Computers & Security 92 (2020) 101739

Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

System log clustering approaches for cyber security applications: A survey

Max Landauer[a,*], Florian Skopik[a], Markus Wurzenberger[a], Andreas Rauber[b]

[a] Austrian Institute of Technology, Austria
[b] Vienna University of Technology, Austria

ARTICLE INFO

Article history:
Received 23 December 2019
Accepted 29 January 2020
Available online 31 January 2020

Keywords:
Log clustering
Cyber security
Log mining
Signature extraction
Anomaly detection

ABSTRACT

Log files give insight into the state of a computer system and enable the detection of anomalous events relevant to cyber security. However, automatically analyzing log data is difficult since it contains massive amounts of unstructured and diverse messages collected from heterogeneous sources. Therefore, several approaches that condense or summarize log data by means of clustering techniques have been proposed. Picking the right approach for a particular application domain is, however, non-trivial, since algorithms are designed towards specific objectives and requirements. This paper therefore surveys existing approaches. It thereby groups approaches by their clustering techniques, reviews their applicability and limitations, discusses trends and identifies gaps. The survey reveals that approaches usually pursue one or more of four major objectives: overview and filtering, parsing and signature extraction, static outlier detection, and sequences and dynamic anomaly detection. Finally, this paper also outlines a concept and tool that support the selection of appropriate approaches based on user-defined requirements.

1. Introduction

Log files contain information about almost all events that take place in a system, depending on the log level. For this, the deployed logging infrastructure automatically collects, aggregates and stores the logs that are continuously produced by most components and devices, e.g., web servers, data bases, or firewalls. The textual log messages are usually human-readable and attached to a time stamp that specifies the point in time the log entry was generated. Especially for large organizations and enterprises, the benefits of having access to long-term log data are manifold: Historic logs enable forensic analysis of past events. Most prominently applied after faults occurred in the system, forensic analysis gives system administrators the possibility to trace back the roots of observed problems. Moreover, the logs may help to recover the system to a non-faulty state, reset incorrect transactions, restore data, prevent losses of information, and replicate scenarios that lead to erroneous states during testing. Finally, logs also allow administrators to validate the performance of processes and discover bottlenecks. In addition to these functional advantages, storing logs is typically inexpensive since log files can effectively be compressed due to a high number of repeating lines.

A major issue with forensic log analysis is that problems are only detected in hindsight. Furthermore, it is a time- and resource-consuming task that requires domain knowledge about the system at hand. For these reasons, modern approaches in cyber security shift from a purely forensic to a proactive analysis (He et al., 2017b). Thereby, real-time fault detection is enabled by constantly monitoring system logs in an online manner, i.e., as soon as they are generated. This allows timely responses and in turn reduces the costs caused by incidents and cyber attacks. On top of that, indicators for upcoming erroneous system behavior can frequently be observed in advance. Detecting such indicators early enough and initiating appropriate countermeasures can help to prevent certain faults altogether.

Unfortunately, this task is hardly possible for humans since log data is generated in immense volumes and fast rates. When considering large enterprise systems, it is not uncommon that the number of daily produced log lines is up in the millions, for example, publicly available Hadoop Distributed File System (HDFS) logs comprise more than 4 million log lines per day (Xu et al., 2009) and small organizations are expected to deal with peaks of 22,000 events per second (Allen and Richardson, 2019). Clearly, this makes

* Corresponding author.
E-mail addresses: max.landauer@ait.ac.at (M. Landauer), florian.skopik@ait.ac.at (F. Skopik), markus.wurzenberger@ait.ac.at (M. Wurzenberger), rauber@ifs.tuwien.ac.at (A. Rauber).

https://doi.org/10.1016/j.cose.2020.101739

manual analysis impossible and it thus stands to reason to employ machine learning algorithms that automatically process the lines and recognize interesting patterns that are then presented to system operators in a condensed form.

One method for analyzing large amounts of log data is clustering. Thus, several clustering algorithms that were particularly designed for textual log data have been proposed in the past. Since most of the algorithms were mainly developed for certain application-specific scenarios at hand, their approaches frequently differ in their overall goals and assumptions on the input data. We were specifically interested to discover the different strategies the authors used to pursue the objectives induced by their use-cases. However, to the best of our knowledge there is no exhaustive survey on state-of-the-art log data clustering approaches that focuses on applications in cyber security. Despite also concerned with certain types of log files, existing works are either outdated or focus on network traffic classification (Ewards et al., 2013), web clustering (Carpineto et al., 2009), and user profiling (Facca and Lanzi, 2005; Vakali et al., 2004). Other surveys address only log parsers rather than clustering (Zhu et al., 2019).

In this paper we therefore create a survey of current and established strategies for log clustering found in scientific literature. This survey is oriented towards the identification of overall trends and highlights the contrasts between existing approaches. This supports analysts in selecting methods that fit the requirements imposed by their systems. In addition, with this paper we aim at the generation of a work of reference that is helpful for all authors planning to publish in this field. Overall, the research questions we address with this paper are as follows:

- What are essential properties of existing log clustering algorithms?
- How are these algorithms applied in cyber security?
- On what kind of data do these algorithms operate?
- How were these algorithms evaluated?

The remainder of the paper is structured as follows. Section 2 outlines the problem of clustering log data and discusses how log analysis is used in the cyber security domain. In Section 3, we explain our method of carrying out the literature study. The results of the survey are stated and discussed in Section 4. We then propose a decision model for selecting appropriate clustering approaches and demonstrate it based on the survey results in Section 5. Finally, Section 6 concludes the paper.

2. Survey Background

Log data exhibits certain characteristics that have to be taken into account when designing a clustering algorithm. In the following, we therefore discuss important properties of log data, outline the reasons why log data is suitable to be clustered and look into application scenarios relevant to cyber security.

2.1. The nature of log data

Despite the fact that log data exists in various forms, some general assumptions on their compositions can be made. First, a log file typically consists of a set of single- or multi-line strings listed in inherent chronological order. This chronological order is usually underpinned by a time stamp attached to the log messages.[1] The messages may be highly structured (e.g., a list of comma-separated values), partially structured (e.g., attribute-value pairs), unstructured (e.g., free text of arbitrary length) or a combination thereof. In addition, log messages sometimes include process IDs (PIDs) that relate to the task (also referred to as thread or case) that generated them. If this is the case, it is simple to extract log traces, i.e., sequences of related log lines, and perform workflow and process mining (Nandi et al., 2016). Other artifacts sometimes included in log messages are line numbers, an indicator for the level or severity of the message (TRACE, DEBUG, INFO, WARN, ERROR, FATAL, ALL, or OFF), and a static identifier referencing the statement printing the message (Bao et al., 2018).

Arguably, log files are fairly different from documents written in natural language. This is not necessarily the case because the log messages themselves are different from natural language (since they are supposed to be human-readable), but rather because of two reasons: (i) Similar messages repeat over and over. This is caused by the fact that events are recurring since procedures are usually executed in loops and the majority of the log lines are generated by a limited set of print statements, i.e., predefined functions in the code that write formatted strings to some output. (ii) The appearances of some messages are highly correlated. This is due to the fact that programs usually follow certain control flows and components that generate log lines are linked with each other. For example, two consecutive print statements will always produce perfectly correlated log messages during normal system behavior since the execution of the first statement will always be followed by the execution of the second statement. In practice, it is difficult to derive such correlations since they often depend on external events and are the result of states and conditions.

These properties allow system logs to be clustered in two different ways. First, clustering individual log lines by the similarity of their messages yields an overview of all events that occur in the system. Second, clustering sequences of log messages gives insight into the underlying program logic and uncovers otherwise hidden dependencies of events and components.

2.2. Static clustering

We consider clustering individual log lines as a static procedure, because the order and dependencies between lines is usually neglected. After such static line-based clustering, the resulting set of clusters should ideally resemble the set of all log-generating print statements, where each log line should be allocated to the cluster representing the statement it was generated by. Examining these statements in more detail shows that they usually comprise static strings that are identical in all messages produced by that statement and variable parts that are dynamically replaced at run time. Thereby, variable parts are frequently numeric values, identifiers (e.g., IDs, names, or IP addresses), or categorical attributes. Note that the generation of logs using mostly fixed statements is responsible for a skewed word distribution in log files, where few words from the static parts appear very frequently while the majority of words appears very infrequently or even just once (Ning et al., 2014; Vaarandi, 2003).

In the following, we demonstrate issues in clustering with the sample log lines shown in Fig. 1. In the example, log messages describe users logging in and out. Given this short log file, a human would most probably assume that the two statements *print("User " + name + " logs in with status " + status)* and *print("User " + name + " logs out with status " + status)* generated the lines, and thus allocate lines {1, 2, 4} to the former and lines {3, 5} to the latter cluster. From this clustering, the templates (also referred to as signatures, patterns, or events) "User * logs in with status *" and "User * logs out with status *" can be derived, where the Kleene star * denotes a wildcard accepting any word at that position.

[1] The order and time stamps of messages do not necessarily have to correctly represent the actual generation of log lines due to technological restrictions appearing during log collection, e.g., delays caused by buffering or issues with time synchronization. A thorough investigation of any adverse consequences evoked by such effects is considered out of scope for this paper.

```
1 :: User Alice logs in with status 1
2 :: User Bob logs in with status 1
3 :: User Alice logs out with status 1
4 :: User Charlie logs in with status -1
5 :: User Bob logs out with status 1
```

Fig. 1. Sample log messages for static analysis.

Beside the high resemblance of the original statements, the wildcards appear to be reasonably placed since all other users logging in or out with any status will be correctly allocated, e.g., "User Dave logs in with status 0".

Other than humans, algorithms lack semantic understanding of the log messages and might just as well group the lines according to the user name, i.e., create clusters {1, 3}, {2, 5}, and {4}, or according to a state variable, i.e., create clusters {1, 2, 3, 5} and {4}. In the latter case, the most specific templates corresponding to the clusters are "User * logs * with status 1" and "User Charlie logs in with status -1". In most scenarios, the quality of these templates is considered to be poor, since the second wildcard of the first template is an over-generalization of a categorical attribute and the second template is overly specific. Accordingly, newly arriving log lines would be likely to form outliers, i.e., not match any cluster template.

With this example in mind we want to point out that there always exist a multitude of different possible valid clusterings and judging the quality of the clusters is eventually a subjective decision that is largely application-specific. For example, investigations regarding user-behavior may require that all log lines generated by a specific user end up in the same cluster. In any way, appropriate cluster quality is highly important since clusters are often the basis for further analyses that operate on top of the grouped data and extracted templates. The next section explores dynamic clustering as such an application that utilizes static cluster allocations.

2.3. Dynamic clustering

As pointed out earlier, log files are suited for dynamic clustering, i.e., allocation of sequences of log line appearances to patterns. However, raw log lines are usually not suited for such sequential pattern recognition, due to the fact that each log line is a uniquely occurring instance describing a part of the system state at a particular point in time. Since pattern recognition relies on repeating behavior, the log lines first have to be allocated to classes that refer to their originating event. This task is enabled by static clustering as outlined in the previous section.

In the following, we consider the sample log file shown in Fig. 2 that contains three users logging into the system, performing some action, and logging out. We assume that these steps are always carried out in this sequence, i.e., it is not possible to perform an action or log out without first being logged in.

```
1 :: User Alice logs in with status 1     :: A
2 :: User Alice performs action open      :: B
3 :: User Alice logs out with status 1    :: C
4 :: User Bob logs in with status 1       :: A
5 :: User Bob performs action write       :: B
6 :: User Charlie logs in with status 1   :: A
7 :: User Bob logs out with status 1      :: C
8 :: User Charlie performs action exec    :: B
9 :: User Charlie logs out with status 1  :: C
```

Fig. 2. Sample log messages and their event allocations for dynamic analysis.

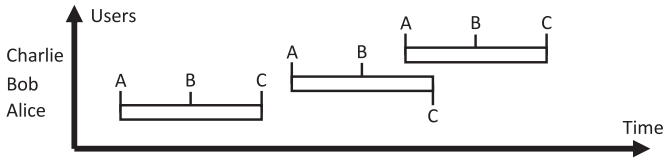


Fig. 3. Sample log events visualized on a timeline.

We assume that the sample log file has been analyzed by a static clustering algorithm to generate the three templates A="User * logs in with status *", B="User * performs action *", and C="User * logs out with status *". It is then possible to assign each line one of the events as indicated on the right side of the figure. In such a setting, the result of a dynamic clustering algorithm could be the extracted sequence A, B, C since this pattern describes normal user behavior. However, the events in lines 6 and 7 are switched, thus interrupting the pattern. Fig. 3 shows that the reason for this issue is caused by interleaved user behavior, i.e., user Charlie logs in before user Bob logs out.

Since many applications are running in parallel in real systems, interleaved processes are commonly occurring in log files and thus complicate the pattern extraction process. As mentioned in Section 2.1, some log files include process IDs that allow to analyze the corresponding logs isolated from interrupting processes and thus resolve this issue. In the simple example from Fig. 2, the username could have been used for this purpose. In addition to interleaved event sequences, real systems obviously involve much more complex patterns, including arbitrarily repeating, optional, alternative, or nested subpatterns.

