**BRNO UNIVERSITY OF TECHNOLOGY**
**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**

**FACULTY OF INFORMATION TECHNOLOGY**
**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**

# LOG FILE ANALYSIS AND ANOMALY DETECTION

**ANALÝZA LOGOVACÍCH SÚBOROV A DETEKCIA ANOMÁLIÍ**

**TERM PROJECT**

**AUTHOR** **Bc. ADAM ZVARA**

**BRNO 2024**

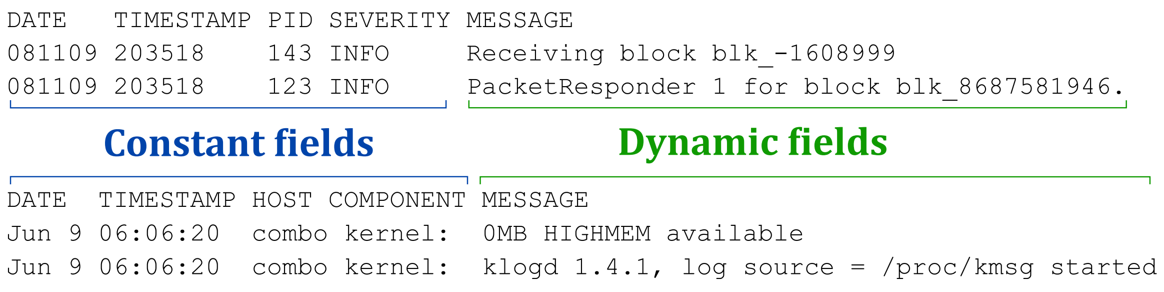
# Contents

Chapter 1

# Log file analysis

Logging is the process of documenting the events occurring within a computer system as it operates. This information is stored in log files, which essentially serve as a chronological record of these events. Log files aggregate information from various sources such as applications, operating systems, or even complex distributed systems or supercomputers. They offer details about the program's activities, including the actions taken, associated parameters, and resulting outcomes. Consequently, they are invaluable for system operators tasked with identifying errors or suspicious activities.

Logs often take the form of unstructured text, which can pose challenges for automated parsing due to their dependence on the preferences of individual programmers, who generate them (often through print statements within programs). Nonetheless, logs can generally be divided into two main parts: constant fields and dynamic fields (as shown in 1.1). **Constant fields**, such as timestamps, severity levels, process IDs (PIDs), or component identifiers, remain unchanged within a single log file. These elements provide context and metadata for understanding the events recorded in the log. **Dynamic fields**, on the other hand, contain specific data, actions, or outcomes associated with individual events.



```
DATE   TIMESTAMP PID SEVERITY MESSAGE
081109 203518    143 INFO     Receiving block blk_-1608999
081109 203518    123 INFO     PacketResponder 1 for block blk_8687581946.
```

```
DATE  TIMESTAMP HOST COMPONENT MESSAGE
Jun 9 06:06:20  combo kernel:  0MB HIGHMEM available
Jun 9 06:06:20  combo kernel:  klogd 1.4.1, log source = /proc/kmsg started
```

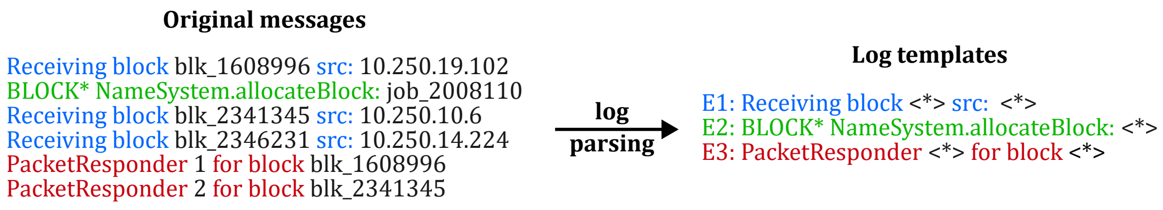
Figure 1.1: HDFS log file (top) and Linux syslog (bottom)

The amount of data generated within large-scale systems can exceed 1PB per day [4], which makes manual inspection impossible. While simple solutions like using regular expressions to match specific keywords such as „error“ or „warning“ can help identify abnormal patterns, it's important to note that these patterns don't necessarily indicate a problem with the system. For instance, in distributed systems employing *failover* mechanisms, processes may be deliberately terminated and moved to other servers for optimization purposes [4]. Filtering logs by severity is another alternative, although logs with lower severity may still contain valuable information, making it unwise to completely discard them.

Algorithms for parsing log files to detect anomalies usually follow the same structure described in He et al. [3]. This structure typically involves three main phases.

