# An Empirical Investigation of Practical Log Anomaly Detection for Online Service Systems

Nengwen Zhao[*]
Tsinghua University; BNRist
Beijing, China

Honglin Wang
BizSeer
Beijing, China

Zeyan Li
Tsinghua University; BNRist
Beijing, China

Xiao Peng
China Everbright Bank
Beijing, China

Gang Wang
China Everbright Bank
Beijing, China

Zhu Pan
China Everbright Bank
Beijing, China

Yong Wu
China Everbright Bank
Beijing, China

Zhen Feng
China Everbright Bank
Beijing, China

Xidao Wen
Tsinghua University; BNRist
Beijing, China

Wenchi Zhang
BizSeer
Beijing, China

Kaixin Sui
BizSeer
Beijing, China

Dan Pei[†]
Tsinghua University; BNRist
Beijing, China

## ABSTRACT

Log data is an essential and valuable resource of online service systems, which records detailed information of system running status and user behavior. Log anomaly detection is vital for service reliability engineering, which has been extensively studied. However, we find that existing approaches suffer from several limitations when deploying them into practice, including 1) inability to deal with various logs and complex log abnormal patterns; 2) poor interpretability; 3) lack of domain knowledge. To help understand these practical challenges and investigate the practical performance of existing work quantitatively, we conduct the first empirical study and an experimental study based on large-scale real-world data. We find that logs with rich information indeed exhibit diverse abnormal patterns (e.g., keywords, template count, template sequence, variable value, and variable distribution). However, existing approaches fail to tackle such complex abnormal patterns, producing unsatisfactory performance. Motivated by obtained findings, we propose a generic log anomaly detection system named *LogAD* based on ensemble learning, which integrates multiple anomaly detection approaches and domain knowledge, so as to handle complex situations in practice. About the effectiveness of *LogAD*, the average F1-score achieves 0.83, outperforming all baselines. Besides, we also share some success cases and lessons learned during our study. To our best knowledge, we are the first to investigate practical log anomaly detection in the real world deeply. Our work is helpful for practitioners and researchers to apply log anomaly detection to practice to enhance service reliability.

## CCS CONCEPTS

• **Software and its engineering → Maintaining software**.

## KEYWORDS

Log Anomaly Detection, Online Service Systems, Practical Challenges

**ACM Reference Format:**
Nengwen Zhao, Honglin Wang, Zeyan Li, Xiao Peng, Gang Wang, Zhu Pan, Yong Wu, Zhen Feng, Xidao Wen, Wenchi Zhang, Kaixin Sui, and Dan Pei. 2021. An Empirical Investigation of Practical Log Anomaly Detection for Online Service Systems. In *Proceedings of the 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '21), August 23–28, 2021, Athens, Greece.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3468264.3473933

## 1 INTRODUCTION

Log data is an important and valuable data source in online service systems, which records detailed information of system running status and user behavior. Log anomaly detection, aiming to identify abnormal system behavior, could assist engineers in identifying incidents promptly and diagnose incidents rapidly [30, 40, 41, 53]. Therefore, log anomaly detection is vital for service reliability and incident management [20, 21].

In traditional monitoring, engineers manually examine logs and write rules (keywords and regular expressions) to detect anomalies based on their domain knowledge. However, as the scale and complexity of service systems increase, it is challenging to detect log anomalies by manual rules due to the following reasons. 1) An online service system involves many components such as hardware, virtual machines, database, and network, generating a large number and variety of logs, about TBs per day [55]. It is tedious

[*]BNRist: Beijing National Research Center for Information Science and Technology
[†]Dan Pei is the corresponding author.

*ESEC/FSE '21, August 23–28, 2021, Athens, Greece*
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8562-6/21/08…$15.00
https://doi.org/10.1145/3468264.3473933

to set manual rules for such numerous and various logs. 2) Setting rules requires intensive domain knowledge, while the manpower of experienced engineers is limited. Besides, different engineers may have different preferences when making rules. 3) Service systems are usually under frequent software changes. Thus the manual rules should be constantly updated and maintained, which is also labor-intensive [31]. In summary, it is time-consuming and error-prone to detect log anomalies via manual rules in large-scale systems. Thus, it is imperative to design an automated log anomaly detection approach to replace manual effort.

To overcome the drawbacks of manual rules, tremendous efforts have been dedicated into automated log anomaly detection [19, 23, 25, 29, 30, 35, 38, 42, 45, 47, 48, 50, 51, 53]. Although the superiority of existing work has been illustrated in their papers, we encounter several practical challenges when applying them to the real world. 1) *Inability to handle various logs and complex abnormal patterns*. In practice, each service component could generate logs, leading to the diversity of log types. Besides, log messages contain rich information, including timestamps, templates, and variables [25], which can be obtained via log parsing [55]. Thus log abnormal patterns are also diverse, such as template count [35, 37, 47], template sequence [18, 25, 26, 38, 39, 52], variable value [25] and variable distribution. Existing approaches, however, are mainly evaluated on limited public datasets (HDFS [46], OpenStack [52], and BGL [2, 40]) and can only deal with one or two types of abnormal patterns. 2) *Poor interpretability*. Existing work simply provides the result of whether the current time is abnormal or not. However, engineers are confused about why it is an anomaly and how the normal pattern should behave. Equipping with an interpretable explanation, engineers could gain some actionable insights to diagnose and mitigate the anomaly more efficiently. 3) *Lack of domain knowledge*. Existing work targets at automated log anomaly detection without human involvement. Incorporating domain knowledge, however, could not only deal with some special situations, but also allow the interaction between humans and algorithms to improve the performance, which is more friendly to engineers. In summary, existing approaches cannot be applied directly to practice.

