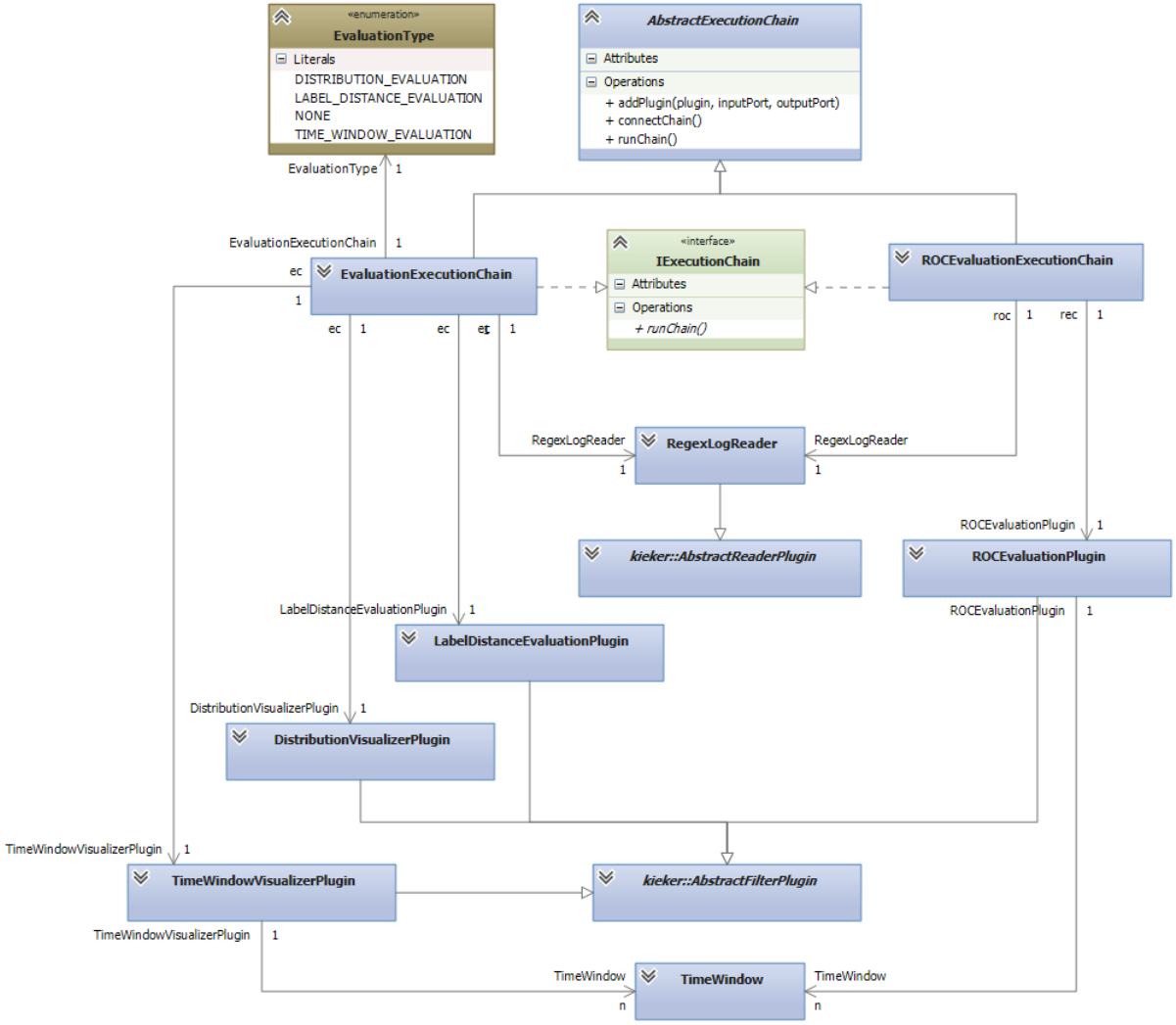


Figure 4.4. Class diagram for the result evaluation part.

Chapter 5

Evaluation

This chapter explains the techniques used to evaluate the results from the anomaly detection. The general idea is to compare the results of the algorithms developed in this work to several other anomaly detection algorithms. The used data set that contains log entries from the supercomputer BlueGene/L provides furthermore certain labels for different kinds of problems in the system. This will be used for the evaluation process to create comparable results. The chapter is basically divided into three parts: The description of the source of the input file, the description of the used evaluation techniques, followed by the actual analysis and comparison of the results generated by the developed algorithms and other anomaly detection algorithms.

5.1 BlueGene/L Architecture and Log Files

This study uses the event logs that have been gathered from the IBM BlueGene/L computer, which is a parallel supercomputer that was installed at Lawrence Livermore National Labs (LLNL) in Livermore, California and listed as the number 1 in the top 500 list of supercomputers in June 2006 [Oliner and Stearley, 2007]. BlueGene/L is based on system-on-a-chip technology developed by IBM and is able to work with 65536 dual-processor nodes which can reach up to 360 teraflops [Gara et al., 2005]. The logging on BlueGene/L is done through the *Machine Monitoring and Control System (MMCS)*. Each mid-plane (two per rack) runs one service node in which one MMCS is executed [Oliner and Stearley, 2007; Adiga et al., 2002]. The events are stored on each node locally until they are pulled and collected with the JTAG-mailbox protocol and then redirected to a central DB2 database [Oliner and Stearley, 2007], which is the source of the log file used in this work. The log file includes all levels of errors and the time can be determined down to microseconds [Oliner and Stearley, 2007]. The system data was collected over 215 days, starting at the beginning of August 2005. In this time, the system produced 4,747,963 event messages which sum up to 1.207 GB of data, containing 348,460 alerts and 41 categories [Oliner and Stearley, 2007]. These were created by executing various tasks, which were amongst others hydrodynamics and climate modeling [Liang, 2007].

An event message in the BlueGene/L log file mostly contains the date and time, the node from which it has been sent, the facility that contains the component where the event has occurred (e.g. MMCS, APP or KERNEL), the event type which specifies through what entity the event has been recorded and the event description. Furthermore, the log file has been provided with labels that indicate several different system states. These labels will be used in the Evaluation (Chapter 5) to compare the anomaly detection results.

5.2 Parameter Adjustment

This section gives a short overview over evaluation results that have been used for parameter adjustment. For each approach that is considered in this approach, there are many parameters to be considered. When it comes to creating the numeric representations for the log files, it has to be determined how many dimensions the result should have. The clustering process also often needs some adjustments like for example the resulting number of clusters or some other threshold value depending on the used algorithm. The anomaly detection algorithms all have unique parameters like, for example, the number of nearest neighbors to consider that also have to be individually adjusted. The result evaluation then needs to decide which and how many anomalies are included in the final result set. All algorithms used generate a ranking of events that are anomalous, so deciding which anomalies are included is a simple task for the result evaluation routine, but deciding how many anomalies to include (i.e. which threshold value indicates an anomalous event) cannot be easily done. Also, in order to get comparable results, it has to be taken care that the parameters are set so that they have the same meaning, if possible.

The following analysis uses the rates defined in Figure 5.1 in order to compare and evaluate the different algorithms with each other and to preset the results of the parameter adjustments as well as general results.

Metric	Equation
True positive rate (Recall)	$\frac{TP}{TP+FN}$
False positive rate	$\frac{FP}{FP+TN}$
Precision	$\frac{TP}{TP+FN}$
F-Measure	$\frac{2 \cdot precision \cdot recall}{precision+recall}$

Figure 5.1. Metrics used for the comparison of the algorithms

5.2.1 Number Representation Adjustments

As mentioned above, the number representations require the dimensions to be set manually. Especially the random index approach cannot be executed without specifying the number of dimensions. The Tf.Idf and feature vector approaches however could be run without specifying the number of dimensions when executed without using the principal component analysis. This would however result in many dimensions that may not contain useful information and would require special analysis techniques in the further steps like, for example, using clustering algorithms that are specifically designed to cluster sparse data.

Getting a good number of dimensions is not easy, since creating the principal components requires a very long time. One way of determining the number of dimensions is to run the algorithms multiple times using another value each time and then take the number that produces the best result. In order to obtain a comparable result, it would be best if the number of dimensions would be constant for each approach, therefore it has to be evaluated whether there is a number that is good for all approaches.

Another parameter that has to be considered when creating the number representations for the Tf.Idf and feature vector approach is the size of the sliding window that is used to group the different events. Those two approaches use a sliding window to determine the number of events that might be temporally correlated and create a multidimensional data point to represent all events currently contained in the sliding window.

The following sections describe the parameter adjustments that have been made for the number representations in order to improve the overall result of the anomaly detection algorithms.

Feature Vector: Varying the dimensions of the PCA output

The feature vector number representation can be varied in two ways:

1. Changing the principal components output dimensions.
2. Changing the sliding window size that determines the time span in which events are grouped into one vector.