## 1.1 Log parsing

The initial step in log analysis is converting raw logging messages (dynamic fields) into structured data. This structured data is then utilized during feature extraction and passed into a machine learning model to discover potential anomalies. During log parsing, similar logging messages, which may share common words but have differing empirical values such as numbers, IP addresses, URLs, and more are grouped together (figure 1.2). By aggregating logs that share similarities, we can assign them a single string representation known as a **log template** or log key. This template encapsulates the cluster of logs, which represents the underlying log event. At the end of this process a **structured log file** is created, where each line of original log file is appended with the corresponding event template.



```
Receiving block blk_1608996 src: 10.250.19.102
BLOCK* NameSystem.allocateBlock: job_2008110
Receiving block blk_2341345 src: 10.250.10.6
Receiving block blk_2346231 src: 10.250.14.224
PacketResponder 1 for block blk_1608996
PacketResponder 2 for block blk_2341345
```

```
E1: Receiving block <*> src:  <*>
E2: BLOCK* NameSystem.allocateBlock: <*>
E3: PacketResponder <*> for block <*>
```

Figure 1.2: Log parsing – empirical values are removed and similar messages are grouped together

Traditionally, log parsing was done using regular expressions to extract specific log events [2]. However, with today's systems and increasing size of log data produced it requires non trivial efforts for manual creation and maintenance of matching rules (especially when a system is constantly being developed). A more automated approach to extracting log templates is using clustering based methods (such as SLCT [5] or LKE [1]) or neural networks (DivLog). Another interesting approach is to use static analysis of source code to extract templates of print statements used to produce logging information [7]. Overview of used log parsing methods as well as their implementation is showcased in the *logparser*[1] project from LogPai [9, 2].

## 1.2 Feature extraction

During feature extraction, the structured logs are grouped together using various windowing techniques to create **log sequences**. Log sequences are then concatenated together to form event count matrix (denoted as $X$), where each entry $X_{ij}$ represents the frequency, with which the event $j$ occurred in the $i$-th log sequence. Anomaly detection methodologies may use different approaches regarding this event count matrix. Some methods operate directly on the event count matrix itself. Others [4] apply weighting functions to each sequence to enhance the significance of certain events within the log sequences.

The type of windowing function used to generate log sequences can significantly impact the accuracy of the detection model. The most frequently used methods are [3] (figure 1.3):

[1] https://github.com/logpai/logparser [accessed 10.04.2024]

1. **Fixed window** – log sequences are constructed from events belonging to fixed (non-overlapping) periods of time.

2. **Sliding window** – similar to fixed windowing, but with the difference that neighboring windows may overlap. Instead of advancing the window by a fixed interval, the window moves by a specified stride.

3. **Session window** – events are grouped together based on session IDs rather than timestamps. Typically, a session ID is assigned to a sequence of events which represents a specific execution path of a program.

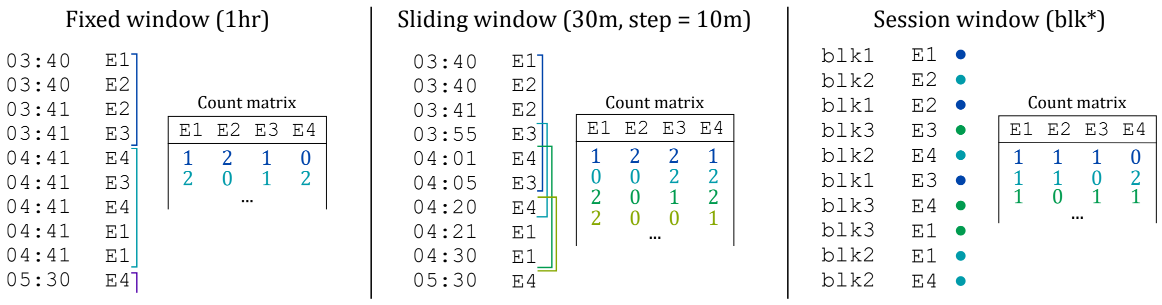


Figure 1.3: Examples of different windowing techniques

## 1.3 Anomaly detection

After feature extraction, the resulting data can undergo further modification, such as dimension reduction using techniques like Principal Component Analysis (PCA), before being transferred into a machine learning model tasked with identifying anomalies within the log data. Over the years, various types of ML models have been utilized, broadly classified into two groups [3].

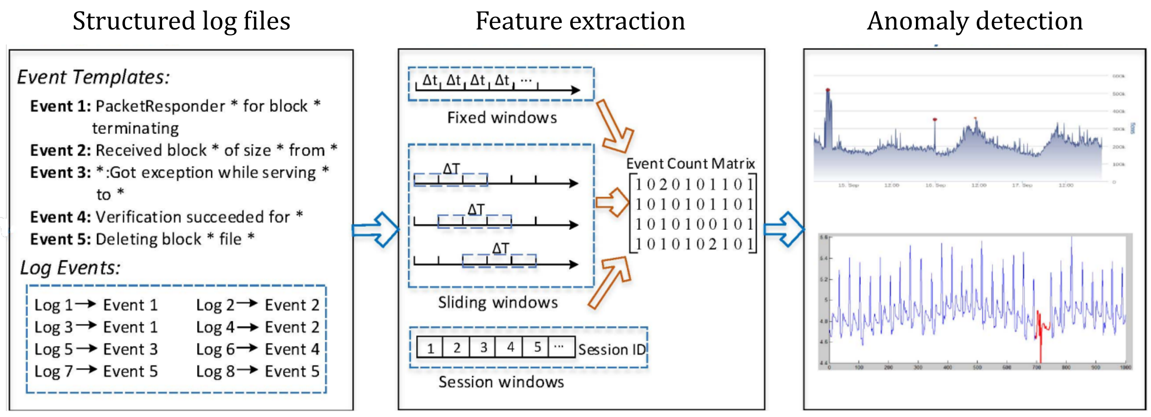
1. **Supervised methods** require labeled data for training, enabling them to learn patterns from examples with known outcomes. Examples include Decision Trees, Support Vector Machines (SVM), and regression models.