While sequence mining is common, it is not the only dynamic clustering technique. In particular, similar groups of log lines can be formed by aggregating them in time-windows and analyzing their frequencies, co-occurrences, or correlations. For example, clustering could aim at generating groups of log lines that frequently occur together. Note that in this setting, the ordering of events is not relevant, but only their occurrence within a certain time interval. The next section outlines several applications of static and dynamic clustering for system security.

2.4. Applications in the security domain

Due to the fact that log files contain permanent documentation of almost all events that take place in a system, they are frequently used by analysts to investigate unexpected or faulty system behavior in order to find its origin. In some cases, the strange behavior is caused by system intrusions, cyber attacks, malware, or any other adversarial processes. Since such attacks often lead to high costs for affected organizations, timely detection and clarification of consequences is of particular importance.

Independent from whether anomalous log manifestations are caused by randomly occurring failures or targeted adversarial activity, their detection is of great help for administrators and may prevent or reduce costs. Clustering is able to largely reduce the effort required to manually analyze log files, for example, by providing summaries of log file contents, and even provides functionalities to automatize detection of anomalous behavior. In the following, we outline some of the most relevant types of anomalies detectable or supported by clustering.

- **Outliers** are single log lines that do not match any of the existing templates or are dissimilar to all identified clusters that are known to represent normal system behavior. Outliers are often new events that have not occurred during clustering or contain highly dissimilar parameters in the log messages. An example could be an error log message in a log file that usually only contains informational and debugging messages.

- **Frequency anomalies** are log events that appear unexpectedly frequent or rare during a given time interval. This may include cases where components stop logging, or detection of attacks that involve the execution of many events, e.g., vulnerability scans.
- **Correlation anomalies** are log events that are expected to occur in pairs or groups but fail to do so. This may include simple co-occurrence anomalies, i.e., two or more events that are expected to occur together, and implication anomalies, where one or more events imply that some other event or events have to occur, but not the other way round. For example, a web server that logs an incoming connection should imply that corresponding log lines on the firewall have occurred earlier.
- **Inter-arrival time anomalies** are caused by deviating time intervals between occurrences of log events. They are related to correlation anomalies and may provide additional detection capabilities, e.g., an implied event is expected to occur within a certain time window.
- **Sequence anomalies** are caused by missing or additional log events as well as deviating orders in sequences of log events that are expected to occur in certain patterns.

Outliers are based on single log line occurrences and are thus the only type of anomalies detectable by static cluster algorithms. All other types of anomalies require dynamic clustering techniques. In addition, anomalies do not necessarily have to be detected using strict rules that report every single violation. For example, event correlations that are expected to occur only in 90% of all cases may be analyzed with appropriate statistical tests.

3. Survey method

In this section we describe our approach to gather and analyze the existing literature.

3.1. Set of criteria

In order to carry out the literature survey on log clustering approaches in a structured way, we initially created a set of evaluation criteria that addresses relevant aspects of the research questions in more detail. The first block of questions in the set of criteria covers purpose, applicability, and usability of the proposed solutions:

P-1 What is the purpose of the introduced approach?
P-2 Does the method have a broad applicability or are there constraints, such as requirements for specific logging standards?
P-3 Is the algorithm a commercial product or has been deployed in industry?
P-4 Is the code of the algorithm publicly accessible?

The next group of questions focuses on the properties of the introduced clustering algorithms:

C-1a What type of technique is applied for static clustering?
C-1b What type of technique is applied for dynamic clustering?
C-2 Is the algorithm fully unsupervised as opposed to algorithms requiring detailed knowledge about the log structures or labeled log data for training?
C-3 Is the clustering character-based?
C-4 Is the clustering word- or token-based?
C-5 Are log signatures or templates generated?
C-6 Does the clustering algorithm take dynamic features of log lines (e.g., sequences) into account?
C-7 Does the algorithm generate new clusters online, i.e., in a streaming manner, as opposed to approaches that allocate log lines to a fixed set of clusters generated in a training phase?
C-8 Is the clustering adaptive to system changes, i.e., are existing clusters adjusted over time rather than static constructs?
C-9 Is the algorithm designed for fast data processing?
C-10 Is the algorithm designed for parallel execution?
C-11 Is the algorithm deterministic?

Since we were aware that a large number of approaches aim at anomaly detection, we dedicated the following set of questions to this topic:

AD-1 Is the approach designed for the detection of outliers, i.e., static anomalies?
AD-2 Is the approach designed for the detection of dynamic anomalies?
AD-3 Is the approach designed for the detection of cyber attacks?

Finally, we defined questions that assess whether and how the approaches were evaluated in the respective articles:

E-1 Did the evaluation include quantitative measures, e.g., accuracy or true positive rates?
E-2 Did the evaluation involve qualitative reviews, e.g., expert reviews or discussions of cluster quality?
E-3 Was the algorithm evaluated regarding its time complexity, i.e., running time and scalability?
E-4 Was at least one existing algorithm used as a benchmark for validating the introduced approach?
E-5 Was real log data used as opposed to synthetically generated log data?
E-6 Is the log data used for evaluation publicly available?

The set of evaluation criteria was then completed for every relevant approach. The process of retrieving these articles is outlined in the following section.

3.2. Literature search

The search for relevant literature was carried out in November 2019. For this, three research databases were used: (i) ACM Digital Library,[2] a digital library containing more than 500,000 full-text articles on computing and information technology, (ii) IEEE Xplore Digital Library,[3] a platform that enables the discovery of scientific articles within more than 4.5 million documents published in the fields computer science, electrical engineering and electronics, and (iii) Google Scholar,[4] a web search engine for all kinds of academic literature.

The keywords used for searching on these platforms were “log clustering” (29,383 results on ACM, 2,210 on IEEE, 3,050,000 on Google), “log event mining” (54,833 results on ACM, 621 on IEEE, 1,240,000 on Google), “log data anomaly detection” (207,821 results on ACM, 377 on IEEE, 359,000 on Google). We did not make any restrictions regarding the date of publication. The titles and abstracts of the first 300 articles retrieved for each query were examined and potentially relevant documents were stored for thorough inspection. It should be noted that a rather large amount of false positives were retrieved and immediately dismissed. The reason why such unrelated articles appeared is that the keywords in the queries were sometimes misinterpreted by the engines, e.g., results related to “logarithm” showed up when searching for “log”. After removing duplicates, this search yielded 207 potentially relevant articles.

[2] https://dl.acm.org/results.cfm.
[3] https://ieeexplore.ieee.org/Xplore/home.jsp.
[4] https://scholar.google.at/.

During closer inspection, several of these articles were discarded. The majority of these dismissed approaches focused on clustering numeric features extracted from highly structured network traffic logs rather than clustering the raw string messages themselves. This is a broad field of research and there exist numerous papers that apply well-known machine learning techniques for analyzing, grouping, and classifying the parsed data (Portnoy et al., 2001). Many other approaches are directed towards process mining from event logs (Van der Aalst et al., 2004), which is an extensive topic considered out of scope for our survey since it relies on log traces rather than simple log data. Furthermore, we discarded papers that introduce approaches for analysis and information extraction from log data, but are not fitted for clustering log lines, such as terminology extraction (Saneifar et al., 2009) and compression (Balakrishnan and Sahoo, 2006). We also dismissed approaches for clustering search engine query logs (Beeferman and Berger, 2000) since they are designed to process keywords written by users rather than log lines generated by programs as outlined in Section 2.1. Articles on protocol reverse engineering are discarded, because they are not primarily designed for processing system log lines and surveys on this topic already exist, e.g., Narayan et al. (2016). Finally, we excluded articles that do not propose a new clustering approach, but apply existing algorithms without modifications on different data or perform comparisons (e.g., Makanju et al., 2009a) as well as surveys. This also includes articles that propose algorithms for subsequent analyses such as anomaly detection, alert clustering, or process model mining, that operate on already clustered log data, but do not apply any log clustering techniques themselves.

After this stage, 50 articles remained. A snowball search was conducted with these articles, i.e., articles referenced in the relevant papers as well as articles referencing these papers were individually retrieved. These articles were examined analogously and added if they were considered relevant. Eventually, we obtained 59 articles and 2 tools that were analyzed with respect to the aforementioned characteristics stated in the set of evaluation criteria. We used these criteria to group articles with respect to different features and discover interesting patterns. The following section discusses the findings.

4. Survey results

We arranged the articles into groups according to the properties ascertained in the set of evaluation criteria. We thereby derived common features that could be found in several articles as well as interesting concepts and ideas that stood out from the overall strategies. In the following, we discuss these insights for every group of questions.

4.1. Purpose and applicability (P)

Four main categories of overall design goals (P-1) were identified during the review process:

- *Overview & filtering*. Log data is usually high-volume data that is tedious to search and analyze manually. Therefore, it is reasonable to reduce the total number of log messages presented to system administrators by removing log events that are frequently repeating without contributing new or any other valuable information. Clustering is able to provide such compact representations of complex log files by filtering out most logs that belong to certain (large) clusters, thus only leaving logs that occur rarely or do not fit into any clusters to be shown to administrators (Jain et al., 2009; Reidemeister et al., 2011).
- *Parsing & signature extraction*. These approaches aim at the automatic generation of log event templates (cf. Section 2.1) for parsing log lines. Parsers enable the allocation of log lines to particular system events, i.e., log line classification, and the structured extraction of parameters. These are important features for subsequent analyses, such as clustering of event sequences or anomaly detection (He et al., 2017b; Wurzenberger et al., 2019).
- *Outlier detection*. System failures, cyber attacks, or other adverse system behavior generates log lines that differ from log lines representing normal behavior regarding their syntax or parameter values. It is therefore reasonable to disclose single log lines that do not fit into the overall picture of the log file. During clustering, these log lines are identified as lines that have a high dissimilarity to all existing clusters or do not match any signatures (Juvonen et al., 2015; Wurzenberger et al., 2017b).
- *Sequences & dynamic anomaly detection*. Not all adverse system behavior manifests itself as individual anomalous log lines, but rather as dynamic or sequence anomalies (cf. Section 2.4). Thus, approaches that group sequences of log lines or disclose temporal patterns such as frequent co-occurrence or correlations are required. Dynamic clustering usually relies on line-based event classification as an initial step and often has to deal with interleaving processes that cause interrupted sequences (Aharon et al., 2009; Jia et al., 2017).

Table 1 shows the determined classes for each reviewed approach. Note that this classification is not mutually exclusive, i.e., an approach may pursue multiple goals at the same time. For example, He et al. (2017b) introduce an approach for the extraction of log signatures and then perform anomaly detection on the retrieved events.

Table 1
Overview of main goals of reviewed approaches. Categorizations are not mutually exclusive.

Purpose of approach	Approaches
Overview & filtering	Aharon et al. (2009), Aussel et al. (2018), Christensen and Li (2013), Jiang et al. (2008), Joshi et al. (2014), Li et al. (2017, 2005), Reidemeister et al. (2011), Gainaru et al. (2011), Gurumdimma et al. (2015), Hamooni et al. (2016), Jain et al. (2009), Jayathilake et al. (2017), Leichtnam et al. (2017), Makanju et al. (2009b), Nandi et al. (2016), Ning et al. (2014), Qiu et al. (2010), Ren et al. (2018), Salfner and Tschirpke (2008), Schipper et al. (2019), Carasso (2012), Taerat et al. (2011), Xu et al. (2009), Zou et al. (2016)
Parsing & signature extraction	Agrawal et al. (2019), Chuah et al. (2010), Du and Li (2016), Fu et al. (2009), Gainaru et al. (2011), Hamooni et al. (2016), He et al. (2017a,b), Jayathilake et al. (2017), Kimura et al. (2014), Kobayashi et al. (2014), Li et al. (2017), Li et al. (2018), Makanju et al. (2009b), Messaoudi et al. (2018), Menkovski and Petkovic (2017), Mizutani (2013), Nagappan and Vouk (2010), Nandi et al. (2016), Ning et al. (2014), Qiu et al. (2010), Zhen (2014), Shima (2016), Taerat et al. (2011), Tang and Li (2010), Tang et al. (2011), Thaler et al. (2017), Tovarňák and Pitner (2019), Vaarandi (2003, 2004), Vaarandi and Pihelgas (2015), Wurzenberger et al. (2019), Zhang et al. (2017), Zhao and Xiao (2016), Zulkernine et al. (2013)
Outlier detection	Juvonen et al. (2015), Leichtnam et al. (2017), Splunk (Carasso, 2012), Wurzenberger et al. (2017a,b)
Sequences & dynamic anomaly detection	Aharon et al. (2009), Chuah et al. (2010), Du et al. (2017), Du and Cao (2015), Fu et al. (2009), Gurumdimma et al. (2015), He et al. (2017a,b), Jia et al. (2017), Kimura et al. (2014), Li et al. (2018), Lin et al. (2016), Nandi et al. (2016), Salfner and Tschirpke (2008), Splunk (Carasso, 2012), Stearley (2004), Vaarandi (2004), Wang et al. (2018), Xu et al. (2009), Zhang et al. (2017), Zhang et al. (2019), Zou et al. (2016)

As expected, most approaches aim at broad applicability and do not make any specific assumptions on the input data (P-2). Although some authors particularly design and evaluate their approaches in the context of a specific type of log protocol (e.g., router syslogs Qiu et al. (2010)), their proposed algorithms are also suitable for any other logging standard. Only few approaches require artifacts specific to some protocol (e.g., Modbus (Wang et al., 2018)) for similarity computation or prevent general applicability by relying on labeled data (Reidemeister et al., 2011) or category labels (e.g., start, stop, dependency, create, connection, report, request, configuration, and other (Li et al., 2005)) for model training, log level information (Du and Cao, 2015) for an improved log similarity computation during clustering, or process IDs for linking events to sequences (Lin et al., 2016). Other approaches impose constraints such as the requirement of manually defined parsers Tang and Li (2010) or access to binary/source code of the log generating system in order to parse logs using the respective print statements (Schipper et al., 2019; Xu et al., 2009; Zhang et al., 2017).