When changing the PCA dimensions, the dimensions itself gain more value due to computing linear correlation and orthogonal transformations for the original (sparse) vector space. As can be seen in Figure 5.2, generating 80 dimensions in the PCA algorithm seems to be the best choice. Interestingly, creating only one dimension is the second best alternative. The one dimension approach even seems to have the lowest number of false positive classified events.

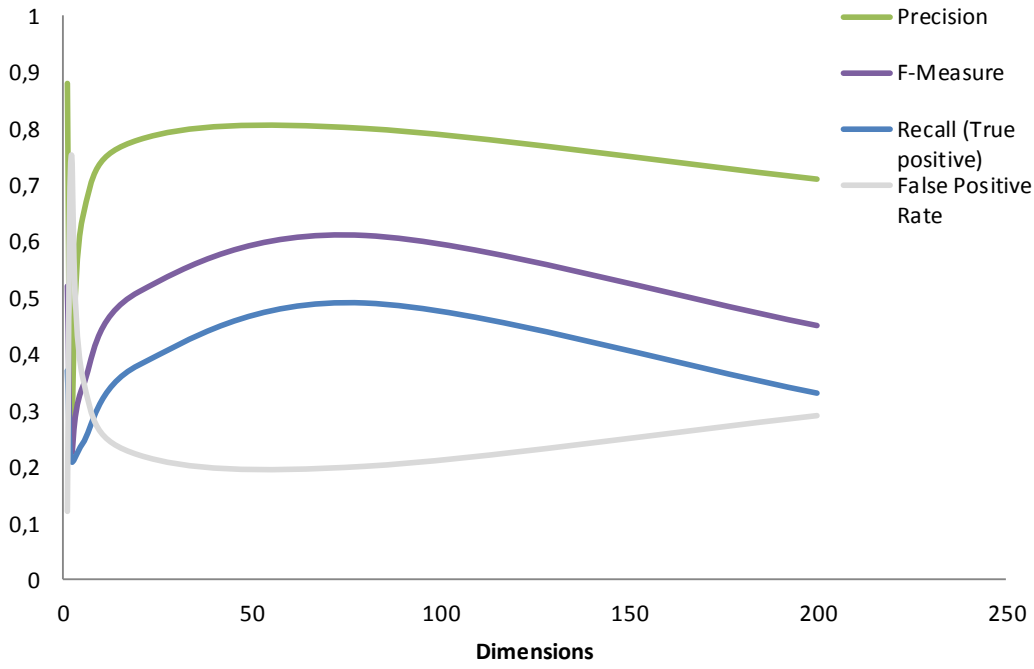


Figure 5.2. Feature Vector representation with 300 seconds sliding window and varying PCA dimensions applied to the position-based anomaly detection method.

Tf.Idf: Varying the dimensions of the PCA output

The Tf.Idf approach can be varied in the same way as the feature vector approach. They both use a sliding window to determine the number of events to group into one instance and use the principal components algorithm to reduce the resulting dimensions. Figure 5.3 shows the results for different dimensions for the Tf.Idf data representation. The diagram shows, that the F-Measure for 80 dimensions (0.65) is the highest, while the F-Measure for 1 dimension (0.27) has the lowest value.

Random Index: Varying the dimensions

The random index number representation does not require dimension reduction algorithms such as principal components. Therefore, there is only one parameter adjustable. The dimensions of the vector space represent the number of events that are grouped together into one data instance. Changing the dimensions therefore means increasing or decreasing the number of correlated events in a data instance. As can be seen in Figure 5.4, the best

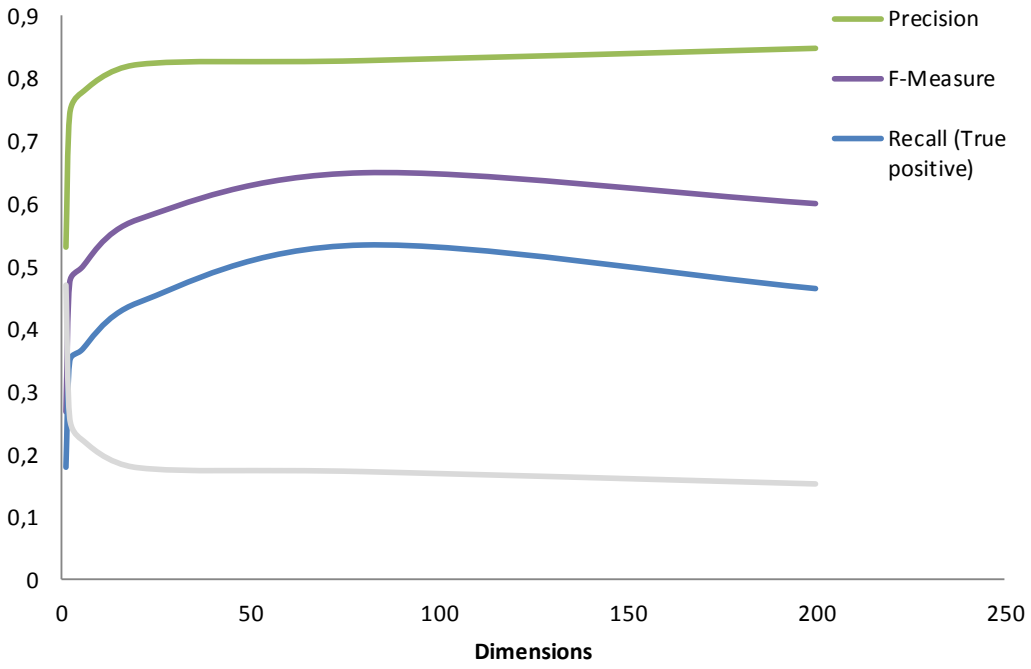


Figure 5.3. Tf.Idf representation with 300 seconds sliding window and varying PCA dimensions applied to the position-based anomaly detection method.

choice for the dimensions is approximately 5 events.

5.2.2 Clustering Parameter Adjustments

Depending on the used cluster technique, there are no parameters to adjust in the algorithm itself. The generated data representation strongly influences the outcome of the clustering. There are however different clustering techniques to choose from that can be used for the task. The chosen algorithm for all further evaluation is the X-Means clustering algorithm because it is fast and produces mostly satisfying results. Figure 5.5 shows the results of the clustering using X-Means and the three different number representations.

5.2.3 Anomaly Detection Parameter Adjustments

The adjustment of the different parameters for the anomaly detection methods is very difficult since they are all different from each other and hard to compare. For the developed

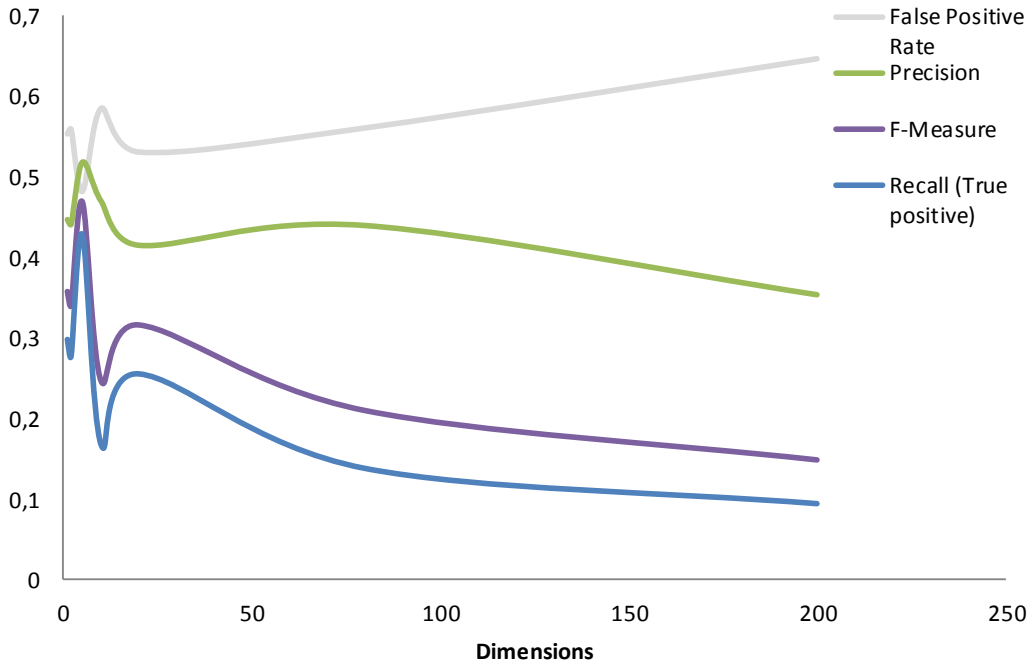
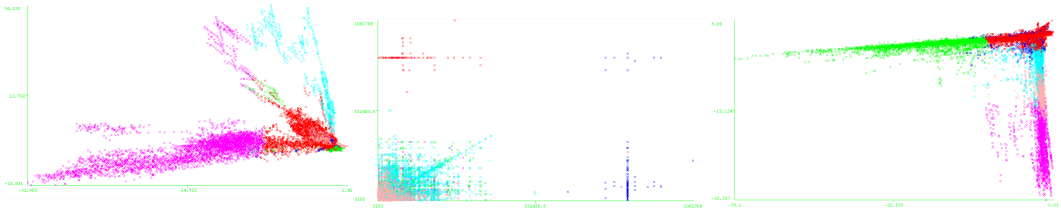


Figure 5.4. Random Index representation with varying dimensions applied to the position-based anomaly detection method.

position-based approach, there is no specific parameter that has to be adjusted. Aside from that, all other algorithms that will be evaluated have at least one parameter that has to be manually defined.

