This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3162857, IEEE Transactions on Dependable and Secure Computing

SwissLog: Robust Anomaly Detection and Localization for Interleaved Unstructured Logs

Xiaoyun Li, Pengfei Chen*, Linxiao Jing, Zilong He, and Guangba Yu

Abstract—Modern distributed systems generate interleaved logs when running in parallel. Identifiers (ID) are always attached to them to trace running instances or entities in logs. Therefore, log messages can be grouped by the same IDs to help anomaly detection and localization. The existing approaches to achieve this still fall short meeting these challenges: 1) Log is solely processed in single components without mining log dependencies. 2) Log formats are continually changing in modern software systems. 3) It is challenging to detect latent performance issues non-intrusively by trivial monitoring tools. To remedy the above shortcomings, we propose SwissLog, a robust anomaly detection and localization tool for interleaved unstructured logs. SwissLog focuses on log sequential anomalies and tries to dig out possible performance issues. SwissLog constructs ID relation graphs across distributed components and groups log messages by IDs. Moreover, we propose an online data-driven log parser without parameter tuning. The grouped log messages are parsed via the novel log parser and transformed with semantic and temporal embedding. Finally, SwissLog utilizes an attention-based Bi-LSTM model and a heuristic searching algorithm to detect and localize anomalies in instance-granularity, respectively. The experiments on real-world and synthetic datasets confirm the effectiveness, efficiency, and robustness of SwissLog.

Index Terms—deep learning; log parsing; anomaly detection; anomaly localization; log correlation

1 Introduction

Reliability and availability are of great importance in large-scale software systems, especially for those systems deployed on cloud servers. Faults including external faults (e.g., resource hogs, node disconnection) and internal software bugs (e.g., an infinite loop, incorrect configuration) are primary culprits of breaking the high availability and reliability of distributed systems. We consider a system that runs according to the expected behavior under sufficient resources as normal, and the deviation from the normal behavior as an anomaly. The above faults are manifested as anomalies by system metrics (e.g., CPU usage spikes), business KPIs (e.g., increase of request errors), and logs (e.g., exception messages). What is worse, they may lead to failures of downstream services, customer drain, and even a huge revenue loss. Take an anomaly in a cloud server as an example. During an upgrade, a snippet of error code caused an I/O hang in many running instances. Millions of services, especially e-commerce services and financial services built on top of cloud servers, suffered a huge revenue loss from this anomaly [2]. Anomaly detection and localization are therefore required to be conducted immediately to mitigate the impact of an anomaly.

Log data is an extensively available data resource in all kinds of software systems. It records system states and critical events at runtime so that developers generally utilize log data to obtain the system status, analyze anomalies. As the scale and complexity of modern computer systems increase, systems generate interleaved logs when processing asynchronous or executing tasks in parallel. It is quite challenging to manually process the exploded volume of interleaved logs (e.g., more than 50 GB of logs generated per hour [3]) and complex dependencies inside distributed components. An effective and efficient data-driven log analysis tool is thus an urgent need.

Recently, data-driven log-based anomaly detection and localization has been widely studied [4]–[12]. However, the existing approaches are built based on strong assumptions which are not easily satisfied in the real-world production environment. There are three major challenges shown as follows when applying the above methods in the production environment.

(i) Log dependencies. IDs are attached to running instances or entities to trace their states and critical events. Correlating logs via the same IDs is a typical approach to separate interleaved logs. The existing approaches correlate them solely for one component, which is not enough in distributed systems. Digging out dependencies between IDs and correlating log messages with the same ID across distributed components can largely help developers to localize anomalies.

(ii) Changing logs. Log formats are changing constantly in practice in those software systems under active development and maintenance. Kabinna, et al. [13] and Zhang, et al. [10] discussed log instability in their prior works. The empirical study shows that there are around 20-45% logs changed throughout the software system lifetime.

(iii) Latent performance issues. Performance issues are the common manifestation of partial failures [14], which refers that partial functionalities are broken, but not all of them. Indeed, partial failures are behind many real-

• *Xiaoyun Li, Pengfei Chen, Linxiao Jing, Zilong He, and Guangba Yu are with the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou 510006, China.*
E-mail: {lixy223, jinglx3, hezlong, yugb5}@mail2.sysu.edu.cn and chenpf7@mail.sysu.edu.cn.
Pengfei Chen is the corresponding author.
This paper is an extended version of ISSRE'20 paper SwissLog [1]

1545-5971 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on September 03,2022 at 06:18:14 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3162857, IEEE Transactions on Dependable and Secure Computing

world outages [2], [15]–[18], hence not latent problems that developers can ignore. In practice, it requires many aspects of system observations to distinguish whether it is a performance issue or not. Although not all latent performance issues can be detected through log data, digging out latent ones as much as possible also helps to enhance the reliability of the systems.

To overcome the above issues, in this study, we propose SwissLog, a robust and unified deep learning-based log anomaly detection and localization tool for unstructured interleaved logs, particularly log messages consisting of IDs and natural languages like "Receiving block blk_123 from IP:1.1.1.1". SwissLog is robust and versatile like the Swiss Army Knife. From real-world log data, we find two common types of log changes, including log sequence order changes and log time interval changes when an anomaly occurs. Considering these two types, our work targets those anomalies manifested on log data involving sequential log anomalies and part of performance issues, which we name latent performance issues in the remainder of the paper.

SwissLog contains two phases, namely the offline phase and the online phase, and consists of four stages, namely relation construction, log parsing, anomaly detection, and anomaly localization. In the offline phase, SwissLog builds relation graphs of IDs from the inter-component logs and correlates interleaved logs with the same IDs. To help analyze logs precisely, a novel and robust dictionary-based log parser without parameter tuning is introduced to extract log templates without losing the semantic meaning of sentences. To the best of our knowledge, we are the first to propose an online data-driven log parser without parameter tuning. Instead of assigning index for log templates, SwissLog encodes semantic information in the templates by Bidirectional Encoder Representation from Transformers (BERT) [19], which makes it robust to newly incoming templates and additionally considers temporal information by projecting it to a high-dimension embedding. Next, SwissLog employs Attention-based Bidirectional Long Short Term Memory (Attn-based Bi-LSTM) to learn the fixed patterns of log data, where the attention mechanism is robust to slight turbulence. In the online phase, SwissLog instantiates the relation between IDs from the incoming logs and parses logs in a streaming manner. The semantic and temporal information are concatenated and fed into the anomaly detection model. When an anomaly is detected, SwissLog raises an alarm and launches the anomaly localization process. Finally, SwissLog utilizes a heuristic algorithm to localize anomalies and reports anomalous instances or entities with a fine granularity.

The key contributions of this paper are four-fold:

- We propose a novel online dictionary-based log parsing method. Compared to previous work, it does not require any parameter tuning and provides predominant effectiveness, robustness, and generalizability.
- We introduce both semantic and temporal information in log sequences to detect various kinds of anomalies including sequential log anomalies and latent performance issues. The latent performance issues are rarely studied in log analysis before.
- We design a heuristic algorithm to localize anomalies at an instance level.
- We collect a real-world log dataset of Hadoop, and apply SwissLog to detect and localize anomalies. Moreover, we evaluate SwissLog on real-world datasets and synthetic datasets. The results prove the effectiveness and robustness of SwissLog. The source code of SwissLog [1] has also been released for reproducible research.

Extended from its preliminary conference version [1], this paper makes several major enhancements including the application of inter-component log correlation, IDs relation construction for unstructured interleaved logs in the distributed system, the heuristic-based anomaly localization in instance-level, the design and implementation of a novel online log parsing method, the advanced time embedding, new experimental comparison on latest log parser both in effectiveness and efficiency, a new real-world dataset to confirm the effectiveness of SwissLog, and code release of SwissLog for reproducible research.

2 Motivation

2.1 Relation Between Identifier Pairs

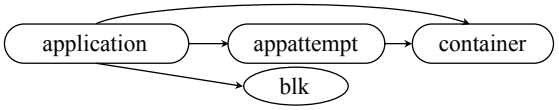


Fig. 1. An example of correlative relation between ID pairs

Each ID identifies an abstracted concept or a concrete resource instance, thus certain relationships (e.g., subordinate, dependent) between IDs during system execution. The log printer usually outputs these IDs in corresponding log messages. So with these IDs, we can reconstruct the inter-component interactions from logs. We can introduce a workflow construction method to profile the distributed system and group log data from different components by known IDs. An example of the correlative relation between ID pairs is presented in Fig. 1. These ID pairs are examples from the task-based distributed system, Hadoop. When a task is executed in Hadoop, Yarn creates an application request for the resource manager. The resource manager allocates computing resources as containers, which is different from containers in Docker, to this request. Also, Hadoop stores files as block replicas (blk) using Hadoop Distributed File System (HDFS). If the container_123 goes wrong, an application attempt (appattempt) launch for the container attempt. We can also check the log messages from the resource manager and node managers by grouping the keyword "container_123" to figure out the root cause. Hence, building a workflow for IDs is an effective method to localize faults in a fine-grained manner based on log messages.

2.2 Diverse Anomalies

It is intuitive to observe system status and inspect potential anomalies from log data. One of the normal sequence is shown in Fig. 2(a). We can observe that the beginning of a normal sequence is to allocate and receive blocks. After a series of operations, this block is eventually added to

1. https://github.com/IntelligentDDS/SwissLog

1545-5971 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on September 03,2022 at 06:18:14 UTC from IEEE Xplore. Restrictions apply.

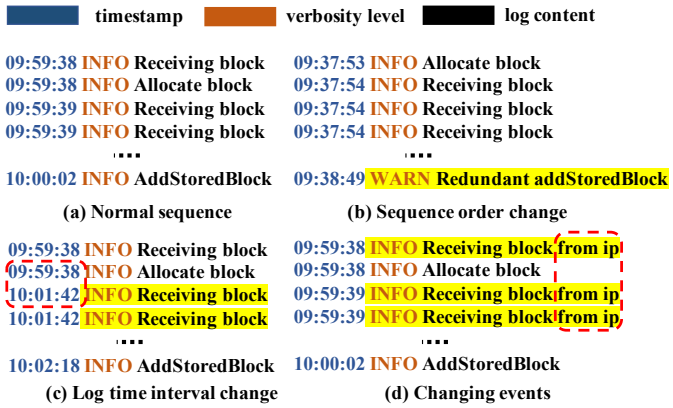


Fig. 2. Four types of log sequences with different changes.

the set of stored blocks, which means the end of a cycle. A large-scale system inevitably encounters faults, part of them manifested in log pattern changes. We target two types of log changes in practice, namely log sequence order change and log time interval change, as shown in Fig. 2(b) and Fig.2(c). We omit some unimportant log messages and only show the key information (i.e. time, verbosity level, simplified log content).

2.2.1 Log sequence order change

An abnormal sequence against the normal one in Fig. 2(a) is depicted in Fig. 2(b), where the abnormal log messages are highlighted in yellow. In this case, the system received a redundant *addStoredBlock* request, causing the log sequence order change. Therefore, sequential log anomalies can be generally observed from their abnormal sequence order. Prior works [4], [5], [7], [9], [10], [20] mostly focus on sequential log anomalies and recognize them by detecting abnormal log sequence order.

2.2.2 Log time interval change

Another kind of anomaly is log time interval change, possibly caused by performance issues, whose example is shown in Fig. 2(c). In contrast to abnormal sequence order, blocks with performance issues usually keep the same sequence order as the normal one. However, performance issues slow down the execution time of specific tasks according to their faulty components. For example, the receiving block in line 3 has a 3000-millisecond latency which is caused by the network congestion. The performance issue here is manifested in the log time interval change. Such performance issues are like buried land mines that may trigger catastrophic outages. Therefore, the topic of detecting performance issues gains lots of attention recently [14], [21]–[24]. But the existing approaches employ static analysis to find performance bugs or an intrusive method to detect them. Either they are difficult to detect performance issues at runtime or they slow down system performance. If we detect potential performance issues by mining time interval changes in log data, the above problems are accordingly solved.

2.3 Changing Events

Modern software systems that need an automated log processing tool are probably under active development and maintenance. Kabinna, et al. [13] examined the stability of logging statements (i.e., the line printing log in source code) via an empirical study. They find that 20-45% of the logging statements change throughout the whole lifetime. Zhang, et al. [10] also conducted a similar empirical study on Microsoft Service X. As reported, up to 30.3% logs are changed in the latest version. Two main possible factors that cause log changes are: 1) Developers add new log statements to source codes. 2) Developers add a few new features and modify the content of log statements. Extra words are thus attached to log statements while not changing their meaning. Fig. 2(d) shows a common case of changing events. String "from ip" is added to the log statement while it keeps the original meaning. Such changes make the log key-based anomaly detection [4]–[7] approaches perform unsatisfactorily since they consider a changed logging statement as a new template. The previous model is thus required to retrain when new templates arrive. Currently, the state-of-the-art method to detect anomalies in changing events is LogRobust [10].

On the basis of the observation in a real-world production environment, we propose SwissLog, a robust and unified deep learning-based log anomaly detection and localization tool. It adopts the typical idea to correlate interleaved logs with the same IDs and introduces the relation between ID pairs construction. Then it uses the correlated log data to detect sequential anomalies and potential performance issues and reports the faulty instances in a fine-grained manner.