2. **Unsupervised methods** do not rely on labeled data for training. Instead, they aim to discover patterns and structures within the data without prior knowledge of class labels. Techniques commonly used in this category include clustering algorithms or invariants mining approaches.

Overview of available detection methods as well as their implementation is showcased in the *loglizer*[2] project from LogPai [3].

[2]https://github.com/logpai/loglizer [accessed 10.04.2024]

To summarize, the log analysis process consists of 3 steps, as shown in figure 1.4: converting raw log messages into structured log events (log parsing), grouping events into log sequences using windowing techniques (feature extraction or log vectorization) and finally deploying various machine learning models to identify anomalies in log files.



**Event 1:** PacketResponder * for block * terminating
**Event 2:** Received block * of size * from *
**Event 3:** *:Got exception while serving * to *
**Event 4:** Verification succeeded for *
**Event 5:** Deleting block * file *

```
Log 1→ Event 1    Log 2→ Event 2
Log 3→ Event 1    Log 4→ Event 2
Log 5→ Event 3    Log 6→ Event 4
Log 7→ Event 5    Log 8→ Event 5
```

$$\begin{bmatrix} 1&0&2&0&1&0&1&1&0&1 \\ 1&0&1&0&1&0&1&1&0&1 \\ 1&0&1&0&1&0&0&1&0&1 \\ 1&0&1&0&1&0&2&1&0&1 \end{bmatrix}$$

Figure 1.4: Stages of log analysis algorithms [3]

# Chapter 2

# LogCluster algorithm

Before discussing the algorithm, let's address some confusion surrounding the term „LogCluster“ when it comes to detecting anomalies in log files. As explained in 1.1, the initial step in analysing log files involves generating log events during the log parsing phase. Lin et al. [4] describes the core ideas of the algorithm, which occur after the log parsing phase. During log parsing, events are created using "log abstraction technique", which references the work "Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis" by Fu et al. [1]. This exact paper is also referenced in a paper providing an overview of log pasing by He et al. [2] as a Log Key Extraction (LKE) method. However, in related literature [6], the LogCluster algorithm is again presented as a parsing method. Adding to the confusion, both the *logparser* and *loglizer* projects implement LogCluster as log parsing tool as well as anomaly detection mechanism. In this paper, LogCluster algorithm which is implemented is the one described in Lin et al. [4].

## 2.1 Algorithm description

The core idea of the LogCluster comes from real-world applications, where engineers frequently employ controlled environments to develop large-scale applications. In this „controlled environment“, we have the opportunity to analyze log patterns and establish a knowledge database. Later, logs generated from production environment can be compared with the knowledge base and if the examined log sequence hasn't been previously seen, it is flagged as an anomaly.

### Log parsing

Log parsing isn't directly integrated into LogCluster – therefore, any of the common log parsing algorithms described in section 1.1 can be used. However, it's essential to note, that different parsing algorithms may yield different results.

### Feature extraction (log vectorization)

Feature extraction begins by deploying session windows on log events, which generates log sequences represented by count matrix. This matrix is further weighted by two functions:

- **IDF-based weighting**: Inverse Document Frequency (IDF) is often used as a term weighting technique, where we treat each event as a term and each log sequence as a document. If an event frequently occurs in multiple log sequences, its discriminative power is lower than event occurring in smaller number of sequences. We then calculate IDF for each event $e$ as:

$$w_{idf}^{e} = \log\left(\frac{N}{n_t}\right)$$

where $N$ is the total number of log sequences and $n_t$ denotes the number of sequences where the event $t$ appears.

- **Contrast-based weighting**: an event, which appears both in controlled environment as well as in production environment has less descriminative power than an event, which only occurs in production environment (unseen events are more likely to reflect failures and thus can be weighted higher):

$$w_{con}^{e} = \begin{cases} 1 & \text{if } t \text{ appears only in production} \\ 0 & \text{otherwise} \end{cases}$$

To calculate the final vector representation of log sequence, both weights are combined and normalization function is applied to IDF weight (in this example the Sigmoid function):

$$w^{e} = 0.5 \cdot \sigma(w_{idf}^{e}) + 0.5 \cdot w_{con}^{e}$$

## Clustering

The objective of this phase is to construct a knowledge base, which serves as a repository of previously observed log sequences. The knowledge base doesn't need to contain only valid sequences but also known failures, for which mitigation actions can be deployed. The process of generating the knowledge base involves two main steps:

**Knowledge base initialization**: a subset of log sequences from controlled environment is used to create initial clusters (bootstrapping phase). **Agglomerative clustering** is chosen as the clustering algorithm, where each sequence is placed into an individual cluster. Distances between clusters are computed using cosine similarity.

