Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

ARTICLE TYPE

amulog: A General Log Analysis Framework for Comparison and Combination of Diverse Template Generation Methods †

Satoru Kobayashi*[1] | Yuya Yamashiro[2] | Kazuki Otomo[2] | Kensuke Fukuda[1,3]

[1]National Institute of Informatics, Tokyo, Japan
[2]Graduate School of Information Science and Technology, The University of Tokyo, Tokyo, Japan
[3]Graduate University for Advanced Studies, Sokendai, Tokyo, Japan

Correspondence
Satoru Kobayashi, the National Institute of Informatics, Tokyo, Japan.
Email: sat@nii.ac.jp

Funding Information
This research was supported by the MIC/SCOPE #191603009.

Summary

One of the ways to analyze unstructured log messages from large-scale IT systems is to classify log messages with log templates generated by template generation methods. However, there is currently no common knowledge pertained to the comparison and practical use of log template generation methods because they are implemented on the basis of diverse environments. To this end, we design and implement amulog, a general log analysis framework for comparing and combining diverse log template generation methods. Amulog consists of three key functions: (1) parsing log messages into headers and segmented messages, (2) classifying the log messages using a scalable template-matching method, and (3) storing the structured data in a database. This framework helps us easily utilize time-series data corresponding to the log templates for further analysis. We evaluate amulog with a log dataset collected from a nation-wide academic network and demonstrate that it classifies the log data in a reasonable amount of time even with over 100,000 log template candidates. The template-matching method in amulog also reduces 75% processing time for template generation and keeps the accuracy when combined with an existing structure-based template generation method. In order to show the effectiveness of amulog in comparing log template generation methods, we demonstrate that the appropriate template generation methods and accuracy metrics largely depend on the purpose of further analysis by comparing the accuracy of six existing log template generation methods with ten different accuracy metrics on amulog.

KEYWORDS:
Network management, Log analysis, Log parsing, Log clustering, Framework

1 | INTRODUCTION

System and network operators need to maintain high availability of IT infrastructures by means of efficient troubleshooting. For troubleshooting, we rely on many kinds of operational data. Log data is one of the most effective data sources for this because, unlike other measurable data, it provides contextual information of system behaviors as literal explanations. However, log data from large-scale information systems is too large to investigate manually. For example, SINET5 [2], a research and education network in Japan, reports about 150,000 log lines in a single day. We need automated analysis methods and tools for analyzing such large-scale log data.

Automated analysis of log data is also difficult because a log message includes an unstructured statement (i.e., MSG in syslog [3]). Figure 1 shows an example of unstructured log messages. The message consists of a structured header part and an unstructured statement part. The structured header describes general information of the appeared event, such as time, place,

†This paper is an extended version of work published in Ref. [1]

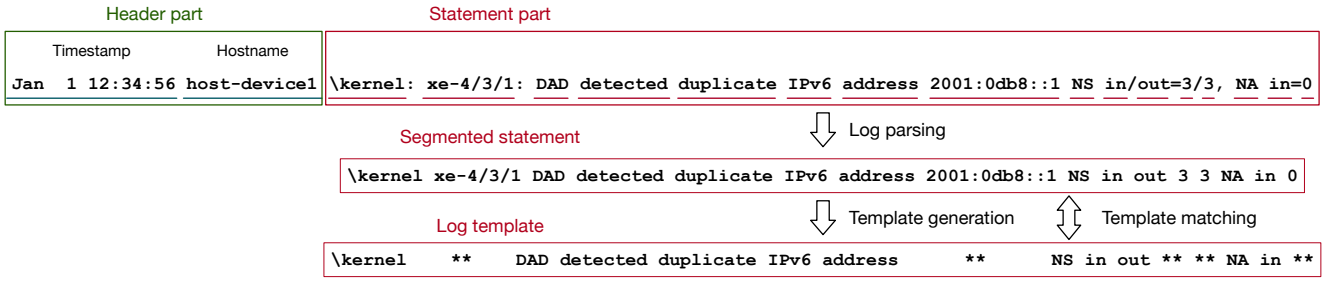

FIGURE 1 Example of log messages as the input of amulog.

and sometimes the classification of the event. The unstructured statement is the main part to describe what happens. The statement is designed for human readability but not reasonable for automated event identification. A major approach to analyze unstructured log data is generating log templates. A log template is a format of unstructured statements in log messages. Log messages belonging to a log template usually contain information on a common system behavior, so template generation is an effective way to classify log messages with their behaviors. In contrast to full-text search[4,5,6], which is another major approach used for keyword-based analysis, the template-based approach is suitable for aggregating and digesting the system behaviors, enabling time-series-based quantitative analysis such as anomaly detection[7,8,9,10,11] and root cause analysis[12,13,14,15].

Many log template generation methods have been proposed in past literature[16,17]. These methods are based on diverse approaches such as source code analysis[18,19,20], clustering[21,22,23,24,25,26,27], prefix-tree approaches[28,8,29], and machine learning[30,31]. As log data plays an important role in multiple IT fields, the template generation methods also have a diversity of assumptions, such as online and offline processing. The difference of assumptions sometimes prevents operators and developers from determining whether the methods match their use in a consistent manner. Therefore, we need a general framework for automated log analyses that does not depend on a specific log template generation method.

In addition, the log template generation methods are evaluated in disparate ways in the past works (see section 5). For example, past works have used disparate accuracy metrics such as the Rand Index[7,29,32], pairwise F-measure[22,33], and parsing accuracy[27,34]. Due to the difference, it is difficult for operators to compare and select log template generation methods with the literature. In our experience, the appropriate accuracy metrics depend on the further use of generated log templates. We need to re-organize the common knowledge of how to compare and evaluate log template generation methods fairly and consistently.

In this paper, we propose amulog, a general framework for template-based log analysis. As a unique feature, amulog consistently uses segmented log statements. Intuitively, log messages are considered as a string and use regular expressions for template matching. However, template matching based on regular expressions is not efficient in terms of the processing time (details in section 4). By applying a constant message segmentation, we can handle the data flow more simply. In addition, the messages segmented with the common rules can be used by multiple template generation methods in a same manner. Therefore, we can easily compare and combine multiple template generation methods on amulog.

Figure 2 shows the schematic system architecture of amulog. The key functions provided by amulog are threefold: (A) parsing log messages into headers and segmented statements with a rule-based parser log2seq[35], (B) classifying log messages with log templates by a tree-based scalable algorithm, and (C) storing the parsed data in a database that enables search and aggregation for further analysis. Two processing modes are available on amulog: online processing for real-time analysis and offline processing for hindsight analysis. We implement amulog as open-source software[36].

First, we discuss the existing log analysis frameworks and their problems. Next, we discuss the design and implementation of amulog to show its effectiveness in practical log analysis (section 3). Then, we evaluate amulog in terms of log template matching performance with a real log data from SINET4[37], a nation-wide academic network (section 4). We confirm that the proposed tree-based algorithm in amulog matches and classifies log messages with given log templates in a reasonable amount of time even if the number of given templates is more than 10^5. In addition, we conduct a comparison of six log template generation methods to determine the applicability of amulog to various algorithms (section 5). This comprehensive comparison work is a main new contribution from the original paper[1]. Through the comparison, we provide a number of findings on how to compare log template generation methods; In summary, we need to select appropriate measurement procedure and accuracy metrics considering the further usage of generated templates. In the comparison, we demonstrate that combining our template matching method with existing log template generation methods largely

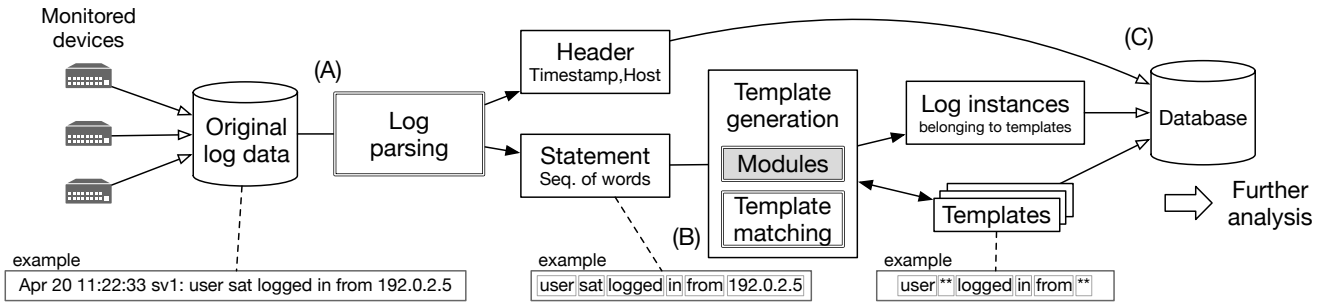

FIGURE 2 Overview of proposed log analysis framework.

decreases the processing time and in some cases improves the accuracy of template generation. In particular on the extended version of existing CRF-based template generation method[30], the template matching reduces 75% processing time and keeps the accuracy. Finally, we discuss the practical availability of amulog (section 6) and summarize the paper (section 7).