3 Design of SwissLog

We first begin with an overview of SwissLog which is presented in Fig. 3. SwissLog comprises two phases, namely the offline processing phase and the online processing phase. In the offline phase, SwissLog first constructs ID relation graph ($\mathcal{R}$) graph (❶). Then, SwissLog adopts a novel log parsing method and extracts multiple templates by tokenizing, dictionarizing, and clustering history log data (❷). These templates are kept as natural sentences instead of event ids. Then the grouped log sequence is transformed into semantic information and temporal information. SwissLog uses BERT encoder to encode semantic information $\boldsymbol{F}$ into embedding $\boldsymbol{E}_{context}$ and projects temporal information $\Delta\boldsymbol{T}$ onto embedding $\boldsymbol{E}_{time}$ (❸). The concatenation of semantic embedding $\boldsymbol{E}_{semantic}$ and time embedding $\boldsymbol{E}_{time}$ as input is fed into Attn-based Bi-LSTM to learn the features of normal, abnormal and performance-anomalous log sequence (❹). At runtime, when a new log arrives, SwissLog correlates those log messages with the same IDs (❺) and instantiates $\mathcal{R}$ graph as $\mathcal{R}_i$ graph (❻). Then it goes through the log parsing step (❼), sentence embedding step and is fed into the pre-trained model from the offline phase. If an anomaly occurs, SwissLog will alarm (❽) and get into the anomaly localization process. Finally, SwissLog analyzes the faulty instance and reports the corresponding instance ID (❾).

We next introduce the design of SwissLog including relation construction (Sec. 3.1), log parsing (Sec. 3.2), sentence embedding (Sec. 3.3), Attn-based Bi-LSTM (Sec. 3.4), anomaly detection (Sec. 3.5), anomaly localization (Sec. 3.6).

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on September 03,2022 at 06:18:14 UTC from IEEE Xplore. Restrictions apply.

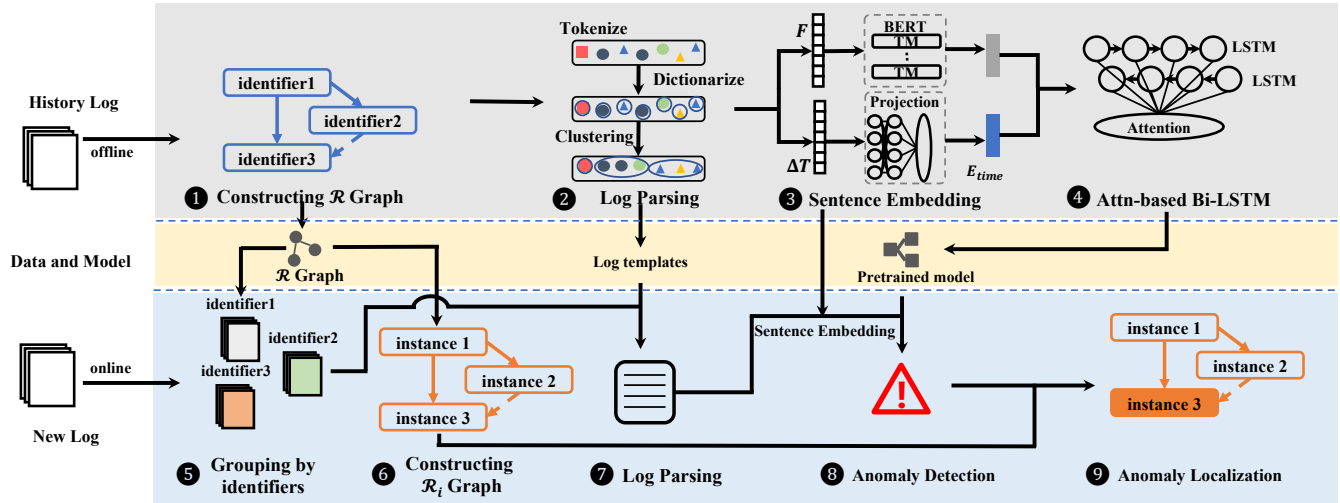


Fig. 3. The overview of SwissLog

3.1 Relation between ID pairs Construction

Nowadays, workflow reconstruction from log data has been widely studied. lprof [25] uses static analysis to find the code execution path containing the log printing statements. Stitch [26] extracts IDs from log messages and builds System Stack Structure graphs to capture the hierarchical relationship between object types, where each ID is of a type. IntelLog [8] further constructs the hierarchical graph from mining log messages and builds the relation between entities.

In this paper, we choose Stitch (more details refer to the original paper [26]) to construct the ID relation graph, which denotes as $\mathcal{R}$. Noting that constructing $\mathcal{R}$ graph is not one of our novel parts, and we apply it to localize anomalies. $\mathcal{R}$ graph is a directed acyclic graph (DAG), where each node represents an ID and each edge captures the hierarchical relation between ID pairs. There are four possible relation between ID pairs: i) empty; ii) 1:1; iii) 1:n, and iv) m:n. The empty relation means that two IDs have no dependency. The relation 1:1 indicates that one may be another alias, which refers to that they can be used interchangeably in some way. The relation 1:n refers that the object dispatches n objects with different IDs. For example, the system allocates multiple containers for an application. The relation between application and container is thus 1:n. While the relation m:n indicates that resources related to m IDs are reused for handling n tasks. Our study utilizes the relation 1:n and m:n to localize anomalies. For the self-explanation of this paper, we give an example to show how to construct the $\mathcal{R}$ graph.

We first use a small set of logs shown in Fig. 4 collected from Hadoop to demonstrate the $\mathcal{R}$ graph construction process. The left column represents the collected component while the right side is the detailed log content with colored IDs. For example, Line 1 is a log message from `resourcemanager.log`, and it shows that the `application01` is related to `appattempt01_01` and `container01_01`. Meanwhile, `namenode.log` records that, system allocates `blk_01` and `blk_02` to `application01` to store data. So we can derive that the relation between ID pairs `application` and `blk` is 1:n. In this way, we can construct a $\mathcal{R}$ graph from Fig. 4.

The $\mathcal{R}$ graph of the above raw log messages (Fig. 4) is

1	ResourceManager	Storing Attempt: application01, AttemptId: appattempt01_01, MasterContainer: container01_01
2	NameNode	Block* allocate blk_01 for /tmp/logs/application01
3	NameNode	Block* allocate blk_02 for /tmp/logs/application01
4	ResourceManager	Assign container01_02 to attempt_m_01
5	Application	Launch container01_02 and attempt_m_01
6	NodeManager	attempt_m_01 using containerId: container01_02
7	ResourceManager	Assign container01_03 to attempt_r_01
8	Application	Launch container01_03 and attempt_r_01
9	NodeManager	attempt_r_01 using containerId: container01_03
10	DataNode	Block* allocate blk_01 for attempt_m_01
11	DataNode	Block* allocate blk_02 for attempt_r_01

Fig. 4. An example of log snippet that contains multiple set of IDs in Hadoop

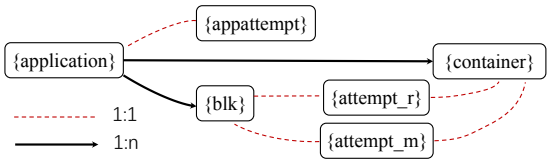


Fig. 5. The $\mathcal{R}$ graph of Hadoop log snippet in Fig. 4

shown in Fig. 5. The red dotted line denotes the relation 1:1 while the black arrow line represents the relation 1:n. We store the static $\mathcal{R}$ mined in the offline phase. When a new log e arrives in the online phase, SwissLog constructs an instantiated $\mathcal{R}_i$ graph, where each node is an ID instance. If the set of ID types in e matches the ID type of one node in $\mathcal{R}$ graph, it will instantiate a node N. After that, SwissLog first checks if N matches any of the existing $\mathcal{R}$ nodes. Otherwise, it instantiates a new instance $\mathcal{R}_i$ graph.

With an instantiated $\mathcal{R}_i$ graph, we can traverse it to localize the anomalous instance and aid operators to identify the precise root cause. The heuristic algorithm that traverses the graph is illustrated in Sec. 3.6 in detail. Furthermore, we explain how it works and discuss a concrete case in Sec. 4.9.

3.2 Log Parsing

Log parsing, the critical stage before executing the downstream log analysis tasks, has been widely studied for many

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on September 03,2022 at 06:18:14 UTC from IEEE Xplore. Restrictions apply.

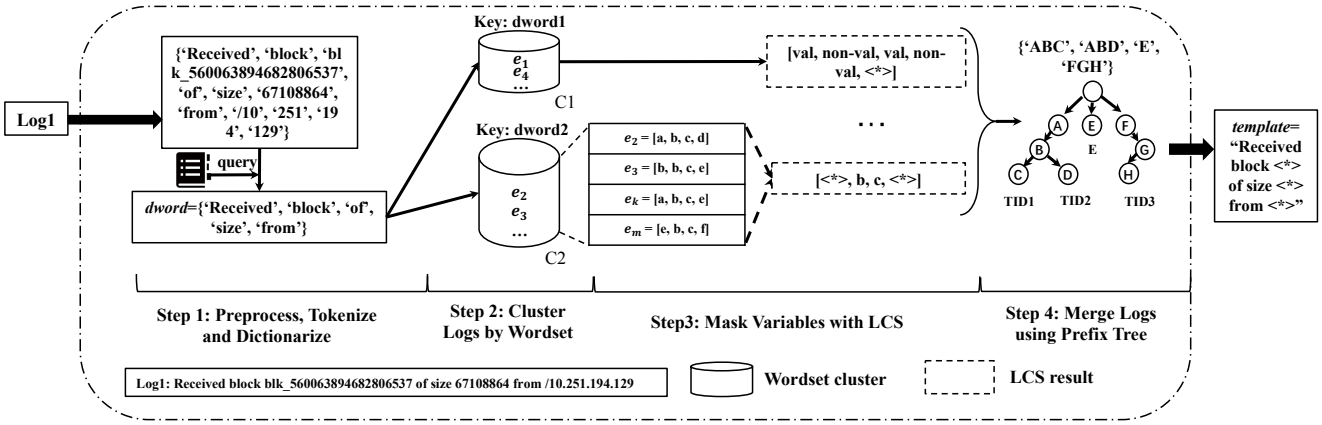


Fig. 6. The workflow of log parsing

years (more details refer to Related Work in Sec. 6). Previous studies generally introduce many parameters such as message-type threshold in Spell [27], similarity threshold, and merge threshold in Drain [28]. Such parameters are tuned after multiple trials. We expect to propose a log parsing method robust to most common logging systems without parameter tuning. Event-based log messages are usually adopted to report system events with natural language. We observe that event-based log messages in Fig. 6 are readable because most of the words in it are valid words, which can be looked up in a dictionary. Such log messages with the same valid words can be visually treated as one template. Inspired by this observation, a dictionary-based approach naturally parses these event-based log messages. Here we propose an online log parser based on a dictionary, which can achieve high accuracy and comparable efficiency. In the following parts, we illustrate the log parser in SwissLog step by step and give an online log parsing case study.

3.2.1 Step 1: Preprocess, Tokenize, and Dictionarize

In the first step, we preprocess the arriving log entries. The preprocessing step is simple but effective for log parsing. Developers only need to program some regular expressions to replace common variables (e.g., IP addresses: 192.168.0.1, Day: Mon/Monday) and special variables in systems. For example, we apply the regular expression "\d+\.\d+\.\d+\.\d+\." to match IP addresses; apply "blk_-?\d+" to find the block id and "container_\d+" to find the container id in Hadoop; and "core\.\d+" to find the core id in BGL.

For each log entry e, we define a slice of log entry as *token*. How to tokenize a complete log entry into appropriate *tokens* is a critical problem in the dictionary-based approach since the parsing result largely depends on it. In addition to using whitespace, the logging system is more likely to use special delimiters such as colon, and quotation marks to separate strings. For better tokenization, we thus utilize five special delimiters, namely $\{, . ; : "\}$ attained from empirical study and whitespace to tokenize log entries. To improve the quality of templates, we also replace those digit tokens with wildcards.

Then we introduce a dictionary to dictionarize all tokens. Given a dictionary $D = \{w_1, w_2, ..., w_n\}$, such that every word w_i can be identified as a valid word. After tokenizing, we first check if each token of log entry e is in the dictionary D. If yes, we put the token into *wordset*. An example is shown in Fig. 6 Step 1. When a raw log message "Received block blk_560063894682806537 of size 67108864 from /10.251.194.129" arrives, it will be separated into 11 *tokens*. SwissLog looks up the dictionary D for every token, then *'Received'*, *'block'*, *'of'*, *'size'*, *'from'* are identified as valid words. Particularly for log-specific concatenated words like "PowerDown", we import an external package *wordninja* [29] to split it into "Power" and "Down" based on the unigram frequencies in English Wikipedia. Finally, we obtain the *wordset* $dword$ containing valid words.

As for the dictionary, developers are able to construct a dictionary from a public corpus (like [30] as in RQ1, which is a large-scale corpus containing 5.2 million English sentences); or customize a dictionary for their systems. The results in RQ1 confirm that the dictionary constructed from one large-scale public corpus can cover most words in different logging systems. In addition, developers can add particular domain knowledge on-demand when the logging systems are updated.