$$\text{Similiarity}(S_i, S_j) = \frac{S_i \cdot S_j}{|S_i| \cdot |S_j|}$$

Two clusters are merged together, if the maximum distance between all pairs of clusters is lower than maximum distance parameter $\theta$. Once initial clusters are established, the representative vector of each cluster is computed using a scoring function, which assigns a score to each item in the current cluster. The item with the lowest score is selected as the representative vector. Example of simplified agglomerative clustering can be seen in figure 2.1.

$$\text{Score}(S_i) = \frac{1}{n-1}\sum_{j=1}^{n}(1 - \text{Similiarity}(S_i, S_j))$$

**Online learning phase**: the rest of the log sequences are used to adjust the clusters. Each vector is added to the knowledge base one by one and if the smallest distance between the new vector and representatives is below a certain threshold, the event is added to that cluster, and the representative vector is recalculated. Otherwise, a new cluster is formed, with the new vector serving as the representative cluster.

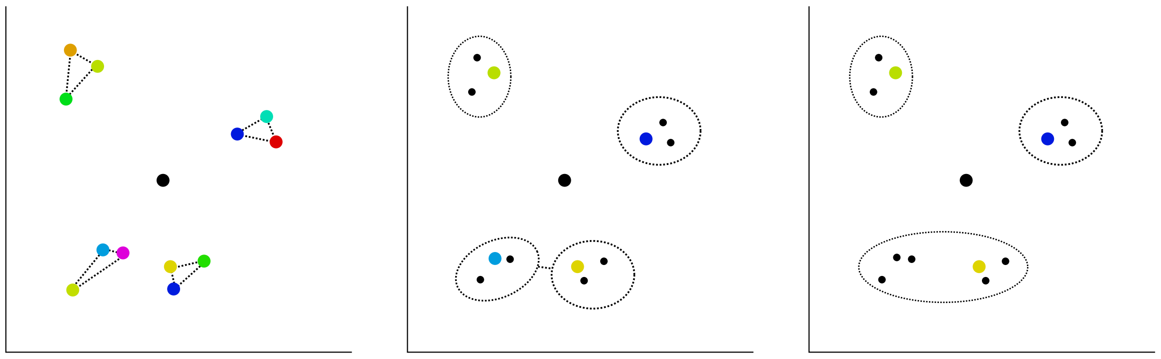


Figure 2.1: Agglomerative clustering: first each vector belongs to its own cluster, then clusters are merged together based on distance of their representative vectors

### Anomaly detection

Once the knowledge base is established, we can identify anomalies by extracting vectors from examined log file one by one, comparing their distances with cluster representatives. If the distance is below a predefined threshold, an anomaly is detected (essentially the same step as in the online learning phase).

# Chapter 3

# HDFS dataset

This section outlines the dataset utilized for evaluating LogCluster. The primary dataset employed for analyzing LogCluster was the labeled HDFS log file obtained from loglizer data samples[1]. However, in the following chapter 4, another validation file was generated for further evaluation. All graphs and results discussed in this chapter are available in the submitted file `analysis.ipynb`.

## 3.1 Dataset description

The HDFS dataset has already undergone parsing using a log parsing algorithm, and thus, it already contains log events. Table 3.1 presents some basic numerical values describing the dataset after the log vectorization phase, during which session windowing was applied. As observed, the dataset contains significantly more normal samples than anomalous samples.

| | |
|---|---|
| **Number of sessions** | 7940 |
| **Normal sessions** $\lvert S_n \rvert$ | 7627 |
| **Anomalous sessions** $\lvert S_a \rvert$ | 313 |
| **Events in** $S_n$ | E2, E3, E5, E6, E9, E11, E16, E18, E21, E22, E25, E26 |
| **Events in** $S_a$ | E2, E3, E5, E6, E7, E8, E9, E10, E11, E13, E14, E15, E16, E18, E21, E22, E25, E26, E27 |
| **Events in** $S_a \setminus S_n$ | E7, E8, E10, E13, E14, E15, E27 |

Table 3.1: Basic description of HDFS dataset

Analyzing the differences between normal and anomalous events (events only occurring in anomalous sessions) can provide valuable insights into identifying anomalies. Events that only appear in anomalous sessions, as shown in table 3.2, often contain keywords such as „exception“ or „interrupted,“ which could indicate abnormal occurrences. Figure A.1 illustrates example of event count matrices, showing unique sequences from normal and anomalous logs. This visualization offers insight into the frequency of occurrence of various log sequences in both normal and anomalous scenarios.