To help understand the role of log anomaly detection in incident management and the above challenges intuitively, we performed the first empirical study based on large-scale real-world data, and obtained three key findings. 1) Log anomaly detection is indeed beneficial to incident discovery and diagnosis. 2) Engineers still prefer to use keyword-based strategy in current practice instead of advanced approaches due to poor interpretability. However, keywords-based strategy is also unsatisfactory, producing false alarms and missing alarms. 3) Log types and abnormal patterns are indeed diverse. We summarized some common-used logs in engineers' daily operation and maintenance and corresponding abnormal patterns through historical incident analysis and interviews with engineers. To further confirm the above challenges quantitatively, we conducted an extensive experimental study to evaluate five typical unsupervised log anomaly detection approaches (PCA [47], LogCluster [35], Invariant Mining [37], DeepLog [25] and LogAnomaly [38]) on ten real-world log datasets from different service components with various abnormal patterns. The results show that all of these approaches perform unstably on different datasets, and the average F1-score is lower than 0.7. Consequently, we could conclude that existing approaches are not generic and effective in the real world.

Motivated by the findings obtained from empirical study and experimental study, we propose a generic and effective log anomaly detection system named *LogAD*, integrating a knowledge base and multiple anomaly detection techniques to address the above challenges. *LogAD* contains four components, data preparation, log preprocessing, log anomaly detection, and alerts with visualization. More specifically, we first collect unstructured raw logs to prepare for anomaly detection. The goal of log preprocessing is to transform raw logs into structured data. In addition to the widely-used log parsing technique (Drain [28] used in *LogAD*), we also import rule matching based on the knowledge base. It is because that not all logs can be well processed with an automated parsing algorithm without human involvement. Rule matching could utilize domain knowledge to extract valuable information and enhance performance (tackle the third challenge). Afterwards, appropriate anomaly detection algorithms are selected based on the configurations in the knowledge base or engineers' selection. *LogAD* fuses multiple anomaly detection techniques to handle complex abnormal types (tackle the first challenge). After identifying anomalies, *LogAD* generates an alert with an interpretable report, which could assist engineers in understanding this alert and gaining insights to diagnose the problem (tackle the second challenge).

We evaluated the performance of *LogAD* on ten log datasets with different abnormal patterns used in the experimental study. The results show that *LogAD* could achieve the best F1-score (0.83 on average) across all datasets. Besides, *LogAD* has been applied in a large company and we also share some success cases to help understand the practical usage of *LogAD*. These cases demonstrate that *LogAD* could assist incident management from multiple aspects (e.g., predicting OOM failure, detecting network issues, and detecting database server down). Moreover, we also share some lessons learned during our study. We believe our work is helpful for practitioners and researchers to apply the technique of log anomaly detection to practice to enhance service reliability.

To sum up, this paper has the following major contributions:

- We point out several practical challenges of log anomaly detection. Besides, we conduct the first empirical study and an experimental study on large-scale real-world data to further support these challenges. Some key observations obtained from our study could provide some guidance for the practical usage of log anomaly detection.
- Inspired by the observations, we propose an effective and generic log anomaly detection system named *LogAD*, which incorporates domain knowledge and multiple anomaly detection approaches to tackle diverse abnormal patterns with good interpretability.
- The effectiveness of *LogAD* is confirmed based on real-world log data. Besides, *LogAD* has been applied in a large company, and we share some real-world cases and lessons learned.

## 2 BACKGROUND

### 2.1 Log Anomaly Detection

Log anomaly detection strives to identify abnormal behavior of systems and users. Following the existing work [31], the general pipeline of log anomaly detection can be summarized with Figure 1,

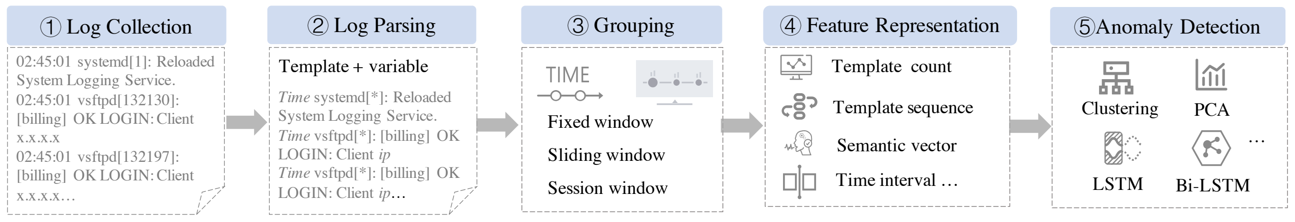

① Log Collection
02:45:01 systemd[1]: Reloaded System Logging Service.
02:45:01 vsftpd[132130]: [billing] OK LOGIN: Client x.x.x.x
02:45:01 vsftpd[132197]: [billing] OK LOGIN: Client x.x.x.x…
② Log Parsing
Template + variable
Time systemd[*]: Reloaded System Logging Service.
Time vsftpd[*]: [billing] OK LOGIN: Client ip
Time vsftpd[*]: [billing] OK LOGIN: Client ip…
③ Grouping
TIME
Fixed window
Sliding window
Session window
④ Feature Representation
Template count
Template sequence
Semantic vector
Time interval …
⑤Anomaly Detection
Clustering
PCA
LSTM
Bi-LSTM
…

**Figure 1: Pipeline of existing log anomaly detection approaches**

including five main steps: log collection, log parsing, grouping, feature representation, and anomaly detection.