Elki Outlier Parameter Adjustments

Most of the external anomaly detection algorithms are nearest-neighbor based and thus have the parameter k. Increasing k might lead to points being more integrated into the whole vector space, while decreasing it could lead to more individual or little data groups. Figure 5.6 shows the F-Measure for different values of k and different outlier algorithms. The diagram shows, that for all algorithms, choosing a k of 5 to 10 leads to the best results. Therefore, the following evaluation of those algorithms will be done using a k of 5.



(a) Feature Vector representation with 80 dimensions, 300 seconds sliding window. Clustered with the X-Means algorithm that produced 6 clusters. The image visualizes the first 2 dimensions (classified the most relevant by PCA).

(b) Random Index with 10 dimensions. Clustered with the X-Means algorithm that produced 5 clusters. The image visualizes the first two dimensions, all other dimension pairs look similar.

(c) Tf.Idf with 80 dimensions, 300 seconds sliding window. Clustered with the X-Means algorithm that produced 5 clusters. The image visualizes the first 2 dimensions (classified the most relevant by PCA).

Figure 5.5. Cluster Visualization for the different number representations.

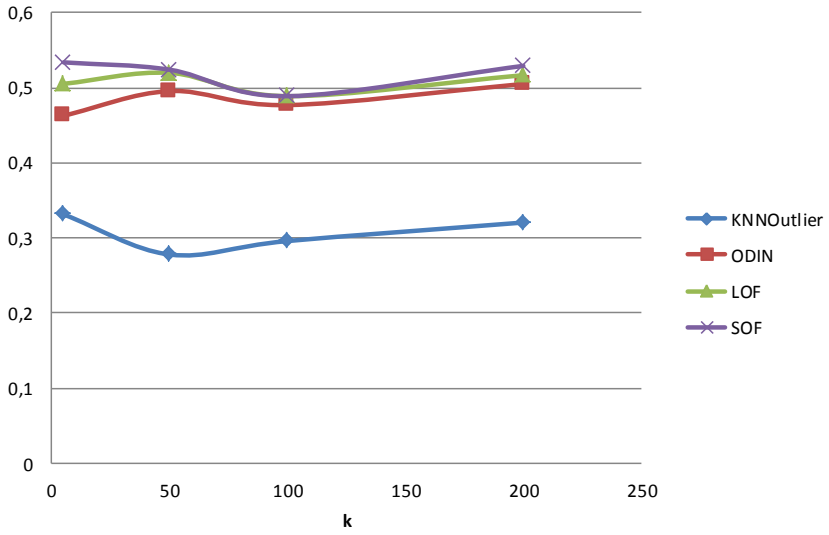


Figure 5.6. F-Measure for different algorithms and different numbers of k that were executed o the Tf.Idf data representation.

5.2.4 Parameter Adjustments for the Result Evaluation

For the determination of true positive, false positive, true negative and false negative events in the result set, a sliding window will be used that determines for every event in the collection, if there is a labeled event within close range. Computing the true and false

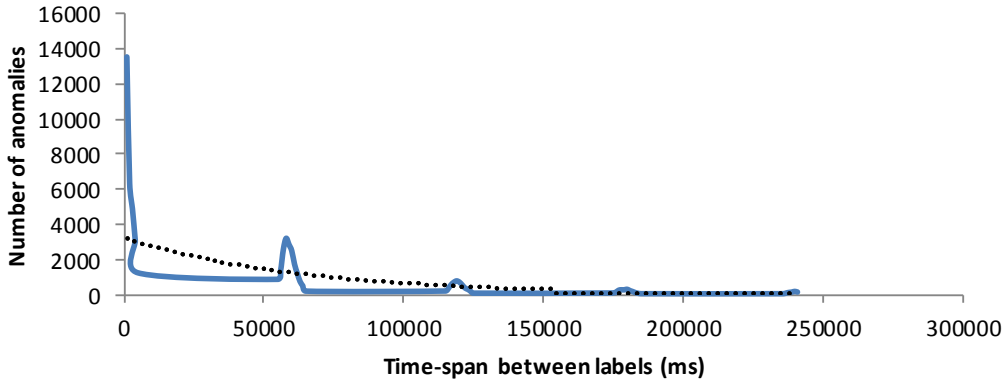


Figure 5.7. Time between Labeled events in the BlueGene/L log file.

positive events cannot be simply done by determining if an "anomalous" event is labeled or not, since anomalous data points are not supposed to be the error messages itself, but instead the events that lead to the erroneous system state.
The problem with analyzing the data set this way is, that the range in which to search for labeled data has to be set manually. It is not necessarily clear in which time span to search - setting the time span too high results in classifying all relevant data points as true positives while setting the time span too low leads to classifying all data points as false positives. One approach to find a good value for the used time span is to visualize the distance between the labels in the log file. This way, it might be possible to determine a good threshold by looking at the average time span between two labels. The threshold for the analysis should then be lower that this average time. Figure 5.7 shows the temporal distances between labeled events in the source log file. The graph shows, that most of the labels occur approximately within a 60 seconds time frame in the log file. As can be seen in Figure 5.8, one of the anomaly detection algorithms that showed relatively good results starts at about a 300x3 second time frame to show good F-measure.

5.3 Evaluation Techniques

This section describes how the true positive, false positive, true negative and false negative numbers are retrieved from an anomaly detection result set. For every anomaly detection algorithm that is being evaluated in this work, a merged result set has been constructed that contains all the events in the original filtered BlueGene/L log file together with the labels and the generated indication for events to be anomalous. The following snippet shows an example of a result file (events shortened).

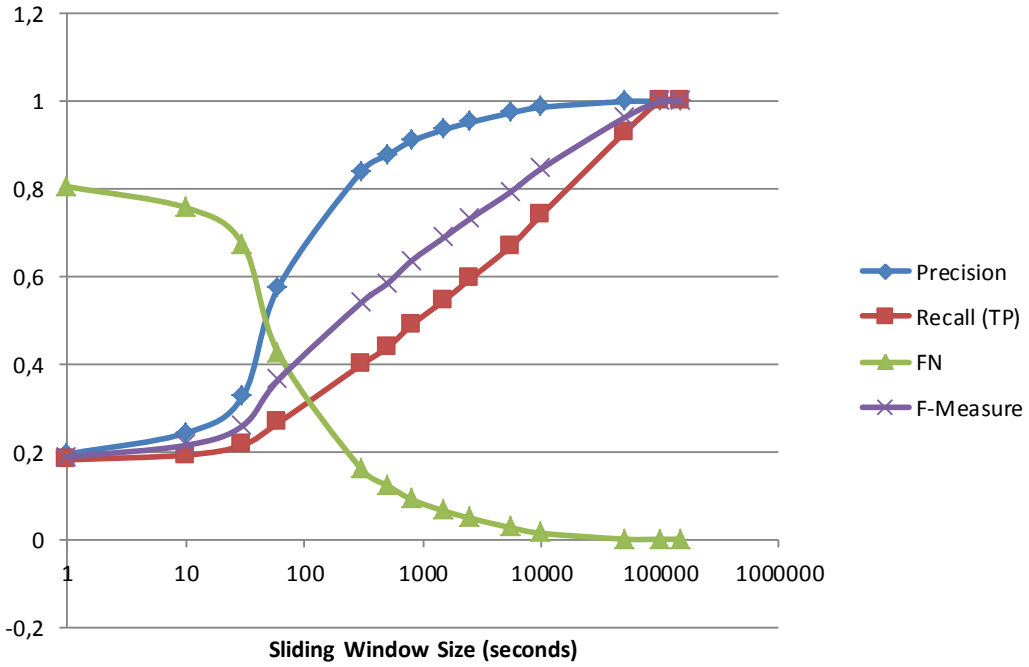


Figure 5.8. Different sliding window sizes for the position-based approach using the feature vector number representation with 80 dimensions and the euclidean metric.

```
...
ANOMALY (0.68) : - 18.05.09.178 KERNEL FATAL RAS  interrupt threshold...0
ANOMALY (0.69) : - 18.05.16.802 KERNEL FATAL RAS  start initialization...0
- 18.05.46.412 KERNEL FATAL RAS  special purpose registers:
K 18.05.56.033 KERNEL FATAL RAS  DDR machine check register: 0x00000000 0x00010010
- 18.06.07.673 KERNEL FATAL RAS  start flushing.........................0
- 18.08.695 KERNEL FATAL RAS  instruction plb error.............0
...
```

The task of the evaluation is now to analyze how the events that have been marked as anomalous correlate with the labels. The following describes how the evaluation extracts these values.