We mentioned in Section 3 that we included two approaches from non-academic literature: Splunk (Carasso, 2012) and Sequence (Zhen, 2014). Splunk is a commercial product (P-3) that offers features that exceed log clustering and is deployed in numerous organizations. However, also the authors of scientific papers share success stories about real-world application in their works, e.g., Lin et al. (2016) describe feedback and results following the implementation of their approach in a large-scale environment and Li et al. (2017) evaluate their approach within a case-study carried out in cooperation with an international company. We appreciate information about such deployments in real-world scenarios, because they validate that the algorithms are meeting the requirements for practical application. Finally, we could only find the original source code of (He et al., 2017a; 2017b; Makanju et al., 2009b; Messaoudi et al., 2018; Shima, 2016; Thaler et al., 2017; Vaarandi, 2003; 2004; Vaarandi and Pihelgas, 2015; Xu et al., 2009; Zhao and Xiao, 2016; Zhen, 2014) online (P-4). In addition, several reimplementations of algorithms provided by other authors exist. We encourage authors to make their code available open-source in order to enable reproducibility.

4.2. Clustering techniques (C)

In the following, we explore different types of applied clustering techniques with respect to their purpose, their applicability in live systems, and non-functional requirements.

4.2.1. Types of static clustering techniques

One of the most interesting findings of this research study turned out to be the large diversity of proposed clustering techniques (C-1a, C-1b). Considering static clustering approaches, a majority of the approaches employ a distance metric that determines the similarity or dissimilarity of two or more strings. Based on the resulting scores, similar log lines are placed in the same clusters, while dissimilar lines end up in different clusters. The calculation of the distance metric may thereby be character-based, token-based or a combination of both strategies (C-3, C-4). While token-based approaches assume that the log lines can reasonably be split by a set of predefined delimiters (most frequently, only white space is used as a delimiter), character-based approaches are typically more flexible, but also computationally more expensive. For example, Juvonen et al. (2015) and Christensen and Li (2013) compute the amount of common n-grams between two lines in order to determine their similarity. Du and Cao (2015), Ren et al. (2018), Salfner and Tschirpke (2008), and Wurzenberger et al. (2017a,b) use the Levenshtein metric to compute the similarity between two lines by counting the character insertions, deletions and replacements needed to transform one string into the other. Taerat et al. (2011), Gurumdimma et al. (2015), Jain et al. (2009), Zou et al. (2016), and Fu et al. (2009) employ a similar metric based on the words of a line rather than its characters. Another simple token-based approach for computing the similarity between two log lines is by summing up the amount of matching words at each position. In mathematical terms, this similarity between log lines a and b with their respective tokens $a_1, a_2, ..., a_n$ and $b_1, b_2, ..., b_m$ is computed by $\sum_{i=1}^{min(n,m)} \mathbb{I}(a_i, b_i)$, where $\mathbb{I}(a_i, b_i)$ is 1 if a_i is equal to b_i and 0 otherwise. This metric is frequently normalized (Aharon et al., 2009; He et al., 2017b; Li et al., 2018; Mizutani, 2013; Ning et al., 2014) and weighted (Hamooni et al., 2016; Tang and Li, 2010). Joshi et al. (2014) use bit patterns of tokens to achieve a similar result. Li et al. (2017) compute the similarity between log lines after transforming them into a tree-like structure. Du and Cao (2015) also consider the log level (e.g., INFO, WARN, ERROR) relevant for clustering and point out that log lines generated on a different level should not be grouped together. Finally, token vectors that emphasize the occurrence counts of words rather than their positions (i.e., the well-known bag of words model) may be used to compute the cosine similarity (Carasso, 2012; Lin et al., 2016; Shima, 2016) or apply k-means clustering (Aussel et al., 2018).

Not all approaches employ distance or similarity metrics. SLCT (Vaarandi, 2003) is one of the earliest published approaches for log clustering. The idea behind the concept of SLCT is that frequent tokens (i.e., tokens that occur more often than a user-defined threshold) represent fixed elements of log templates, while infrequent tokens represent variables. Despite being highly efficient, one of the downsides of SLCT is that clustering requires three passes over the data: The first pass over all log lines retrieves the frequent tokens, the second pass generates cluster templates by identifying these frequent tokens in each line and filling the gaps with wildcards, and the third pass reports cluster templates that represent sufficiently many log lines. Allocating the log lines to clusters is accomplished during the second pass, where each log line is assigned to an already existing or newly generated template.

Density-based clustering appears to be a natural strategy for generating trees (Qiu et al., 2010; Tovarňák and Pitner, 2019; Wurzenberger et al., 2019; Zhao and Xiao, 2016), i.e., data structures that represent the syntax of log data as sequences of nodes that branch into subsequences to describe different log events. Thereby, nodes represent fixed or variable tokens and may even differentiate between data types, e.g., numeric values or IP addresses. The reason why all of the reviewed approaches leveraging trees use density-based techniques is likely attributable to the way trees are built: Log messages are processed token-wise from their beginning to their end; identical tokens in all lines are frequent tokens that result in fixed nodes, tokens with highly diverse values are infrequent and result in variable nodes, and cases in between result in branches of the tree.

4.2.2. Types of dynamic clustering techniques

Several approaches pursue the clustering of log sequences rather than only grouping single log lines (C-6). Thereby, process IDs that uniquely identify related log lines may be exploited to retrieve the sequences (Lin et al., 2016). For example, Fu et al. (2009) use these IDs to build a finite state automaton describing the execution behavior of the monitored system. However, logs that do not contain such process IDs require mechanisms for detecting relations between identified log events. Du and Cao (2015) and Gurumdimma et al. (2015) first cluster similar log lines, then generate sequences by grouping events occurring in time windows and finally cluster the identified sequences in order to derive behavior patterns. Similarly, Salfner and Tschirpke (2008) group generated events that occur within a predefined inter-arrival time and cluster the sequences with a

hidden semi-Markov Model. Also Qiu et al. (2010) measure the inter-arrival time of log lines for clustering periodically occurring events and additionally group the events by derived correlation rules. Kimura et al. (2014) derive event co-occurrences by factorizing a 3-dimensional tensor consisting of the previously identified templates, hosts and time windows. DeepLog (Du et al., 2017) extends Spell (Du and Li, 2016) by computing probabilities for transitions between the identified log events in order to construct a workflow model. Jain et al. (2009) group time-series derived from cluster appearances in a hierarchical fashion. LogSed (Jia et al., 2017) and OASIS (Nandi et al., 2016) analyze frequent successors and predecessors of lines for mining a control flow graph. After first categorizing log messages using probabilistic models (Li et al., 2005) and distance-based strategies (Li et al., 2017), the authors determine the temporal relationships between log events by learning the distributions of their lag intervals, i.e., time periods between events. Other than the previous approaches, Aharon et al. (2009) assume that the order of log lines is meaningless and their algorithm PARIS thus identifies log events that frequently occur together within certain time windows regardless of their order.

We summarize the results in Table 2. For columns C-1a and C-1b, we coded distance-based strategies as (1) and density-based strategies as (2). Note that for static clustering, distances are usually measured between log lines and densities refer to token frequencies, while for dynamic clustering techniques, distances are computed between time-series of event occurrences and densities refer to event frequency counts. Other identified strategies used for static and dynamic clustering are (3) Neural Networks, which are useful for signature extraction (Kobayashi et al., 2014; Menkovski and Petkovic, 2017; Thaler et al., 2017) and event classification (Ren et al., 2018) by Natural Language Processing (NLP) as well as for detecting sequences in the form of Long Short-Term Memory (LSTM) recurrent neural networks (Du et al., 2017; Li et al., 2018; Zhang et al., 2019), (4) iterative partitioning, where groups of log lines are recursively split into subgroups according to particular token positions (Gainaru et al., 2011; He et al., 2017a; Makanju et al., 2009b), (5) Longest Common Substring (LCS), which is a measure for the similarity of log lines (Agrawal et al., 2019; Du and Li, 2016; He et al., 2017a; Jayathilake et al., 2017; Reidemeister et al., 2011; Tang et al., 2011) or sequences of log events (Wang et al., 2018), (6) binary/source code analysis (Schipper et al., 2019; Xu et al., 2009; Zhang et al., 2017), (7) genetic algorithms (Messaoudi et al., 2018), (8) frequent itemset mining (Vaarandi, 2004), (9) statistical modeling (Du et al., 2017; Kimura et al., 2014; Li et al., 2017; 2005; 2018), and (10) graph community extraction (Leichtnam et al., 2017). In addition, a number of approaches employ (11) heuristics for replacing tokens with wildcards if they match specific patterns, e.g., IP addresses or numeric values that most likely represent IDs. While such rules are frequently only used for preprocessing log data before clustering, the approaches by Chuah et al. (2010) and Jiang et al. (2008) suggest that heuristics alone may be sufficient to generate templates. Fig. 4 shows a visual overview of the techniques used in static log clustering. The plot shows that distance-based and density-based techniques are the most common techniques, being used in more than half of all reviewed approaches. Dynamic clustering techniques are less diverse: Most approaches apply statistical methods to generate links between events and rely on event count matrices for grouping and anomaly detection.

4.2.3. Applicability in live systems

Almost all approaches employ self-learning techniques that operate in an unsupervised fashion, i.e., no labeled training data is required for building the model of normal system behavior (C-2). This corresponds to the mentioned ambition of proposing algorithms that are mostly independent of the log structure and allow automatic processing with minimal human interference. However, we identified some approaches that do not follow this tendency and need labeled data for training: Kobayashi et al. (2014) use templates that define which tokens in log messages are fixed or variable, Thaler et al. (2017) also use such templates but mark every character of the log message as fixed or variable, Li et al. (2005) use categorical states that describe the type of log line, and (Reidemeister et al., 2011) use labels that describe types of failures. Other approaches rely on extensive manual work preceding clustering, including the manual extraction of relevant attributes into a common format (Leichtnam et al., 2017) or the definition of parsers (Tang and Li, 2010). Similarly, Chuah et al. (2010) and Zou et al. (2016) incorporate domain knowledge of the log structure in the clustering procedure.

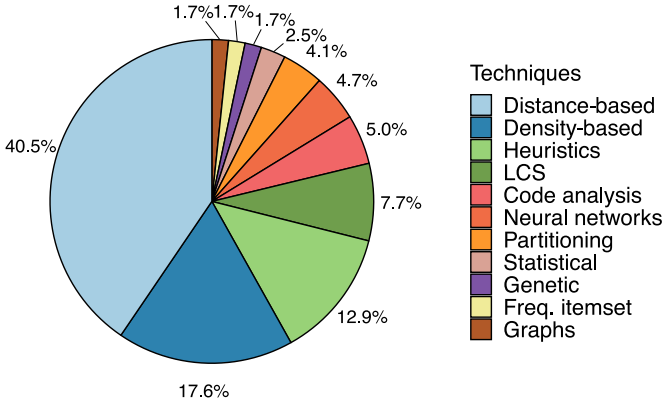


Fig. 4. Relative frequencies of static clustering techniques used in the reviewed articles.

Most articles lack precise investigations of running time complexities and upper bounds due to algorithmic complexity and parametric dependencies. However, we observed that some of the proposed approaches are particularly designed for online clustering (C-7), while others pursue offline or batch clustering. Online clustering means that at any given point in time during the clustering, all the processed log lines are already allocated to clusters. This usually implies that the running time grows at most linearly with the number of processed lines, which is an important property for many real-world applications where log lines are processed in streams rather than limited sets. Note that an allocation of lines to existing clusters in a streaming manner is almost always possible and we therefore only considered approaches that are able to generate new clusters on the fly as capable of online-processing. Typically, the reviewed online algorithms proceed in the following way: First, an empty set of clusters is initialized. Then, for each newly processed log line, the algorithm attempts to find a fitting cluster in the set of clusters. If such a cluster is found, the log line is allocated to it; otherwise, a new cluster containing that line is created and added to the set of clusters. This step is repeated indefinitely (Aharon et al., 2009).

In addition to generating new clusters, approaches that we consider adaptive are also able to modify existing cluster templates when new log lines are received (C-8). Such adaptive approaches are in particular useful when being employed in systems that undergo frequent changes, e.g., software upgrades or source code modifications that affect the logging behavior (Gainaru et al., 2011; Zhang et al., 2019). While non-adaptive approaches usually require a complete reformation of all clusters and templates, adaptive approaches dynamically adjust to the new baseline without the need of instantly “forgetting” all previously learned patterns. Approaches that do not aim at the generation of log templates may achieve

Table 2
Assessed properties regarding clustering techniques assigned to each approach.