*Log Collection.* Log collection mainly contains agent-based method (e.g., Filebeat [4] and Logstash [10]) and syslog-based method (e.g., syslog-ng [17] and Rsyslog [15]). A log message usually contains a timestamp and detailed information indicating what has happened [31]. As systems grow in scale and complexity, logs are generated at an ever-increasing rate. For example, a service system in a large company could generate about 2 TB of log data daily [36].

*Log Parsing.* Logs are usually unstructured texts, generated using the *printf* function with a string template and detailed information. Log parsing aims to transform raw logs into structured templates (constant parts) and variables (variable parts) [55]. Log parsing has been extensively studied in the literature, such as Drain [28], Spell [24], LogMine [27] and LogSig [43]. Zhu et al. conducted an experience report of existing log parsing algorithms on large-scale log datasets [55], demonstrating the superiority of Drain.

*Grouping.* Due to contextual time dependency existing in logs, we cannot consider only a single log message for anomaly detection. Thus we need to split raw logs into a set of log sequences using different grouping strategies. As introduced in [31], common methods include fixed time windows, sliding time windows, and session windows (e.g., splitting HDFS logs using *blockid* ).

*Feature Representation.* Existing approaches have designed various features, such as template count [35, 37, 47] recording the number of each template in the window, template sequence [25, 38] recording the order of task execution, log semantic feature [33, 53] which characterizes the semantic information using the techniques of natural language processing (NLP), and variable value [25] which represents the behavior of some key variables (e.g., response time).

*Anomaly Detection.* Anomaly detection is adopted to identify anomalies from extracted features, containing two categories. 1) Traditional statistical methods. He et al. have provided a detailed experience report [31] about these traditional statistical methods, including PCA [47], LogCluster [35], Invariant Mining [37] and some other classification-based supervised methods [22, 34]. 2) Deep learning based methods. In recent years, deep learning has been widely used in log anomaly detection. DeepLog[25] and LogAnomaly [38] train a prediction model on normal log data and predict which template will arrive using LSTM-based models. Once the arrived log template violates the top-k predicted results, this log is regarded as an anomaly. LogRobust [53] adopts an attention-based Bi-LSTM classification model to detect anomalies based on semantic features. In this paper, we mainly focus on *unsupervised* approaches since it is difficult to obtain high-quality and sufficient anomaly labels to construct supervised classification models.

## 2.2 Practical Challenges

Although many approaches have been proposed to identify log anomalies, they perform not well when applying them to the real world. The significant challenges can be summarized as follows.

### 2.2.1 *Various Logs and Complex Abnormal Patterns.*

Each component in large-scale service systems would generate logs, from hardware-layer (e.g., switch log) to middleware-layer (e.g., Apache access log) and application-layer (e.g., business error log). These logs reflect the system status from different angles and may exhibit different patterns. However, existing approaches are mainly evaluated on several public datasets (HDFS [46], BGL [2] and OpenStack [25]), which are not representative and generic.

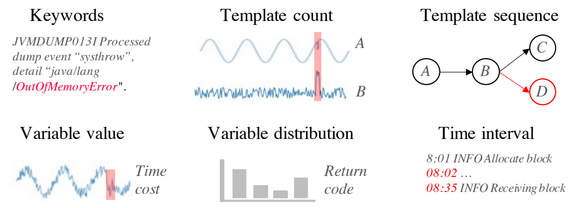
In addition to the variety of logs, logs with rich information could also perform different abnormal patterns. Based on existing work and empirical analysis, we introduce six common abnormal patterns, and Figure 2 presents some intuitive explanations.

- Keywords. Once some negative keywords (e.g., “OutOfMemory”) appear in a log message, this log is abnormal.
- Template count [31, 35, 37, 47]. The number of some templates increases or decreases dramatically.
- Template sequence [18, 25, 26, 38, 39, 52]. Template sequence could reflect the order of task execution. For example, the normal sequence is $A \to B \to C$, while $A \to B \to D$ violating the normal pattern is an anomaly.
- Variable value [25]. Some variables in logs may have specific physical meaning (e.g., time cost), which could be extracted and aggregated into a metric with time-series format. Thus we could identify the anomaly of variable value from the time series.
- Variable distribution. Some variables are categorical, e.g., return code, URL, and IP in Nginx access log, whose distribution should be considered. For example, the increase in 502 (bad gateway) return code would lead to abnormal distribution of HTTP return codes, indicating system unavailability.
- Time interval [33]. Some performance issues (e.g., network congestion) may induce long time intervals between logs. Time interval anomaly could also result in template count anomaly.

In addition to the above log abnormal patterns, there are also other situations. For example, the number of total logs increases or decreases dramatically; new emerging log patterns cannot match any log templates extracted from training data. As a result, a practical log anomaly detection system needs to have the ability to deal with various logs and complex abnormal patterns.