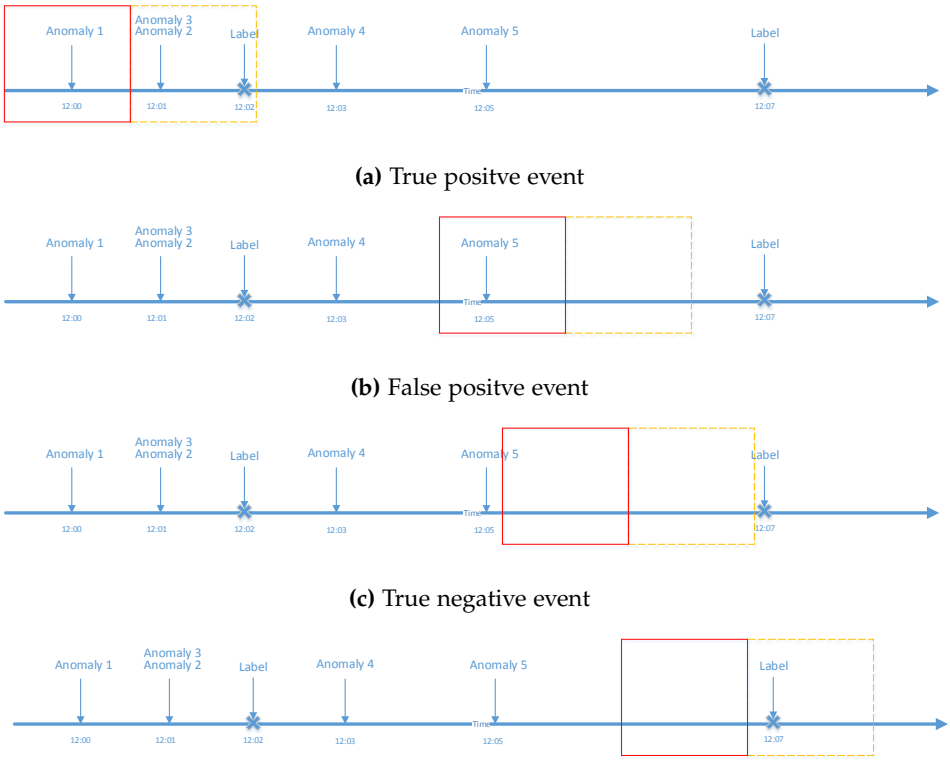
5.3.1 Tp/Fp/Tn/Fn Rates

Unlike for classification algorithms, the anomaly detection results are not necessarily supposed to mark the exact same event that has been labeled. This is because the labeled events are occurring as a result of preceding problems within the system. Therefore, in order to determine these rates, the evaluation cannot simply count every anomaly that had been labeled as true positive and others as false positives. Instead, the evaluation has to look for anomaly-label correlations within several events in a certain time-span. Therefore, the evaluation routine uses two sliding windows of a few seconds (e.g. 300 s) that are sequentially iterating over the log file. These windows contain events that have been generated within a short time. The first time window counts the number of anomalies contained in the current set of events. The number of labels that are contained in the window and additionally those that are contained in the next time window decide on the type of the anomalies: If the windows contain at least one label, the anomalous events are considered true positive. If there are no anomalies in the first time window, the labels count as false negative. For constellations that contain only anomalies but no labels, the false positive rate is increased and sliding windows that neither contain labels nor anomalies are counted as true negative. Figure 5.9 provides examples for each type of event.

The Tp/Fp/Tn/Fn rates for the evaluation will be computed by determining for each sliding window block whether the events contained in it are true positive, false positive, true negative or false negative. For that matter, every label-anomaly pair that is contained in a sliding window block will increase the true positive value, while for all labels in a sliding window block without events that are labeled anomalous, the false positive value will increase. Since sliding window blocks can overlap, every label will be considered and counted only once. If there are multiple anomalies in a block together with only one label, the true positive value will only increase once. Since this does not work for windows that do not contain labels , the true negative and false negative values will increase for each non-overlapping block where no label and no anomalous event is contained in the block, or where one or multiple anomalous events are contained in the block, but no label.

5.4 Comparison of the Anomaly Detection Results

In this section, the anomaly detection algorithms will be compared to each other in order to evaluate which algorithms produce better results for the BlueGene/L fog file as input. The comparison will be done by analyzing for each result set the true positive, false positive, true negative and false negative rates defined as in Figure 5.1 (cf. Pitakrat et al. [2013]; Salfner et al. [2010]). The TP/FP/TN/FN values will be gathered as described previously. In the remainder of this section, the anomaly detection algorithms will be compared with each other by using the three different data representations developed in this work.



(a) True positve event

(b) False positve event

(c) True negative event

(d) False negative event

Figure 5.9. Tp/Fp/Tn/Fn retrieval example. The red box marks the sliding window in which the anomalies as well as the labels are considered while the dashed orange box marks the window in which only labels are considered.

5.4.1 Random Index Data Representation

The random index data representation was created by using a static sliding window of 10 events. Therefore, the dimension of the data set that was used for the anomaly detection algorithms consist of 10 dimensions for each entry. Figure 5.10 shows the calculated F-Measure, Precision, Recall and false negative rates for the random index data representation. As can be seen, the true positive rates for the LOF and ODIN algorithms are significantly higher than that of any other evaluated algorithm and the false negative rates are lower than that of any other algorithm. On the other hand, the precision of the LOF and ODIN algorithms is quite low in comparison to the SOF algorithm. The F-Measure, which shows the harmonic mean of the precision and recall values, indicates, that the LOF and ODIN

algorithms have the overall best score for this data representation. The position-based algorithm that uses the euclidean distance as the distance metric has a slightly higher score than that of the KNNOutlier and SOF algorithm while the position-based algorithm that uses the mahalanobis distance has the overall lowest score. The reason why the position-based algorithm is not as good as the LOF and ODIN algorithm might be, that the k-nearest neighbor distance works better with this data representation, i.e., the static sliding window or the general idea of the random indexing seems to have more beneficial effects for these algorithms compared to the other algorithms. Interestingly, the extended position-based algorithm that includes the rareness shows equal results as the position-based algorithm with the mahalanobis metric.

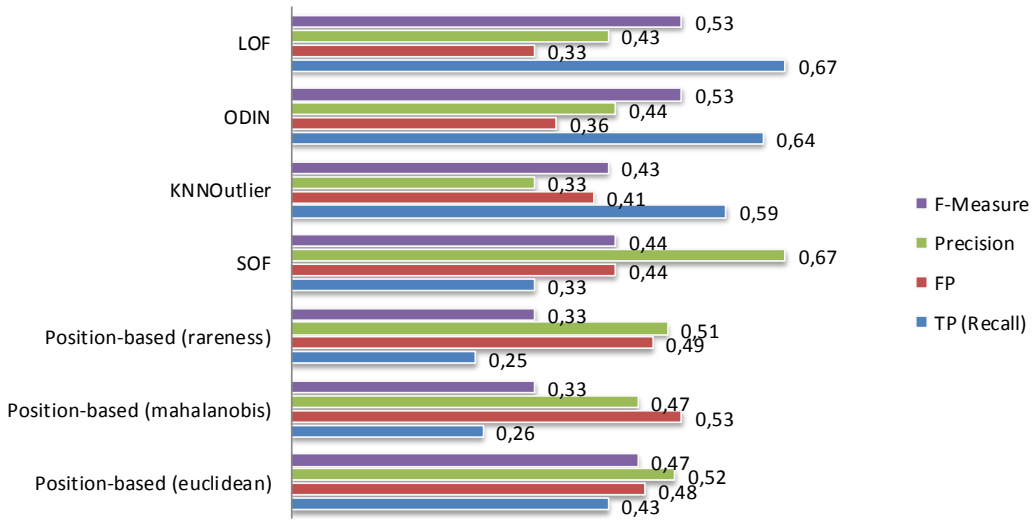


Figure 5.10. Random Indexing Approach - true positive, false positive and false negative rates.

5.4.2 Feature Vector Data Representation

The next data representation that is analyzed is the feature vector approach. The generated numeric representation consists of 80 dimensions, which have been created by first applying the feature vector approach with a sliding window size of 300 seconds. Afterwards, all message types that occurred less than 5 times were removed and ignored in the further steps, which resulted in more than 500 dimensions when applied to the filtered BlueGene/L log file. This result set was then reduced to 80 dimensions by using the principal component analysis algorithm. Figure 5.11 shows, that the precision for the position-based algorithm is significantly higher than the precision for the other algorithms while the recall is the second

lowest. In contrast to the random index representation, the position-based approach seems to be more effective in detecting anomalies based on this representation. The F-Measure for the position-based algorithm that uses the euclidean distance has increased by 0.07 (0.2 increase for the mahalanobis distance) compared to the random index values. The F-Measure for the other algorithms except for the KNNOutlier has however decreased slightly, with KNNOutlier having the smallest F-Measure score of 0.33. Considering, that the construction of the feature vector representation is both more time-consuming as well as more expensive regarding the memory usage, the random index approach seems to be the overall better choice on very large data sets.