[1] https://github.com/logpai/loglizer/blob/master/data/HDFS/HDFS_100k.log_structured.csv [accessed 11.04.2024]

| | | |
|---|---|---|
| Normal events | E2 | Verification succeeded for ⟨∗⟩ |
| | E3 | Served block ⟨∗⟩ to ⟨∗⟩ |
| | E5 | Receiving block ⟨∗⟩ src: ⟨∗⟩ dest: ⟨∗⟩ |
| Anomalous events | E7 | writeBlock ⟨∗⟩ received exception ⟨∗⟩ |
| | E8 | PacketResponder ⟨∗⟩ for block ⟨∗⟩ Interrupted. |
| | E14 | Exception in receiveBlock for block ⟨∗⟩ ⟨∗⟩ |

Table 3.2: Examples of normal and anomalous event templates

## 3.2 Analysis of different weighting functions

### IDF weighting

In the next step of LogCluster, the event count matrix undergoes normalization with an IDF weighting function. The objective of this weighting is to reduce the influence of frequently occurring events. As depicted in figure A.2, the most frequent events (showed in table 3.3) have been weighted, due to their high frequency, they lack discriminatory power in distinguishing anomalous logs. This observation underscores the importance of applying IDF weighting to ensure that less common events contribute more significantly to the anomaly detection process.

| E5 | E26 | E9 | E11 | E22 | E2 | E3 |
|---|---|---|---|---|---|---|
| 22890 | 22887 | 22878 | 22878 | 7627 | 2151 | 204 |

Table 3.3: Most frequent events in normal sessions

### Contrast weighting

Contrast weighting emphasises events, which have not been seen in training phase (anomalous event) and thus assigns even greater weights to them, as seen in figure A.3. Event frequencies are now also normalized to fall within the range $\langle 0, 1\rangle$.

### Evaluation of weighting functions

While the weighting methods applied to this particular dataset effectively emphasize anomalous events (such as E7, E8, etc.) and reduce the impact of frequent events, they may not fully capture differences in the frequency of normal events across sequences. For instance, the presence of „E5“ in normal sessions (figure A.1) shows that some anomalous events emit higher or lower frequencies of this event than normal sequences. However, this trend is not captured in either of the weighted sequences (figures A.2, A.3).

Additionally, upon closer examination of the anomalous sequences, it's evident that some directly overlap with the normal ones. This suggests that the current weighting methods may not sufficiently discriminate between normal and anomalous events in all cases. To further investigate this issue, a simple LogCluster model has been deployed on the testing dataset with `max_distance` parameter value of 0.000001 (this parameter is described in chapter 4). This approach allows us to assess whether the model can accurately distinguish between normal and anomalous events when individual normal sequences are treated as distinct clusters. As indicated in table 3.4, setting the threshold parameter to a value of 0.017, which is relatively high to minimize false positives, would result in the detection of only 23% of overall anomalies. This suggests that the current approach may not effectively capture a significant portion of the anomalies present in the dataset.

| **Distance of anomalous sequence to closest knowledge base centroid** | **Number of classified anomalies** |
|---|---|
| 0.0 | 59 |
| 0.0000324 | 40 |
| 0.0000365 | 139 |
| 0.017 | 1 |
| 0.133 | 72 |
| 0.196 | 1 |

Table 3.4: Distances of anomalous sequences to centroids of knowledge base

# Chapter 4

# Implementation and testing

The implementation of the LogCluster algorithm has been inspired by the framework provided by *loglizer*, with additional enhancements such as contrast weighting and the ability to export the knowledge base.

## 4.1 Project structure

The chosen language for implementation is Python and the source code of the project can be divided into four main parts.

### Dataloading

In the `Dataloader.py` module, the initial step is to load structured log files into DataFrame structures, which serve as the primary data format used throughout the code. Additionally, if a label file is provided, it must only contain binary values, where 0 represents a normal sequence and 1 represents an anomaly sequence.

### Feature extraction

The modules in `FeatureExtractionModels/*.py` implement different types of windowing (although only session windowing was used to validate model, as discussed in chapter 5). Each of the feature extraction models (`SessionWindow.py` and `TimeWindow.py`) provides a `transform` method, which extracts log sequences from loaded structured file, as described in section 1.2. When a label file is provided, the session windowing checks whether all occurrences of extracted lines within a single session window belong to the same category (either normal or anomaly). If time windowing is used, the resulting label of a window is classified as an anomaly if at least one anomaly line is present within that window.

Module `FeatureExtraction.py` encapsulates feature extraction and applies weighting functions to log sequences. After creating log sequences from the training file (using `session/fixed/sliding_windowing` methods), an *event knowledge base* is established to store events from training file. The target log sequences are then vectorized in the same manner using the `transform method`. Following this, weighting of the events can be performed using the `apply_weighting` method. In the case of contrast-based weighting, new events are compared with the event knowledge base extracted from the training dataset.

### Clustering

In the clustering module `LogCluster.py`, a knowledge base with cluster centroids (as described in 2.1) is created from the log sequences generated in the previous step using the `fit` method. Anomaly detection on targeted logs is performed with `predict` method and the model can be also evaluated using the `evaluate` method. The two main parameters of LogCluster model are:

- `max_dist` – the maximum distance of clusters during the agglomerative clustering, when the clusters are merged together.
- `distance_threshold` – if the distance between a new log sequence and the closest cluster exceeds this threshold, the new sequence is identified as an anomaly.

If the targeted logs contain events that do not appear in knowledge base, then the knowledge base centroids are expanded with these events and default values. Conversely, if the knowledge base contains features that are not present in the targeted logs, the targeted logs are expanded with new events and default value 0.

Furthermore, LogCluster supports exporting of the knowledge base, allowing users to save and reuse the model without the need to vectorize and train it before each use.