### 2.2.2 *Poor Interpretability.*

Current work aims to design novel algorithms to improve detection performance. However, most of



```
JVMDUMP013I Processed dump event "systhrow", detail "java/lang/OutOfMemoryError".
```

```
8:01 INFO Allocate block
08:02 ...
08:35 INFO Receiving block
```

**Figure 2: Various log abnormal patterns in the real world**

these algorithms work as a "black box". It is difficult for engineers to intuitively figure out what the anomalies are, why the current time is abnormal, and how the expected normal patterns should behave. Besides, engineers cannot gain any intuitive and actionable insights from the abstract feature vectors. Thus, methods that could reflect the natures of anomalies with a friendly visualization are highly desired in practice. In this way, engineers could understand the results better and investigate the anomaly immediately.

*2.2.3 Lack of Domain Knowledge.* Existing approaches are proposed for automated log anomaly detection with little human involvement. Based on our observations, however, incorporating domain knowledge is necessary due to the variety of logs and abnormal patterns. For example, faced with various logs, which log files are more critical and should be monitored, which anomaly detection approach is appropriate, and for multiple variables in logs, which variables have physical meaning and need to be monitored. All of these should be determined by experienced experts and cannot be completely replaced by automated algorithms. Besides, fusing domain knowledge could help handle some special situations and ensure human-computer interactive log analysis, to enhance the performance and interpretability.

## 3 EMPIRICAL STUDY

To better understand the role of log anomaly detection in incident management and support the above challenges, we conducted an empirical study to answer the following research questions (RQs):

- **RQ1**: What role do logs play in incident management?
- **RQ2**: How about the performance of the current practice of log anomaly detection in the real world?
- **RQ3**: What logs engineers frequently use in daily maintenance, and what abnormal patterns may exist in log data?

To answer these questions, we analyzed tens of thousands of incidents and millions of alerts related to logs over the last two years from a large company, which supports tens of millions of users and contains hundreds of services. Due to the privacy policy, we hid the specific number of incidents. Besides, we also interviewed several experienced engineers from different teams (e.g., business, database, and network) to obtain some practical feedback.

### 3.1 RQ1: Logs in Incident Management

In general, incident ticket records detailed information about how the incident was detected ("Summary" attribute) and the detailed steps of troubleshooting ("Diagnosis process" attribute). To answer this question, we set keywords (i.e., "log") to search the "Summary" and "Diagnosis process" attributes, to count the number of incidents that were discovered via log anomalies and the number of incidents whose diagnosis involved logs. The results are displayed in Figure 3. Only 3.4% of incidents are discovered via log anomalies, probably due to the insensitivity of engineers on log anomalies and the drawbacks of the current practice of log anomaly detection (see §3.2). The role of logs in incident discovery may be further enhanced via a more effective log anomaly detection approach. In terms of incident diagnosis, 30.9% of incidents involve log data in troubleshooting. Considering that some low-quality tickets do not record detailed diagnosis process, the actual percentage could be larger. The results demonstrate that log data is a valuable data source in incident diagnosis. Usually, utilizing logs for troubleshooting also aims to identify log anomalies during the incident [30, 35]. Overall, an effective log anomaly detection approach could not only enable more timely and accurate incident discovery, but also could improve the efficiency of incident diagnosis.

### 3.2 RQ2: Current Practice

The current practice of log anomaly detection in the company we cooperated with is setting keywords (e.g., "error" and "time out") manually. Based on the feedback from engineers, existing approaches, especially deep learning based ones, are black-box and cannot provide intuitive results. Poor interpretability makes it difficult to understand results intuitively and gain actionable insights to diagnose the anomaly. Besides, tuning parameters of black-box models is also challenging. In comparison, configuring keywords based on domain knowledge is more acceptable for engineers (supporting the second and third challenges).

To study the performance of the current practice, we counted the number of alerts generated from logs per day, and the distribution of #log alerts/day is presented in Figure 4. It is clear that there are tens to hundreds of log alerts per day. The percentage of #log alerts/day larger than 100 accounts for 54.6%, and the average #log alerts/day is 282. Based on the results of RQ1 (incident discovery via log anomalies accounts for 3.8%), we infer that the current practice of log anomaly detection may bring some invalid alarms. Moreover, engineers confirm the incapability of keyword-based methods due to the following reasons. 1) In large-scale systems, various and numerous logs are generated per day. It is error-prone and labor-intensive to configure all keywords correctly; 2) Simple keyword strategy could generate some false positives (occurrence of some keywords does not necessarily mean an incident) and false negatives (some keywords are missed, or some incidents are not reflected on keywords) [31]. Typically, to enhance performance and adapt to the dynamic environment, engineers tend to spend significant efforts to maintain and update rules constantly (e.g., adding new "AND" and "OR" conditions). Consequently, a practical and effective log anomaly detection approach to save efforts is in urgent demand.