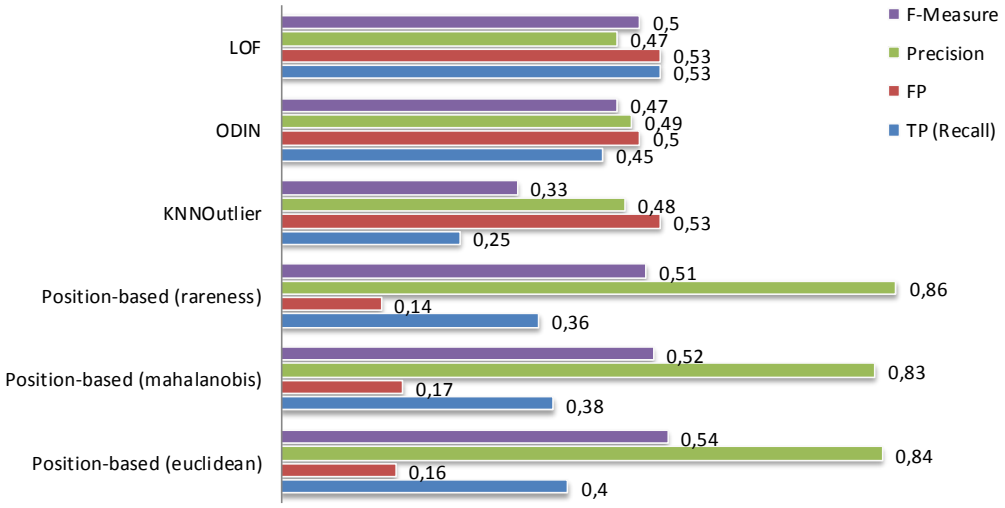


Figure 5.11. Feature Vector Approach - true positive, false positive and false negative rates.

5.4.3 Tf.Idf Data Representation

The last data representation evaluated is the Tf.Idf representation. Figure 5.12 shows the TP/FP/FN rates and the F-Measure for this representation. As can be seen, the euclidean position-based algorithm has the highest F-Measure (0.65) as well as the best precision and false negative rate wheras the SOF algorithm has the best true positive rate but an overall lower F-Measure (0.53). Unlike in the random index representation, the SOF algorithm has a slightly better F-Measure than the LOF and ODIN algorithms. The F-Measure for the euclidean position-based algorithm has the the highest value compared to all other previously evaluated results.

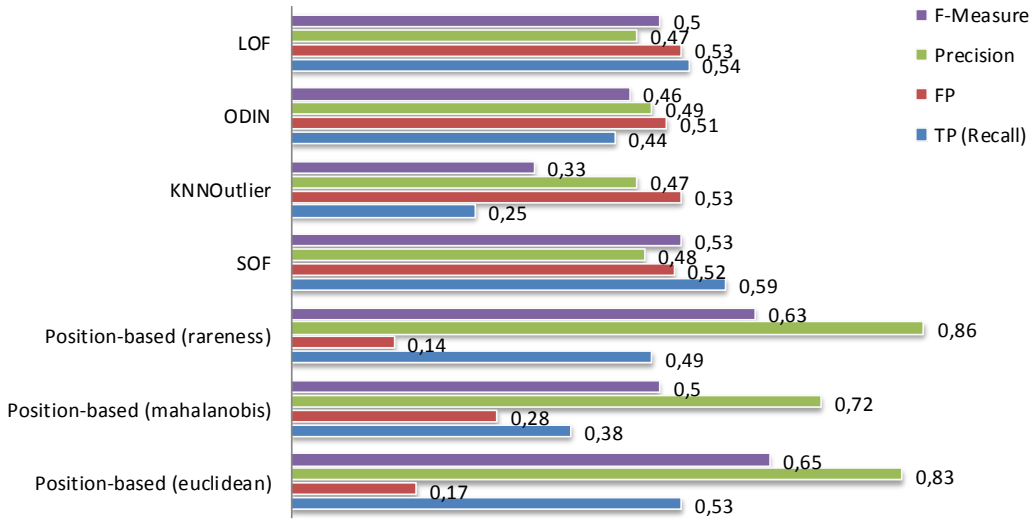


Figure 5.12. Tf.Idf Approach - true positive, false positive and false negative rates.

5.4.4 Overall F-Measure

The overall F-Measure values are shown in Figure 5.13. The feature vector data representation is an obvious bad choice when using the LOF, ODIN and KNNOutlier algorithms, while the random index representation seems to work better for these algorithms than with the developed position-based algorithm. The overall best F-Measure has the euclidean position-based approach using the Tf.Idf data representation, while the KNNOutlier algorithm combined with the feature vector data representation has the overall lowest score.

5.4.5 Evaluation Summary

Table 5.1 shows the used properties and settings as well as the resulting TP, FN and F-Measure for each evaluated algorithm. The position-based approaches were evaluated by using different dimensions and metrics and the results are included in the table as well. As can be seen, the extended position-based algorithm that includes the rareness in the outlier degree only performs well when combined with the Tf.Idf representation. This might indicate, that the rareness of the events does not correlate with anomalous data points as well as the contextual relationships between the events. Table 5.2 lists the execution time and the heap size for each approach. Note, that the heap size for the position-based algorithm include the number representation construction, which used up to 237 MB for the

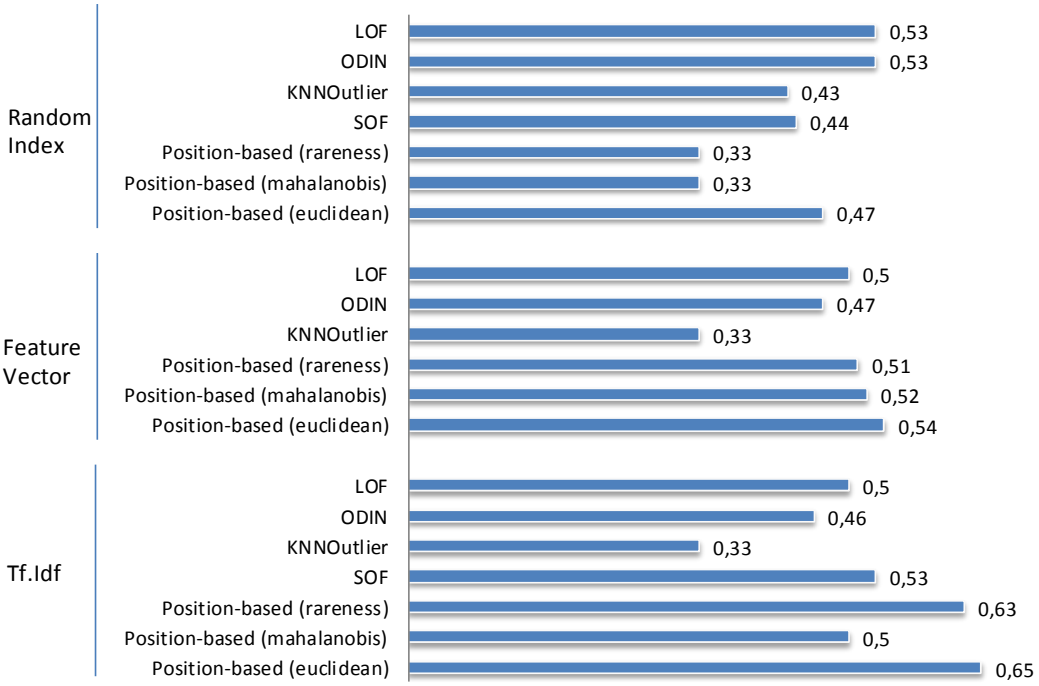


Figure 5.13. Overall F-Measure.

Tf.Idf representation and 248 MB for the Feature vector representation. Figure 5.14 shows the used heap size over time for the different data representations and the position-based algorithm. The numbers used in the diagram in in the table can vary, since they were taken using two different executions. The heap size values in the table have been extracted using an external monitoring software, since the LOF, SOF, ODIN, KNNOutlier algorithms have been executed using the ELKI front end while the actual evaluation was done within the implementation.

5.5 Threads to Validity

This section points out the internal and external threads to the validity of this work. Some of the algorithms that are used in this work use parameters that have a different meaning and thus, they have to be adjusted individually, which makes them naturally difficult to compare. The outliers that were taken from the ELKI Framework (LOF, KNNOutlier,

Table 5.1. Algorithm Overview. SW idicates wheter the algorithm was executed on a dynamic sliding window that considers the time between events or not. The Metric is either the euclidean metric (E) or the mahalanobis metric (M).