## 4.2 Evaluation and testing

The first dataset used for validating the model comprises 100 thousand HDFS log messages, as described in Chapter 3. Another dataset, HDFS250k, was created from the LogHub[1] repository [8], containing 250 thousand lines of the original HDFSv1[2] logs. These logs were parsed using the IPLoM[3] parser from the LogParser repository. The resulting log sequences were manually adjusted to match the log templates from the HDFS100k dataset. The metrics for measuring the quality of the model were accuracy, precision and recall. During the evaluation, the LogCluster model was trained on the 100k dataset and then tested on both the 100k and 250k datasets using the session windowing. Different types of weighting were applied, including none, IDF, and IDF with contrast weighting.

Parameters of the model were selected through the following process: firstly, different values of the `max_distance` parameter were tested to vary the number of clusters created in the initialization phase. Then, the `threshold` value was empirically chosen to maximize the detection of anomalies (true positives) while minimizing the mislabeling of normal sequences as anomalies (false positives). For instance, consider the distances of normal and anomalous sessions to the nearest centroid in the knowledge base, as illustrated in table 4.1. To achieve a balance between accurately classifying normal sessions and effectively detecting a significant portion of anomalous sessions, the value of the threshold parameter can be set in the $\langle 0.013, 0.05 \rangle$ range.

Table 4.2 presents the results when no weighting was applied. Surprisingly, even without weighting, the model performs reasonably well, especially with the `max_dist` parameter set to 0.1 and the `threshold` parameter set to 0.04. This configuration was able to detect 48.6% of the overall anomalies with 98.7% confidence. The results also imply that having more clusters does not necessarily lead to better results. This is because a higher number

[1] https://github.com/logpai/loghub/tree/master [accessed 13.04.2024]
[2] https://github.com/logpai/loghub/blob/master/HDFS#hdfs_v1 [accessed 13.04.2024]
[3] https://github.com/logpai/logparser/tree/main/logparser/IPLoM [accessed 13.04.2024]

| Distance range | Normal sequences | Anomalous sequences |
|---|---|---|
| $\langle 0, 0.008\rangle$ | 6445 | 161 |
| $\langle 0.008, 0.011\rangle$ | 239 | 0 |
| $\langle 0.011, 0.013\rangle$ | 941 | 0 |
| $\langle 0.013, 0.05\rangle$ | 0 | 0 |
| $\langle 0.05, 0.09\rangle$ | 0 | 19 |
| $\langle 0.09, 0.1\rangle$ | 0 | 130 |
| $\langle 0.1, 1\rangle$ | 2 | 3 |

Table 4.1: Distances of normal versus anomalous sequences to centroids

of clusters increases the likelihood that an anomalous sequence will be closer to one of the cluster centroids. However, with fewer clusters, the distances of normal sequences outside these clusters will be larger, leading to a higher chance of mislabeling normal sequences as anomalies (as observed in the last row of table 4.2).

| max_dist | threshold | Clusters | HDFS100k | | | HDFS250k | | |
|---|---|---|---|---|---|---|---|---|
| | | | *Acc* | *Prec* | *Recall* | *Acc* | *Prec* | *Recall* |
| 0.01 | 0.01 | 7 | 0.977 | 1 | 0.428 | 0.974 | 0.924 | 0.447 |
| 0.02 | 0.096 | 6 | 0.977 | 1 | 0.422 | 0.974 | 0.969 | 0.429 |
| 0.05 | 0.09 | 5 | 0.977 | 1 | 0.422 | 0.974 | 0.969 | 0.492 |
| 0.1 | 0.04 | 4 | 0.979 | 0.987 | 0.486 | 0.976 | 0.943 | 0.493 |
| 0.2 | 0.096 | 3 | 0.947 | 0.361 | 0.431 | 0.958 | 0.542 | 0.436 |

Table 4.2: Evaluation of LogCluster with no weighting applied

Table 4.3 shows the results of anomaly detection when IDF weighting is applied to log sequences. The results on 100k dataset are comparable with the unweighted model. However, there's a surprising decline in performance for the 250k log dataset compared to the unweighted model. Finally, table 4.4 shows that the the combination of IDF weighting with contrast weighting produces the poorest performance among the evaluated configurations.

| max_dist | threshold | Clusters | HDFS100k | | | HDFS250k | | |
|---|---|---|---|---|---|---|---|---|
| | | | *Acc* | *Prec* | *Recall* | *Acc* | *Prec* | *Recall* |
| 0.01 | 0.73 | 6 | 0.977 | 0.993 | 0.425 | 0.966 | 0.981 | 0.234 |
| 0.1 | 0.7 | 5 | 0.977 | 0.933 | 0.425 | 0.956 | 0.512 | 0.451 |
| 0.5 | 0.3 | 4 | 0.979 | 0.981 | 0.489 | 0.959 | 0.541 | 0.513 |