3.2.2 Step 2: Cluster Logs by Wordset

Logs with the same *wordset* are possibly generated from one log statement. When a new *wordset* $dword$ arrives, SwissLog looks for the matched group for it. If a group is matched, SwissLog puts the preprocessed log entry into it. Otherwise, SwissLog creates a new cluster for *wordset* $dword$. Assume that $dword_1$, $dword_2$, $dword_3$ are the *wordset* of log entries e_1, e_2, e_3 in Fig. 6, respectively. Since the log entries e_1 and e_2 have different *wordset*, SwissLog creates the new cluster C_1 and C_2 for them separately. *Wordset* $dword_3$ is identical with $dword_2$, so log entry e_3 is consequently categorized into cluster C_2.

Sometimes, a valid word occurs multiple times in one log entry. For example, "120 bytes sent, 80 bytes received". The word `bytes` occurs twice in this log entry, which is easily confused with those log entries with only one `bytes`. Taking the word occurrence into account, we especially use count set $fword$ to store *wordset* occurrence. Hence, only when the *wordset* $dword$ and occurrence $fword$ are identical, can the two log entries be categorized into the same cluster.

3.2.3 Step 3: Mask Variable with LCS

The masking layer is to distinguish the variable part within one cluster. Generally, common sequences of log entries in

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on September 03,2022 at 06:18:14 UTC from IEEE Xplore. Restrictions apply.

the same cluster can be regarded as the constant parts, while the changing part can be viewed as the variable parts. Next, we introduce token-level Longest Common Sequence (LCS) to help us mask all variable parts in a cluster with wildcards.

LCS is to find the longest common sequence among a sequence set. We first define token-level subsequence. Suppose Σ is a universe of *tokens*. Given any sequence $\alpha = \{a_1, a_2, ..., a_m\}$, such that $a_i \in \Sigma$. Then a subsequence of α is defined as $\{a_i,\ a_{i+1},\ ...,\ a_j\}$, where $i \in \mathbb{Z}^+$ and $1 \leq i \leq j \leq m$. A common subsequence is a subsequence of both sequence α_1 and α_2. Given four log entries e_2, e_3, e_k, e_m in the cluster C_2 (as shown in Step 3 of Fig. 6), a common subsequence between e_2 and e_k is $\{a, b, c\}$. Finally, we obtain $\{b, c\}$ as the token-level LCS of e_2, e_3, e_k, e_m.

After clustering by *wordset*, the preprocessed log entries e_1 and e_4 are in the cluster C_1. Assume that the log entry e_1 is "Received signal *, code=*, errno=*, address=0x000001b0" and log entry e_4 is "Received signal *, code=*, errno=*, address=0x000001f2". The cluster stores all *tokens*, containing valid words and non-valid words (e.g., errno). Then we apply token-level LCS algorithm to those log entries in cluster C_1. Finally, {'Received', 'signal', 'code', 'errno', 'address'} can be found as LCS of cluster C_1, and we replace the remainder parts by wildcards. Noting that non-valid words can also be treated as constant parts if they are the common subsequence in all log entries in the cluster. Therefore, SwissLog shows a good robustness to those words out-of-vocabulary in logging systems.

3.2.4 Step 4: Merge Logs using Prefix Tree

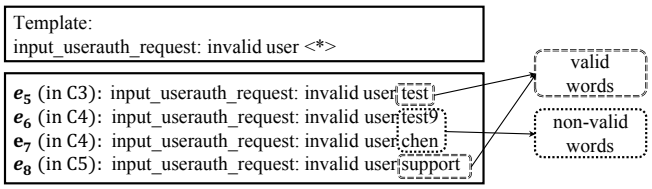


Fig. 7. A snippet of OpenSSH log data

After executing step 3, we attain rough templates from different clusters. Here we will encounter two cases. The first case is that variable parts only contain valid words. For example, "Container * transitioned from NEW to LOCALIZING" and "Container * transitioned from LOCALIZING to SCHEDULED". Container state changes from one to another may come from the same log printing statement but they have different physical meanings. Therefore, we tend to split them in our study.

The other important case that the variables in the groundtruth template contain both valid words and non-valid words (val and non-val in Fig. 6) should be carefully considered. An example shown in Fig. 7 can well present this issue. Log entries e_5-e_8 are excerpted from real-word OpenSSH log data [31]. We can observe that the last word are `test` (valid word), `test9` (non-valid word), `chen` (non-valid word), `support` (valid word). Compared with $dword_5$, $dword_4$ involves valid words `test`, hence log entry e_5 and e_6 are grouped into cluster C_3 and C_4. But intuitively, they should be clustered into one template and the last word should be the variable part in groundtruth. The prefix tree has been applied in log analysis before [27], [28], [32], here we employ it so as to merge non-valid words and valid words in the second case.

The Prefix Tree, an ordered tree data structure, is often used to store a dynamic set. The root of the prefix tree points to an empty string and all the descendants of a node in the prefix tree have a common prefix string with that node. Step 4 in Fig. 6 shows an example of a prefix tree structure. Keys are listed in the nodes and final string values are below them. Given a set of strings $strs = \{ABC, ABD, E, FGH\}$, they are indexed by the prefix tree. String *ABD* traverses the whole tree starting from the root to check if there exists a common prefix. Then it finds *ABC*, so their leaf nodes point to the same parent node B. While string *E* and *FGH* branch out because they have no common prefix.

The input of step 4 are the masked templates attained from different clusters. One *wordset* corresponds to one template. We first sort all *wordset* in alphabetical order. Specially, we place the wildcard as the first rank before all alpha orders. It helps the prefix tree to distinguish the second case and merge valid words into wildcards. In this way, the example shown in Fig. 7 can be eventually clustered into one template.

3.2.5 Online Log Parser

If the logging system is evolving due to active development and maintenance, we also propose an online version log parser to update templates. The requirements of online log parsers are 1) Efficient. The log parsing speed should catch up with the log generating speed. 2) Accurate. Parsing results should be accurate and provide good input for downstream log analysis tasks. In this part, we present an online version log parser.

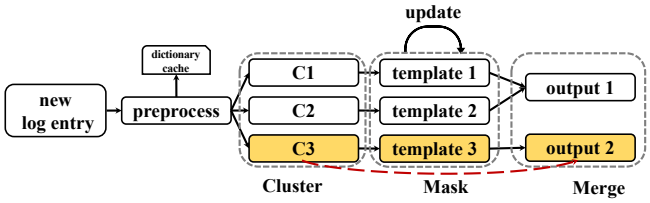


Fig. 8. Online log parser of SwissLog

The whole process of the online log parser is shown in Fig. 8. When a new log entry arrives, it first goes through the preprocessing layer. Although we introduce a dictionary containing 10k words, most of the words may not occur in this logging system. For example, only 69 valid words are found in 10m HDFS logs. Therefore, we apply a dictionary cache, which stores those valid words that have occurred, to enhance the efficiency of the dictionarizing process.

Then the log entry is allocated to the matched cluster according to its *wordset*. Each cluster maintains one template in the masking layer. The masking layer applies LCS between the new log entry and the existing template. If the template is updated, the merging step would reconstruct the prefix tree following the steps in Sec. 3.2.4. Finally, we get the output from the merged prefix tree. In practice, LCS is a high-cost process in log parsing. Therefore, we also introduce a cut-off algorithm to avoid some unnecessary masking. Consider the cluster $C3$ where $dword = \{'Received', 'block', 'of', 'size', 'from'\}$ and according *template3*="Received block * of size * from *". We can

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on September 03,2022 at 06:18:14 UTC from IEEE Xplore. Restrictions apply.

observe that the *template3* is minimal as the *template3* only contains valid words. It is impossible to mask more variable parts in the same clusters. Such templates are named mature templates, which are colored with yellow in Fig. 8. Once the template is mature, we will jump the masking layer and the merging layer and directly get the final output (following the red dotted line in Fig. 8).

3.2.6 Time Complexity Analysis

To satisfy the efficient requirements of the online log parser, we analyze the time complexity of the log parser in SwissLog. Supposed that m, n, and s are the number of message types, raw log messages, and the length of a log entry respectively. Here we first analyze time complexity for one log entry. Step 1 *tokenize and preprocess* is executed in a streaming manner, so its complexity is $O(1)$. Step 2 costs $O(s * logd)$ to lookup dictionary, where d is the size of the dictionary and it is stored as a tree structure in a computer. Supposed that c is the number of clusters, step 2 costs $O(logc)$ to search the matched group. In the step itmask layer, supposed that c is the number of clusters. Finding LCS between two sequences costs $O(s^2)$, then to find common sequences using LCS costs $O(s^2)$. In Step 4, before constructing a prefix tree, we need to sort all log entries to ensure the effectiveness of log parsing and it costs $O(logm)$. Then constructing a prefix tree also costs $O(m)$. The complexity of step 4 is $O(m + logm)$. Therefore, the worse complexity of log parser in SwissLog without cache mechanisms is near $O(s*logd+logc+s^2+m+logm)$. With cache mechanisms, we can largely reduce the time complexity to $O(s*logd+logc)$. In practice, the above parameters are much smaller than log size, which are obviously constant, so the time complexity is near $O(n)$.

3.3 Sentence Embedding

After log parsing, one log entry is eventually stored as the matched log template with only its constant parts. Prior works adopt log keys to embed sentences with the matched log template keys but they lose valuable semantic information inside the natural language texts. Besides the semantic information, SwissLog additionally introduces temporal information as features to make it capable of capturing more kinds of faults. Although the goal of SwissLog is to localize anomalies in systems with IDs, it is also able to detect faults in those systems without IDs. Therefore, a sequence linked with IDs or split by sliding windows is transformed into semantic information T and temporal information ΔT. Then we encode two kinds of information with the following methods.

3.3.1 Semantic Embedding

Case 1: "Expected quotacontroller.Sync to still be running but it is **blocked**. %v",err

Case 2: "`{"metadata":{"ownerReferences":[{"apiVersion":"%s","kind":"%s","name":"%s","uid":"%s","controller":true,"**block**OwnerDeletion":true}],"uid":"%s"}}`" ,m.controllerKind.GroupVersion(), m.controllerKind.Kind, m.Controller.GetName(), m.Controller.GetUID(), rs.UID)

Fig. 9. Two log cases extracted from Kubernetes

Log formats are under active evolution. Yet, the key meaning of changing log statements stays unchanged as we discussed in Sec. 2.3. Sentence embedding is therefore introduced to encode templates into vectors to preserve the key meaning of log messages. Word2Vec [33] has been widely used in the existing approaches to embed words into vectors. But it only performs the limited utility meeting the case in Fig. 9. There are two log cases extracted from Kubernetes source codes. Both cases 1 and 2 contain the word `block`. `block` in case 1 is a `verb` which means to prevent something from happening, developing, or making progress. While `block` in case 2 is a `noun` which represents that there exists a data block to be processed. It produces the same word embedding with Word2Vec for the word `block`. It will probably confuse downstream works and lead to false alarming. To overcome the challenges of polysemous words and changing events in log data, we need an advanced word embedding approach.

The pre-trained language representation gains considerable progress in the NLP field, especially BERT developed by Google. Google released the pre-trained language model which has trained on Wikipedia corpus and Book corpus. The public corpus does cover some special semantics and domain knowledge in computer science such as *admin* and *root*. Compared to other embedding methods, the large pre-trained language model provides a sufficient word database to encode words more precisely. Our work targets readable event-based log messages, so that language model pre-trained on public corpus is able to capture their meanings. As described in the initial paper of BERT [19], there are two usages based on specific downstream tasks: fine-tuning and feature extraction. We adopt the latter to get the semantic embedding.

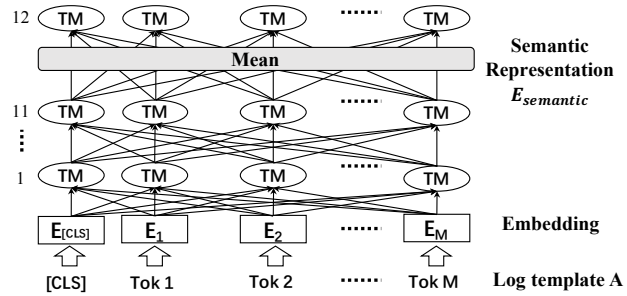


Fig. 10. The structure of BERT

Fig. 10 shows a simplified structure of BERT. As we only use the feature extraction part of BERT, the rest of BERT will not be shown in this paper. Log template A is first tokenized into M tokens as listed in Fig. 10 (Tok means Token). BERT particularly adds a [CLS] token at the beginning of the sentence, marking the starting position of a sentence. The embedding layer generates an embedding vector $\boldsymbol{E}_i$ involving token embedding, sentence embedding, and transformer positional embedding for each token including [CLS], where i refers to the ith word in the sentence. Then embedding vectors $\boldsymbol{E}_i$ are fed into transformer encoders (TM in Fig. 10) as model inputs. A self-attention layer is particularly added in the transformer encoder to acquire other word information in log messages. Therefore, when processing a log message, the attention mechanism builds a correlation among all other words in this sentence. After that, the

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on September 03,2022 at 06:18:14 UTC from IEEE Xplore. Restrictions apply.

output of self-attention is transferred to two feed-forward layers to learn the further position and word vector relation.