Algorithm	Representation	SW	Metric	k	Dim	|SW|	Precision	TP	FP	F
Position-based	Tf.Idf	y	E		80	300	0,83	0,53	0,17	0,65
		y	M		80	300	0,72	0,38	0,28	0,5
		y	E		1	300	0,53	0,18	0,47	0,27
		y	E		2	300	0,74	0,35	0,26	0,47
		y	E		5	300	0,78	0,36	0,22	0,5
		y	E		20	300	0,82	0,44	0,18	0,57
		y	E		200	300	0,85	0,46	0,15	0,6
	Feature Vector	y	E		80	300	0,84	0,4	0,16	0,54
		y	M		80	300	0,83	0,38	0,17	0,52
		y	E		1	300	0,88	0,37	0,12	0,52
		y	E		2	300	0,91	0,38	0,09	0,53
		y	E		5	300	0,64	0,24	0,36	0,34
		y	E		20	300	0,78	0,38	0,22	0,51
		y	E		200	300	0,71	0,33	0,29	0,45
	Random Index	n	E		10	-	0,47	0,25	0,53	0,33
		n	M		10	-	0,47	0,26	0,53	0,33
		n	E		1	-	0,45	0,3	0,55	0,36
		n	E		2	-	0,44	0,28	0,56	0,34
		n	E		5	-	0,52	0,43	0,48	0,47
		n	E		20	-	0,41	0,17	0,59	0,24
		n	E		200	-	0,35	0,09	0,65	0,15
		n	E		80	-	0,44	0,14	0,56	0,24
Position-based (rareness)	Tf.Idf	y	E	5	80	300	0,86	0,49	0,14	0,63
	Feature Vector	y	E	5	80	300	0,85	0,36	0,14	0,51
	Random Index	y	E	5	80	300	0,51	0,25	0,49	0,33
SOF	Tf.Idf	y	E	5	80	300	0,48	0,59	0,52	0,53
		y	E	50	80	300	0,5	0,55	0,5	0,52
		y	E	100	80	300	0,49	0,49	0,49	0,49
		y	E	200	80	300	0,52	0,54	0,48	0,53
	Random Index	n	E		10	-	0,67	0,33	0,44	0,44
KNNOutlier	Tf.Idf	y	E	5	80	300	0,47	0,25	0,53	0,33
		y	E	50	80	300	0,48	0,2	0,52	0,28
		y	E	100	80	300	0,5	0,21	0,5	0,3
		y	E	200	80	300	0,53	0,23	0,47	0,32
	Feature Vector	y	E		80	300	0,11	0,1	0,89	0,1
	Random Index	n	E		10	-	0,33	0,59	0,41	0,43
ODIN	Tf.Idf	y	E	5	80	300	0,49	0,44	0,51	0,46
		y	E	50	80	300	0,51	0,48	0,49	0,5
		y	E	100	80	300	0,49	0,47	0,51	0,48
		y	E	200	80	300	0,5	0,51	0,5	0,51
	Feature Vector	y	E		80	300	0,13	0,15	0,87	0,14
	Random Index	n	E		10	-	0,44	0,64	0,36	0,53
LOF	Tf.Idf	y	E	5	80	300	0,47	0,54	0,53	0,51
		y	E	50	80	300	0,5	0,54	0,5	0,52
		y	E	100	80	300	0,49	0,49	0,51	0,49
		y	E	200	80	300	0,52	0,51	0,48	0,52
	Feature Vector	y	E		80	300	0,14	0,18	0,86	0,16
	Random Index	n	E		10	-	0,43	0,67	0,33	0,53

SOF and ODIN) are mostly using the k-nearest neighbor distances in order to determine anomalous data points. The position-based approach developed in this work however is not based on the k-nearest neighbors and therefore has no such parameter. A possible parameter for this approach would instead be the weight of the global and local outlier factors described in Section 3.5.1. In the study, every algorithm that is based on the

Table 5.2. Execution Time and Memory Usage. SW idicates wheter the algorithm was executed on a dynamic sliding window that considers the time between events or not.

Algorithm	Representation	SW	Metric	Time (S)	Heap (MB)	Used Heap (MB)
Position-based	Tf.Idf	y	euclidean	4156	986	757
		y	mahalanobis	4432	795	624
	Feature Vector	y	euclidean	1511	872	639
		y	mahalanobis	1766	851	579
	Random Index	n	euclidean	100	168	117
		n	mahalanobis	139	185	163
SOF	Tf.Idf	y	euclidean	4707	144	71
	Random Index	n	euclidean	234	288	118
KNNOutlier	Tf.Idf	y	euclidean	4717	162	131
	Feature Vector	y	euclidean	2210	198	129
	Random Index	n	euclidean	223	190	95
ODIN	Tf.Idf	y	euclidean	4957	178	120
	Feature Vector	y	euclidean	2102	206	135
	Random Index	n	euclidean	235	187	98
LOF	Tf.Idf	y	euclidean	4789	181	135
	Feature Vector	y	euclidean	2374	233	209
	Random Index	n	euclidean	226	272	113

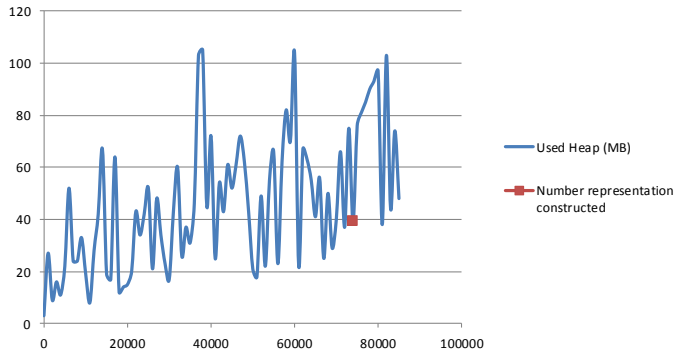
k-nearest neighbor distance was executed with the same parameter for k. However, that does not necessarily mean, that they are comparable with each other in the context of the different techniques used by each algorithm.

Another aspect is the number of anomalies that were considered for each approach. In order to get a more comparable result, the number of anomalies was set to be a constant number (6000 anomalies) for each approach. However, setting the number to a constant also comes with the disadvantage, that the degree to which an outlier is anomalous has to be ignored. It is possible, that the different algorithms produce more precise results by carefully evaluating the outlierness of the events and determining the number of anomalies individually for each approach. This however would lead to an even more hard-to-compare result set since the chances to detect an outlier increases with the overall number of anomalies that are considered.

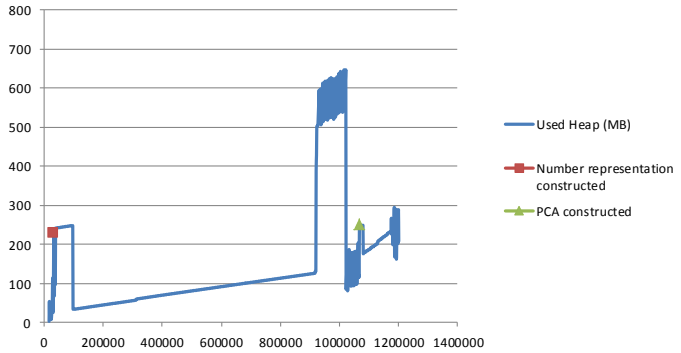
When comparing the number representations with each other, another thread to validity is the difference in the sliding window for the random index approach and the other two representations. The random index representation requires a constant number of events that are grouped into one instance, while the feature vector and Tf.Idf approach can vary the number of events for each data instance. Therefore, the last two representations can consider the temporal difference between the events in the log file, while the random index approach simply groups a constant number of events together into one instance.

A thread to the external validity of this study is, that the only log file that was used was produced by the BlueGene/L supercomputer. Therefore, the algorithms and number

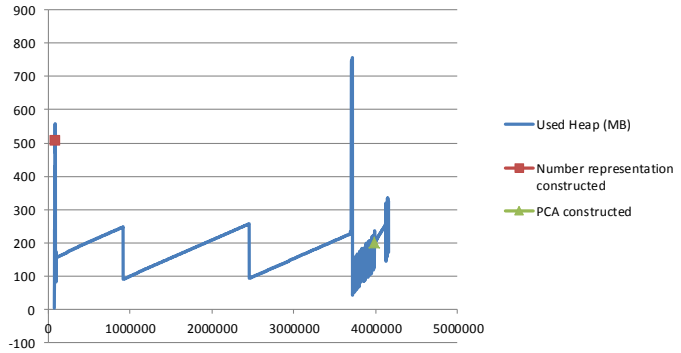
representations have not been tested with other log files, which means, that the results might look differently for other inputs.



(a) Used heap size for the position-based anomaly detection algorithm using the random index data representation



(b) Used heap size for the position-based anomaly detection algorithm using the feature vector data representation



(c) Used heap size for the position-based anomaly detection algorithm using the Tf.Idf data representation

Figure 5.14. Used heap size over time for the position-based algorithm and different number representations.