The contributions of this paper are as follows. (1) We design and implement a general log analysis framework for comparing and combining multiple log template generation methods. (2) We propose a scalable template-matching algorithm, which can improve the performance of other log template generation methods when combined. (3) We provide a guideline for comparing and selecting the template generation method through our comprehensive comparison work.

2 | RELATED WORKS

Some earlier works have proposed log analysis frameworks and platforms for monitoring computer systems. Logstash[38] and Napalm-logs[39] are popular tools widely used in commercial networks and services. These tools can parse unstructured logs into structured data, but they need pre-defined log formats for all log events to be parsed. It is a hard task for system operators to define all the log formats manually.

The other approach, without pre-defined log formats, is the full-text search based on Elasticsearch[40]. Bai[4] proposed a real-time log search engine with Elasticsearch and HBase. Cuong et al[5] constructed a log management system with LogStash[38] and Elasticsearch, where LogStash functions as a formatting and forwarding engine of the log messages using rule-based filters[41]. Futhermore, Hayabusa[6] is a full-text search engine for log data based on a distributed and parallelized SQLite database. These approaches focus on the full-text search of massive log data and are not appropriate for template-based log analysis (i.e., event-oriented time-series analysis) because it is difficult to extract log instances for a template with a keyword search.

Other works have discussed the selection of log template generation methods for log analysis. El-Masri et al.[16] reviewed 17 state-of-the-art log template generation algorithms selected from 89 research papers. Landauer et al.[17] surveyed more than 50 articles about log template generation methods. These works are simple qualitative comparisons based on paper surveys but are nevertheless helpful to select template generation methods. There are multiple frameworks with automated log template generation such as FLAP[42] and AECID-PG[43], but they are basically implemented for one particular log template generation method.

Similar to our work, Zhu et al.[34,33] implemented logparser[44], a framework to compare the accuracy of log template generation methods. They conducted a quantitative comparison of 13 log template generation methods with open log datasets of supercomputers and applications. However, logparser does not provide functions for further log analysis (such as template matching) and does not support combination of template generation methods (i.e., logparser does not satisfy the amulog's requirements (A)-(C), explained in subsection 3.1).

3 | AMULOG

3.1 | Requirements

There are three requirements for a general log analysis framework.

(A) The framework needs to preprocess the messages uniformly before applying log template generation methods. Most of the log template generation methods use segmented log messages (i.e., a sequence of words), but many practical log messages cannot be segmented by simple methods (details in subsection 3.3). In addition, the difference of segmentation rules among the template generation methods prevents us from combining and comparing them in a consistent manner. A suitable log segmentation rule depends on the dataset rather than the template generation method, so we need log segmentation functions not in the method but in the framework.

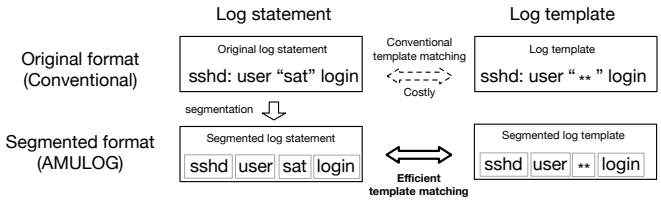

FIGURE 3 Difference of data format between amulog and conventional systems.

(B) The framework needs to match log messages with existing log templates. A log analysis framework should consider various use cases, such as accurate log templates being partially available without template generation or manual template modification required for precise analysis. Log template matching enables the framework to remap log templates and their instances to satisfy these practical requirements.

(C) The framework needs to provide the parsed and classified data in an appropriate style for further analysis, as system operators intend to search log data with the appropriate schema for their troubleshooting in mind.

3.2 | Overview of amulog

To satisify these three requirements, we propose amulog, a general framework for template-based log analysis, which is mainly designed for comparison and combination of diverse log template generation methods. The key idea of amulog is to use segmented log statements and segmented log templates consistently. Figure 3 shows an example of the segmented log data. Many log template generation methods use some word segmentation techniques in their template generation processes, but the implementations of these methods finally restore the segmented templates to the original format[21,34]. In contrast, we keep the log statements segmented and store them in a database. By applying a constant message segmentation, we can handle the data flow more simply and compare (or combine) template generation methods in a consistent manner.

Figure 2 illustrates the system architecture of amulog. It has three main components corresponding to the three requirements: (A) log parsing, (B) template matching, and (C) database storing. Amulog assumes line-based log data collected from multiple devices using logging platforms such as syslog as the input. The log data must contain at the very least a time stamp, source hostname, and free-format statement (for more details, see section 6). Users can search and aggregate the log messages with time stamp, hostname, and template identifier from the database storage by means of amulog functions afterward.

Here, we explain the data flow of amulog in the case of online processing (e.g., amulog incrementally receives a log message as an input). Amulog first parses an input log message into header items (i.e., time stamp and hostname) and a sequence of words corresponding to the free-format log statement with in the log parsing component (log2seq). Next, amulog determines a corresponding log template for the segmented statement by means of log template generation modules (i.e., estimates log templates dynamically) or log template matching (i.e., uses pre-generated log templates). Finally, amulog stores the data structured with the above two steps in a database. The stored data includes time stamps, source hostnames, log template identifier, and all words in the statement.

In amulog, the existing log template generation methods are considered as the modules for log template generation. A template generation module receives a log message as an input, and output the corresponding log template (and its template identifier). Log template matching with known templates can also be treated as one of the log template generation modules because they have common input and output. Therefore, they can be easily compared and combined in a consistent manner. Log template generation methods are combined in pipeline style; the first module tries to generate a log template for the input, and if it fails the second module tries (e.g., basically using template matching, and estimating log templates for unfamiliar log messages).

3.3 | Log parsing

Log messages have two parts of information in each line: a structured header part (e.g. time stamp and source hostname) and an unstructured statement part (e.g., event description) The format of the header part depends on the configuration of the logging system (the same as syslog[3]). The statement part is free-format, but basically has a potential template because system programs output the log statement by filling the replacers (e.g., format specifiers) in the template with variables. For human readability, the statement part is usually semi-formalized (i.e., partially using natural language), so in many cases it can be segmented into a sequence of words, just like other languages can be. Many log template generation algorithms use this feature of the statement part.

One simple approach is defining separator symbols[43] for datasets. However, there is a problem with the word segmentation process of the log template generation; some log messages are inconsistent with the usage of symbol strings. The inconsistent symbol strings can be uses as both the separators of words and part of the variables. Figure 1 shows an example of such log messages in our real dataset in which some of the sensitive variables (such as IP addresses) have been replaced. Focusing on the slash symbols ("/"), some of them are used as separators of words, while others are part of the variable, which is an interface name in this case. If we split the statement with a word

segmentation rule that uses slashes as separators, the interface name "xe-4/3/1" will be torn into smaller pieces that lose the information of interface. In contrast, if we leave the slash in the word segmentation, the words "in" and "out" are not available in the log template generation and further analysis. In this example, colon symbols (":") also cause a similar problem. No single static rule can parse this log statement correctly.