SwissLog leverages an off-the-shelf service *bert-as-service* [34] which uses BERT as a sentence encoder and runs it as a service. Google Research released pre-trained models on the website [35]. SwissLog selects the BERT base model involving a 12-layer of transformer encoders and 768-hidden units of each transformer for semantic embedding. Each output per token from each layer can be used as a word embedding. The first layer is close to the initial word embedding while the last layer may be biased to the training of downstream tasks. Choosing a word embedding from these is then a trade-off. Xiao, et al. [34] researched this problem and suggested generating word embedding in the last second layer. Hence, we take the average of the hidden state of the encoding layer on the time axis to get the final semantic embedding $\boldsymbol{E}_{semantic}$.

3.3.2 Time Embedding

The existing approaches either perform static analysis to find performance bugs or an intrusive method to detect performance issues. Log time interval anomalies, referring to the anomalous elapsed time, are probably caused by latent performance issues. As shown in Fig. 2(c), timestamps are recorded in each log message in most logging systems. Hence, it is possible to use the timestamp to calculate elapsed time between two log messages. In a stable system, executing common program paths between two log statements costs a relatively steady time, thus we introduce the time interval information into anomaly detection.

We calculate the time difference Δt between two log messages e_1 and e_2, and then obtain a temporal differential sequence $\Delta \boldsymbol{T} = \{\Delta t_1, \Delta t_2, ..., \Delta t_i, ...\}$, where i refers to the time axis in time series. Additionally, minus one is used to pad the beginning of the time series. For example, we obtain temporal sequence $\Delta \boldsymbol{T} = \{-1, 0, 3, 0, ...\}$ in seconds in Fig. 2(c). Intuitively, we can observe that Δt is closely related to the former event e_1. For example, the IO task shows a smaller Δt while the scheduling task shows a greater Δt. Even in the normal operation, the time interval vibrates in a task-related time range. But the attention model in Sec. 3.4 assigns a lower weight to such task-related time interval. Therefore, SwissLog is also robust to logs generated from various task types. Also, we standardize all temporal data by removing the mean and scaling to unit variance so as to receive trainable data.

However, 1-dimension temporal data exhibits limited information. It is better to extend 1-dimension temporal data to a higher dimension embedding. Li, et al. [36] proposed a time-dependent event representation method. Inspired by their work, we encode Δt using soft one-hot encoding.

The first step is to project the scalar value Δt onto a d-dimension vector space. As presented in Eq. 1, we multiply Δt with a randomly-initialized weight vector $\boldsymbol{W}$ and then add a randomly-initialized biases vector $\boldsymbol{b}$, where p is the projection size. After the above linear transformation, we apply a softmax function to catch the importance vector $\boldsymbol{s}$ of the obtained projection vector. The function $softmax(\cdot)$ is used to re-scale a tensor, making its elements lie in the range $[0, 1]$ and sum to 1 along with a selected dimension.

$$\boldsymbol{s^i} = softmax(\Delta t^i \boldsymbol{W} + \boldsymbol{b}), \text{where } \boldsymbol{W} \in \mathbb{R}^p, \boldsymbol{b} \in \mathbb{R}^p \quad (1)$$

Then we weight all rows in the randomly-initialized embedding matrix $\boldsymbol{E}_s$ with vector values in $\boldsymbol{s} = \{s^1, s^2, ..., s^{n-1}\}$. Finally, we get the time embedding vector $\boldsymbol{E}_{time}$.

$$\boldsymbol{E}_{time} = \boldsymbol{s}\boldsymbol{E}_s, \text{where } \boldsymbol{E}_s \in \mathbb{R}^{p \times d} \quad (2)$$

3.4 Attn-based Bi-LSTM

After sentence embedding, each log message is transformed into a semantic vector $\boldsymbol{E}_{semantic}$ and a time embedding vector $\boldsymbol{E}_{time}$. We obtain the concatenation $\boldsymbol{V} = concat(\boldsymbol{E}_{semantic}, \boldsymbol{E}_{time})$, so each log sequence is represented as a list of vectors (like $[\boldsymbol{V}_1, \boldsymbol{V}_2, ..., \boldsymbol{V}_T]$). Taking such vectors as input, SwissLog adopts the Attn-based Bi-LSTM neural network in Fig. 11 for detecting diverse anomalies.

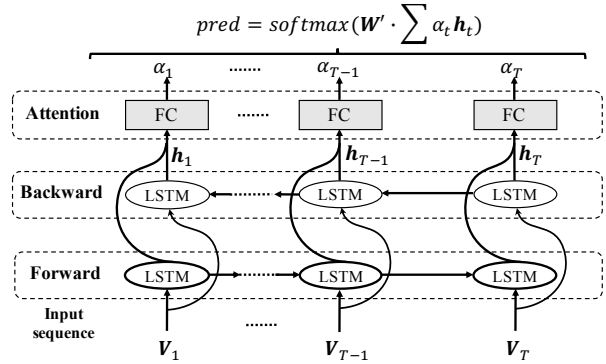


Fig. 11. The architecture of Attn-Bi-LSTM.

The LSTM network, a variant of Recurrent Neural Network (RNN), is capable of capturing contextual information for sequential data. Incorporating gating mechanisms, LSTM can have the ability to remove or add information to the cell state and finally decide what information to go through. It allows neural networks to dynamically exhibit temporal behavior. The LSTM network consists of three layers: input layer, hidden neurons layer, and output layer. At each time step, LSTM calculates the new cell state c_t and new hidden state $\boldsymbol{h}_t$ using the input state $\boldsymbol{V}_t$ and transferred hidden state $\boldsymbol{h}_{t-1}$. Bi-LSTM is an extension of LSTM. It particularly adds a hidden neuron layer in a backward direction and calculates each hidden state h_t at time t through concatenating from both directions as input to output layers.

Like verbosity levels in log statements, different log messages show different importance in a log sequence. To mitigate the impact of noisy or unimportant log statements, attention mechanisms are therefore introduced to Bi-LSTM to assign different weights to different log statements. Noisy or unimportant log statements will tend to be given low attention. The attention function α_t at time t is implemented with a fully connected layer (i.e., FC layer in Fig. 11), which performs the following calculation,

$$\alpha_t = tanh(\boldsymbol{W'}_t^{\alpha} \cdot \boldsymbol{h}_t). \quad (3)$$

Here, $\boldsymbol{W'}_t^{\alpha}$ denotes the trainable weight matrix of the attention layer at time t. The function $tanh(\cdot)$ is kind of an activation function. Then, all the hidden states multiply their corresponding α_t and are further summed to get a summarized hidden state vector. Finally, a prediction output is calculated by applying a softmax layer to the summarized

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on September 03,2022 at 06:18:14 UTC from IEEE Xplore. Restrictions apply.

hidden state vector. The computation is formulated in Eq. 4, with $\boldsymbol{W}'$ representing the softmax layer weight.

$$pred = softmax(\boldsymbol{W}' \cdot (\sum_{t=0}^{T} \alpha_t \cdot \boldsymbol{h}_t)) \tag{4}$$

At the training stage, we calculate the cross-entropy as loss function and use the Adam optimizer [37] to train networks. The cross-entropy is formulated in Eq. 5, where $\boldsymbol{y}^{(i)}$ denotes the one-hot representation of the label (normal or abnormal) of the ith log sequence and $\hat{\boldsymbol{y}}^{(i)}$ refers to its prediction.

$$H\left(\boldsymbol{y}^{(i)}, \hat{\boldsymbol{y}}^{(i)}\right) = -\sum_{j=1}^{2} y_j^{(i)} \log \hat{y}_j^{(i)}. \tag{5}$$

3.5 Anomaly Detection

In the offline phase, we obtain a pre-trained Attn-based Bi-LSTM model for anomaly detection using history logs. When a set of new log messages arrives, it first goes through log parsing and sentence embedding. Then the obtained vectors as input are fed into the pre-trained model. Finally, the Attn-based Bi-LSTM can detect if an anomaly occurs. Pay attention to that when SwissLog makes decisions based on a session of log messages correlated by a common ID such as block ID. Therefore, an anomaly can be robustly reported until the session is closed. In other words, SwissLog works in a near real-time mode like LogRobust [10].

3.6 Anomaly Localization

During detection, we tag the detected instance IDs as anomalous instances. We have already constructed an instantiated DAG $\mathcal{R}_i$ graph in Sec. 3.1. Only after there are no updates on the $\mathcal{R}_i$ graph, it will launch the anomaly localization progress. It is intuitive that if an anomaly occurs in the low-level instances, it will propagate to high-level instances. Following this intuition, we design a heuristic algorithm to localize anomalies in instance granularity.

SwissLog first obtains a set of anomalous instances $A = \{ID1, ID2, ...\}$ in the $\mathcal{R}_i$ graph. We name the node with zero in-degree as root and nodes with zero out-degree as leave nodes. SwissLog scans all *ID* in set A: i) if *ID* is a leaf node, SwissLog stops searching and return the anomalous *ID*. ii) if *ID* is not a leaf node, SwissLog searches its children nodes and check if it is anomalous. If yes, SwissLog removes the *ID* out of A. Otherwise, continue searching. SwissLog eventually returns the set of anomalous instances, thus achieving the instance-level anomaly localization.

4 Experimental Evaluations

In this section, we evaluate the effectiveness and robustness of SwissLog for diverse anomalies by answering the following questions:

- **RQ1**: How effective and robust is the proposed log parser?
- **RQ2**: How efficient is the proposed online log parser?
- **RQ3**: How effective is the BERT encoder on anomaly detection? Do other log parsers perform as well as the proposed log parser using BERT encoder?
- **RQ4**: How robust is SwissLog on those log data with changing events?
- **RQ5**: Can SwissLog detect log time interval changes? How sensitive is SwissLog to log time deviations?
- **RQ6**: How much overhead is introduced by the anomaly detection part in both offline phase and online phase?
- **RQ7**: How to localize anomalies using SwissLog?

All experiments in this paper are conducted on a server equipped with two 24-core CPU, 128GB RAM, and one NVIDIA GeForce GTX 1080 Ti GPU.

4.1 Experiment Setups

In this paper, we evaluate the proposed approach on the following datasets.

4.1.1 Test Environment Setup

Hadoop [38] is a well-known open-source and task-based software for reliable, scalable, distributed computing. IDs inside Hadoop help developers to search the execution paths. Our testbed built on Hadoop (version 3.1.4) is a 5-node cluster that contains 4 slaves and 1 master. The topology of our testbed is shown in Fig. 12. We run Hadoop jobs like WordCount (counting words in files) and Sort (sorting words in files) to generate workload to simulate the real-world traffic. According to our experiments, both WordCount and Sort are CPU-sensitive and IO-sensitive jobs, so they are easily affected by the testbed resource environment. Namenode runs on the master node, manages the file system namespace, and maintains the file system tree. It is mainly responsible for checking the allocation and completion of the block, collecting the states of each slave node after a certain interval, and outputting the log when the state changes. DataNodes in slaves read and write HDFS data to local file systems. Yarn is adopted in Hadoop, in which the master is resource manager, responsible for the resource (e.g. container) management and scheduling of the whole cluster; the slave is node manager, which is responsible for resource management and task start of a single node. We collect all logs on nodes by filebeats [39], which is a lightweight log shipper. Next, the log data are stored in an elasticsearch cluster.

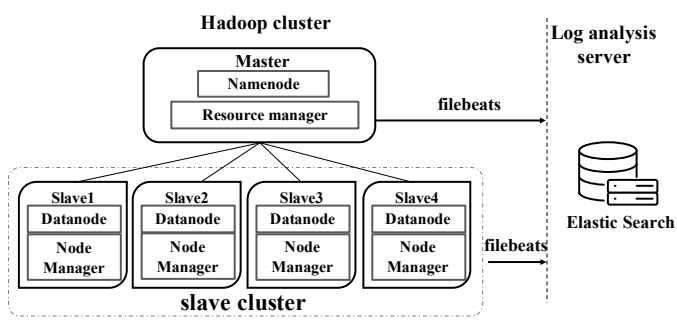


Fig. 12. The topology of Hadoop testbed

4.1.2 Chaos Engineering

Due to the limited amount of faults during normal execution, we choose to conduct chaos engineering on our testbed. Chaos engineering is a simple but effective way

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on September 03,2022 at 06:18:14 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3162857, IEEE Transactions on Dependable and Secure Computing

to generate faults. In our study, we use chaosblade [40], an open-source experimental injection tool developed by Alibaba, to create chaos. Our study injects faults both proactively and passively.

TABLE 1
Injected faults and related methods, details and their impacts

Faults Type	Methods	Seq	Perf
Drop heartbeat	Passive	✓	✗
Packet receive delay	Active	✓	✓
Block receive delay	Active	✓	✓
CPU full load	Passive	✓	✗
JVM CPU full load	Passive	✓	✓
Network delay	Passive	✓	✓
Network drop	Passive	✓	✗
Disk read burn	Passive	✓	✗
Disk write burn	Passive	✓	✗

We inject nine types of faults and present their information in Tab. 1. The column "Seq" and "Perf" means that the injected fault caused anomalous log changes in sequence or anomalous log time interval changes. For example, we increase CPU usage to 100%, which is in a passive way to generate chaos. Such CPU full load only generates abnormal log sequences but no log time interval anomalies. The other example is a packet delay, which is injected in an active way. We delay the call of the packet receive function, thus showing a high latency in receiving data. Such delays can be observed as log time interval anomalies. We ran Hadoop in different time periods and collected data for eleven days. During execution, we randomly inject various types of faults in Tab. 1 into Hadoop to create chaos, and each fault lasts five minutes, resulting in generating error log messages.