### 3.3 RQ3: Log Types and Log Abnormal Patterns

As introduced in §2.2, various logs and complex log abnormal patterns exist in practice. To confirm it, we counted the percentage of alerts generated from different kinds of logs. Intuitively, logs, where engineers set alerting rules, are more critical than others. As shown

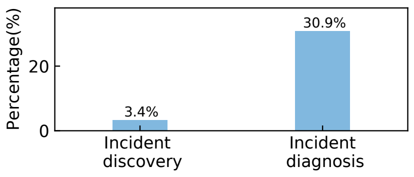


**Figure 3: The percentage of logs used in incident discovery and diagnosis**

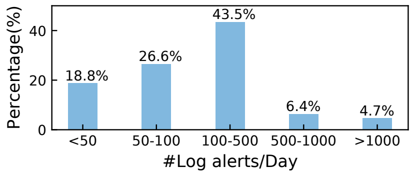


**Figure 4: The distribution of the number of log alerts per day**

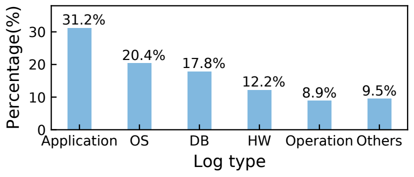


**Figure 5: The percentage of different types of log alerts**

**Table 1: Some typical common-used log files, corresponding abnormal patterns and incident scenarios**

| Type | Example logs | Abnormal patterns | Incident scenarios |
|---|---|---|---|
| Application | Application error | Template count (e.g., #"connection refused" increases), new log pattern (unknown error messages) | System unavailability, application error |
| Operation system | Linux, Windows, AIX | Keywords (e.g., "soft lockup" and "page allocation failure"), #total logs, template count (e.g., #"creating session" decreases) | OOM, node hang, hardware failures |
| Database | Oracle, DB2, MySQL | Keywords (e.g, "ORA-", "Level: Error"), new log pattern | Database server down, performance degradation, SQL defect |
| Hardware (Network devices) | Switch, F5 | Keywords (e.g., illegal state transition, address conflict detected), new log pattern, template count | IP address conflicted |
| Hardware (Storage) | Ceph | Keywords (e.g., "no reply", and "time out"), new log pattern | Disk failure, space exhaustion |
| Distributed system | HDFS, OpenStack | Keywords (e.g., "IOException"), template sequence (wrong order of execution), new log pattern | Task failed, instance failure |
| Operation | User behavior | Template count, template sequence | Illegal operation, security issue |
| Middleware (Web server) | Nginx access, Apache access | Variable distribution (e.g., return code, and IP address), #total logs | System unavailability, illegal user access, security threat |
| Middleware (JVM GC) | CMS, G1, parallel | Keywords ("OutOfMemory"), template count (e.g.,#"FullGC" increases), variable value (GC time cost) | GC overhead limit exceeded, full heap space, memory leak |
| Middleware (Message queue) | IBM MQ | Keywords (e.g., "error occur", "ended abnormally", and "exception"), new log pattern | Message queue stuck |

in Figure 5, it is clear that the logs generated from application, operation systems (OS), database (DB), hardware (HW), and operation account for a substantial part (about 90%). To further support it, we interviewed several engineers from different teams. Our goal is to understand which logs engineers find most valuable, which logs are frequently inspected during daily maintenance, and what abnormal patterns these logs typically exhibit. Based on the discussion results, we summarized some typical common-used logs, the corresponding abnormal patterns, and potential incident scenarios, as shown in Table 1. We obtain two key findings from this table. First, the abnormal patterns existing in logs are indeed diverse, and moreover, a log file may exhibit more than one abnormal pattern. For example, the anomalies in JVM Garbage Collection (GC) logs contain the appearance of keywords "OutOfMemory", the increase in "FullGC" template (template count), and the rise of GC time cost and heap space (variable value). Second, different abnormal behavior in a log file may indicate different potential incidents. Taking Nginx access logs recording HTTP requests as an example, the anomaly of return code distribution (e.g., 502 accounts for a large part) may indicate system unavailability. The anomaly of IP address distribution (e.g., the increase in the number of an IP) may indicate an illegal user is frequently trying to access the website. Therefore, log anomaly detection is a complex task, and various situations should be taken into consideration, which a single algorithm cannot solve.

## 3.4 Summary

- Log anomaly detection is valuable for incident management.
- Engineers tend to use keyword-based method in current practice due to better interpretability. However, the performance of keyword-based strategy is far from satisfying.
- Log types and abnormal patterns are indeed diverse in practice. Thus a practical, effective and generic log anomaly detection approach to tackling such complex situations is in urgent demand.

## 4 EXPERIMENTAL STUDY

To further illustrate the performance of existing approaches in practice quantitatively, we conducted an experimental study based on several real-world datasets. We selected five popular unsupervised log anomaly detection approaches, including statistical approaches (PCA [47], LogCluster [35] and IM [37]) and deep learning based approaches (DeepLog [25] and LogAnomaly [38]), aiming to answer the following two research questions:

- **RQ4**: How about the effectiveness of the studied approaches on various real-world log data with different abnormal patterns?
- **RQ5**: How about the efficiency of the studied approaches?