In response to this issue, we propose log2seq, a rule-based log message parser that converts string log messages into headers and segmented log statements. The configuration for log2seq consists of two rules: the header parsing rule and the statement segmentation rule. The header parsing rule is a simple regular expression to parse the header information, which helps amulog accept diverse header format of log messages as input. The statement segmentation rule is a collection of multiple actions for word segmentation. The key idea of log2seq in statement segmentation is to fix known variable words including inconsistent symbol strings (e.g., "xe-4/3/1" in Figure 1) while parsing. The actions can be classified into two types: SPLIT and FIX. A SPLIT action is a rule to separate words segmented by some separators. The rule is specified with a regular expression of separator symbol strings. A FIX action is a rule to specify words that should not be segmented further. This rule is specified with some regular expressions of known variable words. The user customizes the configuration of these rules by modifying the orders and corresponding regular expressions in these actions.

For example, a sample configuration of log2seq consists of four steps. (1: header parsing rule) First, we parse the header part using a static rule of regular expressions. (2: SPLIT action) Next, we split the statement part (the remaining part after removing the header part) into word sequences using standard separator symbols (e.g., spaces and brackets). (3: FIX action) Then, we fix the known variable words that should not be separated later (e.g., IPv6 address). (4: SPLIT action) Finally, we split the words by inconsistent symbols (e.g., slashes and colons). These rules can parse log messages including inconsistent symbols such as Figure 1 as expected.

In our experience, the configuration of log2seq largely depends on the vendor of the devices and applications. Still, the configuration can be easily shared in engineer communities because it does not include any internal information of the systems and devices.

The source code of log2seq is publicly available on GitHub[35], and can be easily installed as a PyPI package.

3.4 | Template matching

Log template generation must deal with two issues: determining the log template structure (structure issue) and classifying the log messages with the templates (classification issue). Solving the structure issue enables us to obtain an abstracted log representation and to determine variable words in templates. These are important when we compare variables in the messages and apply NLP-based log analysis methods that focus on descriptive messages[10,11]. On the other hand, solving the classification issue is required for event-oriented time-series analysis. Many existing works based on network log analysis (including anomaly detection[7,8,9,10,11] and root cause analysis[12,13,15]) classify log messages with log templates to generate the time series of log events.

However, not all template generation methods satisfy both issues. For example, a set of generated templates from source code analysis[18,19,20] is not mapped to the real log messages (i.e., does not solve the classification issue). These methods need template matching to use classified log data in further analysis. Another example appears when we use external log template lists (e.g., third-patry documents) for log analysis. Such template lists are usually not mapped to our log messages. Furthermore, in practical terms, we would need to modify log templates generated automatically by some methods if they have any errors. Editing log templates usually breaks the consistency of models for automated template generation, so log template matching is required to re-map the modified templates to the log messages.

To design a log template-matching method, it is important to consider the scalability of the log template matching to accept diverse template generation methods. Log data from large-scale systems have a large number of potential log template candidates (much more than the number of appearing log templates in actual logs), which prevents fast template matching. Also, the number of template candidates can be larger if the templates are generated automatically and include any errors.

A straightforward approach for log template matching is to use regular expressions. However, template matching based on regular expressions is not efficient because a log template usually includes multiple variables. For example, some log templates found in our dataset (see subsection 4.1) have more than ten variable words in one log statement. The degree of freedom of the regular expressions is too large for efficient calculation. Besides, the computation complexity is $O(n)$ in the straightforward approaches, where n is the number of the log templates for template matching.

To resolve this issue, we propose a log template-matching algorithm based on a prefix-tree. In this algorithm, we first make a search tree of the given log templates, as shown in Figure 4. Every node has a word or a wildcard, and some nodes have a log template identifier that indicate the end of the templates. This tree consists of three log templates: "user ** logged in from **", "user name invalid", and "user name ** removed" ("**" represents a variable word). The searching algorithm is a kind of depth-first one: the search process

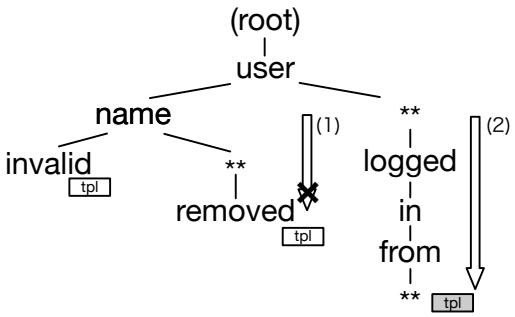

FIGURE 4 Example of a template search tree.

starts from the root node and the next node is determined by matching the words with the nodes from the top of the input statement. There can be several potential branches of the searching paths because some nodes are the wildcard for variable words. In that case, we first select a node with a static word and then search another node with a wildcard. If there is no next matching node, the searching path is discarded and searching for the next path begins. If the node after passing all words in the input statement has a log template identifier, the searching process finishes.

For example, assume the log statement "user name logged in from 192.0.2.5" is given for matching with the tree in Figure 4 (it is unlikely, but this is just an example). First, amulog traces branch (1) and fails with the fourth word "removed". Next, amulog traces branch (2) and succeeds to obtain a corresponding log template identifier. Finally, amulog obtains a log template identifier registered at the end of branch (2) as an identifier corresponding with the input.

The computational complexity of the tree-based template-matching method is $O(2^m)$ in the worst case where m is the number of words in the input statement. This is because of the forks formed by the wildcard nodes in the search tree. However, it is a rare case that amulog needs to search for multiple paths for a log message with practical log templates. The paths to be searched increase only when a word matches both nodes of the equal static word and that of a wildcard, but template generation methods usually unify them because most of the words in log messages are used for either a static word or a variable. In addition, m is typically small enough in practical log templates (13 words on average in our ground truth templates).

3.5 | Data storage

We also need to consider the schema of the database storage. Currently amulog supports SQLite or MySQL as data storage. Amulog records the time stamp, source hostname, and template identifier of each log message for further analysis. The time stamp and the hostname are parsed with log2seq. The template identifier is determined by means of log template matching or log template generation.

Also, amulog records all words in the log statement for detailed analysis. Intuitively, it would seem that we only need to record variable words instead of all words in the log statement. However, some template generation methods (especially incremental methods based on clustering approaches, such as Drain[27]) change the definition of the log templates afterward. If we only record the variable words in the log statement, the definition change of the log templates will destroy the data structure. In addition, by recording all words, we can manually edit the log template definitions afterward (see section 6).

3.6 | Implementation

Amulog is implemented in Python 3. The source code of amulog is available on GitHub[36].

Amulog provides two processing modes: online and offline. Online processing is designed for continuous data collection and real-time log analysis. The online processing works with small memory costs and supports interruption and resumption of template generation. Offline processing is designed for hindsight analysis. The offline processing works with all template generation methods, and some of them support parallel processing (see section 5). Amulog currently supports multiple log template generation methods including online methods (e.g., Drain[27], LenMa[45], FT-Tree[8], and CRF[30]) and offline methods (e.g., Dlog[29]). Amulog also supports parallel processing (including log parsing and template matching) for offline processing.

4 | EVALUATION

4.1 | Dataset

We use a set of backbone network syslog data obtained from SINET4[37] to evaluate amulog performance (in this section) and conduct further analysis (section 5). SINET4 is a nationwide R&E network connecting over 800 organizations in Japan. The network consists of eight core routers and over 100 Layer-2 switches provided by multiple vendors.

We manually generate ground truth log templates for the dataset. There are 1,788 different log templates in 15 months of log data (35 million lines in total). Figure 5 shows the number of log instances in the dataset. This empirical distribution is long-tailed in log-scale, which means most log messages belong to only a small number of log templates.

In the following experiments, we use an Ubuntu 18.04 server (x86_64) equipped with an Intel(R) Xeon(R) Silver 4110 (2.10 GHz) and 64 GB of memory.