Approach	C-1a	C-1b	C-2	C-3	C-4	C-5	C-6	C-7	C-8	C-9	C-10
Agrawal et al. (2019) (Logan)	5, 11		✓		✓	✓		✓	✓	✓	✓
Aharon et al. (2009) (PARIS)	1	1	✓		✓	✓	✓	✓		✓	
Aussel et al. (2018)	2, 11		✓		✓					~	
Christensen and Li (2013)	1		✓	✓				✓	✓	~	✓
Chuah et al. (2010) (Fdiag)	11	2, 9	~		✓	✓	✓	✓		~	
Du and Li (2016) (Spell)	1, 5		✓		✓	✓		✓	✓	✓	
Du et al. (2017) (DeepLog)	Du and Li (2016)	2, 3, 9	✓		✓		✓	✓	~	~	
Du and Cao (2015)	1, 11	1, 2	✓	✓			✓				
Fu et al. (2009)	1, 11	9	✓		✓	✓	✓		✓		✓
Gainaru et al. (2011) (HELO)	4		✓		✓	✓		✓	✓	~	
Gurumdimma et al. (2015)	1	1, 2, 9	✓		✓		✓				
Hamooni et al. (2016) (LogMine)	1		✓		✓	✓		✓	✓	✓	✓
He et al. (2017a) (POP)	4, 5		✓		✓	✓				✓	✓
He et al. (2017b) (Drain)	1		✓		✓	✓		✓	✓	✓	
Jain et al. (2009)	1	1, 2	✓		✓		✓	✓	✓	~	
Jayathilake et al. (2017)	5		✓	✓	✓	✓					
Jia et al. (2017) (LogSed)	Vaarandi (2003)	2, 9	✓	✓	✓	✓	✓			✓	✓
Jiang et al. (2008)	11		✓		✓	✓				✓	
Joshi et al. (2014)	1		✓		✓	✓		✓	✓	✓	
Juvonen et al. (2015)	1		✓	✓				✓			
Kimura et al. (2014)	9	9	✓		✓	✓	✓			~	
Kobayashi et al. (2014)	3				✓	✓					
Leichtnam et al. (2017) (STARLORD)	10				✓						
Li et al. (2005)		9			✓		✓			~	
Li et al. (2017) (FLAP)	1	9	✓		✓	✓	✓			~	
Li et al. (2018)	1, 11	3, 9	✓		✓	✓	✓	✓	✓	~	
Lin et al. (2016) (LogCluster)	Fu et al. (2009)	1	✓		✓		✓				
Makanju et al. (2009b) (IPLoM)	4		✓		✓	✓				✓	
Menkovski and Petkovic (2017)	1, 3		✓		✓	✓				~	
Messaoudi et al. (2018) (MoLFI)	7		✓		✓	✓				~	
Mizutani (2013) (SHISO)	1		✓	✓	✓	✓		✓	✓	✓	
Nagappan and Vouk (2010)	2		✓		✓	✓				✓	
Nandi et al. (2016) (OASIS)	1, 2, 5	9	✓	✓	✓	✓	✓	✓		✓	✓
Ning et al. (2014) (HLAer)	1		✓		✓	✓		✓	✓		✓
Qiu et al. (2010)	2, 11	9	✓		✓	✓	✓			~	
Reidemeister et al. (2011)	1, 2, 5			✓	✓	✓				✓	
Ren et al. (2018)	1, 3, 11			✓	✓					~	
Salfner and Tschirpke (2008)	1	9	✓	✓	✓		✓			~	
Schipper et al. (2019)	6		✓	✓	✓	✓				~	
Zhen (2014)	11		✓	~	✓	✓		✓	✓	✓	
Shima (2016) (LenMa)	1		✓		✓	✓		✓	✓	~	
Splunk Carasso (2012)	1		✓		✓			✓		✓	
Stearley (2004)(Teiresias)	2, Vaarandi (2003)	9	✓		✓	✓	✓				
Taerat et al. (2011) (Baler)	1		✓		✓	✓		✓		~	
Tang and Li (2010) (LogTree)	1		~		✓	✓		✓		✓	
Tang et al. (2011) (LogSig)	5		✓		✓	✓				✓	
Thaler et al. (2017)	3			✓		✓				~	
Tovarňák and Pitner (2019)	2, Nagappan and Vouk (2010)		✓		✓	✓				✓	✓
Vaarandi (2003) (SLCT)	2		✓		✓	✓				✓	
Vaarandi (2004) (LogHound)	Vaarandi (2003)	8	✓		✓	✓	✓			✓	
Vaarandi and Pihelgas (2015) (LogCluster)	2		✓		✓	✓				✓	
Wang et al. (2018)	1	5	✓		✓	✓	✓			~	
Wurzenberger et al. (2017a)	1		✓	✓						✓	
Wurzenberger et al. (2017b)	1		✓	✓				✓		✓	
Wurzenberger et al. (2019) (AECID-PG)	2		✓		✓	✓				~	
Xu et al. (2009)	6	2, 9	✓		✓	✓	✓			✓	✓
Zhang et al. (2017) (GenLog)	6	6	✓		✓	✓	✓			✓	
Zhang et al. (2019) (LogRobust)	2, 11, He et al. (2017b)	3	✓		✓	✓	✓	✓	✓	~	
Zhao and Xiao (2016)	2, 11		✓		✓	✓					
Zou et al. (2016) (UiLog)	1	9	~	✓	✓	✓	✓		~	~	
Zulkernine et al. (2013) (CAPRI)	2	9	✓	✓	✓	✓	✓	✓		✓	

adaptive behavior by only considering the most recently added log lines as relevant for clustering (Christensen and Li, 2013).

4.2.4. Non-functional requirements

The further columns provide information on whether the approaches were particularly designed for high efficiency (C-9) or parallel execution (C-10). Note that we considered a comparative evaluation on the efficiency of all algorithms out of scope for this survey, but rather assessed whether the authors particularly designed the algorithm for high log throughput, for example, by employing data structures or methods that enable fast data processing. In general, such an evaluation is difficult, because the running time often depends on the type of log data, parameter settings and data preprocessing.

Finally, we assessed that most algorithms operate in a deterministic fashion (C-11). However, some exceptions particularly make use of randomization, for example genetic algorithms (Messaoudi et al., 2018), randomized hash functions (Joshi et al., 2014), randomly initialized fields (Juvonen et al., 2015) and all approaches that rely on neural networks.

4.3. Anomaly Detection (AD)

According to our set of evaluation criteria, we group the approaches with respect to their ability to detect static or dynamic anomalies and discuss the origin of anomalies that are typically detected in the reviewed articles.

4.3.1. Static outlier detection

As mentioned before, not all reviewed articles primarily pursue anomaly detection (cf. Table 1) and thus do not include discussions about the effectiveness of their detection capabilities. However, the patterns or groups of log lines resulting from the clustering can always be used for the detection of anomalies. For example, log lines that are very dissimilar to all clusters or do not match any of the retrieved patterns are considered outliers (AD-1). New and previously unseen lines are usually regarded as suspicious and should be reported. In addition, clusters that are unusually small or very distant to all other clusters may indicate anomalous groups of log lines. Clearly, domain knowledge is required to interpret the retrieved lines and Hamooni et al. (2016) add that a keyword search on the new logs is an effective measure for system administrators to locate and interpret the occurred event.

SLCT (Vaarandi, 2003) and LogCluster (Vaarandi and Pihelgas, 2015) allocate all log lines in an outlier cluster if they do not match any of the generated log templates, i.e., patterns that represent each cluster. They used logs collected from a mail server and found that the identified outliers correspond to errors and unauthorized access attempts. In a similar manner, Wurzenberger et al. (2017b), Stearley (2004) and Splunk (Carasso, 2012) identify rare log lines that do not end up in large clusters as outliers. HLAer Ning et al. (2014) offers two possibilities for outlier detection: an online method based on pattern matching as well as an offline method that uses the same similarity score used for clustering. Similarly, Wurzenberger et al. (2017a) defines a similarity function for outlier detection and further mentions that small clusters contain potentially interesting log lines. Juvonen et al. (2015) detect outliers without the need for pattern matching. They inject cross-site scripting (XSS) attacks and the resulting log lines are located far away from all the other log lines when being projected into an euclidean space.

4.3.2. Dynamic anomaly detection

Other than detecting outliers, some algorithms aim at the detection of failure patterns (AD-2). Thereby, the retrieval of distinct and expressive descriptors is regarded as the main goal. For example, Baler (Taerat et al., 2011) identifies patterns corresponding to failure modes of the system CPU and memory errors. Such fault types are also detected by Zou et al. (2016) who group alerts that occur within time windows. Categories of these alerts thereby include errors caused by the network, failed authentications, peripheral devices and the web engine.

In addition, some approaches support root-cause analysis, where the identification of log events occurring in the past that relate to detected failures is pursued. Thereby, algorithms utilize the learned temporal dependencies between log events for such reasoning. Chuah et al. (2010) and Kimura et al. (2014) particularly focus on root-cause analysis and identify temporal dependencies by correlating event occurrences within time windows. However, Li et al. (2017, 2005) point out that a correct selection of the time window sizes is often difficult, and therefore propose a solution that relies on lag time distributions rather than time windows.

It is non-trivial to derive dynamic properties from clusterings. LogTree (Tang and Li, 2010) supports manual detection by displaying patterns of cluster appearances. In their case study, misconfigurations in HTML files were detected. For an analytical detection, Drain (He et al., 2017b) gradually fills an event count matrix that keeps track of the number of occurrences of each log event. They then use principal component analysis for detecting unusual points in the resulting matrix. Similarly, Xu et al. (2009) use PCA for detecting anomalies in high-dimensional message count vectors and additionally consider state variables for filling the matrix. Du and Cao (2015) detect anomalous system behavior by applying a distance metric on time-series derived from event frequencies.

Beside unusual frequencies of occurring events, the execution order of certain log line types may be used as another indicator for anomalies. Zulkernine et al. (2013) derive correlation rules from line patterns that frequently occur together. Fu et al. (2009) learn the execution order of events and detect deviations from this model as work flow errors. In addition, they identify performance issues by measuring the execution times of newly occurring log sequences and compare them with the learned behavior. Jia et al. (2017) also detect unknown logs as redundancy failures, deviations from execution orders as sequence anomalies and deviations from interval times as latency anomalies. Beside sequence errors, Nandi et al. (2016) make use of a control flow-graph in order to also detect changes of branching distributions, i.e., changes of occurrence probabilities of certain log events in sequences. DeepLog (Du et al., 2017) trains a Long Short-Term Memory (LSTM) neural network with such workflow transition probabilities and automatically detects any deviations from the learned behavior. A different approach is taken by Gurumdimma et al. (2015), who detect failure patterns of sequences rather than single events.

4.3.3. Cyber attack detection

Finally, we noted that although many approaches are directed towards anomaly detection, these anomalies are almost always considered to be random or naturally occurring failures rather than targeted cyber attacks (AD-3), such as denial-of-service and scanning attacks (Du et al., 2017; Wang et al., 2018), injections (Juvonen et al., 2015; Wurzenberger et al., 2017b), or unauthorized accesses (Reidemeister et al., 2011; Vaarandi, 2003). We assume that the reasons for this trend are manifold: (i) Failures may be more common than attacks in the considered systems and thus pose a higher risk, (ii) attacks are implicitly assumed to produce artifacts similar to failures and can thus be detected using the same methods, and (iii) lack of log files containing cyber attacks for evaluation.

4.4. Evaluation (E)

In the following, we investigate the procedures of evaluating approaches presented in the reviewed papers. In particular, we discuss what kind of evaluation techniques were applied to assess fulfillment of functional and non-functional requirements, and whether the results are reproducible.

4.4.1. Evaluation techniques

Every reviewed clustering approach includes at least some kind of experiments and discussion of results. As shown in Table 3, a majority of authors use quantitative metrics for validating and evaluating their proposed concepts (E-1). We identified three main approaches to quantitative evaluation: (i) Unsupervised methods that do not require a labeled ground truth for comparison. There exist various possibilities for estimating the quality of the clustering in an unsupervised fashion. Menkovski and Petkovic (2017) show that the consistency of the clustering can be assessed by measures such as the Silhouette Coefficient (Rousseeuw, 1987), which measures the relation between inter- and intra-cluster distances. Kimura et al. (2014) estimate the quality of the log templates heuristically, by assuming that all tokens without numbers should end up as fixed elements in the generated templates. This measure is easy to compute, but has the disadvantages that it may produce incorrect results in cases where

Table 3
Assessed properties regarding the evaluation carried out in each approach.