## 4.1 Datasets and Measurements

We collected ten real-world log datasets from different service components, as displayed in Table 2. *D*1 and *D*2 are application error

**Table 2: Details of our experimental datasets**

| Dataset | Type | #Logs | #Pos/#Neg |
|---|---|---|---|
| $D1$ | Application error | 26,918 | 3/720 |
| $D2$ | Application error | 84,139 | 7/1446 |
| $D3$ | User operation | 9,080 | 2/38 |
| $D4$ | Nginx access | 2,856,793 | 32/3036 |
| $D5$ | JVM GC (CMS) | 217,613 | 13/398 |
| $D6$ | JVM GC (Parallel) | 64,208 | 27/469 |
| $D7$ | DB2 | 16,133 | 2/74 |
| $D8$ | Linux system | 771,083 | 4/768 |
| $D9$ | Linux system | 3,227,843 | 5/1459 |
| $D10$ | Linux system | 1,087,956 | 6/1288 |

logs which record the application-layer error information of a service. $D3$ is user operation log, which records the users' actions on servers (such as logging in, logging out, and executing commands). $D4$ is Nginx access log recording HTTP requests. $D5$ and $D6$ are collected from JVM GC with two different GC strategies (CMS and parallel). $D7$ is DB2 database log. $D8$, $D9$, and $D10$ are Linux system logs from three servers, where different tasks are running. The abnormal patterns of these datasets are diverse (see Table 1).

We adopt precision, recall, and F1-score as evaluation measurements, which have been widely used in existing work [25, 31, 38]. About data labeling, it is inappropriate to label single log message since we need to refer to the contextual information to determine the current time is abnormal or not. Based on the grouping methods discussed in [31], we split all logs by sliding time window (window size is 10 minutes and step size is 5 minutes). Motivated by the label of public datasets (e.g., HDFS and OpenStack), we label a time window as an anomaly if an incident (e.g., illegal access in $D4$, frequent FullGC in $D5$ and $D6$, server unavailability in $D8$, $D9$ and $D10$) occurs during the window. All log messages during the time window are regarded as a whole. The number of positive (abnormal) and negative (normal) samples are presented in Table 2.

## 4.2 Experiment Setup

We split each dataset into a training set and a testing set in the ratio of 6:4, where the training set is all normal data. It is easy to obtain sufficient normal data in practice since services tend to run stably without failures. For the studied approaches, we used the parameters provided by their papers. The log parsing approach we adopted is Drain [28], whose superiority has been demonstrated in [55]. The implementation of Drain is based on LogParser [9] and the implementation of statistical approaches (PCA, LogCluster, and IM) is based on loglizer [8, 31]. All approaches are implemented by Python with widely-used libraries, including NumPy [11], pandas [12], scikit-learn [16], Pytorch [13], etc. Our experimental study was conducted on Ubuntu 18.04.1 with 24-core Intel Xeon(R) CPU E5-2620 v3@2.40GHz, 64GB memory, 64-bit operating system.

## 4.3 RQ4: Effectiveness

Table 3 presents the precision, recall, and F1-score of five studied approaches on different datasets. Clearly, although these approaches achieve promising performance on several public datasets (HDFS [46], BGL [2] and OpenStack [52]), they perform not well and unstably on various real-world data. The average F1-score of the five approaches could only achieve 0.48~0.66. Besides, deep learning based approaches perform better than traditional statistical approaches, indicating the superiority of deep learning techniques.

The reasons for the unsatisfactory performance can be summarized as three points. First, the abnormal patterns of public log datasets are relatively simple. Taking HDFS as an example, some abnormal blocks contain negative keywords (e.g., "IOException", "Unexpected error", and "does not belong to any file"). The number of total logs of some abnormal blocks is far fewer than normal blocks. We also tried to set some manual rules to detect anomalies from HDFS dataset, and the F1-score could also achieve about 0.86. Therefore, it is inappropriate to demonstrate the efficacy and practicality of log anomaly detection algorithms using such simple datasets. Second, these approaches perform unstably with high standard deviations due to various abnormal patterns. These approaches fail to consider comprehensive information in logs, which limits the types of anomalies (see Figure 2) they can detect. For example, PCA and LogCluster deal with template count; DeepLog deals with template sequence and variable value; LogAnomaly tackles template count and template sequence. Third, the poor performance of log parsing could destroy detection accuracy. We observe that even the state-of-the-art log parsing algorithm (Drain) performs not well on some logs, requiring manual adjustment using domain knowledge. For example, access log ($D4$) composing with IP, URL, and return code, could be transformed into structural information with domain rule, while it is unsuitable for log parsing algorithm based on word frequency, resulting in low F1-score on all approaches.

The results further support the proposed challenges in §2.2 quantitatively. Thus we need to propose a practical and generic log anomaly detection system to overcome the challenges.

## 4.4 RQ5: Efficiency

Time efficiency is a vital factor for log anomaly detection, to guarantee timely incident discovery and efficient incident diagnosis. Table 4 presents the average time cost on ten experimental datasets, including offline training time on all training samples (including log parsing) and online detection time on a 10-minute time window (one sample). We find that the average online prediction time of these approaches is in a satisfactory range, less than 2 seconds (nearly real time). About offline training, statistical approaches require relative short time, about several minutes. In comparison, DeepLog and LogAnomaly require tens of minutes to train the model. It is because they adopt template sequence as features which is very large, and LSTM model also takes much time to train. Considering that the training process is conducted offline and will not destroy the efficiency of anomaly detection, long training time is also acceptable. However, faced with TBs logs generated per day, high training cost may take up many computation resources, and make the model cannot update flexibly to accommodate to software changes.