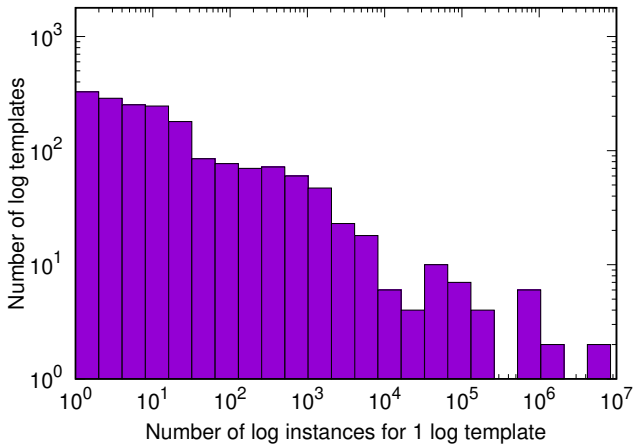

FIGURE 5 Log-log-scale histogram of log instances in our dataset.

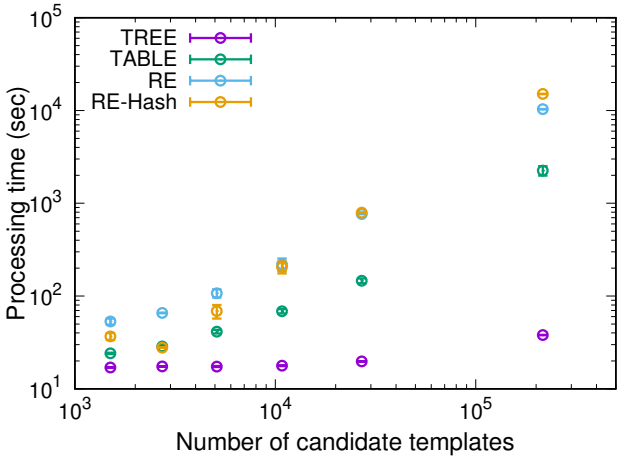

FIGURE 6 Processing time of template-matching methods.

4.2 | Log template matching

We demonstrate the scalability of the proposed template matching method to the number of log template candidates by the comparison of the processing times of various log template-matching approaches.

Four template-matching approaches are used for the comparison: TREE, TABLE, RE, and RE-Hash. TREE is our proposed tree-based method described in subsection 3.4. TABLE is a simple method for segmented log messages. It searches the corresponding template with the same length (i.e., the same number of words) of log statement and matches all words except wildcards. TREE and TABLE depend on the message segmentation (i.e., use log2seq). RE uses regular expressions for template matching. We automatically generate a regular expression for a given log template. RE-Hash is also based on regular expressions, but it classifies the regular expressions by the hash of initial characters in the log statement (we use five characters; e.g., "/kern" in Figure 1) to decrease the number of regular expressions to match a message. RE and RE-Hash do not translate the input log statements into a sequence of words. The computational complexity of TABLE and RE is $O(n)$, where n is the number of given log templates.

For the comparison, we measure the average processing time over ten trials with shuffling the order of the log template candidates to generate the matching models. This is because the processing time of some template-matching methods (especially TABLE and RE) depends on the order of the given log templates. As explained in subsection 4.1, most log messages belong to only a small number of templates. Therefore, if the frequently appearing templates were on the top part of the given templates, linear search methods (TABLE and RE) would succeed in matching the messages in a very short amount of time. That is why we need to shuffle the templates for a fair comparison.

We also generate six template sets for the comparison using the implemented log template generation methods [1]. We evaluate the processing time to generate an amulog database from 1-day log data (76,719 lines, a part of the testing data in section 5) with the four different template-matching methods. The processing time includes all of the standard amulog processes: initializing the matching model from the templates, parsing input messages with log2seq, classifying the messages by the template matching, and generating a database.

Figure 6 shows the log-log-scale comparison of the processing time with the four template matching methods. As explained above, the measured processing time is the average of ten trials with shuffled templates and the error bars show their standard error. Our proposed method, TREE, took about 40 seconds even if more than 10^5 templates were given as template candidates [2]. We can see that the difference in the processing time was small with a smaller number of template candidates, but this difference became larger for more template candidates. The processing time of template matching with TREE was nearly constant, and the increase for larger template candidates stemmed from the model initialization step (because the number of given templates was larger than the number of log messages). The processing time of TABLE depended largely on the number of templates, but even so it was faster than RE and RE-Hash. This is because the message segmentation enabled us the partial matching of messages. On the TABLE algorithm, we do not need to match all words in a template if the top word is neither a matching word nor a wildcard. In contrast, RE and RE-Hash need to match all of

[1] The six methods are Manual, Drain, Match+CRFe, CRFe, RegEx, and CRF. These methods are introduced in section 5. The number of templates is 1,504, 2,726, 5,108, 10,793, 26,986, and 215,934, respectively.

TABLE 1 Log template generation algorithms for comparison

Method	Online	Offline	Parallel	Preparation
Drain[27] [3]	✓	✓		–
LenMa[45]	✓	✓		–
FT-tree[29]	✓	✓		scanning [4]
Dlog[8]		✓		–
RegEx	✓	✓	✓	–
CRF[30]	✓	✓	✓	training

the message with the regular expressions, as wildcards in regular expressions allow any length of variable. Besides, RE-Hash was not effective compared to RE in this result. In our dataset, most templates in the larger template sets start with variable wildcards, so the hash of initial characters cannot decrease the candidate templates for matching. In summary, the amulog design with segmenting messages is effective for efficient template matching, and the proposed tree-based method improves the scalability of template matching.

The scalable log template matching is also helpful to fast template generation by the combination of template matching and template generation methods (see subsection 5.4).

5 | APPLICATION

To determine the applicability of amulog to various template generation methods, we demonstrate a comparison of existing log template generation methods with the dataset explained in 4.1.

5.1 | Template generation methods for comparison

We compare six different existing methods, consisting of cluster-based and structure-based methods, as shown in Table 1.

The first four methods are cluster-based, classifying log messages into clusters on the basis of message similarity and heuristics. Drain[27] is an online template generation method based on heuristics and clustering with common words. LenMa[45] is an online method using the character length of words for message clustering. FT-tree[29] is an online method with a tree-based method that prunes minor words as variables. Dlog[8] is an offline method with a tree-based clustering and aggregation of common parts. LenMa and FT-tree use the heuristic knowledge that a top word is always a part of the templates. We disable this heuristic rule in the algorithms because the messages in our dataset often start with variables. As Drain also uses this knowledge, we leave it available because it avoids failures with top variables by preprocessing for commonly used variables (digit values).

The latter two methods are structure-based. In these methods, template generation is considered as a problem to classify words in log messages into descriptions (i.e., words to form templates) and variables (i.e., words replaced into wildcards in the templates). RegEx is a simple method based on regular expressions. We manually created regular expressions for commonly used variables in log messages (digits, timestamps, IP addresses, MAC addresses, and interface/device names). We consider words matched with the regular expressions as variables and other words as descriptions. CRF[47,30] is an NLP-like approach that estimates template structures (i.e., descriptions and variables in a log message) for log messages. CRF requires manually labeled training data for supervised learning, but for tagging, it does not depend on other messages, which means parallel processing can be easily applied. We also extended the CRF-based template generation (CRFe) in two ways: (1) using mid-labels generated by regular expressions (same as RegEx) and (2) sampling training data for manual labeling with a preliminary clustering by a simplified SLCT[48] to learn more about minor templates.

In addition, we used combined methods with the tree-based template-matching method of amulog. As the method combination in amulog is implemented as a pipeline processing for each line, the combined template generation is only available for online methods (i.e., Dlog cannot be combined with template matching). In the combined methods, the log template matching enables to process log messages fast by matching them to known log templates. A combined method conducts log template generation as follows: Initially, there is an empty log template search tree. For each input log message, the method first searches existing templates with the tree. If not found, the method generates a log template with the combined existing method. Then the generated log template is added to the search tree, and used for following input log messages.