Approach	E-1	E-2	E-3	E-4	E-5	E-6
Agrawal et al. (2019) (Logan)	✓	✓	✓	✓	✓	
Aharon et al. (2009) (PARIS)		✓		✓	✓	
Aussel et al. (2018)	✓			✓	✓	
Christensen and Li (2013)	✓		~	✓	✓	
Chuah et al. (2010) (Fdiag)		✓			✓	✓
Du and Li (2016) (Spell)	✓		✓	✓	✓	✓
Du et al. (2017) (DeepLog)	✓	✓				✓
Du and Cao (2015)	✓			✓	✓	✓
Fu et al. (2009)	✓	✓		✓		
Gainaru et al. (2011) (HELO)	✓			✓	✓	✓
Gurumdimma et al. (2015)	✓			✓	✓	✓
Hamooni et al. (2016) (LogMine)	✓		✓	✓	✓	
He et al. (2017b) (POP)	✓		✓	✓	✓	✓
He et al. (2017a) (Drain)	✓		✓	✓	✓	✓
Jain et al. (2009)	✓		✓		✓	✓
Jayathilake et al. (2017)					✓	
Jia et al. (2017) (LogSed)	✓		✓		✓	✓
Jiang et al. (2008)	✓	✓		✓	✓	✓
Joshi et al. (2014)	✓			✓	✓	
Juvonen et al. (2015)			✓		✓	
Kimura et al. (2014)	✓	✓			✓	
Kobayashi et al. (2014)	✓		✓	✓	✓	
Leichtnam et al. (2017) (STARLORD)		✓			✓	✓
Li et al. (2005)	✓	✓			✓	
Li et al. (2017) (FLAP)	✓	✓	✓		✓	
Li et al. (2018)		✓	✓		✓	
Lin et al. (2016) (LogCluster)	✓	✓		✓	✓	
Makanju et al. (2009b) (IPLoM)	✓			✓	✓	✓
Menkovski and Petkovic (2017)	✓		✓	✓	✓	✓
Messaoudi et al. (2018) (MoLFI)	✓			✓	✓	
Mizutani (2013) (SHISO)		✓	✓	✓	✓	✓
Nagappan and Vouk (2010)				✓	✓	
Nandi et al. (2016) (OASIS)	✓		✓		✓	
Ning et al. (2014) (HLAer)		✓	✓	✓	✓	✓
Qiu et al. (2010)	✓	✓			✓	
Reidemeister et al. (2011)	✓			✓		
Ren et al. (2018)	✓			✓	✓	✓
Salfner and Tschirpke (2008)	✓				✓	
Schipper et al. (2019)	✓		✓		✓	
Zhen (2014)						
Shima (2016) (LenMa)		✓	✓	✓	✓	✓
Carasso (2012)						
Stearley (2004) (Teiresias)		✓		✓	✓	
Taerat et al. (2011) (Baler)		✓		✓	✓	
Tang and Li (2010) (LogTree)	✓		✓	✓	✓	✓
Tang et al. (2011) (LogSig)	✓		✓	✓	✓	
Thaler et al. (2017)	✓			✓		
Tovarňák and Pitner (2019)	✓	✓	✓	✓	✓	✓
Vaarandi (2003) (SLCT)			✓		✓	
Vaarandi (2004) (LogHound)			✓		✓	
Vaarandi and Pihelgas (2015) (LogCluster)			✓	✓	✓	
Wang et al. (2018)	✓					
Wurzenberger et al. (2017a)	✓		✓			
Wurzenberger et al. (2017b)	✓					
Wurzenberger et al. (2019) (AECID-PG)	✓			✓	✓	✓
Xu et al. (2009)	✓	✓	✓		✓	
Zhang et al. (2017) (GenLog)	✓	✓	✓		✓	✓
Zhang et al. (2019) (LogRobust)	✓				✓	✓
Zhao and Xiao (2016)	✓		✓		✓	
Zou et al. (2016) (UiLog)	✓	✓	✓		✓	✓
Zulkernine et al. (2013) (CAPRI)		✓		✓	✓	✓

the heuristics do not apply and that it is only reasonably applicable for log files where such heuristics are known to fit the data. Alternatively, Li et al. (2017) make use of event coverage during clustering, a measure for the goodness of a set of cluster descriptors with respect to their similarities to all log lines. The problem with such strategies is that it is typically difficult to obtain interpretable and comparable results and thus most of the reviewed approaches do not take unsupervised evaluation into consideration. (ii) The grouped log lines are compared to a manually crafted ground truth of cluster allocations. This allows the computation of the accuracy, precision, recall and F-score of the approach. Different strategies for computing these metrics are possible. For example, He et al. (2017b) count two log lines generated by the same event grouped in the same cluster as true positive; two lines generated by different events grouped in the same cluster as false positive and two lines generated by the same event grouped in different clusters as false negative. Contrary to such a line-based measure, Du and Li (2016) evaluate their approach with a more strict focus on clusters. They measure the accuracy by the number of lines allocated to correct clusters, where a cluster is counted

correct if all and only all log lines of a particular type from the ground truth are allocated to the same cluster. The results of line-based and cluster-based evaluations can be very different: Consider a clustering result containing one large cluster. A line-based accuracy measure will show good results as long as many log lines of that type end up in the same clusters, even if a portion of the lines end up in other clusters or a few misclassifications occurred. The accuracy measured in cluster-based evaluation on the other hand will indicate poor results when only one or few misclassifications occur in that cluster, since this causes that all contained lines are considered as incorrectly classified. Kobayashi et al. (2014) measure the accuracy by inspecting the templates rather than the associated log lines. In particular, they count the number of fields correctly identified as fixed or variable in each generated log template. Hamooni et al. (2016) apply a similar approach but also take types of fields, e.g., string, number or IP, into account. This approach appears particularly useful when obtaining a ground truth or labeling all log lines is not possible, but the number and structure of expected cluster templates can be determined. (iii) The quality of the clustering is assessed by its ability to detect anomalies. In this case, a ground truth of known anomalies is required for counting the true positives (correctly identified anomalies), false positives (incorrectly identified anomalies), false negatives (missed anomalies), and true negatives (correctly classified non-anomalous instances). The advantage of this method is that it does not require labels for log lines or knowledge about all clusters. However, anomaly-based evaluation relies on a data set containing anomalies and only measures the quality of the clustering indirectly, i.e., it is possible that an inappropriate detection mechanism is responsible for a poor detection accuracy even though the clustering is of good quality.

A number of approaches also qualitatively assess the clustering (E-2). This is especially common for approaches that aim at the extraction of log signatures. For example, Taerat et al. (2011) discuss the appropriateness of the number of clusters and outliers based on domain knowledge about the used log data. Moreover, they manually check whether unnecessary signatures exist or generated patterns are too general and thus lead to overgrouped clusters. In cases where a ground truth of expected signatures is available, differences and overlaps between the generated and expected patterns can be determined (e.g., Fu et al., 2009). Because of the ambiguities of what is considered an appropriate clustering, experts or administrators with domain knowledge about the specific real-world use cases are occasionally consulted for labeling the data (Xu et al., 2009) or validating the results (Aharon et al., 2009; Li et al., 2017; 2005; Lin et al., 2016; Qiu et al., 2010; Stearley, 2004).

4.4.2. Evaluation of Non-functional Requirements

Given that many approaches are particularly designed for fast processing of log lines, a high number of articles also include an empirical evaluation of running time requirements (E-3). Thereby, both the total time necessary to process a specific log file Messaoudi et al. (2018) as well as the scalability of the algorithm with respect to the number of processed log lines Mizutani (2013) are relevant characteristics.

4.4.3. Comparisons and Reproducibility

Most evaluations include thorough comparisons with one or multiple widely-applied approaches (E-4). For example, HLAer (Ning et al., 2014) is compared by Hamooni et al. (2016) with their algorithm LogMine regarding the accuracy of the generated signatures and SLCT Vaarandi (2003) is used as a benchmark by Joshi et al. (2014) for comparing the quality of the clustering and by Stearley (2004) regarding outlier detection. Fig. 5 shows an overview of approaches that are frequently used as benchmarks. Note that it is common that more than one approach is used for comparison, in which case the approaches were added in proportionally. As visible in the plot, the most frequently used algorithms for benchmarking are SLCT (Vaarandi, 2003) and IPLoM (Makanju et al., 2009b). It is also remarkable that all approaches visible in the plot are mainly used for signature extraction. This suggests that there exist more renowned standards for signature extraction than for other clustering approaches. It must however be noted that a majority of the reviewed articles employ signature extraction and thus dominate this statistic.

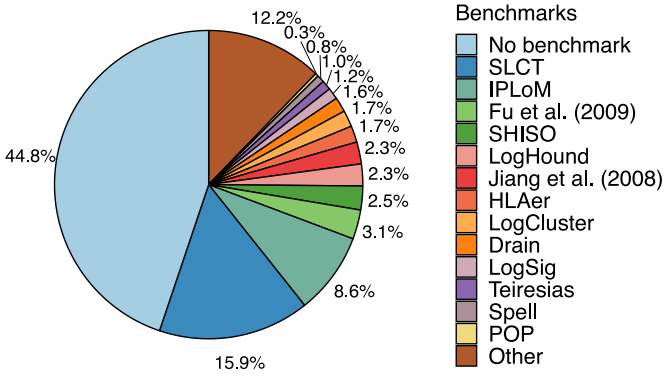


Fig. 5. Relative frequencies of benchmarks used for evaluation.

Most of the articles were evaluated with logs collected from real-world computer systems (E-5). Due to confidentiality of these logs, not all of them are publicly available (E-6). The most common open-source data sets used in the reviewed articles are the supercomputer logs[5] Blue Gene/L, Thunderbird, RedStorm, Spirit, Liberty, etc. (Agrawal et al., 2019; Chuah et al., 2010; Du and Li, 2016; Du and Cao, 2015; Gainaru et al., 2011; Gurumdimma et al., 2015; He et al., 2017a; 2017b; Jain et al., 2009; Jiang et al., 2008; Makanju et al., 2009b; Messaoudi et al., 2018; Ning et al., 2014; Tovarňák and Pitner, 2019; Wurzenberger et al., 2019); other sources are Hadoop Distributed File System (HDFS) logs[6] (Agrawal et al., 2019; Aussel et al., 2018; Du et al., 2017; He et al., 2017b; Messaoudi et al., 2018; Tovarňák and Pitner, 2019; Wurzenberger et al., 2019; Xu et al., 2009; Zhang et al., 2019), system logs[7] (Mizutani, 2013; Shima, 2016), and web logs[8] (Zulkernine et al., 2013). The artificially generated network logs data[9] used by Du et al. (2017) is particularly interesting, because it comes with a ground truth and information on the attacks that were injected during the data collection. Fig. 6 shows an overview of log data sources used in the reviewed papers. Note that approaches that use multiple logs, e.g., supercomputer and HDFS logs, were added in proportionally, and evaluation on non-available data ("Private data") was only counted when there was no evaluation on publicly available data. As visible in the plot, almost 60% of the reviewed approaches involve non-reproducible evaluation.

4.5. Discussion

Based on the results outlined in the previous section, several findings can be derived. In the following, we discuss identified issues with the overall problem of clustering log data, disadvantages of employed clustering techniques, and frequently encountered issues in evaluation.

[5] https://www.usenix.org/cfdr-data.
[6] http://iiis.tsinghua.edu.cn/~weixu/sospdata.html.
[7] http://log-sharing.dreamhosters.com/.
[8] http://cs.queensu.ca/~farhana/supporting-pages/capri.html.
[9] https://www.cs.umd.edu/hcil/varepository/VAST%20Challenge%202011/challenges/MC2%20-%20Computer%20Networking%20Operations/.

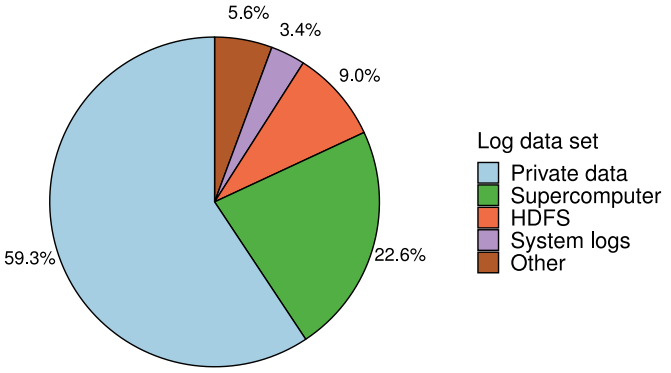


Fig. 6. Relative frequencies of log data used for evaluation.

4.5.1. Problem domains

We did not expect to see such a high number of articles primarily focused on the extraction and generation of signatures, while comparatively few articles are oriented towards anomaly detection. Especially static outlier detection, i.e., the identification of log lines with unusual structure or content, appears to be more of a by-product rather than a main feature of signature generating approaches. This may of course be attributable to the fact that such a detection is often a trivial subsequent step to any clustering method. On the other hand, dynamic anomaly detection such as correlation analysis and especially the identification of sequences of log lines appears to be of a higher relevance and the problem of missing sequence identifiers (process IDs discussed in Section 2.1) is tackled with various strategies.

4.5.2. Techniques

We were surprised to observe discords regarding some general assumptions on the nature of log files. First of all, we noted a tendency to employ token-based approaches rather than character-based approaches. We attribute this to the fact that token-based strategies are generally computationally less expensive and align better with heuristics, for example, replacing numeric values with wildcards before carrying out a more sophisticated clustering procedure. Despite these advantages, we think that character-based approaches have high potential of generating more precise cluster templates. We have already pointed out in Wurzenberger et al. (2017a) that token-based approaches are unable to correctly handle words that differ only slightly, e.g., URLs or words with identical semantic meaning such as “u.user” and “t.user” that are frequently found in SQL logs. Moreover, choosing a set of delimiters that are used to split the log messages into tokens is not trivial in practice, because different sets of delimiters may be required for appropriately tokenizing different log messages (Jiang et al., 2019; Tovarňák and Pitner, 2019).