## 5 APPROACH

Motivated by the findings obtained from the empirical study and experimental study, we propose a generic log anomaly detection system named *LogAD* to tackle the practical challenges together in §2.2, which could handle various logs and complex abnormal

**Table 3: Precision (P), recall (R) and F1-score (F1) of studied approaches on experimental datasets**

| Approach | PCA | | | LogCluster | | | IM | | | DeepLog | | | LogAnomaly | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Dataset | P | R | F1 | P | R | F1 | P | R | F1 | P | R | F1 | P | R | F1 |
| *D*1 | 0.40 | 1.00 | 0.57 | 0.38 | 1.00 | 0.55 | 0.51 | 1.00 | **0.68** | 0.64 | 0.33 | 0.44 | 0.45 | 0.67 | 0.54 |
| *D*2 | 0.56 | 0.43 | 0.48 | 0.50 | 0.57 | 0.53 | 0.34 | 0.43 | 0.38 | 0.71 | 0.86 | **0.78** | 0.71 | 0.86 | **0.78** |
| *D*3 | 0.67 | 0.50 | 0.57 | 0.72 | 0.50 | 0.59 | 0.43 | 1.00 | **0.60** | 0.43 | 0.50 | 0.46 | 0.62 | 0.50 | 0.55 |
| *D*4 | 0.20 | 0.44 | 0.28 | 0.24 | 0.53 | **0.33** | 0.32 | 0.21 | 0.26 | 0.17 | 0.38 | 0.23 | 0.22 | 0.34 | 0.27 |
| *D*5 | 0.92 | 1.00 | **0.96** | 0.80 | 1.00 | 0.89 | 0.87 | 1.00 | 0.93 | 0.91 | 1.00 | 0.95 | 0.91 | 1.00 | 0.95 |
| *D*6 | 0.64 | 1.00 | 0.78 | 0.86 | 0.81 | 0.83 | 0.82 | 1.00 | 0.90 | 0.93 | 0.96 | 0.94 | 0.92 | 1.00 | **0.96** |
| *D*7 | 0.42 | 0.50 | 0.46 | 0.44 | 0.50 | 0.47 | 0.58 | 0.25 | 0.35 | 0.57 | 1.00 | **0.73** | 0.53 | 1.00 | 0.69 |
| *D*8 | 0.10 | 0.25 | 0.14 | 0.35 | 0.25 | 0.29 | 0.14 | 0.25 | 0.18 | 1.00 | 0.50 | **0.66** | 1.00 | 0.50 | **0.66** |
| *D*9 | 0.18 | 0.60 | 0.28 | 0.28 | 1.00 | 0.44 | 0.44 | 0.40 | 0.42 | 0.32 | 1.00 | 0.48 | 0.54 | 0.60 | **0.57** |
| *D*10 | 0.29 | 0.33 | 0.31 | 0.43 | 0.33 | 0.37 | 0.29 | 1.00 | 0.45 | 0.74 | 0.50 | 0.60 | 1.00 | 0.50 | **0.67** |
| Mean F1 (Std) | 0.48 (0.24) | | | 0.53 (0.19) | | | 0.52 (0.24) | | | 0.63 (0.22) | | | 0.66 (0.19) | | |

**Table 4: Average time cost comparison**

| Approach | PCA | LogCluster | IM | DeepLog | LogAnomaly |
|---|---|---|---|---|---|
| Training (min) | 0.89 | 5.40 | 4.60 | 34.67 | 48.54 |
| Detection (s) | 0.23 | 0.56 | 0.38 | 1.02 | 1.78 |

patterns. More specifically, *LogAD* contains four steps: data preparation, log preprocessing, log anomaly detection, and alerts with visualization. There are three core ideas of *LogAD*. First, *LogAD* integrates multiple anomaly detection techniques to deal with complex scenarios (tackle the first challenge). The second one is importing a knowledge base fusing expert experience, which could assist log preprocessing (§5.1), selecting corresponding algorithms (§5.2) and setting keywords (tackle the third challenge). The last one is *LogAD* provides an intuitive alert report with good interpretability for engineers to understand the alerts (tackle the second challenge). In the following, we will introduce three main components in detail.

## 5.1 Log Preprocessing

The goal of log preprocessing is to transform unstructured logs into structured information. In the experimental study (§4), we observe that even the state-of-the-art log parsing algorithm (Drain [28]) performs not well on some logs and often requires manual adjustments on templates (e.g., merging and deleting). Besides, automated log parsing fails to handle complex situations. For example, the formats of log timestamps are diverse in practice, about tens of types, e.g., "05/01/21 21:40:00", "05/01/2021 09:40:00 PM" and "20210501|214000|895". It is a daunting task for an automated algorithm to identify various timestamps accurately without domain knowledge. To address the problem, *LogAD* imports a *knowledge base* to assist log preprocessing from three perspectives. First, the templates of some generic logs (e.g., Oracle and DB2) are usually fixed. Thus we could store these templates as knowledge. In this way, if we want to detect anomalies on similar logs, it will save resources and efforts of log parsing and manual adjustments. Second, for some special logs which are inappropriate to use parsing algorithm (e.g., access log), *LogAD* could directly adopt rule matching to transform raw logs into structured information (e.g., access logs can be split into timestamp, IP, URL, and return code). Third, some key variables (e.g., heap space in GC logs) could be selected for further variable anomaly detection. Consequently, equipping log parsing with a knowledge base, raw log data could be well transformed into structured information for further anomaly detection.