4.1.3 Anomaly Labeling

We collect log messages from different components in Hadoop, including datanode logs, namenode logs, and so on. Here we select the crucial IDs as examples to correlate logs, namely block ID. We then select a labeling strategy to label log sequential anomalies and log time interval anomalies.

TABLE 2
Detected true problems in Hadoop

Anomaly description	Occurrences
Replica already exist exception	7983
Fail to transfer block	6294
IO operation got exception	1464
Premature EOF from inputStream	591
Socket timeout exception	134
Closed by interrupt exception	115
Pending reconstruction monitor time out	21
Replica not found	9
Join on writer thread time out	4
Fail to delete replica, replicaInfo not found	5
File not found, blockid is not valid	1

Labeling log sequential anomalies. Range-based labeling, namely labeling all log samples during the failure time as anomalies, is a widely used way to distinguish injected anomalies in metrics. But in log messages, anomalies are not always reflected in all injected faults. For example, we have injected CPU full load into Hadoop. It is intuitive to observe that CPU usage climbs up to 100% during the injection. But due to the CPU scheduling mechanisms, not all applications are affected during the injection. So we carefully find 11 true problems in HDFS (Tab. 2) that occur after chaos engineering and manually label them as anomalies. For example, network delay causes anomaly "Fail to transfer block * to *" which totally occur 6,294 times in our collected datasets. We label all blocks containing anomalies in Tab. 2 as log sequential anomalies.

Labeling log time interval anomalies. We calculate time differences ΔT for all blocks and find that receiving block operation is easily affected by active injection. So we label those blocks that are obviously slower than others as log time interval anomalies.

4.1.4 Datasets

TABLE 3
The Details of Log Datasets

Log Type	#Messages	#Temp	Seq	Perf
HDFS [41]	11,175,629	30	✓	✗
Blue Gene /L [42]	4,747,963	377	✓	✗
Android [41]	30,348,042	76,923	✗	✗
Hadoop-blk	2,949,569	109	✓	✓

Real-world Datasets. Logpai [43] is a log parser benchmark that adopts 16 real-world log datasets ranging from distributed systems, supercomputers, operating systems, mobile systems, server applications, to standalone software including HDFS, Hadoop, Spark, Zookeeper, BGL, HPC, Thunderbird, Windows, Linux, Android, HealthApp, Apache, Proxifier, OpenSSH, OpenStack, and Mac. The above log datasets are provided by LogHub [31]. Each dataset contains 2,000 log samples with its ground truth tagged by a rule-based log parser. Besides sampled datasets, we select datasets collected from three representative systems to evaluate the proposed approach. The details are shown in Tab. 3.

HDFS log dataset is collected from a cluster on Amazon EC2 platform with 203 nodes [4], containing 11,175,629 raw log messages. The abnormal behaviors in HDFS were manually labeled by studying HDFS code and by consulting with Hadoop experts, including sequential order anomalies like the anomaly "Replica immediately deleted" and some exception logs like "Receive block exception". More details refer to the original paper [4].

BGL dataset is a supercomputing system log dataset collected by Lawrence Livermore National Labs (LLNL) [44]. The anomalies in BGL are manually determined by its system administrators. The log messages in these anomalies probably contain the description of exceptions. A log example is that "ciod: Error creating node map from file [...]". More details refer to the original paper [44].

The Android dataset provided by Loghub [31] records Android framework states and we only use the Android dataset to evaluate the log parser part.

We also collect a new log dataset from a 5-node Hadoop cluster deployed in our testbed, and details are shown in (Sec. 4.1.3). Dataset grouped by block ID is named as Hadoop-blk. Hadoop-blk contains 2,949,569 raw log messages with sequential anomalies labels and log time interval anomaly labels. We use the Hadoop-blk dataset to confirm the ability of SwissLog to detect log sequential anomalies and log time interval anomalies.

1545-5971 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on September 03,2022 at 06:18:14 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3162857, IEEE Transactions on Dependable and Secure Computing

Synthetic HDFS Data. To evaluate SwissLog, we synthesize new test datasets by simulating the real-world situation discussed in Sec. 2. Two possible types that may occur are injected in HDFS log data illustrated as below:

- Changing events. Only unimportant words are inserted or removed, without changing the key meaning of sentences. Therefore, the labels of changing log data stay unchanged. We apply this injection with a specific ratio ranging from 5% to 30% to the original HDFS log data. Also, we tag the changed log message as a new log template key.
- Log time interval anomalies. Only those latent performance issues that do not change the log sequence order are considered. We keep the same log sequence order and apply the time interval latency injection to mimic CPU hog, memory hog, disk write burn, and network delay with a ratio 5% into those logs whose original time interval is less than 2. We label the injected sessions as log time interval anomalies.

For simplicity, we name the dataset injected with changing events as *TestingEvent* and log time interval anomalies as *TestingPerf*.

4.2 Experiment Evaluations

4.2.1 Evaluation Metrics

We leverage the widely used metrics, namely *Precision*, *Recall*, and *F1-score* to measure the effectiveness of anomaly detection in SwissLog. Besides, the parsing accuracy (PA) metric is introduced to qualify the effectiveness of an automated log parser. Compared to previous metrics, evaluation using PA is more rigorous because partially matched templates are also considered incorrect. The detailed definitions of them are as follows, where TP, FP, FN represent True Positive, False Positive, and False Negative respectively.

- **Parsing Accuracy**: $PA = \frac{count(correct\ event\ ID\ group)}{count(all\ event\ ID\ group)}$. The ratio of correctly parsed log messages over the total number of log messages.
- **Precision**: $P = \frac{TP}{TP+FP}$. The percentage of correctly detected anomalies amongst all detected anomalies.
- **Recall**: $R = \frac{TP}{TP+FN}$. The percentage of correctly detected anomalies amongst all real anomalies.
- **F1-Score**: $F1 = \frac{2*P*R}{P+R}$. The harmonic mean of Precision and Recall.

4.2.2 Implementation and Parameters Setting

6,000 normal and 6,000 abnormal blocks from real-world datasets are randomly sampled for training. The neural network is trained using Adam optimizer [37]. We use a weight decay of 0.0001 and set the initial learning rate to 0.001. We set the hidden dim to 128. The training epoch is 30 and the mini-batch size is set to 32. We use cross-entropy as the loss function. We implement SwissLog with Python 3.7, Pytorch 1.3.

4.3 RQ1: The Effectiveness and Robustness of Log Parser

Our dictionary-based log parser requires a small size and effective dictionary. Developers can construct a dictionary from public corpora. Here we select one open-source corpus including 5.2 million sentences, which is accessible on [30]. After splitting this corpus with the space delimiter, we collect 588,054 distinct words. Noting that not every occurred word is valid (e.g., location name), we set an occurrence threshold to filter common valid words. We first conduct the statistics on these distinct words and observe that those words only occur once or twice among these distinct words occupy around 62%. After multiple trials, we set a relatively excellent threshold that hits the balance between efficiency and effectiveness. The dictionary finally remains only 18,653 common words. Our experiments confirm that such a dictionary from a public corpus can cover almost all the valid words in logging systems. In the evaluation, we will use these 18,653 common words as the dictionary D to recognize valid words.

To answer RQ1, we utilize a sampled dataset and a large dataset to figure out the effectiveness and robustness of SwissLog. The sampled dataset is quick and effective to test the effectiveness and robustness of log parsers. LogPai [43] provide an easy-to-run benchmark spanning 16 datasets from different systems and implement 14 log parsers including offline log parsers (i.e., SLCT, AEL, IPLoM, LKE, LFA, LogSig, LogCluster, LogMine, and MoLFI) and online log parsers (i.e., SHISO, LenMa, Spell, Drain), which is open-source on Github [2]. We evaluate the effectiveness and robustness of SwissLog on the LogPai benchmark and additionally evaluate Logram [3] [45], which also works based on dictionary. The results are shown in Tab. 4. Due to the limited space, we only present the state-of-the-art (SOTA) result in LogPai (i.e, the best score of the specific dataset shown in the LogPai benchmark [43]). In particular, the better result among SOTA in LogPAI, Logram, and SwissLog is highlighted in bold font with a gray block.

Overall, we observe that SwissLog shows almost the best accuracy in all datasets including the acceptable accuracy in Mac logs and comparable good accuracy in HPC logs. It also presents the excellent robustness of our dictionary-based log parser. Even more, SwissLog can parse HDFS, BGL, Windows, Apache, OpenSSH datasets with 100% accuracy. Note that we only utilize 2,000 samples for testing, thus a 100% accuracy is possible to achieve. The average of SwissLog is up to 0.962, which is much more than other log parsers by 10%. Indeed, parsing accuracy evaluates the ratio of correctly parsed groups but does not evaluate the accuracy of variables extraction. Hence, we manually check the parsed templates and observe that almost all variables in groundtruth templates can be recognized in SwissLog. Logram applies the n-gram dictionaries and automatically sets the threshold for different datasets. But the results show that it achieves a relative log accuracy among these datasets because it tends to overparse events (e.g., templates are full of variables). From this remarkable result, we can indicate that the dictionary-based method is close to the visual reflection of humans, thus the better results.

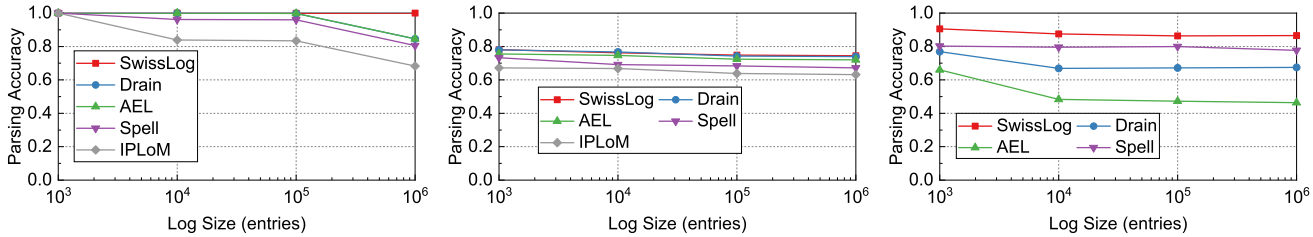
However, SwissLog shows an unsatisfactory performance on the Mac logs. Consider three groundtruth templates of Mac logs shown in Fig. 15. Kernel records the

2. https://github.com/logpai/logparser
3. https://github.com/BlueLionLogram/Logram

1545-5971 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on September 03,2022 at 06:18:14 UTC from IEEE Xplore. Restrictions apply.

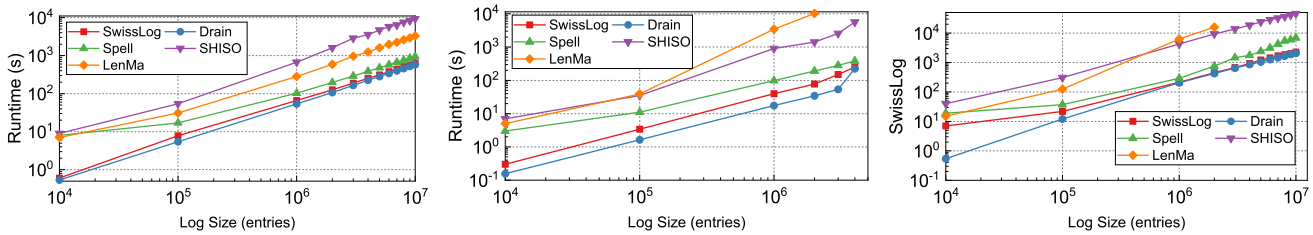
TABLE 4
Comparisons among different log parsers of Parsing Accuracy on Different Log Datasets

Dataset		HDFS	Hadoop	Spark	Zookeeper	BGL	HPC	Thunderbird	OpenStack	Mac
Accuracy	**SwissLog**	**1.000**	**0.992**	**0.997**	**0.985**	**0.970**	0.910	**0.992**	**1.000**	0.840
	SOTA in LogPAI	**1.000**	0.957	0.994	0.967	0.963	0.903	0.955	0.871	**0.872**
	Logram	0.809	0.451	0.357	0.724	0.587	**0.911**	0.554	0.816	0.568
Dataset		Windows	Linux	Andriod	HealthApp	Apache	Proxifier	OpenSSH	Average	
Accuracy	**SwissLog**	**1.000**	**0.869**	**0.954**	**0.901**	**1.000**	**0.990**	**1.000**	**0.962**	
	SOTA in LogPAI	0.997	0.701	0.919	0.822	**1.000**	0.967	0.925	0.865	
	Logram	0.0.694	0.140	0.906	0.267	0.313	0.0.504	0.611	0.575	



(a) HDFS (b) BGL (c) Android

Fig. 13. Comparisons among different log parsers of parsing accuracy on different volumes of logs



(a) HDFS (b) BGL (c) Android

Fig. 14. Comparisons among different log parsers of runtime on different volumes of logs

PM response took <*> ms (<*>, powerd)
PM response took <*> ms (<*>, QQ) → <*>
PM response took <*> ms (<*>, WeChat)

Fig. 15. An example of Mac log templates

response time of different tasks. They are clustered into one template in SwissLog since "powerd" and "wechat" are non-valid words. At first glance, it is not hard for us to classify these three templates into one category, in that the difference is the service name, *powerd* (power management daemon process), *QQ*, *WeChat*. In this case, according to the different characteristics of tasks, they should be divided into three templates. However, in other cases, where we can simply treat them as the variable part, three templates should be merged into one template. We need to adjust the dictionary used in preprocessing step according to different template discriminants. In this case, we can add the word "powerd" and "wechat" to the dictionary, then SwissLog will not merge these templates.