5.2 | Accuracy metrics and comparison settings

In this work, the accuracy of log template generation corresponds to the comparison of the log template matching results with the ground truth templates and the log template generation results with compared method. There also exist multiple

[2] The amulog implementation is highly improved from the past paper[1], and now it only takes less than 50% processing time compared with the previous version.

[3] POP[46], an extended method of Drain, is offline-only but can be processed in parallel.

[4] In online processing, FT-tree requires a static scan of all or at least a part of the log data to survey word appearance distribution.

TABLE 2 Accuracy metrics for log template generation.

Metrics	Category	Decision	Count
Word accuracy (WA) [30]	structure	word label	words
Line accuracy (LA) [30]	structure	line labels	lines
Template accuracy (TA) [30]	structure	line labels	templates
Template word accuracy (TWA)	structure	word label	templates
Rand Index (RI) [49]	cluster	line pairs	line pairs
Adjusted Rand Index (ARI) [50]	cluster	line pairs	line pairs
Pairwise F-measure (Fm) [51]	cluster	line pairs	line pairs
V-measure score (Vm) [52]	cluster	line entropy	lines
Parsing accuracy (PA) [34]	cluster	exact match	lines
Cluster accuracy (CA)	cluster	exact match	templates

accuracy metrics for log template generation. Past works have used disparate metrics such as the Rand Index [7,29], pairwise F-measure [22,53,33,27,46,43], and parsing accuracy [34]. In this experiment, we utilize the ten metrics shown in Table 2 (TWA and CA are newly defined in this paper).

These accuracy metrics can be classified into two categories: structure metrics and cluster metrics. Structure metrics measure whether the structure of generated log templates is correct or not. We use four different structure metrics, three of which are proposed in our previous paper [30]. There are two aspects to explain the difference between the four structure metrics. One is the difference in the decision unit to evaluate. WA and TWA evaluate whether each word is labeled (i.e., classified as description or variable) correctly or not, and LA and TA evaluate whether the words in each log line are all labeled correctly or not. The other is the difference in the counting unit to evaluate. WA and LA simply count the number of words or lines, depending largely on the distribution of log instances (see also subsection 4.1). In constrast, TA and TWA are the average of the scores for each log templates, which means the metrics are weighted to be fair to each log templates.

On the other hand, cluster metrics measure whether the message clusters classified by the templates are correct or not. RI, ARI, and Fm are the pair-wise metrics that are traditionally used to evaluate clustering results. PA is designed for log template generation, which strictly matches the generated clusters with the ground truth. CA is an extended metrics of PA, which is weighted for clusters instead of log instances, the same as TA and TWA. Vm, which is also similar to PA, is a combined value of two entropy-based support metrics, homogeneity (Ho) and completeness (Co). Ho measures whether the log instances in one ground truth cluster are correctly classified into one estimated cluster. Opposite to the Ho, Co measures whether the log instances in one estimated cluster are corresponding to only one ground truth cluster. These two metrics respectively correspond to two kinds of failure cases in strict cluster matching (similar to PA): over-division and over-aggregation. From the viewpoint of template structures, cluster over-division is caused by false estimation of variables as descriptions, and cluster over-aggregation is caused by false estimation of descriptions as variables.

We compare the six log template generation algorithms with these ten accuracy metrics on amulog. We use the first three months of log data as training and the following twelve months for evaluation. For the online clustering methods, the training data is preliminarily loaded into the model before evaluation (This is mandatory for the FT-tree preprocessing. For other methods, this stabilizes the template structure in the online evaluation). For the CRF-based methods, we annotated 1,000 items sampled from the training data and used them to generate the CRF model. In both cases, we only use the results of testing data, not the training data, for calculating the metrics. Note that the template-matching method can help the annotation step: we manually prepare a list of answer log templates, and amulog annotates the items by the template-matching method with the list.

5.3 | Results and findings of metrics

We first need to discuss the usage of these ten accuracy metrics in comparing log template generation methods. In this section, we demonstrates some findings through the comparison results shown in Table 3.

Online and offline measurement should be distinguished in structure metrics: For the online clustering methods (Drain, LenMa, and FT-tree), we can conduct two kinds of accuracy measurement: offline and online. Offline evaluation is the intuitive measurement that calculates after all evaluation data processing has finished. In the online evaluation, we incrementally calculate the structure metrics with the generated template just after each line is processed. This will cause a difference of the structure metrics between offline and online because the template structures change as cluster components are added during data processing. Table 4 compares the accuracy of offline and online measurement in structure metrics. In the comparison, the structure metrics values were different between the online and offline evaluations: specifically, many values were better in the online evaluation. Figure 7 helps understand the reason. In this example, the second line of the estimated cluster members is an unexpected log instance (i.e., over-aggregation). In the online measurement, top three words were correctly labeled when processing the first line, but failed when processing the following four lines because the second line changes the template structure of this cluster. In contrast in the offline measurement, the accuracy of all lines are calculated with the final template structure. Therefore, over-aggregation caused the performance degradation in

TABLE 3 Performance comparison of log template generation algorithms.

Method	Stats		Structure metrics				Cluster metrics						Support metrics	
Algorithm	Time	#Clusters	WA	LA	TA	TWA	RI	ARI	Fm	Vm	PA	CA	Ho	Co
(Answer)	–	1,504	–	–	–	–	–	–	–	–	–	–	–	–
Drain	3,241	2,726	0.933	0.559	0.221	0.817	0.997	0.988	0.990	0.984	0.582	0.408	0.971	0.997
LenMa	15,552	2,285	0.868	0.235	0.134	0.727	0.977	0.918	0.932	0.949	0.086	0.368	0.971	0.928
FT-tree	2,940	13,380	0.765	0.233	0.067	0.633	0.996	0.988	0.990	0.949	0.201	0.266	0.995	0.907
Dlog	8,238	9,549	0.698	0.000	0.053	0.665	0.994	0.981	0.985	0.972	0.066	0.394	0.997	0.948
RegEx	5,318	26,986	0.867	0.040	0.260	0.870	0.921	0.675	0.717	0.724	0.349	0.525	0.999	0.567
CRF	11,992	215,934	0.985	0.949	0.167	0.808	0.999	0.996	0.998	0.915	0.930	0.521	1.000	0.843
CRFe	12,155	10,793	0.940	0.529	0.560	0.941	0.999	0.998	0.998	0.990	0.278	0.745	1.000	0.979
Match+Drain	3,043	2,648	0.933	0.559	0.221	0.803	0.997	0.988	0.990	0.984	0.581	0.406	0.971	0.997
Match+LenMa	3,059	832	0.867	0.235	0.131	0.716	0.997	0.988	0.990	0.984	0.085	0.369	0.969	0.999
Match+FT-tree	3,047	9,530	0.834	0.203	0.068	0.670	0.996	0.988	0.990	0.953	0.007	0.305	0.988	0.919
Match+RegEx	3,025	21,246	0.867	0.041	0.269	0.872	0.921	0.676	0.718	0.725	0.351	0.545	1.000	0.568
Match+CRF	3,012	18,330	0.993	0.966	0.344	0.898	1.000	1.000	1.000	0.996	0.968	0.596	1.000	0.991
Match+CRFe	3,022	5,108	0.941	0.530	0.576	0.943	1.000	0.998	0.999	0.991	0.811	0.803	1.000	0.982

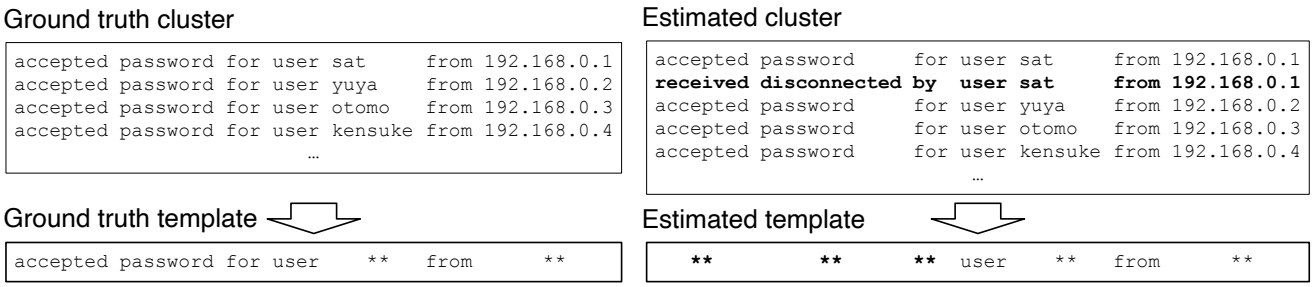

FIGURE 7 A simplified example of over-aggregation in clustering-based template generation methods. One additional unexpected log message significantly changes the structure of log templates. In online evaluation, messages only after the unexpected message are considered failed to determine the structure of top three words. In offline evaluation, all the messages in the cluster are considered failed.