Chapter 6

Related Work

Fronza et al. [2013] developed a failure prediction algorithm for log files that uses support vector machines and Random Indexing (RI) similar to the data representation researched in this work. Originally, the concept was proposed by Sahlgren [2005], who used random indexing as an alternative to the latent semantic analysis. His research found, that random indexing is an efficient and scalable alternative to the standard method [Sahlgren, 2005]. The main advantage of the RI representation in his approach is, that no co-occurrence matrix has to be built and no dimension reduction phase has to be done which lead to a significant speedup of the procedure. The original construction of the random index vectors (called *context vectors* in the study) is to first build atomic *index vectors* for each context (like documents or words) that are unique, high dimensional vectors with varying $+1$ and -1 entries. While scanning the text, these vectors are added to the context vectors of the particular context. This produces a pseudo-unique vector for each context that can be retrieved iteratively by scanning the text. The construction of the random index vectors therefore can be seen as a reversal to the traditional approach where first the co-occurrence matrix was calculated and afterwards the context vectors were extracted from the matrix.

The concept of the Tf.Idf data representation used in this work is a well-known technique which is very common in textual information retrieval tasks [Forman, 2008; Blei et al., 2003]. Han and Karypis [2000] developed a supervised centroid-based algorithm for document classification that, based on the tf.idf representation and the cosine measure, classifies a document collection into different classes by determining the similarity of a document and the k cluster centers in the data set. The document is then assigned to the class to which it is the most similar. The hypothesis of this study as to why the centroid-based algorithm performed well in comparison with other algorithms, is that the centroids represent the average behavior of the documents contained in it. By classifying a document based on its similarity or distance to each centroid, the algorithm measures how much the document fits into this average behavior. This process allows for dynamic adjustments for classes with different densities and is able to consider the dependencies between different terms within a class. This is similar to the explanation given in this work, which is, that the cluster centroids of the position-based algorithm represent the average contextual relationship and

behavior between different events in the contained cluster. The farther away a document is from any cluster center, the more outstanding, i.e. anomalous it becomes.

The idea of distance based outliers was originally developed by Knox and Ng [1998], who defined outliers to be data points for which there are less than k neighbor data points within a distance d. The two presented algorithms in their study where the first one was a simple implementation for their definition with a time complexity of $\mathcal{O}(\delta N^2)$ with δ being the number of dimensions and N the number of data points. The second algorithm was a cell-based approach whose complexity could be reduced to $\mathcal{O}(c^\delta + N)$ where c is inversely proportional to the cell length $\frac{d}{2\sqrt{\delta}}$ with d being a parameter for the algorithm. The distance-based approach has been picked up by various other studies, including the KNNOutlier algorithm mentioned in Section 2.6.

In contrast to the existing work, this study focuses on the contextual relationship between textual information in log file-like input files. Therefore, the focus lies more on the interpretation of the in this work developed umber representations, which are designed to group temporally close events into one single data point. The position-based anomaly detection algorithm presented in this work was developed in order to analyze whether the number representations of this work arrange the data points around different centroid points that indicate normalized instances for data point relations. In contrast to that, most of the anomaly detection algorithms that were considered in this study are based on the density of regions in the data set or are otherwise utilizing the k-nearest neighbors for determining the local outlierness relative to neighbor points.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

With increasingly larger systems and workflows, log files increase in size and complexity as well. A manual search for system errors in log files therefore becomes continually superfluous, while automated search algorithms become more and more indispensable. However, the contextual relationships between log file events and the importance of the context between them itself makes it very hard to algorithmically determine relevant events in a log file. The goal of this work is to develop and analyze techniques that might ease the process of finding these relations in a data set. In this work, three different numerical representations have been developed and applied to the BlueGene/L log file. All three representations are designed to represent multiple events together as one single data instance in the vector space. This enables an implicit contextual analysis of the data points by all outlier algorithms that are executed on these data representations. Based on the data representations, various existing anomaly detection algorithms and one new approach have been analyzed and compared with each other. The *Random Index* representation, which uses randomly generated numbers in order to represent the different types of events, enables a data representation that can be computed fast and efficiently and does not need dimension reduction techniques as a post-processing step. The Tf.Idf data representation uses the popular Tf.Idf values to create the vector space. The resulting dimensions for this approach are however proportional to the number of unique terms in the log file, which makes dimension reduction in the post-processing step necessary. The same holds for the third data representation, the *feature vector* approach. This approach creates a new dimension for each encountered type of event and represents an instance by adding the vectors of all included events into a new vector. The last two data representations are able to vary the number of events that are contained in a single data instance, which has the advantage, that the temporal difference between events in the log file can be considered. The random index representation however uses a constant number of events for each data instance. This approach is therefore not able to consider the time between events but is, on the other hand, able to consider the order of the events within the data instance.

The developed anomaly detection algorithm is a clustering-based approach that orders all data points according to their distance to the cluster center. The resulting outlier factor is

then computed based on that distance, as well as the size and outlierness of the cluster in which the data point is included in. Since the feature vector and random index approach are however not able to consider the rareness of an event, an extension to this position-based algorithm was presented, that uses the rareness of an event to compute the outlier degree. Compared to the other two representations, the random index approach resulted in an overall lower F-Measure for the position-based algorithm, while the KNNOutlier and LOF algorithm delivered the best results using this data representation. The overall best results were computed using the position-based algorithm combined with the Tf.Idf data representation and the euclidean metric.

The evaluation of the anomaly detection algorithms indicates, that the developed algorithm has an advantage to the other algorithms (when using the feature vector or Tf.Idf data representation), which are mostly using the k-nearest neighbor distances to evaluate their outlier degree.

The hypothesized reason for that is, that for log files like the BlueGene/L log file, the center of the cluster might refer to some kind of normalized event that represents the type of events that this cluster contains. The farther away a data point is from that center, the more anomalous it becomes. Using kNN instead, the data points are locally compared with their neighbor data points and the distance from a data point to its neighbor points is used for outlier detection. This however only measures the local outlierness of data points in context to their neighbors. In log files, the occurrence of similar events in the data set does not necessarily mean, that these events are non-anomalous, since the same error at different times is no indicator for an event to be non-anomalous. Since the position-based approach computes the distance to a center data point instead, the occurrence of similar events does not have such a high impact on the distances compared to the k-nearest neighbor approach.

7.2 Future Work

This study has shown, that the use of centroid-based approaches for detecting outliers is a good choice for analyzing log files. The random index data representation currently uses a static sliding window that consists of a specified number of events. A future improvement for this representation would therefore be to extend the algorithm so, that the number of events contained in an instance can be varied for each data point and to evaluate how much this change affects the results of the different anomaly detection algorithms. Another extension to this work is to compare more outlier detection algorithms with each other and to determine whether centroid-based approaches create generally better results than k-nearest neighbor approaches when using the provided data representations. A possible issue of the position-based algorithm presented in this work is, that the clustering step of the algorithm can be sensitive to outliers in the data set. Another preprocessing step in

order to decrease the impact of these outliers may therefore result in a better detection of anomalies in the data set. For example, the program could be executed twice, where the results of the first run is used to remove the anomalies from the data set while clustering the data in the second execution.

Bibliography

[Achtert et al. 2008] E. Achtert, H.-P. Kriegel, and A. Zimek. Elki: a software system for evaluation of subspace clustering algorithms. In *Scientific and Statistical Database Management*, page 580–585, 2008.

[Adiga et al. 2002] N. R. Adiga, G. Almási, G. S. Almasi, Y. Aridor, R. Barik, D. Beece, R. Bellofatto, G. Bhanot, R. Bickford, M. Blumrich, et al. An overview of the bluegene/l supercomputer. In *Supercomputing, ACM/IEEE 2002 Conference*, page 60, 2002.

[Agarwal and Procopiuc 2002] P. K. Agarwal and C. M. Procopiuc. Exact and approximation algorithms for clustering. *Algorithmica*, 33(2):201–226, 2002. ISSN 0178-4617.

[Aizawa 2003] A. Aizawa. An information-theoretic perspective of tf–idf measures. *Information Processing & Management*, 39(1):45–65, 2003. ISSN 0306-4573.

[Arora et al. 1998] S. Arora, P. Raghavan, and S. Rao. Approximation schemes for euclidean k-medians and related problems. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, page 106–113, 1998.

[Blei et al. 2003] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

[Boriah et al. 2008] S. Boriah, V. Chandola, and V. Kumar. Similarity measures for categorical data: A comparative evaluation. In *In Proceedings of the eighth SIAM International Conference on Data Mining*, pages 243–254, 2008.

[Breunig et al. 2000] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. Lof: identifying density-based local outliers. In *ACM Sigmod Record*, volume 29, page 93–104, 2000.

[Chandola et al. 2009] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Computing Surveys (CSUR)*, 41(3):15, 2009. ISSN 0360-0300.