Besides the sampled dataset, we further evaluate SwissLog towards three large datasets. The comparison of parsing accuracy is shown in Fig. 13. The horizontal axis represents log size which increases in logarithm and the vertical axis denotes parsing accuracy over different amounts of log statements. We compare SwissLog with the best four log parsers in LogPai benchmark [43], namely Drain, AEL, Spell, and IPLoM. It is worth noting that IPLoM is excluded from the Android dataset since it consumes too much time to finish parsing the data.

From Fig. 13, we observe that the effectiveness of SwissLog outperforms other log parsers on the HDFS and Android datasets while it is slightly higher than others on the BGL dataset. When the volume of logs increases, the parsing accuracy of SwissLog drops very slightly. Hence, we can indicate from the above results on the sampled and large datasets that SwissLog has excellent effectiveness and robustness on different volumes and different types of log entries.

Next, we need to figure out the effectiveness of log parsers towards precise downstream anomaly detection task (the anomaly detection experimental results refers to RQ2). We select the top 2 log parsers in LogPai benchmark [43], namely AEL and Drain, to parse the HDFS dataset in Tab. 3 into log templates. Then these log templates are fed to BERT to acquire the semantic embedding.

TABLE 5
Ablation Results on Log Parsers

LogParser	Precision	Recall	F1-Score
Drain	0.95	0.96	0.96
AEL	0.96	0.97	0.97
SwissLog	**0.97**	**1.00**	**0.99**

Tab. 5 presents the ablation results on different log parsers. Compared with other log parsers, SwissLog achieves the best score of 0.99 in *F1-Score*. The biggest difference between SwissLog and other approaches is that SwissLog extracts more valuable valid words which provide wealthy information for sentence embedding. Drain and AEL tend to overparse logs as variable parts so that the log templates lose lots of valuable semantic information. For example, "... Exception: Interrupted receiveBlock" and "... Exception: Broken pipe" are pieces of two log messages. Since Drain merges similar templates in a log bin, Drain is likely to merge these two log templates with different

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on September 03,2022 at 06:18:14 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3162857, IEEE Transactions on Dependable and Secure Computing

meanings into one log template (i.e., Exception: *) as long as their similarity is higher than the similarity threshold. AEL also faces the same challenges. It tends to misparse two log messages like "PacketResponder ..." and "writeBlock ..." into one log template when it merges log messages within the categorized log bin.

The intuition behind the dictionary-based log parser is that constant parts probably contain valid words while variable parts are likely to involve non-valid words. So SwissLog cannot parse correctly if the variable parts always contain valid words. For example, if the value of variable *username* only contains 'guest' and 'user', SwissLog could mistake them into two templates. For further improvement, we can introduce a *human-in-the-loop* procedure to merge these templates into one template with human supervision. More specifically, highly similar templates are recommended to operators to decide to merge or not.

4.4 RQ2: The Efficiency of Online Log Parser

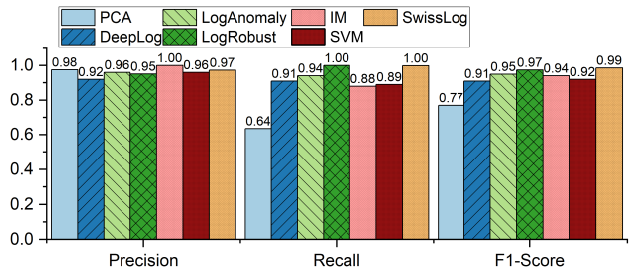
Efficiency plays an important role in online log parser. The speed to parse logs should be faster than the speed that the system generates logs. To evaluate the efficiency of our online version log parsers, we compare other online log parsers (i.e., Drain, Spell, SHISO, LenMa) on three large datasets involving HDFS, BGL, and Android. Large log sizes can test the scalability of online log parsers and large log template sizes in Android can figure out if template size will affect the parsing efficiency. All log parsers including SwissLog are fed with log messages one by one, and they incrementally update templates as the log size grows.

The results of efficiency evaluation are presented in Fig. 14. The horizontal axis refers to log size in logarithm scale and the vertical axis refers to the time cost in different online log parsers. Since LenMa costs too much time in the large-scale BGL dataset and Android dataset, the results of LenMa are not shown. We can observe that SwissLog achieves a relatively excellent efficiency and Drain spent the least time on parsing HDFS and BGL. Both of SwissLog and Drain follow a near-linear growth rate as the log size grows. The reason is that the time complexity of Drain is $O((l + cs)n)$, where l is the depth of DAG. The fixed-depth prefix tree largely advances the efficiency but also leads to accuracy degradation. Indeed, a small gap in efficiency can be ignored in practice. Therefore, we can confirm that SwissLog can achieve relatively high efficiency in practice.

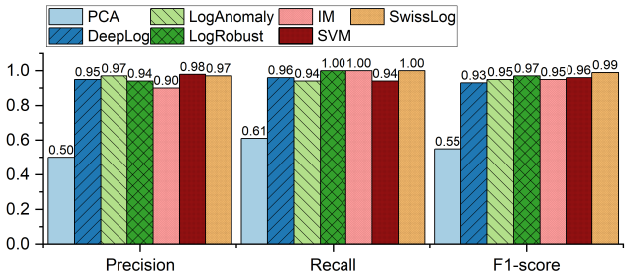
4.5 RQ3: The Effectiveness of Semantic-based Model

In this part, we intend to evaluate the effectiveness of the semantic-based model in SwissLog. We specially select two kinds of datasets: with IDs (HDFS) and without IDs (BGL). We construct *sessions* by correlating logs with IDs if so, while using a sliding window if not. Consequently, we conduct experiments on original datasets HDFS and BGL with two kinds of labels, namely normal sequence, and sequential log anomalies. For the HDFS dataset, we correlate log messages with the same block id named *session* in advance. For the BGL dataset, we apply a sliding window with a length of 20 entries to construct a sequence *session*.

We adopt the proposed log parser of SwissLog to extract log templates. Then we employ two supervised method (i.e., LogRobust [10], SVM), three unsupervised methods (i.e., IM [5], DeepLog [7], LogAnomaly [9], PCA [4]), and SwissLog to detect anomalies. DeepLog [7] is a log key-based anomaly detection model and it leverages LSTM to learn the pattern of normal sequence. LogRobust [10] encodes log templates using Word2Vec and leverages Attn-based Bi-LSTM to learn and detect anomalies. LogAnomaly [9] accurately extracts the semantic and syntax information from log templates. IM [5] mines the invariants among log events from log event count vectors and identifies those log sequences that violate the invariant relationship as anomalies.



(a) HDFS (with IDs)



(b) BGL (without IDs)

Fig. 16. Comparisons among different anomaly detection approaches on HDFS (with IDs) and BGL (without IDs)

The comparison results of evaluation metrics *Precision/Recall/F1-Score* on different datasets are shown in Fig. 16. A lower *Precision* means that more anomalies cannot be detected while a higher *Recall* means more manual works. Compared with other competitive approaches, SwissLog with Attn-based Bi-LSTM achieves a better balance between *Precision* and *Recall*. It achieves a very high *F1-Score* up to 0.99 and 0.99 in HDFS and BGL, respectively. This is because Attn-based Bi-LSTM can capture the log sequential order precisely so that log sequential order anomalies like "replica immediately deleted" in the HDFS dataset can be detected precisely. BERT further provides enriched semantic information so as to detect those anomalies that contain exception messages. The detection capability of SwissLog covers almost all anomalies mentioned in HDFS and BGL datasets.

4.6 RQ4: The Robustness on Changing Log Data

As we discussed in Sec. 2, changing events inevitably occur in modern software systems under active development and maintenance. In this part, we evaluate the effectiveness of the semantic-based anomaly detection model on changing log data. Two competitive approaches, namely DeepLog and LogRobust are chosen as the baselines. DeepLog leverages log key to identify templates while SwissLog and LogRobust utilize sentence embedding. We use the model

1545-5971 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on September 03,2022 at 06:18:14 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3162857, IEEE Transactions on Dependable and Secure Computing

trained by the original dataset to predict the *TestingEvent* dataset. Since the injected events are changed, we tag them as new templates in DeepLog. The experimental results on *TestingEvent* are shown in Fig. 17.

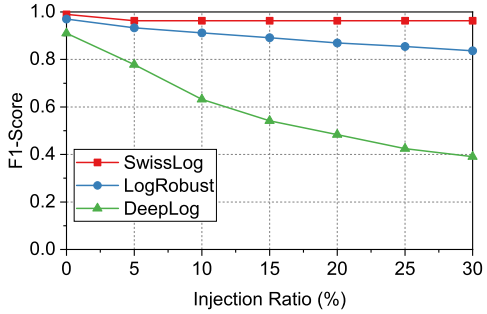


Fig. 17. F1-Score on the dataset TestingEvent

In Fig. 17, the horizontal axis denotes the injection ratio and the vertical axis denotes the *F1-Score* of different anomaly detection models. We can observe that the *F1-Score* of SwissLog, LogRobust, and DeepLog with injection ratio 5% are 0.96, 0.93, 0.78 respectively. Semantic-based models (i.e., LogRobust, SwissLog) achieve a better *F1-Score* than log key-based models (i.e, DeepLog). The reason is that the log key-based model treats those changing events as new templates, which probably results in false alarms. Semantic-based models utilize sentence embedding to encode templates, which extend the 1-dimension sentence array to a 2-dimension sentence embedding matrix. Hence, sentence embedding brings robustness to changing log data as expected. As the injected ratio increases, the *F1-Score* of LogRobust starts to drop while SwissLog still maintains a high *F1-Score*. For example, under the injection ratio of 30%, the *F1-Score* of SwissLog is higher than 0.9, but LogRobust can only achieve 0.84. Compared to a 2-dimension unordered sentence embedding array in LogRobust, BERT encoders in SwissLog capture the contextual information in templates and encode changing log data with similar vectors. Hence, SwissLog with BERT encoders is almost not affected by changing events, showing good robustness.

4.7 RQ5: The Effectiveness and Sensitivity of Time Embedding

SwissLog targets diverse anomalies including log sequential anomalies and log time interval anomalies. To answer RQ5, we need to verify the effectiveness of time embedding in SwissLog using synthetic HDFS dataset *TestingPerf* and real-world dataset Hadoop-blk. We compare SwissLog with those models using different time embedding: 1) 1-dimension raw time (i.e., Raw time in Tab. 6). 2) The mean of $\boldsymbol{E}_{semantic}$ and $\boldsymbol{E}_{time}$ (i.e., Mean in Tab. 6). The comparison among them is shown in Tab. 6. We can observe that SwissLog with the proposed time embedding achieves 0.92 in *Precision* while others are less than 0.7. Raw time shows the worst results 0.70 in *F1-Score* since it only contains raw information without preprocessing. The mean embedding loses part of semantic information and temporal information, therefore it obtains 0.76 in *F1-Score*. According to the result, the effectiveness of time embedding is confirmed.

Intuitively, the higher dimension d of time embedding may lead to the accurate prediction at the cost of training

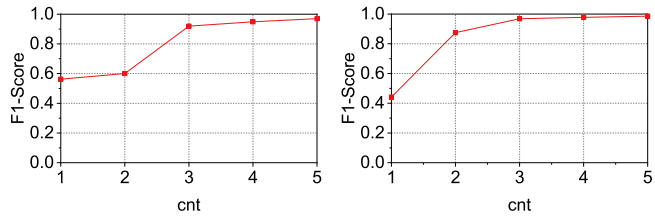
TABLE 6
Results on different operation for time embedding

Time embedding	Precision	Recall	F1-Score
Raw time	0.68	0.71	0.70
Mean	0.67	0.94	0.76
SwissLog (d=768)	**0.92**	**0.99**	**0.95**

TABLE 7
Results on different dimension for the proposed time embedding

Model	Precision	Recall	F1-Score
SwissLog (d=2)	0.95	0.99	0.97
SwissLog (d=5)	**0.98**	0.99	**0.99**
SwissLog (d=10)	0.96	**1.00**	0.98
SwissLog (d=50)	0.95	**1.00**	0.97
SwissLog (d=100)	0.97	**1.00**	0.98
SwissLog (d=768)	0.92	0.99	0.95

and predicting time. Hence, we attempt to figure out the impact of dimension d and hit a good balance between effectiveness and efficiency. We conduct experiments on dataset HDFS under different dimensions d of time embedding and present the results in Tab. 7. We can observe that dimension d has a very limited impact on the results. All models with different dimension achieve more than 0.95 in *Precision* and almost 1.00 in *Recall*. Yet, as the dimension d grows, the elapsed time will also increase. Since our goal is to detect anomalies as much as we can, we select dimension $d = 5$ as our model parameter where the model can achieve 0.99 in *F1-Score*. When applying the time embedding for logs from a new system, we also recommend to conduct experiments on a small set of new logs and derive the best dimension d.