We also observed that several token-based algorithms compare tokens only at identical positions. The problem with such a strategy is that optional tokens or tokens consisting of multiple words shift the positions of the remaining tokens in the log line. This may cause otherwise similar log lines to incorrectly end up in different clusters (Tovarňák and Pitner, 2019). While some articles such as Makanju et al. (2009b) explicitly state or implicitly assume that the order of words is relevant for clustering, others (e.g., Vaarandi and Pihelgas, 2015) particularly design their algorithms to be resilient to word shifts. Since optional words and free text are common in most unstructured log files, we recommend to carry out research on approaches that alleviate these issues.

We stated in Section 3.2 that approaches on protocol reverse engineering (Narayan et al., 2016) are excluded from this survey, because of their focus on network protocols rather than system log data. However, we see the application of algorithms from protocol reverse engineering for the generation of log signatures as a potentially interesting area for future research. The reason for this is that existing protocol reverse engineering approaches often consider both character-based matching through n-grams as well as the positions of these n-grams or tokens relevant for the extraction of protocol syntaxes. Adapting concepts from protocol reverse engineering may thus effectively alleviate the previously described issues with existing log signature extraction approaches.

4.5.3. Benchmarking & Evaluation

Despite of the fact that SLCT (Vaarandi, 2003) is one of the first algorithms designed for log clustering, it is still regarded as de facto standard due to its open-source availability. However, more recent articles demonstrated its weaknesses and proposed alternative clustering strategies that largely improved the quality of clusters and signatures. In particular, SLCT generates overly general clusters consisting of only wildcards, which obviously cover a large number of log lines but provide little to no information for the user, as well as overly specific patterns of similar log lines (Taerat et al., 2011). We therefore suggest to employ more recent alternative approaches for benchmarking in future articles. In addition, SLCT and other standards such as LogHound (Vaarandi, 2004), LogCluster (Vaarandi and Pihelgas, 2015), and IPLoM (Makanju et al., 2009b) only operate on fixed-size log files and are not able to incrementally process log lines for clustering. However, since stream processing is essential for grouping logs in real-world environments where frequent reclusterings on training sets are not an option, we argue that algorithms capable of such online analyses are superior regarding their applicability.

We observed that evaluating log clustering approaches is far from trivial. In order to quantitatively determine the quality of the generated clusters, anomalies or patterns, ground truth consisting of labeled log data or at least expected signatures are required. Moreover, since log data collected from specific sources often exhibits peculiarities, proper evaluation should always be carried out on multiple data sets. However, generating labeled data usually requires time-consuming manual work. Thus, open-source labeled log data would be highly beneficial for objective comparisons and would allow researchers to benchmark approaches in a thorough manner. We were pleased to see that He et al. (2017a,b) not only provide the code of their algorithms, but also reimplement other approaches and further collect log data sets including labels[10] that specify which log lines belong to the same clusters, i.e., originate from the same log events. This enables reproducibility and proper comparisons among different approaches. We hope to see more researchers contributing or making their data and code accessible to the public.

Finally, we also point out that only few authors injected actual attacks in their data sets, but rather targeted failures and errors. While these could of course be artifacts of attacks, we suggest to use real attack scenarios in future evaluations. Since anomaly detection is applied in intrusion detection solutions, use cases more closely related to cyber threats have the potential to expand the possible application areas.

5. Approach selection

We identified and assessed several properties of log clustering algorithms in the previous sections and used them to group existing approaches into meaningful classes. Many of the identified characteristics are essential for achieving application-specific

[10] https://github.com/logpai.

goals and the prepared tables and discussions support administrators and analysts in making appropriate selections for their systems at hand. However, in practice it is often difficult to link the objectives of the analysis to characteristics of clustering algorithms. This is due to the fact that there is usually not one, but a range of relevant attributes, where each attribute may be of different importance for the analysts.

For illustrating this issue, consider a scenario where an expert is asked to pick a particular log clustering algorithm. In this scenario, a vendor for log management solutions employs security analysts who implement a module for a Security Information and Event Management (SIEM) tool. Their customers are large-scale enterprises that deploy the log management solutions in industrial production systems consisting of numerous IoT-devices, including microprocessors and sensors, that are connected to servers and networks. Since it is difficult to analyze these complex systems with standard procedures, the customers report their demands to the security analysts who gather all the received information. It turns out that one of the main issues is that many components regularly undergo modifications such as firmware updates and thus change their logging behavior, which is not supported by the tools. In addition to such reconfigurations, the structures of the log lines follow no particular standards and can therefore not be analyzed with simple existing parser models. The customers struggle to detect and interpret occasionally occurring error messages since they lack overview of the log data, which is constantly produced in massive amounts. After careful consideration of these insights, the security analysts decide to solve this problem by first extracting the log signatures and then classifying the log data in order to present it in a more structured way. This further enables the creation of statistical profiles for each log event and is a basis for the generation of rules for alerting. However, due to the fact that a rather large number of log clustering approaches exist, it is not obvious which one to pick. Therefore, we describe a process of identifying one or a small number of appropriate solutions in the following.

When searching for existing clustering approaches, going through all available options one by one often involves time-consuming tasks such as reviewing each approach multiple times and carrying out pairwise comparisons. As a solution, we propose a model following the weighted sum method for multi-objective optimization (Marler and Arora, 2004) that eases the decision process. As a first step, we suggest to analyze the attributes and properties of the prevalent situation that elicited the need for log clustering and derive the requirements on the algorithms subsequently. In particular, we identify three areas that determine which approach fits best to a given situation and should be considered before reviewing existing approaches:

1. *Objectives.* Defining which objectives are pursued limits the search space, since most algorithms are designed towards specific goals and are thus unable to fulfill all demands. The approaches reviewed in this article were divided in four major categories regarding their overall purpose (cf. Section 4.1). We point out again that such classifications are usually not mutually exclusive and may be further refined, e.g., in this survey we did not differentiate between the identification of frequently occurring log sequences and the detection of anomalous log sequences. In addition, other limiting factors related to the application of the algorithm should be considered, e.g., source code may be required in order to utilize the solution without the need for reimplementations.
2. *Log data.* Several characteristics of log data that influence the performance and quality of log clustering were already outlined in Section 2.1. In particular, some clustering algorithms enable advanced techniques such as the extraction of log traces, but demand that log lines adhere to certain standards or include special artifacts such as process IDs. Depending on whether these features are of relevance and the log data fulfills the requirements, it is possible to exclude a number of approaches from further consideration. Moreover, a thorough investigation of the log data at hand yields further insights into requirements on the clustering algorithms, for example, log data may be difficult to tokenize using only white space as a delimiter, contain mostly key-value pairs, involve multi-line log messages, etc.
3. *System requirements.* Log lines are generated under certain conditions determined by the system. For example, when a high log level is set and it is common that many events occur in a system, clustering algorithms that are able to handle the resulting high quantities of log lines are essential. On the other hand, efficiency may be less important when a system is analyzed forensically, i.e., not in real-time. Similarly, system landscapes that are subject to frequent modifications require algorithms that are able to handle dynamic changes of the cluster structures. All such restrictions caused by the system that generates the log lines indicate which approaches may be better suited than others.

Additional attributes that may be of relevance may be derived from meta information of the approaches, e.g., their evaluation outlined in the respective papers. Since we gathered comprehensive data in course of carrying out this survey, we considered all attributes that were used in the survey for the approach selection procedure.

The next step of our model is to assign a weight to each attribute that represents its relevance to the analysis. These weights are then multiplied with the values ascertained for the respective properties of the reviewed approaches. For the survey results presented in the tables in Section 4, this corresponds to multiplying the weight of a specific attribute with 1.0 if the approach fulfills that property (✓), 0.5 if the approach partially fulfills that property (~) and 0.0 if the approach does not fulfill that property. Since some of the attributes are categorical and not mutually exclusive, e.g., the clustering techniques, one-hot encoding was used to avoid problems with ordinal values (Harris and Harris, 2007). The weighted attributes are then summed up row-wise in the tables, i.e., the score of each approach is the sum of the weighted attributes. Note that the user-defined weights thereby act similar to a query, i.e., attributes that are weighted higher have a larger influence on the score, attributes that are weighted 0.0 do not influence the score and attributes with scores < 0.0 negatively affect the score.

It is then possible to rank the approaches according to the computed scores, where approaches that are ranked higher fulfill more relevant attributes. In order to increase the understandability and comparability of the scores, we suggest to divide all scores by their maximum, i.e., normalize them in the range [0, 1].

In order to apply these principles to the scenario that was proposed earlier in this section, the weights for the attributes have to be defined. In the scenario, the security analysts primarily pursue log signature extraction, thus the respective attributes are weighted 1.0, and also desire a better overview on the data, which is weighted 0.5. In addition, the constraints from the system and data demand online, adaptive and fast processing of log lines, which are thus weighted 1.0.

In order to evaluate this scenario and also give other users with different scenarios at hand the possibility to make use of our findings, we implemented the ranking model as a web-based application that is accessible online.[11] Fig. 7 shows the main interface, consisting of an input form for weighting the attributes, the list containing the ranked approaches and their respective scores, and

[11] http://81.189.135.168/selection/.

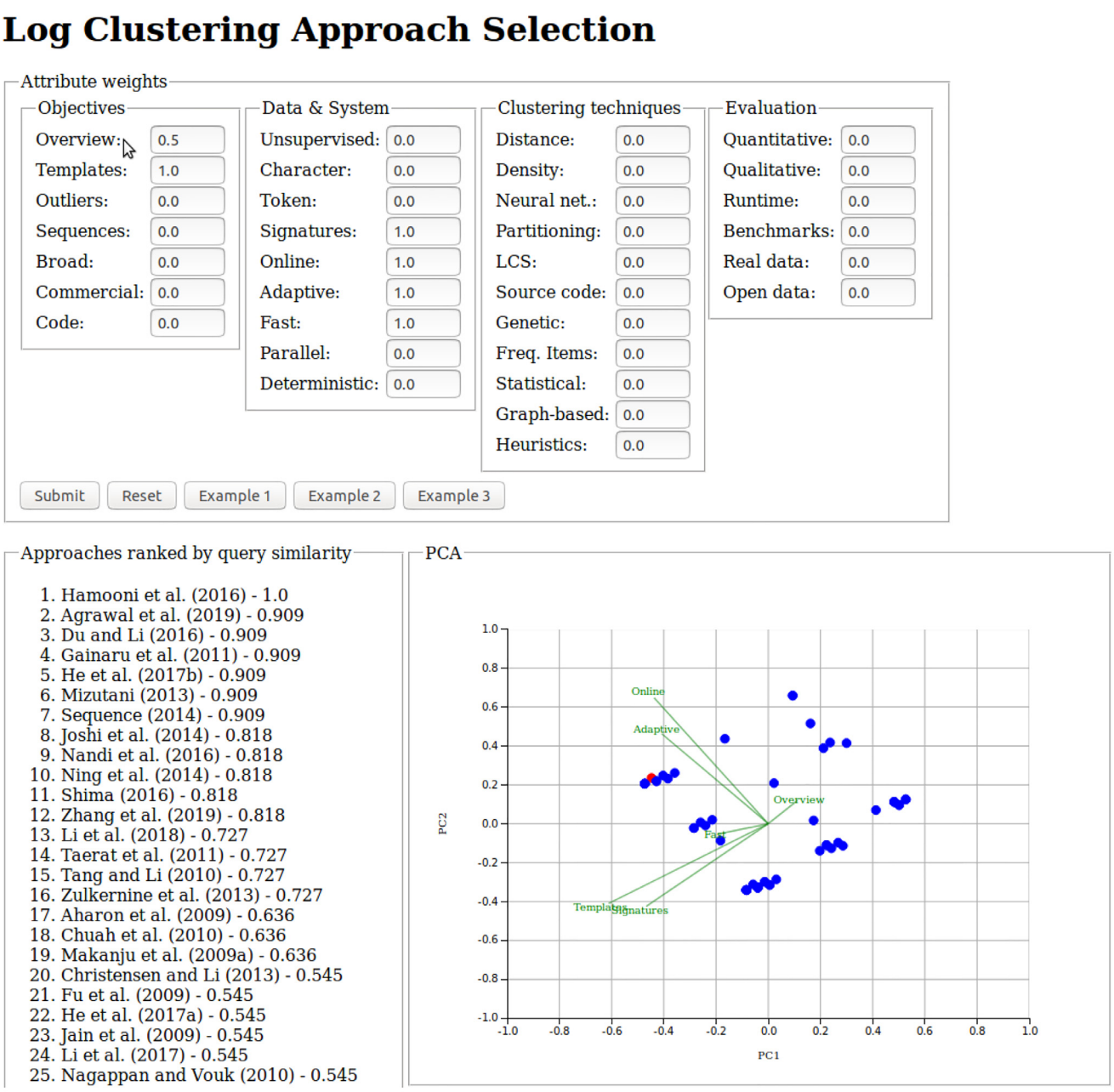


Fig. 7. Web interface of the Log Clustering Approach Selection tool. Top: Input form for attribute weights. Bottom left: Approaches ranked by their respective similarity to the query. Bottom right: PCA plot showing clusters of similar approaches.

a plot showing a principal component analysis (PCA) (Jolliffe, 2005) of all approaches (blue points) and all feature axes (green lines) with a corresponding attribute weight other than 0.0. The names of the approaches appear when the user hovers the mouse pointer over any point and the red point signalizes the approach that achieved the highest score. Clicking the "Submit" button recalculates the ranking and generates a new plot for the currently entered weights, the "Reset" button resets all weights to 0.0, and the "Example" buttons fill the weights for predefined scenarios. "Example 2" corresponds to the scenario outlined in this section.