TABLE 4 Comparison of online and offline measurements in structure metrics.

Algorithm	Offline	WA	LA	TA	TWA
Drain		0.933	0.559	0.221	0.817
	✓	0.903	0.572	0.121	0.813
LenMa		0.868	0.235	0.134	0.727
	✓	0.812	0.055	0.153	0.656
FT-tree		0.765	0.233	0.067	0.633
	✓	0.765	0.233	0.009	0.602
Match+Drain		0.933	0.559	0.221	0.803
	✓	0.902	0.572	0.125	0.817
Match+LenMa		0.867	0.235	0.131	0.716
	✓	0.811	0.055	0.237	0.803
Match+FT-tree		0.834	0.203	0.068	0.670
	✓	0.834	0.203	0.012	0.599

the structure metrics in the offline measurement. Hereafter, we focus on online measurement results (Table 3 is all online measurement).

Over-division and over-aggregation can be measured by Vm support metrics.: As explained in offline-online comparison, over-division and over-aggregation make different effect on the measurement results. With Ho and Co, the supporting metrics of Vm, we can measure how much over-division and over-aggregation exists in the comparison results (see subsection 5.2). For example in Table 3, Drain and LenMa had smaller Ho scores (i.e., more over-aggregation). This can also be confirmed in Table 4: the offline-online difference is larger in Drain and LenMA than in FT-tree especially on unweighted metrics (WA and LA).

It is important to understand the two failure cases, over-division and over-aggregation, in the template generation results. One reason is that the failures of some template generation methods can be biased for each of the cases. In the six methods, Drain and LenMa have both over-division and over-aggregation. In comparison, failures in other methods are biased to over-division. The other reason is that the presentation of generated log templates depends on failure cases. If a template has an over-division failure, the generated templates still have much variables, so the over-division should be reduced for anonymizing log data. On the other hands, if a template has an over-aggregation failure, the generated template is too much masked (see also Figure 7). The over-aggregation should be reduced for some visualization purposes.

Pairwise metrics are not effective in long-tailed dataset: In Table 3, pair-wise cluster metrics (RI, ARI, and Fm) were nearly stuck to the upper limit. As shown in Figure 5, the log template appearance in our dataset is long-tailed. Pair-wise values depend on the square of the number of log lines, where the values only demonstrate major clusters in the dataset.

Line-counting metrics are for trend analysis, and template-counting metrics are for anomaly analysis: To clarify the difference between the counting unit (i.e., lines or clusters), we focus on the difference between CRF and CRFe. The difference stems mainly from extension (2) of CRFe: sampling training data. CRF randomly selects training data, which means frequently appearing messages are intensively learned (according to the log instance distribution shown in Figure 5, 90% of the training data would belong to only ten templates if randomly sampled). In contrast, CRFe selects training data including minor log templates, which means the CRFe model focuses more on minor log templates than frequent ones. This difference is confirmed by the results in Table 3: CRF was superior, especially on LA and PA (i.e., counting lines), and CRFe was superior on TA, TWA, and CA (i.e., counting templates). To this end, we should select appropriate metrics for further analysis usage of log data. If we focus largely on the time-series trend of frequent messages (e.g., access-log trend analysis), we should focus on line-counting metrics such as LA, PA, and Vm. On the other hand, if we focus on anomalous events that rarely appear (e.g., troubleshooting of error logs), we should focus on template-counting metrics such as TA, TWA, and CA.

Structure metrics and cluster metrics should be used together: Especially with clustering-based template generation methods, large cluster metrics do not mean large structure metrics. This is because, in clustering-based method, a template structure is obtained as the common parts of log instances in a cluster. Even if the cluster is completely accurate, some variables can be consistent in a dataset and be considered as descriptions. The template structure is expected accurate for example in semantic analysis[54] or variable analysis. For these purposes, we should consider not only cluster metrics but also structure metrics.

Line-complete metrics and cluster-complete metrics are too peaky to understand template generation results: We can see that some metrics, especially LA and TA, were too peaky for comparison. These metrics consider the labeling of a log message as failed even with one failed label (i.e., line-complete) and are sometimes ineffective for comparison because they cannot distinguish partial failures from nearly complete failures. PA and CA also have a similar problem, as they consider log instances in a cluster as all failed even with one excess or deficiency message (i.e. cluster-complete). In terms of over-aggregation, this is reasonable because one unexpected message can break the template structure completely as shown in Figure 7. However, in terms of over-division, these metrics are too sensitive. We can see this with the results for Dlog and CRFe: almost all failures on these methods were due to over-division (i.e., large Ho and small Co), and they had a large Vm but comparatively small PA even though Vm and PA are similar metrics depending on the number of lines. Therefore, we should avoid these peaky metrics (LA, TA, PA, and CA) for precise comparison of template generation methods.

Recommendation: From these findings, if one intends to design log template generation methods for general further usage, the recommendation is using TWA and Vm for evaluation, and additionally using Vm's supporting metrics (Ho and Co) for validation. Vm shows the accuracy for standard usage such as trend analysis, and TWA shows the accuracy for anomaly analysis and template-structure-required analysis. The supporting metrics will help understand what kind of failures (i.e., over-division or over-aggregation) exist in the results.

5.4 | Comparison results of template generation methods

On the basis of above, we briefly compare the template generation methods. Drain is the state-of-the-art method in this field, and it outperformed the other clustering methods in terms of accuracy. Even so, it had more over-aggregation failures than the other methods, which may result in some minor log messages being buried into a cluster of major messages with different template structures (as in Figure 7). If we intend to avoid over-aggregation for further analysis, we should also consider other methods such as Dlog. Moreover, if we allow a certain amount of training data to be manually annotated, CRF or CRFe would be the best method. These CRF-based methods are especially effective in terms of structure metrics, which has a big impact on further log analysis considering variable values of log instances (e.g., protocol-specific analysis).

In addition, we can see combining template matching and existing methods is effective in many cases. For the methods that require large processing time (especially LenMa and CRFs), template matching reduces processing time to the same level as Drain and FT-tree. In particular, the processing time of CRFe is 75% reduced if combined with template matching. Also, template matching does not largely change the accuracy in any metrics. In particular for CRFs, the accuracy metrics (notable in PA and Vm) are improved with template matching. This is because CRF sometimes fails to label variables that appear in multiple different templates. The messages with these variables can be easily merged with appropriate template clusters by template matching. Therefore, template matching is effective especially when combined with structure-based methods.

6 | DISCUSSION

Requirements for combination of template generation methods: In this paper, we demonstrated the combination of template matching and existing template generation methods. They are combined in pipeline style; one method generates templates for a part of input accurately, and the other method generates templates for the remaining part of input. As explained in subsection 5.1, this approach is only available for online template generation. For effective combination, the combined passing methods (i.e., methods except the last one) need to be more accurate than the following methods and only determine a part of messages. LogParse[32] uses a similar approach to combine multiple template generation methods, but it has some limitations (see section 2).