[Chu et al. 2002] S.-C. Chu, J. F. Roddick, and J.-S. Pan. An efficient k-medoids-based algorithm using previous medoid index, triangular inequality elimination criteria, and partial distance search. In *Data Warehousing and Knowledge Discovery*, page 63–72. Springer, 2002.

[Cutting and Pedersen 1989] D. Cutting and J. Pedersen. Optimization for dynamic inverted index maintenance. In *Proceedings of the 13th annual international ACM SIGIR conference on Research and development in information retrieval*, page 405–411, 1989.

[Ding and He 2004] C. Ding and X. He. K-means clustering via principal component analysis. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, page 29, New York and NY and USA, 2004. ACM. ISBN 1-58113-838-5.

[Eberle and Holder 2009] W. Eberle and L. Holder. Applying graph-based anomaly detection approaches to the discovery of insider threats. In *Intelligence and Security Informatics, 2009. ISI '09. IEEE International Conference on*, page 206–208, 2009.

[Forman 2008] G. Forman. Bns feature scaling: an improved representation over tf-idf for svm text classification. In *Proceedings of the 17th ACM conference on Information and knowledge management*, page 263–270, 2008.

[Fronza et al. 2013] I. Fronza, A. Sillitti, G. Succi, M. Terho, and J. Vlasenko. Failure prediction based on log files using random indexing and support vector machines. *Journal of Systems and Software*, 86(1):2–11, 2013.

[Gara et al. 2005] A. Gara, M. A. Blumrich, D. Chen, G.-T. Chiu, P. Coteus, M. E. Giampapa, R. A. Haring, P. Heidelberger, D. Hoenicke, G. V. Kopcsay, et al. Overview of the blue gene/l system architecture. *IBM Journal of Research and Development*, 49(2.3):195–212, 2005.

[Hall et al. 2009] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[Han and Karypis 2000] E.-H. S. Han and G. Karypis. *Centroid-based document classification: Analysis and experimental results*. Springer, 2000.

[Hangal and Lam 2002] S. Hangal and M. S. Lam. Tracking down software bugs using automatic anomaly detection. In *Proceedings of the 24th International Conference on Software Engineering*, ICSE '02, page 291–301, New York and NY and USA, 2002. ACM. ISBN 1-58113-472-X.

[Hautamäki et al. 2004] V. Hautamäki, I. Kärkkäinen, and P. Fränti. Outlier detection using k-nearest neighbour graph. In *ICPR (3)*, page 430–433, 2004.

[Holmes et al. 1994] G. Holmes, A. Donkin, and I. H. Witten. Weka: A machine learning workbench. In *Proceedings of the 1994 Second Australian and New Zealand Conference on Intelligent Information Systems*, page 357–361, 1994.

[Huang and Qin 2004] T. Huang and X. Qin. Detecting outliers in spatial database. In *2004 IEEE First Symposium on Multi-Agent Security and Survivability*, page 556–559, 2004.

[Kanungo et al. 2002] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):881–892, 2002.

[Knox and Ng 1998] E. M. Knox and R. T. Ng. Algorithms for mining distancebased outliers in large datasets. In *Proceedings of the International Conference on Very Large Data Bases*, page 392–403, 1998.

[Lazarevic et al. 2003] A. Lazarevic, L. Ertoz, V. Kumar, A. Ozgur, and J. Srivastava. A comparative study of anomaly detection schemes in network intrusion detection. In *Proceedings of the 2003 SIAM International Conference on Data Mining*, page 25–36. Society for Industrial and Applied Mathematics, Philadelphia and PA, 2003. ISBN 978-0-89871-545-3.

[Leung and Leckie 2005] K. Leung and C. Leckie. Unsupervised anomaly detection in network intrusion detection using clusters. In *Proceedings of the Twenty-eighth Australasian Conference on Computer Science - Volume 38*, ACSC '05, page 333–342, Darlinghurst and Australia and Australia, 2005. Australian Computer Society, Inc. ISBN 1-920-68220-1.

[Liang 2007] Y. Liang. *Failure Analysis, Modeling, and Prediction for BlueGene/L*. ProQuest, 2007.

[Liang et al. 2005] Y. Liang, Y. Zhang, A. Sivasubramaniam, R. K. Sahoo, J. Moreira, and M. Gupta. Filtering failure logs for a bluegene/l prototype. In *DSN 2005. Proceedings. International Conference on Dependable Systems and Networks*, page 476–485, 2005.

[MacQueen et al. 1967] J. MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, page 281–297, 1967.

[Mahajan et al. 2009] M. Mahajan, P. Nimbhorkar, and K. Varadarajan. The planar k-means problem is np-hard. In *WALCOM: Algorithms and Computation*, volume 5431 of *Lecture Notes in Computer Science*, page 274–285. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-00201-4.

[Oliner and Stearley 2007] A. Oliner and J. Stearley. What supercomputers say: A study of five system logs. In *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, 2007. DSN'07*, page 575–584, 2007.

[Pelleg et al. 2000] D. Pelleg, A. W. Moore, et al. X-means: Extending k-means with efficient estimation of the number of clusters. In *International Conference on Machine Learning*, page 727–734, 2000.

[Phua et al. 2010] C. Phua, V. C. S. Lee, K. Smith-Miles, and R. W. Gayler. A comprehensive survey of data mining-based fraud detection research. *CoRR*, abs/1009.6119, 2010.

[Pitakrat et al. 2013] T. Pitakrat, A. van Hoorn, and L. Grunske. A comparison of machine learning algorithms for proactive hard disk drive failure detection. In *Proceedings of the 4th international ACM Sigsoft symposium on Architecting critical systems*, page 1–10, 2013.

[Pitakrat et al. 2014] T. Pitakrat, J. Grunert, O. Kabierschke, F. Keller, and A. van Hoorn. A framework for system event classification and prediction by means of machine learning. 2014. Submitted for review.

[Ramaswamy et al. 2000] S. Ramaswamy, R. Rastogi, and K. Shim. Efficient algorithms for mining outliers from large data sets. *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, 29(2):427–438, 2000. ISSN 0163-5808.

[Ramos 2003] J. Ramos. Using tf-idf to determine word relevance in document queries. In *Proceedings of the First Instructional Conference on Machine Learning*, 2003.

[Ray 2004] A. Ray. Symbolic dynamic analysis of complex systems for anomaly detection. *Signal Processing*, 84(7):1115–1130, 2004. ISSN 0165-1684.

[Reynolds and Reynolds 1977] H. T. Reynolds and H. T. Reynolds. *The analysis of cross-classifications*. Free Press New York, 1977.

[Sahlgren 2005] M. Sahlgren. An introduction to random indexing. In *Methods and applications of semantic indexing workshop at the 7th international conference on terminology and knowledge engineering, TKE*, volume 5, 2005.

[Sahoo et al. 2004] R. K. Sahoo, M. S. Squillante, A. Sivasubramaniam, and Y. Zhang. Failure data analysis of a large-scale heterogeneous server environment. In *International Conference on Dependable Systems and Networks*, page 772–781, 2004.

[Salfner et al. 2010] F. Salfner, M. Lenk, and M. Malek. A survey of online failure prediction methods. *ACM Computing Surveys (CSUR)*, 42(3):10, 2010. ISSN 0360-0300.

[Tang et al. 1990] D. Tang, R. K. Iyer, and S. S. Subramani. Failure analysis and modeling of a vaxcluster system. In *Fault-Tolerant Computing, 1990. FTCS-20. Digest of Papers., 20th International Symposium*, page 244–251, 1990.

[Valdman 2001] J. Valdman. Log file analysis. *Department of Computer Science and Engineering (FAV UWB)., Tech. Rep. DCSE/TR-2001-04*, 2001.

[van Hoorn et al. 2009] A. van Hoorn, M. Rohr, W. Hasselbring, J. Waller, J. Ehlers, S. Frey, and D. Kieselhorst. Continuous monitoring of software services: Design and application of the kieker framework. 2009.

[van Hoorn et al. 2012] A. van Hoorn, J. Waller, and W. Hasselbring. Kieker: A framework for application performance monitoring and dynamic software analysis. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, page 247–248, 2012.

[Xu et al. 2009] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan. Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, SOSP '09, page 117–132, New York and NY and USA, 2009. ACM. ISBN 978-1-60558-752-3.

[Zheng et al. 2009] Z. Zheng, Z. Lan, B.-H. Park, and A. Geist. System log pre-processing to improve failure prediction. In *IEEE/IFIP International Conference on Dependable Systems & Networks, 2009. DSN'09*, page 572–577, 2009.

Decleration

I declare that this thesis is the solely effort of the author. I did not use any other sources and references than the listed ones. I have marked all contained direct or indirect statements from other sources as such. Neither this work nor significant parts of it were part of another review process. I did not publish this work partially or completely yet. The electronic copy is consistent with all submitted copies.

---

place, date, signature