We included PCA because it provides a convenient way of visually comparing all approaches with respect to multiple attributes in a single plot. Similar approaches are located close to each other and thus form clusters. As a consequence, it is easy to find alternative approaches in case that the best ranked approach is not applicable. On top of that, following the direction of the feature axes allows the reader to discover approaches that fulfill or lack the respective attributes represented by these axes. Finally, by nature of the PCA, related axes are somewhat aligned with each other. For example, the figure shows that approaches that pursue the extraction of templates ("Templates") are very likely also approaches that make use of signatures as means of clustering ("Signatures"). This allows users to discover attributes that usually occur together, which are useful insights when the search is performed iteratively, i.e., the query is repeatedly modified depending on the results observed in the ranking and plot.

The figure shows that the approach by Hamooni et al. (2016) achieved the highest score and is thus the best fitting solution for the situation outlined in the scenario. Note that the score of 1.0 does not imply that all attributes with a weight not equal to 0.0 are necessarily fulfilled, since the scores are always computed relatively to the maximum of all scores, i.e., there is always at

least one approach with a score of 1.0. The figure also shows that there are several other approaches close to the point that represents the highest-scoring approach. Not surprisingly, these are the approaches that also achieved high scores in the ranking. As mentioned, it is now possible to discover groups of approaches that are related to each other by following the direction of the feature axes. For example, the group of approaches in the bottom of the plot deal with signature extraction but do not support online or adaptive log processing.

We briefly outline the two remaining default examples we provide on the website of our selection tool. Example 1 searches for online log clustering approaches that are capable of detecting anomalies in single lines or sequences of lines. Accordingly, the fields “Outliers”, “Sequences”, and “Online” are weighted 1.0. Such a use-case could be typical for an enterprise environment, where analysts are mainly interested in the automatic detection of failures or attacks, but the generation of templates or application of particular algorithms is of less relevance. The results show that only Splunk (Carasso, 2012) fulfills all requirements set by the scenario. Example 3 on the other hand represents a more specific search on character-based approaches for outlier detection. In addition, approaches that enable fast processing and were evaluated on real data should be ranked higher. This setting could refer to a scenario where it is known that splitting log data into tokens is difficult (cf. Section 4.5) and produced in fast rates, e.g., logs collected from databases. For such a scenario, our model selection tool assigns the highest rank to the approach by Juvonen et al. (2015).

6. Conclusion

Log clustering plays an important role in many application fields, including cyber security. Accordingly, a great number of approaches have been developed in the past. However, existing approaches are often designed towards particular objectives, make specific assumptions on the log data and exhibit characteristics that prevent or foster their application in certain domains. This makes it difficult to select one or multiple approaches for a use case at hand. In this paper we therefore carried out a survey that groups clustering approaches according to attributes that support such decisions. For this, we created a set of evaluation criteria that breaks down aspects of clustering approaches that are relevant with respect to objectives, clustering techniques, anomaly detection, and evaluation. We assessed these attributes using 59 approaches proposed in scientific literature as well as 2 non-academic solutions.

We found that approaches usually pursue one or more of four major objectives: overview and filtering, parsing and signature extraction, static outlier detection, and sequences and dynamic anomaly detection. We were further able to identify different categories of clustering techniques, including partitioning, neural networks, source code analysis as well as token-based, character-based, distance-based and density-based techniques. We investigated which approaches are suitable for detecting specific types of anomalies in the log data, discussed how the evaluation is carried out in the reviewed articles and made several suggestions for future work. Finally, we presented our tool for selecting log clustering approaches based on the data collected throughout the survey and a list of attributes weighted by the user. The tool ranks the approaches according to their ability to fulfilling the queried attributes and visualizes the resulting groups in a PCA plot.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was partly funded by the FFG projects INDICAETING (868306), IoT4CPS (863129), and DECEPT (873980), and the EU H2020 project GUARD (833456).

References

Agrawal, A., Karlupia, R., Gupta, R., 2019. Logan: a distributed online log parser. In: Proceedings of the 35th International Conference on Data Engineering (ICDE). IEEE, pp. 1946–1951. doi:10.1109/ICDE.2019.00211.

Aharon, M., Barash, G., Cohen, I., Mordechai, E., 2009. One graph is worth a thousand logs: uncovering hidden structures in massive system event logs. In: Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Springer, pp. 227–243. doi:10.1007/978-3-642-04180-8_32.

Allen, R., Richardson, B., 2019. Neural network, that’s the tech; to free your staff from, bad regex. [Online; accessed 19-December-2019].

Aussel, N., Petetin, Y., Chabridon, S., 2018. Improving performances of log mining for anomaly prediction through nlp-based log parsing. In: Proceedings of the International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS). IEEE, pp. 237–243. doi:10.1109/MASCOTS.2018.00031.

Balakrishnan, R., Sahoo, R.K., 2006. Lossless compression for large scale cluster logs. In: Proceedings of the 20th International Parallel & Distributed Processing Symposium. IEEE, p. 7. doi:10.1109/IPDPS.2006.1639692.

Bao, L., Li, Q., Lu, P., Lu, J., Ruan, T., Zhang, K., 2018. Execution anomaly detection in large-scale systems through console log analysis. J. Syst. Softw. 143, 172–186. doi:10.1016/j.jss.2018.05.016.

Beeferman, D., Berger, A., 2000. Agglomerative clustering of a search engine query log. In: Proceedings of the 6th International Conference on Knowledge Discovery and Data Mining. ACM, pp. 407–416. doi:10.1145/347090.347176.

Carasso, D., 2012. Exploring splunk. CITO Research, New York, USA, p. 156.

Carpineto, C., Osiński, S., Romano, G., Weiss, D., 2009. A survey of web clustering engines. ACM Comput. Surv. (CSUR) 41 (3), 17:1–17:38. doi:10.1145/1541880.1541884.

Christensen, R., Li, F., 2013. Adaptive log compression for massive log data. In: Proceedings of the International Conference on Management of Data. ACM, p. 1283. doi:10.1145/2463676.2465341.

Chuah, E., Kuo, S.-h., Hiew, P., Tjhi, W.-C., Lee, G., Hammond, J., Michalewicz, M.T., Hung, T., Browne, J.C., 2010. Diagnosing the root-causes of failures from cluster log files. In: Proceedings of the International Conference on High Performance Computing (HiPC). IEEE, pp. 1–10. doi:10.1109/HIPC.2010.5713159.

Du, M., Li, F., 2016. Spell: Streaming parsing of system event logs. In: Proceedings of the 16th International Conference on Data Mining (ICDM). IEEE, pp. 859–864. doi:10.1109/ICDM.2016.0103.

Du, M., Li, F., Zheng, G., Srikumar, V., 2017. Deeplog: anomaly detection and diagnosis from system logs through deep learning. In: Proceedings of the Conference on Computer and Communications Security. ACM, pp. 1285–1298. doi:10.1145/3133956.3134015.

Du, S., Cao, J., 2015. Behavioral anomaly detection approach based on log monitoring. In: Proceedings of the International Conference on Behavioral, Economic and Socio-cultural Computing (BESC). IEEE, pp. 188–194. doi:10.1109/BESC.2015.7365981.

Ewards, V., et al., 2013. A survey on signature generation methods for network traffic classification. Int. J. Adv. Res. Comput. Sci. 4 (2). doi:10.26483/ijarcs.v4i4.1594.

Facca, F.M., Lanzi, P.L., 2005. Mining interesting knowledge from weblogs: a survey. Data Knowl. Eng. 53 (3), 225–241. doi:10.1016/j.datak.2004.08.001.

Fu, Q., Lou, J.-G., Wang, Y., Li, J., 2009. Execution anomaly detection in distributed systems through unstructured log analysis. In: Proceedings of the 9th International Conference on Data Mining (ICDM’09). IEEE, pp. 149–158. doi:10.1109/ICDM.2009.60.

Gainaru, A., Cappello, F., Trausan-Matu, S., Kramer, B., 2011. Event log mining tool for large scale hpc systems. In: Proceedings of the European Conference on Parallel Processing. Springer, pp. 52–64. doi:10.1007/978-3-642-23400-2_6.

Gurumdimma, N., Jhumka, A., Liakata, M., Chuah, E., Browne, J., 2015. Towards detecting patterns in failure logs of large-scale distributed systems. In: Proceedings of the International Parallel and Distributed Processing Symposium Workshop (IPDPSW). IEEE, pp. 1052–1061. doi:10.1109/IPDPSW.2015.109.

Hamooni, H., Debnath, B., Xu, J., Zhang, H., Jiang, G., Mueen, A., 2016. Logmine: fast pattern recognition for log analytics. In: Proceedings of the 25th International Conference on Information and Knowledge Management. ACM, pp. 1573–1582. doi:10.1145/2983323.2983358.

Harris, D.M., Harris, S.L., 2007. Chapter 3 - sequential logic design. In: Digital Design and Computer Architecture. Morgan Kaufmann, Burlington, pp. 103–165. doi:10.1016/B978-012370497-9/50004-4.

He, P., Zhu, J., He, S., Li, J., Lyu, M.R., 2017. Towards automated log parsing for large-scale log data analysis. Trans. Depend. Secure Comput. doi:10.1109/TDSC.2017.2762673.

He, P., Zhu, J., Zheng, Z., Lyu, M.R., 2017. Drain: an online log parsing approach with fixed depth tree. In: Proceedings of the International Conference on Web Services (ICWS). IEEE, pp. 33–40. doi:10.1109/ICWS.2017.13.

Jain, S., Singh, I., Chandra, A., Zhang, Z.-L., Bronevetsky, G., 2009. Extracting the textual and temporal structure of supercomputing logs. In: Proceedings of the International Conference on High Performance Computing (HiPC). IEEE, pp. 254–263. doi:10.1109/HIPC.2009.5433202.

Jayathilake, P., Weeraddana, N., Hettiarachchi, H., 2017. Automatic detection of multi-line templates in software log files. In: Proceedings of the 17th International Conference on Advances in ICT for Emerging Regions (ICTer). IEEE, pp. 1–8. doi:10.1109/ICTER.2017.8257824.

Jia, T., Yang, L., Chen, P., Li, Y., Meng, F., Xu, J., 2017. Logsed: anomaly diagnosis through mining time-weighted control flow graph in logs. In: Proceedings of the 10th International Conference on Cloud Computing (CLOUD). IEEE, pp. 447–455. doi:10.1109/CLOUD.2017.64.

Jiang, J., Versteeg, S., Han, J., Hossain, M.A., Schneider, J.-G., Leckie, C., Farahmandpour, Z., 2019. P-gram: positional n-gram for the clustering of machine-generated messages. IEEE Access 7, 88504–88516. doi:10.1109/ACCESS.2019.2924928.

Jiang, Z.M., Hassan, A.E., Hamann, G., Flora, P., 2008. An automated approach for abstracting execution logs to execution events. J. Softw. 20 (4), 249–267. doi:10.1002/smr.v20:4.

Jolliffe, I., 2005. Principal Component Analysis. American Cancer Society.

Joshi, B., Bista, U., Ghimire, M., 2014. Intelligent clustering scheme for log data streams. In: Proceedings of the International Conference on Intelligent Text Processing and Computational Linguistics. Springer, pp. 454–465. doi:10.1007/978-3-642-54903-8_38.

Juvonen, A., Sipola, T., Hämäläinen, T., 2015. Online anomaly detection using dimensionality reduction techniques for http log analysis. Comput. Netw. 91, 46–56. doi:10.1016/j.comnet.2015.07.019.

Kimura, T., Ishibashi, K., Mori, T., Sawada, H., Toyono, T., Nishimatsu, K., Watanabe, A., Shimoda, A., Shiomoto, K., 2014. Spatio-temporal factorization of log data for understanding network events. In: Proceedings of the Conference on Computer Communications (INFOCOM). IEEE, pp. 610–618. doi:10.1109/INFOCOM.2014.6847986.

Kobayashi, S., Fukuda, K., Esaki, H., 2014. Towards an nlp-based log template generation algorithm for system log analysis. In: Proceedings of the 9th International Conference on Future Internet Technologies. ACM, pp. 11:1–11:4. doi:10.1145/2619287.2619290.

Leichtnam, L., Totel, E., Prigent, N., Mé, L., 2017. Starlord: Linked security data exploration in a 3d graph. In: Proceedings of the Symposium on Visualization for Cyber Security (VizSec). IEEE, pp. 1–4. doi:10.1109/VIZSEC.2017.8062203.

Li, T., Jiang, Y., Zeng, C., Xia, B., Liu, Z., Zhou, W., Zhu, X., Wang, W., Zhang, L., Wu, J., et al., 2017. Flap: an end-to-end event log analysis platform for system management. In: Proceedings of the 23rd International Conference on Knowledge Discovery and Data Mining. ACM, pp. 1547–1556. doi:10.1145/3097983.3098022.

Li, T., Liang, F., Ma, S., Peng, W., 2005. An integrated framework on mining logs files for computing system management. In: Proceedings of the 11th International Conference on Knowledge Discovery in Data Mining. ACM, pp. 776–781. doi:10.1145/1081870.1081972.