Table 4.3: Evaluation of LogCluster with IDF weighting applied

| max_dist | threshold | Clusters | HDFS100k | | | HDFS250k | | |
|---|---|---|---|---|---|---|---|---|
| | | | *Acc* | *Prec* | *Recall* | *Acc* | *Prec* | *Recall* |
| 0.00001 | 0.01 | 7 | 0.970 | 1 | 0.240 | 0.966 | 1 | 0.237 |
| 0.0001 | 0.01 | 6 | 0.970 | 1 | 0.240 | 0.966 | 1 | 0.237 |
| 0.01 | 0.01 | 5 | 0.970 | 1 | 0.240 | 0.966 | 1 | 0.237 |

Table 4.4: Evaluation of LogCluster with IDF and contrast weighting applied

The LogCluster's overall performance in anomaly detection appears to be suboptimal. As indicated in chapter 3, there is a considerable number of anomalous sequences that closely resemble normal sequences. Consequently, the model can only identify up to 50% of the total anomalies. To test this limitation, a synthetic HDFS file was created from normal sequences and injected with sequences exhibiting the most significant differences in events, as illustrated in figure A.2. The results, presented in table 4.5, confirm that the LogCluster model successfully detected more anomalous sequences than those present in the original HDFS log files.

| **weighting** | **max_dist** | **threshold** | *Acc* | *Prec* | *Recall* |
|---|---|---|---|---|---|
| none | 0.1 | 0.04 | 0.999 | 0.750 | 0.667 |
| IDF | 0.1 | 0.07 | 1 | 0.818 | 1 |
| IDF and contrast | 0.0001 | 0.01 | 1 | 1 | 0.889 |

Table 4.5: Evaluation of LogCluster on synthetic HDFS dataset

# Chapter 5

# Conclusion

The objective of this term project was to gain familiarity with anomaly detection from log files. The LogCluster model offered an intuitive approach to achieve this, making it the natural choice for this project. However, the demo version of the LogCluster tool implemented in *loglizer* had limitations, such as only supporting HDFS datasets, utilizing a sliding window for feature extraction, and lacking support for contrast weighting. To address these constraints, a more flexible implementation was developed. This implementation allows for the use of different datasets, incorporates contrast weighting, and provides the capability to import and export pre-trained models. On the contrary, certain features such as the online learning phase were omitted, as they aren't expected to directly influence the performance score of the LogCluster model. Instead, they primarily serve to lower the memory consumption during the training phase. Given that we're working with relatively small datasets where memory isn't a concern, these features were deemed unnecessary for this project.

This paper describes different types of weighting functions utilized in the LogCluster model and evaluates their performance through tests. The initial intention behind the project was to explore different windowing approaches, as session windowing is not always feasible. However, due to the LogCluster model's limited effectiveness in anomaly detection even with session windowing, alternative approaches were abandoned.

# Bibliography

[1] Fu, Q., Lou, J.-G., Wang, Y. et al. Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis. In: *2009 Ninth IEEE International Conference on Data Mining*. 2009, p. 149–158. DOI: 10.1109/ICDM.2009.60. Available at: https://ieeexplore.ieee.org/document/5360240.

[2] He, P., Zhu, J., He, S. et al. An Evaluation Study on Log Parsing and Its Use in Log Mining. In: *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 2016, p. 654–661. DOI: 10.1109/DSN.2016.66. Available at: https://ieeexplore.ieee.org/document/7579781.

[3] He, S., Zhu, J., He, P. et al. Experience Report: System Log Analysis for Anomaly Detection. In: *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*. 2016, p. 207–218. DOI: 10.1109/ISSRE.2016.21. Available at: https://ieeexplore.ieee.org/document/7774521.

[4] Lin, Q., Lou, H. Z. J.-G., Zhang, Y. et al. Log clustering based problem identification for online service systems. In: *Proceedings of the 38th International Conference on Software Engineering Companion*. New York, NY, USA: Association for Computing Machinery, 2016, p. 102–111. ICSE '16. DOI: 10.1145/2889160.2889232. ISBN 9781450342056. Available at: https://doi.org/10.1145/2889160.2889232.

[5] Vaarandi, R. A data clustering algorithm for mining patterns from event logs. In: *Proceedings of the 3rd IEEE Workshop on IP Operations & Management (IPOM 2003) (IEEE Cat. No.03EX764)*. 2003, p. 119–126. DOI: 10.1109/IPOM.2003.1251233. Available at: https://ieeexplore.ieee.org/document/1251233.

[6] Vaarandi, R., Kont, M. and Pihelgas, M. Event log analysis with the LogCluster tool. In: *MILCOM 2016 - 2016 IEEE Military Communications Conference*. 2016, p. 982–987. DOI: 10.1109/MILCOM.2016.7795458. Available at: https://ieeexplore.ieee.org/document/7795458.

[7] Xu, W., Huang, L., Fox, A. et al. Detecting large-scale system problems by mining console logs. In: *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*. New York, NY, USA: Association for Computing Machinery, 2009, p. 117–132. SOSP '09. DOI: 10.1145/1629575.1629587. ISBN 9781605587523. Available at: https://doi.org/10.1145/1629575.1629587.

[8] Zhu, J., He, S., He, P. et al. Loghub: A Large Collection of System Log Datasets for AI-driven Log Analytics. In: *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. Los Alamitos, CA, USA: IEEE Computer Society,

Oct 2023, p. 355–366. DOI: 10.1109/ISSRE59848.2023.00071. Available at: https://doi.ieeecomputersociety.org/10.1109/ISSRE59848.2023.00071.