Practical use of combined template generation: The combination of template-matching method is also useful for more practical situations. In actual operation, we sometimes have partial lists of accurate log templates. For example in network fields, the partial log templates can be obtained from vendor's documentation of network devices. These template lists are used in two different approaches. One approach is for combined template matching. By adding the template lists as initial knowledge of the template search tree, amulog can generate accurate log templates for more log inputs. The other approach is for CRF training data. With the template lists, we can generate annotated training data more easily. Note that generating training data from some lists may cause a training bias, so it is better to add manually generated training data that follows missing templates in the lists. These two approaches can be used together, which will largely improve the accuracy of log template generation.

Manual modification of log templates: Amulog supports manual modification of log template structures after automated log template generation. This modification not only changes template structure but also re-organize log message clusters. It will help more flexible operation, for example when we have known important log messages that must be classified accurately. If one needs to prepare ground truth log templates of a dataset for evaluating log template generation methods, the manual modification function is also helpful to make it.

If we simply modify log templates generated by online clustering methods, it may lose consistency of online clustering. However, if the method is combined with template matching, we can modify the templates on the search tree and keep online clustering consistent. Therefore, the template matching is also helpful for flexible management. Note that if one intends to modify log templates manually, he/she should select an automated log template generation method that has smaller over-aggregation. In our experience, we can easily fix over-divided templates by just replacing variables into wildcards. In contrast, it is difficult to fix over-aggregated templates because we need to refer cluster members to reorganize accurate templates.

Input data format: We used log examples shown in Figure 1, which is described in the default format of rsyslog[55]. Amulog is mainly designed for syslog data, but it also accepts line-based logs including at least a timestamp and an unstructured statement (Amulog also requires a hostname, but it can be a dummy in single systems). Log2seq accepts any format of the input logs satisfying this requirements with the customizable header parser if the header format is common in a dataset. The header part of recorded syslog data may include other parameters: facility, severity, and app-name are optional annotation information of the following unstructured message, and procid and msgid usually do not include information for system troubleshooting. In amulog, the optional annotation information can be parsed with log2seq and used as annotation tags of log templates. The annotation tags can be used for search and classification of log time-series in further analysis (e.g., for domain knowledge in causal analysis[14]).

Parameter tuning: Amulog is also helpful for parameter tuning of log template generation methods. To illustrate this, we selected a set of parameters for Drain by an automated parameter tuning on the annotated three-months dataset (corresponding to the three-months training data used in subsection 5.2). Drain has two parameters, $depth$ and st, and the appropriate parameters depend on the dataset[27]. We maximized Vm and obtain $depth = 3$ and $st = 0.5$. Note that the obtained parameters change if we use different accuracy metrics to be maximized, so it is important to use appropriate accuracy metrics for the purposes of further analysis to select a reasonable set of parameters.

Limitation: As amulog consistently uses segmented log statements, there is a limitation that amulog cannot distinguish log templates with common words but different separator symbols. In our experience, it is a rare case that two such templates have different meanings and their clusters should be divided. Instead, the minor changes of separator symbols are mainly caused by the updates of softwares or firmwares. For the long term analysis, we do not need to distinguish templates with different separator symbols. Similar problems appear if we use log templates extracted from software source codes on amulog. There are some literature that use log templates extracted from source codes[18,19,56]. These templates are not parsed as the sequence of words, and cannot be segmented directly because the variable part can include separator symbols that are not visible in the templates. To use these templates on amulog, we additionally need their log instances to apply segmentation accurately. Still, the scalable template-matching method of amulog is very helpful to use these extracted templates because

the number of extracted templates is extremely large compared to the ones that actually appear in managed systems. For example, Yamashiro et al.[56] generated 28,148 templates from source code[57] of routing services of Vyatta, an open source router.

7 | CONCLUSION

In this paper, we have proposed the design and implementation of a general framework, amulog, for template-based log analysis. Our design is characterized by a simplified data flow while using segmented log messages consistently. We implemented amulog considering the three issues facing framework design: rule-based log parsing, tree-based template matching, and database schema. Our evaluation demonstrated that the proposed template-matching algorithm is scalable enough to match 1-day log messages (76,000 lines) with more than 10^5 templates in 40 seconds. To demonstrate the applicability of amulog to the comparison and combination of log template generation methods, we conducted a comprehensive comparison of existing log template generation methods with real network log data on amulog. We clarified that there is no one best log template generation method; rather, we need to select which methods to use by considering the purposes of the further analysis. We also confirmed that the combination of template matching with existing methods is effective to decrease processing time of log template generation.

As future work, we will implement state-of-the-art template generation methods as external modules for amulog. We will also share the log2seq configurations for major devices and applications. In addition, we will consider further log analysis approaches based on amulog.

ACKNOWLEDGEMENTS

This work is supported by the MIC/SCOPE #191603009.

References

1. Kobayashi S, Yamashiro Y, Otomo K, Fukuda K. amulog: A General Log Analysis Framework for Diverse Template Generation Methods. In: *Proceedings of the 16th International Conference on Network and Service Management (CNSM)* IFIP/IEEE. ; 2020: 1-5.

2. Kurimoto T, Urushidani S, Yamada H, et al. SINET5: A low-latency and high-bandwidth backbone network for SDN/NFV Era. In: *Proceedings of the IEEE International Conference on Communications (ICC)* IEEE. ; 2017: 1-7.

3. Gerhards R. The Syslog Protocol. RFC 5424; 2009.

4. Bai J. Feasibility analysis of big log data real time search based on Hbase and ElasticSearch. In: *Proceedings of the 2013 International Conference on Computing, Networking and Communications (ICNC)* IEEE. ; 2013: 1166-1170.

5. Cuong TV, Nam NV. An Efficient Log Management System. *VNU Journal of Computer Science and Communication Engineering* 2016; 32(2): 43-48.

6. Abe H, Shima K, Miyamoto D, et al. Distributed Hayabusa: Scalable Syslog Search Engine Optimized for Time-Dimensional Search. In: *Proceedings of the Asian Internet Engineering Conference (AINTEC)* ACM. ; 2019: 9-16.

7. Kimura T, Ishibashi K, Mori T, et al. Spatio-temporal factorization of log data for understanding network events. In: *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)* IEEE. ; 2014: 610-618.

8. Li T, Ma J, Sun C. Dlog: diagnosing router events with syslogs for anomaly detection. *The Journal of Supercomputing* 2018; 74: 845-867.

9. Otomo K, Kobayashi S, Fukuda K, Esaki H. Latent Variable based Anomaly Detection in Network System Logs. *IEICE Transactions on Information and Systems* 2019; E102.D(9): 1644-1652.

10. Meng W, Liu Y, Zhu Y, et al. Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs. In: *Proceedings of the IJCAI International Joint Conference on Artificial Intelligence* IJCAI. ; 2019: 4739-4745.

11. Zhang X, Xu Y, Lin Q, et al. Robust log-based anomaly detection on unstable log data. In: *Proceedings of the ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)* ACM. ; 2019: 807-817.

12. Zheng Z, Yu L, Lan Z, Jones T. 3-Dimensional root cause diagnosis via co-analysis. In: *Proceedings of the International Conference on Autonomic Computing and Communications (ICAC)* ACM. ; 2012: 181-190.

13. Kobayashi S, Otomo K, Fukuda K, Esaki H. Mining causes of network events in log data with causal inference. *IEEE Transactions on Network and Service Management* 2018; 15(1): 53-67.

14. Kobayashi S, Otomo K, Fukuda K. Causal analysis of network logs with layered protocols and topology knowledge. In: *Proceedings of the 15th International Conference on Network and Service Management (CNSM 2019)* IFIP/IEEE. ; 2019: 1-9.

15. Li T, Ma J, Pei Q, Shen Y, Lin C, Ma S. AClog: Attack Chain Construction Based on Log Correlation. In: *Proceedings of the 2019 IEEE Global Communications Conference (GLOBECOM)* IEEE. ; 2019: 1-6.