(a) Injected seconds=1 (b) Injected seconds=2

Fig. 18. F1-Score on TestingPerf

The sensitivity to time violation is crucial in detecting log time interval anomalies. Here we inject additional 1-second and 2-second latency (i.e., the latency between two consecutive log messages) to mimic log time interval anomalies, where the count of injected faults ranges from 1 to 5. As shown in Fig. 18, when the injected number of log time interval anomalies and the injected latency increase, SwissLog predicts more accurately in log time interval anomalies. It is reasonable that SwissLog shows an unsatisfactory score in count 1 and 2. If SwissLog is sensitive to all time violations, it will report a large volume of alarms resulting in unnecessary effort to analyze anomalies. When the injected number is bigger than 2, SwissLog can achieve a high *F1-Score* which is more than 0.9. The results reveal the sensitive response of SwissLog to time violation.

To further confirm the effectiveness of time embedding in practice, we conduct experiments on a real-world dataset Hadoop-blk. Here we label sequential anomalies and log time interval anomalies as anomalies in other methods. We can observe from Fig. 19 that SwissLog is able to detect diverse anomalies especially for log time interval anomalies and achieves the highest *F1-Score* 0.97. Other methods, namely PCA, LogRobust, SVM, DeepLog and IM, can only

1545-5971 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on September 03,2022 at 06:18:14 UTC from IEEE Xplore. Restrictions apply.

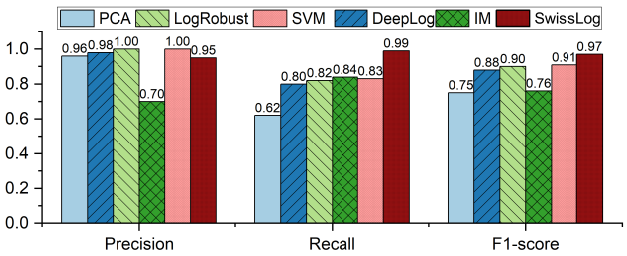


Fig. 19. The Results of anomaly detection on real-world datasets Hadoop-blk

detect sequential anomalies, so they show poor performance on identifying log time interval anomalies.

4.8 RQ6: The overhead of Anomaly Detection Part

TABLE 8
The offline training time and online inference time of HDFS dataset among different models

Model	Training time (ms/seq)	Inference time (ms/seq)
SwissLog	41.67	10.64
LogRobust	9.63	4.48
DeepLog	2.84	2.36
IM	14.85	$2.9 * e^{-5}$
PCA	$8.64 * e^{-4}$	$8.19 * e^{-2}$
SVM	$6.86 * e^{-3}$	$2.16 * e^{-3}$

Apart from the experiments on the effectiveness and robustness of SwissLog, we additionally evaluate the training time of the anomaly detection model in the offline phase and the inference time in the online phase on the HDFS dataset. We adopt the metric milliseconds per sequence (ms/seq) to measure the runtime of SwissLog.

We present a runtime comparison results of anomaly detection part among deep learning-based model (DL, i.e., SwissLog, LogRobust, DeepLog) and non deep learning-based model (non-DL, i.e., IM, PCA, SVM) in Tab. 8. The training time refers to the training cost in the offline phase while the inference time means the detection time in the online phase. Since DL model needs to train via multiple epochs, we only show the runtime of one epoch here. We can observe that DL models are much slower than non-DL models because of the high complexity of DL models and the insufficient resources of our experimental environments. The training time of the proposed model is 4x higher than others. It is acceptable since the overhead in the offline training phase will not affect the runtime of anomaly detection in the online phase. The online inference time of the proposed model is around 2x higher than LogRobust. Reasonably, our model is a three-classification task while LogRobust is a two-classification task, so learning and predicting an additional class may introduce more time.

In practice, a large-scale system can generate more than 50 GB of logs per hour [3], namely 120 million log lines (i.e., 6 million log sequences if one sequence contains 20 log lines). It requires around 1000 minutes according to Tab. 8 due to the limited computation power and device memory under the environment of one Nvidia GeForce GTX 1080 Ti GPU. The online detection time largely depends on the provisioned resources, and we have additionally performed SwissLog in parallel under the environment with 4 Nvidia Tesla V100 GPUs. The results show that we only spend around 50 minutes processing such a huge volume of log data.

4.9 RQ7: Detect and Localize Anomalies in Hadoop

To further illustrate the process of anomaly detection and localization, we apply SwissLog to give a case study in a real-world system, Hadoop. The $\mathcal{R}$ graph of Hadoop cluster is shown in Fig. 20. There are totally six types in Hadoop. Relation between types is linked with arrow lines, where the relationship 1:1 is colored by a red dotted line and the relationship 1:n is colored by a black line. For example, an application generates multiple blocks, so their relationship is 1:n. In our study, the relationship 1:1 is removed.

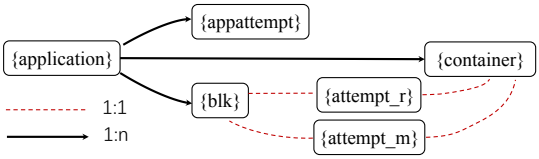


Fig. 20. The subgraph $\mathcal{R}$ graph of Hadoop

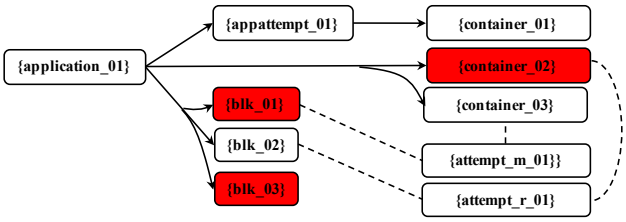


Fig. 21. The simplified $\mathcal{R}_i$ graph of Hadoop

Next, we give an anomaly case in the real world. First, a set of incoming logs gradually initiates the $\mathcal{R}$ graph as $\mathcal{R}_i$ graph. In practice, application_01 totally generates 14 blocks and 16 containers. Due to the limited space, we only show the simplified $\mathcal{R}_i$ graph including 3 blocks, 3 containers, one appattempt, one attempt_r, and one attempt_m in Fig. 21.

First, SwissLog detects anomalies in log data grouped by block ID and labels blk_01 as an anomalous block. After that, an exception is caught in blk_03 so SwissLog also labels blk_03 as an anomalous block. Finally, a killing signal is caught by SwissLog in container_02, which is tagged as a faulty container. The anomalous instances set is, therefore, $A = \{blk\_01, blk\_03, container\_02\}$ and all the faulty instances in A are colored by red in Fig. 21. Then we apply the heuristic searching algorithm to the $\mathcal{R}_i$ graph. Instance node blk_01, blk_03, and container_02 are leaf nodes in $\mathcal{R}_i$ graph, consequently, SwissLog reports the anomalous instances blk_01, blk_03, and container_02. So we can localize anomalies in instance-level in Hadoop (blk and container in this case). Such information can largely help developers to analyze the root cause. For example, we can observe that not all blocks and containers encounter anomalies here. We can indicate that it might be a node failure in slave clusters. With the correlated logs of blk_01, blk_03, and container_02 in SwissLog, we can find that a network exception occurs in slave 2. Hence, SwissLog can help us quickly localize and analyze root causes when meeting anomalies.

5 Discussion

Threats To Validity. We discuss threats in three aspects: 1) The log parser of SwissLog is based on a dictionary. However, 18,653 common words in our filtered dictionary cannot cover all of the valid words in logs, especially those words

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on September 03,2022 at 06:18:14 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3162857, IEEE Transactions on Dependable and Secure Computing

of domain knowledge. A portion of terminology, such as "Hadoop", is not included in them. It is quite challenging to find a suitable dictionary for all software systems generally. Instead, customizing a particular dictionary for a software system accordingly is a better choice. For example, it is easy to collect specific valid words by analyzing a small set of log data in the offline phase. 2) Unstable performance in practice. Although the time interval of the normal sequence seems unchanged, different tasks still have various execution times under different environments. Developers need to retrain the model once the system environment changes. 3) Unstable relation construction. We utilize Stitch to construct relations in the Hadoop case, but it shows unstable performance in other systems. The manual check is required to ensure the accuracy of $\mathcal{R}$ graph.

Limitations. We discuss two major limitations here: 1) SwissLog is limited to the anomalies manifested in log data. On one hand, SwissLog cannot detect those anomalies (e.g., high CPU usage) exposed on KPI data instead of log data. On the other hand, SwissLog uses log time interval change to dig out potential performance issues as many as possible, but may not all of them. Because not all performance anomalies are correlated with logs. 2) SwissLog is limited to those continually changing log statements while their key meanings stay unchanged. If the changed logging statement alternates its meaning or the incoming log message is quite different from the previous ones, the pre-trained SwissLog may fail to capture the meaning, leading to bad results. So in such cases, SwissLog should be retrained.

6 Related Work

Workflow Construction. Non-intrusive workflow construction is the mainstream way to reconstruct the workflow of a targeted system. CloudSeer [6] construct workflow and monitor it from interleaved logs. lprof [25] uses static analysis to find IDs that can identify logs of different requests from the same component. Stitch [26] also provides a non-intrusive and builds a dependency graph as a S^3 graph based on finding IDs in logs. The S3 graph presents the relationships between IDs such as one to one, one to more. Rather than focusing on the coarse-grained IDs, IntelLog [8] is able to provide abundant workflow information such as entity names to users. It utilizes NLP-based approaches to construct hierarchical workflow graphs to present the relation between entities. Besides leveraging log data, LRTrace [46] also profiles actual resource consumptions of an application at runtime in a fine-grained manner.

Log Parsing. Log parsing is the fundamental step of log analysis works that have been widely studied. Xu, et al. [4] and Nagappan, et al. [47] parsed logs by generating regular expressions based on source codes. However, not all projects are open-source online in practice. Moreover, existing log parsing approaches can be divided into several categories. 1) Similarity based clustering: LKE [48], LogSig [49], LogMine [50], SHISO [51] and LenMa [52] compute distances between two log messages or their signature and then cluster them based on similarity. Similarity based clustering methods generally require a preset threshold to determine whether the templates belong to one cluster. Parameter tuning is labor-intensive, time-consuming, and non-universe. 2) Frequency based clustering: a set of constant items occurs frequently in logs, so mining the frequency of items is a straightforward way to parse logs automatically. SLCT [53], LFA [54] and LogCluster [55] firstly record frequency of items and then group them into multiple groups. 3) Heuristics by searching tree: Drain [28] and Spell [27] utilize a tree structure to parse log into multiple templates. Drain leverages fixed-depth prefix tree to split log messages into different nodes. Each layer of the prefix tree is one feature extracted from logs such as the length feature, first token feature. 4) Dictionary-based: Logparse [56] and Logram [45] shows the possibility on parsing logs with dictionary. Both of them construct a dictionary from existing templates. Logparse learns the features of template words and variable words, then they build a word classifier to classify log messages. Logram leverages n-gram dictionaries instead of single tokens to split the constant n-gram and variable n-gram. 5) Deep learning based: NuLog [57] proposed a neural log parsing based on a self-attention model and formulates the parsing task as masked language modeling (MLM). However, deep learning based log parsing methods are inefficient. They require much more resources than other log parsers to execute.

Anomaly Detection. Existing anomaly detection approaches mainly focus on sequential log anomalies. They can be mainly separated into data mining methods and deep learning methods.

Data mining methods include supervised learning methods and unsupervised learning methods. 1) Supervised: by training labeled log data, supervised methods (e.g., decision tree [58], support vector machines [59], regression-based technique [60]) can learn the fixed pattern of different labeled log. Consequently, they generally achieve a higher score than unsupervised methods. But it is time-consuming to label a large volume amount of historical data for training. Moreover, they cannot detect a black swan, which may not be involved in historical data. 2) Unsupervised: unsupervised methods take unlabeled history data to train. This kind of method generally constructs a normal space and an abnormal space for normal sequence and abnormal sequence, respectively [4], [5]. The strength of unsupervised methods is unnecessary to label log data. But similar to supervised methods, a black swan is also hard to detect.

With the prevalence of deep learning, anomaly detection models based on deep learning are widely studied [7], [9], [10], [20]. Deep learning methods go through parsing log, model training, and model predicting. 1) Log key-based models: log key-based models first parse log statements into templates and tag them with log keys. Du, et al. [7] adopted LSTM while Vinayakumar et, al. [20] trained stacked-LSTM to model the sequential patterns of normal and abnormal sessions. However, when source codes update for a new version, the old-trained log key-based model will treat them as new templates which leads to unsatisfactory performance. 2) Semantic-based models: As log data contains wealthy semantic information of system states, NLP techniques are utilized to analyze log data. Meng, et al. [9] trained LSTM considering the synonyms and antonyms with word vectors. However, it also takes the log count vector as inputs that are not robust to the changing log data. Zhang, et al. [10] leveraged Attention-Based Bi-LSTM to detect anomalies.

1545-5971 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on September 03,2022 at 06:18:14 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3162857, IEEE Transactions on Dependable and Secure Computing

But Word2Vec and TF-IDF ignore the contextual information in sentences. In our work, we use BERT to capture the contextual semantic meaning in sentences.

7 Conclusion and Future Work

Aiming at tackling the challenges of complex log dependencies, changing events, and log time interval anomalies in practice, we propose SwissLog in this paper, a robust anomaly detection and localization tool for interleaved unstructured logs. SwissLog targets diverse anomalies including log sequential anomalies and log time interval anomalies, and achieves the instance-grained anomaly localization. We have conducted experiments on real-world datasets and synthetic datasets to evaluate the effectiveness, efficiency, and robustness of SwissLog. The results show that our approach outperforms others. In the future, we plan to apply SwissLog to more kinds of systems and design a flexible incremental updating mechanism to adapt to new log anomaly patterns.