[9] Zhu, J., He, S., Liu, J. et al. Tools and benchmarks for automated log parsing. In: *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice*. IEEE Press, 2019, p. 121–130. ICSE-SEIP '19. DOI: 10.1109/ICSE-SEIP.2019.00021. Available at: https://doi.org/10.1109/ICSE-SEIP.2019.00021.

# Appendix A

# Analysis of HDFS dataset

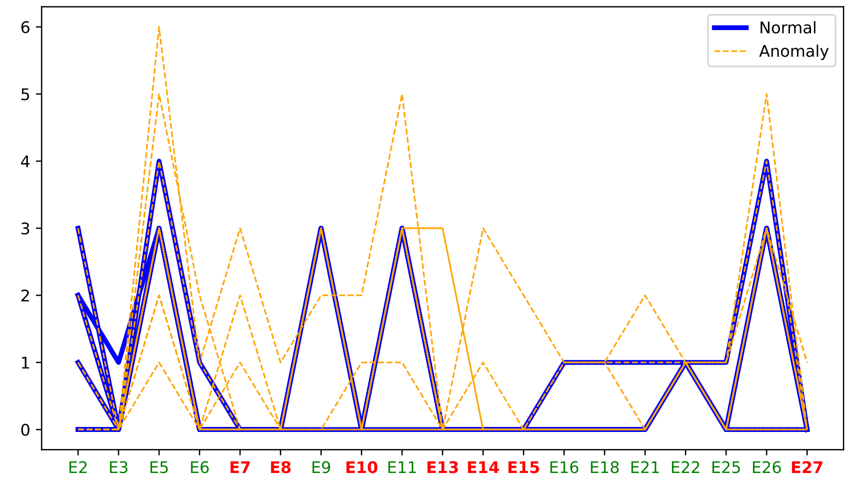


Figure A.1: Count matrix for unique sequences found HDFS dataset, displayed as a graph. Green labels (x-axis) represent events found in normal sequences and red labels represent events found only in anomalous sequences.

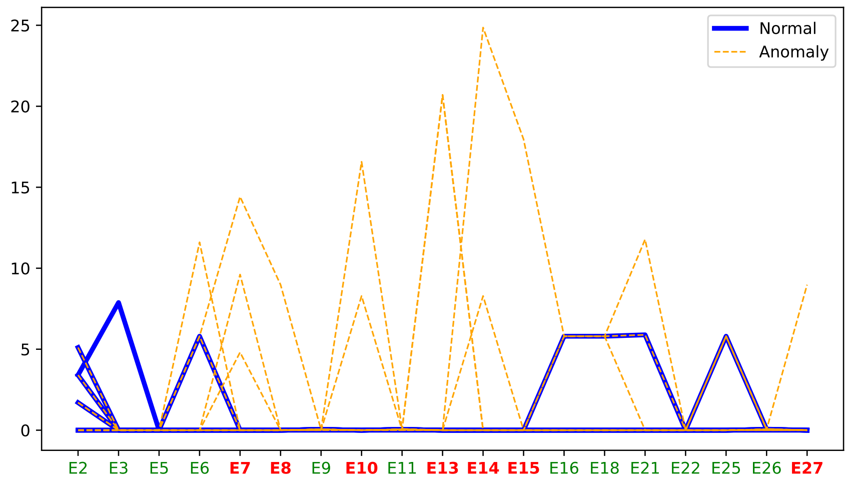


Figure A.2: Count matrix from A.1, where IDF weighting function has been applied to each log sequence.

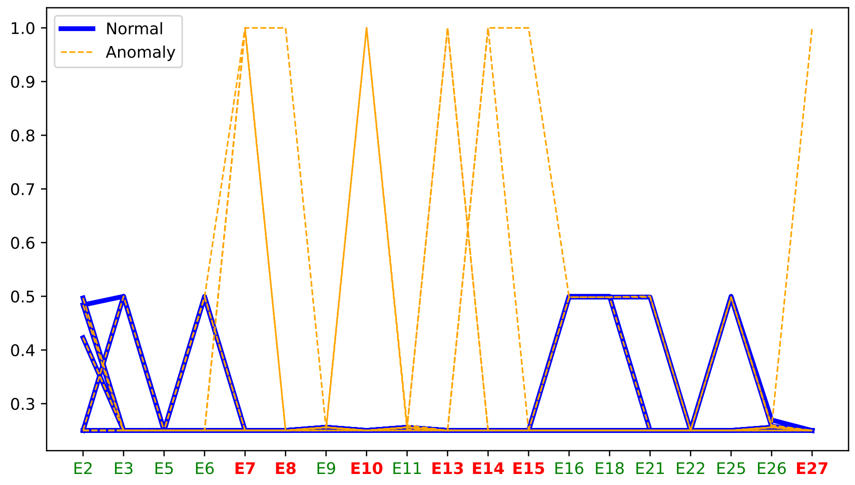


Figure A.3: Weighted matrix from A.2 with applied contrast based weighting.