16. El-Masri D, Petrillo F, Guéhéneuc YG, Hamou-Lhadj A, Bouziane A. A systematic literature review on automated log abstraction techniques. *Information and Software Technology* 2020; 122: 1-23.

17. Landauer M, Skopik F, Wurzenberger M, Rauber A. System log clustering approaches for cyber security applications: A survey. *Computers and Security* 2020; 92(101739): 1-17.

18. Xu W, Huang L, Fox A, Patterson D, Jordan MI. Detecting large-scale system problems by mining console logs. In: *Proceedings of the 22nd ACM Symposium on Operating Systems Principles (SOSP)* ACM. ; 2009: 117-132.

19. Tak BC, Tao S, Yang L, Zhu C, Ruan Y. LOGAN: Problem Diagnosis in the Cloud Using Log-Based Reference Models. In: *Proceedings of the IEEE International Conference on Cloud Engineering (IC2E)* IEEE. ; 2016: 62-67.

20. Zhang M, Zhao Y, He Z. GenLog: Accurate Log Template Discovery for Stripped X86 Binaries. In: *Proceedings of the IEEE Computer Society Computers, Software, and Applications Conference (COMPSAC)* IEEE. ; 2017: 337-346.

21. Vaarandi R, Pihelgas M. LogCluster - A Data Clustering and Pattern Mining Algorithm for Event Logs. In: *Proceedings of the 11th International Conference on Network and Service Management (CNSM)* IFIP/IEEE. ; 2015: 1-8.

22. Makanju A, Zincir-Heywood AN, Milios EE. Clustering event logs using iterative partitioning. In: *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)* ACM. ; 2009: 1255-1264.

23. Tang L, Li T, Perng Cs. LogSig: Generating System Events from Raw Textual Logs. In: *Proceedings of the ACM Conference on Information and Knowledge Management (CIKM)* ACM. ; 2011: 785-794.

24. Mizutani M. Incremental Mining of System Log Format. In: *Proceedings of the 2013 IEEE International Conference on Services Computing (SCC)* IEEE. ; 2013: 595-602.

25. Hamooni H, Debnath B, Xu J, Zhang H, Jiang G, Mueen A. LogMine: Fast Pattern Recognition for Log Analytics. In: *Proceedings of the ACM Conference on Information and Knowledge Management (CIKM)* ACM. ; 2016: 1573-1582.

26. Kimura T, Watanabe A, Toyono T, Ishibashi K. Proactive Failure Detection Learning Generation Patterns of Large-scale Network Logs. *IEICE Transactions on Communications* 2019; E102.B(2): 306-316.

27. He P, Zhu J, Zheng Z, Lyu MR. Drain: An online log parsing approach with fixed depth tree. In: *Proceedings of the 2017 IEEE International Conference on Web Services (ICWS)* IEEE. ; 2017: 33-40.

28. Qiu T, Tech G, Ge Z, Park F, Pei D, Xu JJ. What Happened in my Network? Mining Network Events from Router Syslogs Categories and Subject Descriptors. In: *Proceedings of the 2010 Internet Measurement Conference (IMC)* ACM SIGCOMM. ; 2010: 472-484.

29. Zhang S, Meng W, Bu J, Yang S, Liu Y, Pei D. Syslog Processing for Switch Failure Diagnosis and Prediction in Datacenter Networks. In: *Proceedings of the 25th IEEE/ACM International Symposium on Quality of Service (IWQoS)* IEEE/ACM. ; 2017: 1-10.

30. Kobayashi S, Fukuda K, Esaki H. Towards an NLP-based log template generation algorithm for system log analysis. In: *Proceedings of the 9th International Conference on Future Internet Technologies (CFI)* ACM. ; 2014: 1-4.

31. Nedelkoski S, Bogatinovski J, Acker A, Cardoso J, Kao O. Self-Supervised Log Parsing. *arXiv* 2020: 1-16.

32. Meng W, Liu Y, Zaiter F, et al. LogParse: Making Log Parsing Adaptive through Word Classification. In: *Proceedings of the 29th International Conference on Computer Communications and Networks (ICCCN 2020)* IEEE. ; 2020: 1-9.

33. He P, Zhu J, He S, Li J, Lyu MR. An Evaluation Study on Log Parsing and Its Use in Log Mining. In: *Proceedings of the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* IEEE/IFIP. ; 2016: 654-661.

34. Zhu J, He S, Liu J, et al. Tools and Benchmarks for Automated Log Parsing. In: *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)* IEEE. ; 2019: 121-130.

35. log2seq. https://github.com/amulog/log2seq; .

36. amulog. https://github.com/amulog/amulog; .

37. Urushidani S, Aoki M, Fukuda K, et al. Highly available network design and resource management of SINET4. *Telecommunication Systems* 2014; 56: 33-47.

38. Logstash. https://www.elastic.co/jp/products/logstash; .

39. Napalm-logs. https://napalm-logs.readthedocs.io/en/latest/; .

40. Elasticsearch. https://www.elastic.co/products/elasticsearch; .

41. Vaarandi R, Niziński P. Comparative analysis of open-source log management solutions for security monitoring and network forensics. In: *Proceedings of the European Conference on Information Warfare and Security (ECCWS)* ACPI. ; 2013: 278-287.

42. Li T, Jiang Y, Zeng C, et al. FLAP: An end-to-end event log analysis platform for system management. In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* ACM. ; 2017: 1547–1556.

43. Wurzenberger M, Landauer M, Skopik F, Kastner W. AECID-PG: A tree-based log parser generator to enable log analysis. In: *Proceedings of the 4th IEEE/IFIP International Workshop on Analytics for Network and Service Management (AnNet 2019)* IFIP/IEEE. ; 2019: 7–12.

44. logpai/logparser. https://github.com/logpai/logparser; .

45. Shima K. Length Matters: Clustering System Log Messages using Length of Words. *arXiv* 2016: 1-10.

46. He P, Zhu J, He S, Li J, Lyu MR. Towards Automated Log Parsing for Large-Scale Log Data Analysis. *IEEE Transactions on Dependable and Secure Computing* 2018; 15(6): 931–944.

47. Lafferty J, Mccallum A, Pereira FCN. Conditional Random Fields : Probabilistic Models for Segmenting and Labeling Sequence Data. In: *Proceedings of the International Conference on Machine Learning (ICML)* IMLS. ; 2001: 282-289.

48. Vaarandi R. A data clustering algorithm for mining patterns from event logs. In: *Proceedings of the 3rd IEEE Workshop on IP Operations & Management (IPOM)* IEEE. ; 2003: 119-126.

49. M. Rand W. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association* 1971; 66(336): 846-850.

50. Hubert L, Arabie P. Comparing partitions. *Journal of Classification* 1985; 2: 193-218.

51. Basu S, Banerjee A, Mooney RJ. Active semi-supervision for pairwise constrained clustering. In: *Proceedings of the 2004 SIAM international conference on data mining* SIAM. ; 2004: 333–344.

52. Rosenberg A, Hirschberg J. V-Measure: A conditional entropy-based external cluster evaluation measure. In: *Proceedings of the 2007 Conference on Empirical Methods on Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)* ACL. ; 2007: 410-420.

53. Tang L, Li T. LogTree: A framework for generating system events from raw textual logs. In: *Proceedings of the IEEE International Conference on Data Mining (ICDM 2010)* IEEE. ; 2010: 491–500.

54. Otomo O, Kobayashi S, Fukuda K, Esaki H. Latent Semantics Approach for Network Log Analysis: Modeling and its application. In: *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM 2021)* IFIP/IEEE. ; 2021: 1-9.

55. rsyslog. https://www.rsyslog.com; .

56. Yamashiro Y, Kobayashi S, Fukuda K, Esaki H. Network Log Template Generation from Open Source Software. *IEICE Technical Report (in japanese)* 2018; 118(204): 15–22.

57. vyos-legacy/vyatta-quagga. https://github.com/vyos/vyatta-quagga/tree/napa; .

□