Acknowledgment

The research is supported by the National Key Research and Development Program of China (2019YFB1804002), the Key-Area Research and Development Program of Guangdong Province (No. 2020B010165002), the National Natural Science Foundation of China (No. 61802448, No. U1811462), the Basic and Applied Basic Research of Guangzhou (No. 202002030328), and the Natural Science Foundation of Guangdong Province (No. 2019A1515012229). The corresponding author is Pengfei Chen.

References

[1] X. Li, P. Chen, L. Jing, Z. He, and G. Yu, "Swisslog: Robust and unified deep learning based log anomaly detection for diverse faults," in *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2020, pp. 92–103.

[2] "Alibaba cloud reports io hang error in north china," https://equalocean.com/technology/20190303-alibaba-cloud-reports-io-hang-error-in-north-china, 2019, [Online].

[3] H. Mi, H. Wang, Y. Zhou, M. R.-T. Lyu, and H. Cai, "Toward fine-grained, unsupervised, scalable performance diagnosis for production cloud computing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1245–1255, 2013.

[4] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *SOSP'09: Proc. of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*. ACM, 2009, pp. 117–132.

[5] J.-G. Lou, Q. Fu, S. Yang, J. Li, and B. Wu, "Mining program workflow from interleaved traces," in *SIGKDD'10: Proc. of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2010, pp. 613–622.

[6] X. Yu, P. Joshi, J. Xu, G. Jin, H. Zhang, and G. Jiang, "Cloudseer: Workflow monitoring of cloud infrastructures via interleaved logs," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 2, pp. 489–502, 2016.

[7] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *SIGSAC'17: Proc. of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 1285–1298.

[8] A. Pi, W. Chen, S. Wang, and X. Zhou, "Semantic-aware workflow construction and analysis for distributed data analytics systems," in *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*, 2019, pp. 255–266.

[9] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun *et al.*, "Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs." in *IJCAI'19: Proc. of the 28th International Joint Conference on Artificial Intelligence*, 2019, pp. 4739–4745.

[10] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li *et al.*, "Robust log-based anomaly detection on unstable log data," in *ESEC/FSE'19: Proc. of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 2019, pp. 807–817.

[11] L. Yang, J. Chen, Z. Wang, W. Wang, J. Jiang, X. Dong, and W. Zhang, "Plelog: Semi-supervised log-based anomaly detection via probabilistic label estimation," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 2021, pp. 230–231.

[12] D.-Q. Zou, H. Qin, and H. Jin, "Uilog: Improving log-based fault diagnosis by log analysis," *Journal of computer science and technology*, vol. 31, no. 5, pp. 1038–1052, 2016.

[13] S. Kabinna, C.-P. Bezemer, W. Shang, M. D. Syer, and A. E. Hassan, "Examining the stability of logging statements," *Empirical Software Engineering*, vol. 23, no. 1, pp. 290–333, 2018.

[14] C. Lou, P. Huang, and S. Smith, "Understanding, detecting and localizing partial failures in large system software," in *NSDI'20: Proc. of the 17th USENIX Symposium on Networked Systems Design and Implementation*, 2020, pp. 559–574.

[15] "Gocardless service outage on october 10th, 2017," https://gocardless.com/blog/incident-review-api-and-dashboard-outage-on-10th-october, 2017, [Online].

[16] "Office 365 update on recent customer issues," https://blogs.office.com/2012/11/13/update-on-recent-customer-issues/, 2017, [Online].

[17] "Google compute engine incident 17008," https://status.cloud.google.com/incident/compute/17008, 2017, [Online].

[18] "Twilio billing incident post-mortem: Breakdown, analysis and root cause." https://bit.ly/2V8rurP, 2013, [Online].

[19] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[20] R. Vinayakumar, K. Soman, and P. Poornachandran, "Long short-term memory based operation log anomaly detection," in *ICACCI'17: 2017 International Conference on Advances in Computing, Communications and Informatics*. IEEE, 2017, pp. 236–242.

[21] P. Huang, C. Guo, L. Zhou, J. R. Lorch, Y. Dang, M. Chintalapati, and R. Yao, "Gray failure: The achilles' heel of cloud-scale systems," in *HotOS'17: Proc. of the 16th Workshop on Hot Topics in Operating Systems*, 2017, pp. 150–155.

[22] P. Huang, C. Guo, J. R. Lorch, L. Zhou, and Y. Dang, "Capturing and enhancing in situ system observability for failure detection," in *OSDI'18: Proc. of the 13th USENIX Symposium on Operating Systems Design and Implementation*, 2018, pp. 1–16.

[23] D. J. Dean, H. Nguyen, X. Gu, H. Zhang, J. Rhee, N. Arora, and G. Jiang, "Perfscope: Practical online server performance bug inference in production cloud computing infrastructures," in *SOCC'14: Proc. of the ACM Symposium on Cloud Computing*, 2014, pp. 1–13.

[24] D. J. Dean, H. Nguyen, P. Wang, X. Gu, A. Sailer, and A. Kochut, "Perfcompass: Online performance anomaly fault localization and inference in infrastructure-as-a-service clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 6, pp. 1742–1755, 2015.

[25] X. Zhao, Y. Zhang, D. Lion, M. F. Ullah, Y. Luo, D. Yuan, and M. Stumm, "lprof: A non-intrusive request flow profiler for distributed systems," in *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, 2014, pp. 629–644.

[26] X. Zhao, K. Rodrigues, Y. Luo, D. Yuan, and M. Stumm, "Non-intrusive performance profiling for entire software stacks based on the flow reconstruction principle," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016, pp. 603–618.

[27] M. Du and F. Li, "Spell: Streaming parsing of system event logs," in *ICDM'16: Proc. of the 16th International Conference on Data Mining*. IEEE, 2016, pp. 859–864.

[28] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *ICWS'17: 2017 IEEE International Conference on Web Services*. IEEE, 2017, pp. 33–40.

[29] "wordninja," https://github.com/keredson/wordninja, 2021, [Online].

[30] "English corpus," https://storage.googleapis.com/nlp_chinese_corpus/translation2019zh.zip, 2021, [Online].

[31] S. H. Pinjia He, Jieming Zhu and M. R. Lyu, "Loghub: A large collection of system log datasets for ai-powered log analytics,"

1545-5971 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on September 03,2022 at 06:18:14 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3162857, IEEE Transactions on Dependable and Secure Computing

in *ESEC/FSE'19: Proc. of the 27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 2019.

[32] W. Meng, Y. Liu, F. Zaiter, S. Zhang, Y. Chen, Y. Zhang, Y. Zhu, E. Wang, R. Zhang, S. Tao *et al.*, "Logparse: Making log parsing adaptive through word classification."

[33] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *ICML'14: Proc. of the 31st International Conference on Machine Learning*, 2014, pp. 1188–1196.

[34] H. Xiao, "bert-as-service," https://github.com/hanxiao/bert-as-service, 2018.

[35] "Bert pretrained models," https://github.com/google-research/bert, 2021, [Online].

[36] Y. Li, N. Du, and S. Bengio, "Time-dependent representation for neural event sequence prediction," *arXiv preprint arXiv:1708.00065*, 2017.

[37] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[38] "Hadoop," https://hadoop.apache.org/, 2021, [Online].

[39] "filebeats," https://www.elastic.co/beats/filebeat, 2021, [Online].

[40] "chaosblade," https://github.com/chaosblade-io, 2021, [Online].

[41] "Loghub datasets," https://zenodo.org/record/3227177, 2021, [Online].

[42] "Bluegene/l message types," https://www.usenix.org/cfdr-data#hpc4, 2019, [Online].

[43] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, "Tools and benchmarks for automated log parsing," in *ICSE(SEIP)'19: Proc. of the 41st International Conference on Software Engineering: Software Engineering in Practice*. IEEE Press, 2019, pp. 121–130.

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

[45] H. Dai, H. Li, C. S. Chen, W. Shang, and T.-H. Chen, "Logram: Efficient log parsing using n-gram dictionaries," *IEEE Transactions on Software Engineering*, 2020.

[46] A. Pi, W. Chen, X. Zhou, and M. Ji, "Profiling distributed systems in lightweight virtualized environments with logs and resource metrics," in *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*, 2018, pp. 168–179.

[47] M. Nagappan, K. Wu, and M. A. Vouk, "Efficiently extracting operational profiles from execution logs using suffix arrays," in *ISSRE'09: Proc. of the 20th International Symposium on Software Reliability Engineering*. IEEE, 2009, pp. 41–50.

[48] Q. Fu, J.-G. Lou, Y. Wang, and J. Li, "Execution anomaly detection in distributed systems through unstructured log analysis," in *ICDM'09: Proc. of the 9th IEEE International Conference on Data Mining*. IEEE, 2009, pp. 149–158.

[49] M. Mizutani, "Incremental mining of system log format," in *SCC'13: 2013 IEEE International Conference on Services Computing*. IEEE, 2013, pp. 595–602.

[50] H. Hamooni, B. Debnath, J. Xu, H. Zhang, G. Jiang, and A. Mueen, "Logmine: Fast pattern recognition for log analytics," in *CIKM'16: Proc. of the 25th ACM International on Conference on Information and Knowledge Management*. ACM, 2016, pp. 1573–1582.

[51] K. Q. Zhu, K. Fisher, and D. Walker, "Incremental learning of system log formats," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 1, pp. 85–90, 2010.

[52] K. Shima, "Length matters: Clustering system log messages using length of words," *arXiv preprint arXiv:1611.03213*, 2016.

[53] R. Vaarandi, "A data clustering algorithm for mining patterns from event logs," in *IPOM'03: Proc. of the 3rd IEEE Workshop on IP Operations & Management*. IEEE, 2003, pp. 119–126.

[54] M. Nagappan and M. A. Vouk, "Abstracting log lines to log event types for mining software system logs," in *MSR'10: Proc. of the 7th IEEE Working Conference on Mining Software Repositories*. IEEE, 2010, pp. 114–117.

[55] R. Vaarandi and M. Pihelgas, "Logcluster-a data clustering and pattern mining algorithm for event logs," in *CNSM'15: Proc. of the 11th International Conference on Network and Service Management*. IEEE, 2015, pp. 1–7.

[56] W. Meng, Y. Liu, F. Zaiter, S. Zhang, Y. Chen, Y. Zhang, Y. Zhu, E. Wang, R. Zhang, S. Tao *et al.*, "Logparse: Making log parsing adaptive through word classification," in *2020 29th International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2020, pp. 1–9.

[57] S. Nedelkoski, J. Bogatinovski, A. Acker, J. Cardoso, and O. Kao, "Self-supervised log parsing," *arXiv preprint arXiv:2003.07905*, 2020.

[58] M. Chen, A. X. Zheng, J. Lloyd, M. I. Jordan, and E. Brewer, "Failure diagnosis using decision trees," in *ICAC'04: Proc. of the first International Conference on Autonomic Computing*. IEEE, 2004, pp. 36–43.

[59] Y. Liang, Y. Zhang, H. Xiong, and R. Sahoo, "Failure prediction in ibm bluegene/l event logs," in *ICDM'07: Proc. of the 7th IEEE International Conference on Data Mining*. IEEE, 2007, pp. 583–588.

[60] M. Farshchi, J.-G. Schneider, I. Weber, and J. Grundy, "Experience report: Anomaly detection of cloud application operations using log and cloud metric correlation analysis," in *ISSRE'15: Proc. of the 26th International Symposium on Software Reliability Engineering*. IEEE, 2015, pp. 24–34.

Xiaoyun Li is currently pursuing the Ph.D. degree with the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China. She received her BE degree from Sun Yat-sen University, in 2019. Her current research areas include log analysis and AI-driven operations.

Pengfei Chen is currently an associated professor in the School of Computer Science and Engineering of Sun Yat-sen University. Meanwhile, he is a Ph.D. advisor. Dr. Chen graduated from the department of computer science of Xi'an Jiaotong University with a Ph.D. degree in 2016. Now, he is interested in distributed systems, AIOps, cloud computing, Microservice and BlockChain. Especially, he has strong skills in cloud computing. So far, Dr. Chen has published more than 50 papers in some international conferences including IEEE INFOCOM, WWW, ACM/IEEE CCGRID, ICSOC, IEEE ICWS, IEEE ICPADS and journals including IEEE TDSC, IEEE TNNLS, IEEE TR, IEEE TSC, IEEE TETC, IEEE TCC. He serves as of program committee member of multiple conferences and reviewers of some internal journals such as IEEE Transactions on Cybernetics, Information Science, and Neurocomputing.

Linxiao Jing received the B.E. degree from School of Computer Science and Technology, Xidian University, Xi'an, China, in 2019. He is currently pursuing M.E. degree with the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China. His current research areas include cloud computing and AI-driven operations.

Zilong He received his BE degree and MS degree from Sun Yat-sen University, China, in 2019 and 2021. He is now a phd student at School of Computer Science and Engineering of Sun Yat-sen University, China. His current research areas include anomaly detection algorithms, AI driven operations.

Guangba Yu received his master degree from Sun Yat-Sen University, China, in 2020. He is now a phd student at School of Computer Science and Engineering with Sun Yat-Sen University, China. His current research areas include distributed system, cloud computing, and AI driven operations.

1545-5971 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on September 03,2022 at 06:18:14 UTC from IEEE Xplore. Restrictions apply.