Li, Z., Davidson, M., Fu, S., Blanchard, S., Lang, M., 2018. Converting unstructured system logs into structured event list for anomaly detection. In: Proceedings of the 13th International Conference on Availability, Reliability and Security. ACM, pp. 15:1–15:10. doi:10.1145/3230833.3230855.

Lin, Q., Zhang, H., Lou, J.-G., Zhang, Y., Chen, X., 2016. Log clustering based problem identification for online service systems. In: Proceedings of the 38th International Conference on Software Engineering Companion. ACM, pp. 102–111. doi:10.1145/2889160.2889232.

Makanju, A., Zincir-Heywood, A.N., Milios, E.E., et al., 2009. Extracting message types from bluegene/l's logs. In: Proceedings of the SOSP Workshop on the Analysis of System Logs (WASL).

Makanju, A.A., Zincir-Heywood, A.N., Milios, E.E., 2009. Clustering event logs using iterative partitioning. In: Proceedings of the 15th International Conference on Knowledge Discovery and Data Mining. ACM, pp. 1255–1264. doi:10.1145/1557019.1557154.

Marler, R.T., Arora, J.S., 2004. Survey of multi-objective optimization methods for engineering. Struct. Multidiscip. Optim. 26 (6), 369–395. doi:10.1007/s00158-003-0368-6.

Menkovski, V., Petkovic, M., 2017. Towards unsupervised signature extraction of forensic logs. In: Proceedings of the 26th Benelux Conference on Machine Learning, pp. 154–160. doi:10.1007/978-3-319-71273-4_25.

Messaoudi, S., Panichella, A., Bianculli, D., Briand, L., Sasnauskas, R., 2018. A search-based approach for accurate identification of log message formats. In: Proceedings of the 26th International Conference on Program Comprehension (ICPC'18). ACM doi:10.1145/3196321.3196340.

Mizutani, M., 2013. Incremental mining of system log format. In: Proceedings of the International Conference on Services Computing (SCC). IEEE, pp. 595–602. doi:10.1109/SCC.2013.73.

Nagappan, M., Vouk, M.A., 2010. Abstracting log lines to log event types for mining software system logs. In: Proceedings of the 7th Working Conference on Mining Software Repositories (MSR). IEEE, pp. 114–117. doi:10.1109/MSR.2010.5463281.

Nandi, A., Mandal, A., Atreja, S., Dasgupta, G.B., Bhattacharya, S., 2016. Anomaly detection using program control flow graph mining from execution logs. In: Proceedings of the 22nd International Conference on Knowledge Discovery and Data Mining. ACM, pp. 215–224. doi:10.1145/2939672.2939712.

Narayan, J., Shukla, S.K., Clancy, T.C., 2016. A survey of automatic protocol reverse engineering tools. ACM Comput. Surv. (CSUR) 48 (3), 40:1–40:26. doi:10.1145/2840724.

Ning, X., Jiang, G., Chen, H., Yoshihira, K., 2014. Hlaer: a system for heterogeneous log analysis.

Portnoy, L., Eskin, E., Stolfo, S., 2001. Intrusion detection with unlabeled data using clustering. In: Proceedings of the Workshop on Data Mining Applied to Security (DMSA), pp. 5–8.

Qiu, T., Ge, Z., Pei, D., Wang, J., Xu, J., 2010. What happened in my network: mining network events from router syslogs. In: Proceedings of the 10th Conference on Internet Measurement. ACM, pp. 472–484. doi:10.1145/1879141.1879202.

Reidemeister, T., Jiang, M., Ward, P.A., 2011. Mining unstructured log files for recurrent fault diagnosis. In: Proceedings of the International Symposium on Integrated Network Management (IM). IEEE, pp. 377–384. doi:10.1109/INM.2011.5990536.

Ren, R., Cheng, J., Yin, Y., Zhan, J., Wang, L., Li, J., Luo, C., 2018. Deep convolutional neural networks for log event classification on distributed cluster systems. In: Proceedings of the International Conference on Big Data. IEEE, pp. 1639–1646. doi:10.1109/BigData.2018.8622611.

Rousseeuw, P.J., 1987. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. J. Comput. Appl. Math. 20, 53–65. doi:10.1016/0377-0427(87)90125-7.

Salfner, F., Tschirpke, S., 2008. Error log processing for accurate failure prediction. In: Proceedings of the 1st USENIX Workshop on the Analysis of System Logs (WASL).

Saneifar, H., Bonniol, S., Laurent, A., Poncelet, P., Roche, M., 2009. Terminology extraction from log files. In: Proceedings of the International Conference on Database and Expert Systems Applications. Springer, pp. 769–776. doi:10.1007/978-3-642-03573-9_65.

Schipper, D., Aniche, M., van Deursen, A., 2019. Tracing back log data to its log statement: from research to practice. In: Proceedings of the 16th International Conference on Mining Software Repositories. IEEE Press, pp. 545–549. doi:10.1109/MSR.2019.00081.

Shima, K., 2016. Length matters: clustering system log messages using length of words. Comput. Res. Reposit. (CoRR) abs/1611.03213.

Stearley, J., 2004. Towards informatic analysis of syslogs. In: Proceedings of the International Conference on Cluster Computing. IEEE, pp. 309–318. doi:10.1109/CLUSTR.2004.1392628.

Taerat, N., Brandt, J., Gentile, A., Wong, M., Leangsuksun, C., 2011. Baler: deterministic, lossless log message clustering tool. Comput. Sci.-Res. Dev. 26 (3–4), 11. doi:10.1007/s00450-011-0155-3.

Tang, L., Li, T., 2010. Logtree: A framework for generating system events from raw textual logs. In: Proceedings of the 10th International Conference on Data Mining (ICDM). IEEE, pp. 491–500. doi:10.1109/ICDM.2010.76.

Tang, L., Li, T., Perng, C.-S., 2011. Logsig: generating system events from raw textual logs. In: Proceedings of the 20th International Conference on Information and Knowledge Management. ACM, pp. 785–794. doi:10.1145/2063576.2063690.

Thaler, S., Menkonvski, V., Petkovic, M., 2017. Towards a neural language model for signature extraction from forensic logs. In: Proceedings of the 5th International Symposium on Digital Forensic and Security (ISDFS). IEEE, pp. 1–6. doi:10.1109/ISDFS.2017.7916497.

Tovarňák, D., Pitner, T., 2019. Normalization of unstructured log data into streams of structured event objects. In: Proceedings of the Symposium on Integrated Network and Service Management (IM). IEEE, pp. 671–676.

Vaarandi, R., 2003. A data clustering algorithm for mining patterns from event logs. In: Proceedings of the 3rd Workshop on IP Operations & Management (IPOM 2003). IEEE, pp. 119–126. doi:10.1109/IPOM.2003.1251233.

Vaarandi, R., 2004. A breadth-first algorithm for mining frequent patterns from event logs. In: Intelligence in Communication Systems. Springer, pp. 293–308. doi:10.1007/978-3-540-30179-0_27.

Vaarandi, R., Pihelgas, M., 2015. Logcluster - a data clustering and pattern mining algorithm for event logs. In: Proceedings of the 11th International Conference on Network and Service Management (CNSM). IEEE, pp. 1–7. doi:10.1109/CNSM.2015.7367331.

Vakali, A., Pokorný, J., Dalamagas, T., 2004. An overview of web data clustering practices. In: Proceedings of the International Conference on Extending Database Technology. Springer, pp. 597–606. doi:10.1007/978-3-540-30192-9_59.

Van der Aalst, W., Weijters, T., Maruster, L., 2004. Workflow mining: discovering process models from event logs. Trans. Knowl. Data Eng. (9) 1128–1142. doi:10.1109/TKDE.2004.47.

Wang, P.-H., Liao, I.-E., Kao, K.-F., Huang, J.-Y., 2018. An intrusion detection method based on log sequence clustering of honeypot for modbus tcp protocol. In: Proceedings of the International Conference on Applied System Invention (ICASI). IEEE, pp. 255–258. doi:10.1109/ICASI.2018.8394581.

Wurzenberger, M., Landauer, M., Skopik, F., Kastner, W., 2019. Aecid-pg: A tree-based log parser generator to enable log analysis. In: Proceedings of the Symposium on Integrated Network and Service Management. IEEE, pp. 7–12.

Wurzenberger, M., Skopik, F., Fiedler, R., Kastner, W., 2017. Applying high-performance bioinformatics tools for outlier detection in log data. In: Proceedings of the 3rd International Conference on Cybernetics (CYBCONF). IEEE, pp. 1–10. doi:10.1109/CYBConf.2017.7985760.

Wurzenberger, M., Skopik, F., Landauer, M., Greitbauer, P., Fiedler, R., Kastner, W., 2017. Incremental clustering for semi-supervised anomaly detection applied on log data. In: Proceedings of the 12th International Conference on Availability, Reliability and Security. ACM, pp. 31:1–31:6. doi:10.1145/3098954.3098973.

Xu, W., Huang, L., Fox, A., Patterson, D., Jordan, M.I., 2009. Detecting large-scale system problems by mining console logs. In: Proceedings of the 22nd Symposium on Operating Systems Principles. ACM, pp. 117–132. doi:10.1145/1629575.1629587.

Zhang, M., Zhao, Y., He, Z., 2017. Genlog: Accurate log template discovery for stripped x86 binaries. In: Proceedings of the 41st Annual Computer Software and Applications Conference (COMPSAC), 1. IEEE, pp. 337–346. doi:10.1109/COMPSAC.2017.137.

Zhang, X., Xu, Y., Lin, Q., Qiao, B., Zhang, H., Dang, Y., Xie, C., Yang, X., Cheng, Q., Li, Z., et al., 2019. Robust log-based anomaly detection on unstable log data. In: Proceedings of the 27th Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ACM, pp. 807–817. doi:10.1145/3338906.3338931.

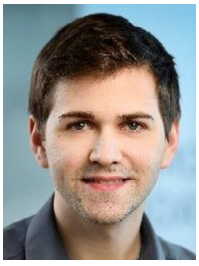
Zhao, Y., Xiao, H., 2016. Extracting log patterns from system logs in large. In: Proceedings of the International Parallel and Distributed Processing Symposium Workshops. IEEE, pp. 1645–1652. doi:10.1109/IPDPSW.2016.110.

Zhen, J., 2014. Sequence website. http://sequencer.io/.

Zhu, J., He, S., Liu, J., He, P., Xie, Q., Zheng, Z., Lyu, M.R., 2019. Tools and benchmarks for automated log parsing. In: Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice. IEEE Press, pp. 121–130. doi:10.1109/ICSE-SEIP.2019.00021.

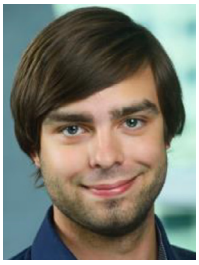
Zou, D.-Q., Qin, H., Jin, H., 2016. Uilog: Improving log-based fault diagnosis by log analysis. J. Comput. Sci. Technol. 31 (5), 1038–1052. doi:10.1007/s11390-016-1678-7.

Zulkernine, F., Martin, P., Powley, W., Soltani, S., Mankovskii, S., Addleman, M., 2013. Capri: A tool for mining complex line patterns in large log data. In: Proceedings of the 2nd International Workshop on Big Data, Streams and Heterogeneous Source Mining. ACM, pp. 47–54. doi:10.1145/2501221.2501228.



Dipl.-Ing. Max Landauer finished his Bachelor's Degree in Business Informatics at the Vienna University of Technology in 2016. In 2017, he joined the Austrian Institute of Technology in 2017 where he carried out his Master Thesis. He started his PhD studies in 2018 and is currently employed as a Junior Scientist at AIT. His main research interests are anomaly detection and log data analysis.

Dr. Florian Skopik, CISSP, CISM, CCNP-S joined the Austrian Institute of Technology in 2011 and is the Thematic Coordinator of AIT's cyber security research program. He coordinates national and international (EU) research projects, as well as the overall research direction of the team. The main topics of his projects are focusing on smart grid security, the security of critical infrastructures and national cyber security. He published more than 100 scientific conference papers and journal articles and holds some 20 industry-recognized security certifications, Florian is member of various conference program committees and editorial boards, as well as standardization groups, such as ETSI TC Cyber and OASIS CTI. Florian is IEEE Senior Member, Member of the Association for Computing Machinery (ACM) and Member of the International Society of Automation (ISA).



DI Markus Wurzenberger finished his Bachelor's Degree in Mathematics in Science and Technology in 2013. In 2014 he joined AIT as a freelancer and finished his Master's Degree in Technical Mathematics in 2015. In the end of 2015 he joined AIT as Junior Scientist and is working on national and international projects in the context of anomaly detection. In 2016 he started his PhD studies in Computer Science.

Andreas Rauber is Head of the Information and Software Engineering Group (IFS) at the Department of Information Systems Engineering (ISE) at the Vienna University of Technology (TU-Wien). He furthermore is president of AARIT, the Austrian Association for Research in IT and a Key Researcher at Secure Business Austria (SBA-Research. He has published numerous papers in refereed journals and international conferences and served as PC member and reviewer/Editorial Board Member for several major journals, conferences and workshops. He is a member of the Association for Computing Machinery (ACM), The Institute of Electrical and Electronics Engineers (IEEE), the Austrian Society for Artificial Intelligence (ÖGAI). His research interests cover the broad scope of digital libraries and information spaces, including specifically text and music information retrieval and organization, information visualization, as well as data analysis, neural computation and digital preservation.