## 5.2 Log Anomaly Detection

As presented in §3, log messages contain rich information (template, variable, and timestamp) and exhibit diverse log abnormal patterns. Therefore, it is notoriously hard for a single algorithm to work well faced with such complex situations. In our system, we leverage the idea of ensemble learning and integrate multiple anomaly detection techniques [45]. In this way, engineers could select one or more suitable anomaly detection components based on domain knowledge to detect specific abnormal patterns for different logs. Although existing work [45] adopts multiple approaches (PCA, LogCluster and IM) to detect anomalies more accurately, it is not designed for tackling various abnormal patterns. For template count, we propose to adopt LSTM-based multivariate time-series anomaly detection technique (§5.2.2). Besides, it is the first time to notice variable distribution, and we utilize the popular distribution distance measurement (JS divergence) to detect anomalies (§5.2.5). The techniques of template sequence (§5.2.3) and variable value (§5.2.4) are from existing work. In the following, we will introduce the anomaly detection techniques equipped in *LogAD* in detail.

5.2.1 *Keywords.* Although keyword-based strategy is relatively simple, it cannot be replaced since it is the most direct way to identify failures. Based on our experience, setting rare, specific and fatal keywords(e.g., "OutOfMemory" and "IOException") is more valuable and accurate. The keywords of different logs(see Table 1) are maintained in the knowledge base. Once the keyword appears in online logs, an alert would be generated.

5.2.2 *Template Count.* Existing work (e.g., PCA [47]) encodes a template count feature vector for each time window and detects anomalies on this vector, which is not intuitive for engineers. In *LogAD*, we characterize the behavior of template count using template time series at a regular interval (e.g., one minute). The number of a template per minute should follow a normal pattern. In this way, engineers could grasp an overview of the patterns of template count. Usually, a log file may have tens to hundreds of templates. Detecting anomalies on each template series would consume many resources and incur numerous alerts. Based on our observation,

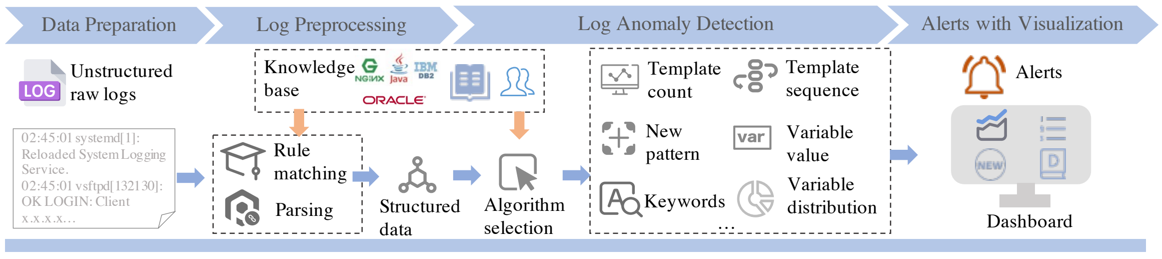


**Figure 6: Overview of *LogAD***

there exists some relationship among the number of different templates. For example, the decrease in template *A* may lead to an increase in template *B*. Thus we could use the multivariate time series anomaly detection technique (LSTM-based algorithm [32] used in *LogAD*) and regard all template series as an overall entity, which could not only enhance accuracy and identify potential correlated anomalies but also save computation resources.

*5.2.3 Template Sequence.* Template sequence typically profiles the process of task execution (e.g., batch tasks and distributed system tasks). Recently, some approaches adopt LSTM-based models to learn the normal template sequence [25, 38]. Once the arrived log template violates the top-k predicted template, it is regarded as an anomaly. However, directly applying LSTM-based approaches fails to provide the normal sequence intuitively, which is unfriendly to engineers. In general, the workflow of task execution can be characterized as a finite state automaton (FSA). Thus it would be better to provide a normal workflow visualization generated from logs so that engineers can have a global view of the task execution process. In *LogAD*, we build a FSA based on [25] since it has overcome several drawbacks of other related work (e.g., CloudSeer [52]). When online detection, once the arrived logs violate the workflow graph, they are considered as anomalies.

*5.2.4 Variable Value.* Some variables in logs with specific physical meaning are vital and should be monitored closely, e.g., time cost and heap space in GC logs. The variable value can be aggregated into time series with a regular interval (e.g., one minute). Thus we could leverage the technique of time series anomaly detection to detect the anomalies of variable value. Specifically, we find that there are usually two types of variable value series, seasonal and stationary. We first utilize auto-correlation coefficient [44, 54] to determine the variable series is seasonal or not. Then we adopt Holt-winters for seasonal series [49] and 3-$\sigma$ for stationary series.

*5.2.5 Variable Distribution.* Some variables are categorical, instead of continuous value, for example, IP address, URL, and return code in access logs; city, ISP, version, and browser in transaction logs. Taking an intuitive example, the number of requests from different cities in normal situations tends to follow a regular distribution. Once the number of requests from a city decreases dramatically, the distribution changes accordingly, indicating local system unavailability or city ISP issues. Thus for categorical variables, we mainly focus on the distribution anomaly. *LogAD* utilizes Jensen-Shannon (JS) divergence [7] to characterize the distance between two distributions (ranging from 0 to 1). Specifically, the normal range of distribution distance can be learned from historical data. When online detection, once the distribution of current data is far away from the historical data (exceeding the threshold), we can infer current time is an anomaly.

In summary, here we just introduce several typical anomaly detection algorithms. During practical usage, the anomaly detection module is extensible, and engineers could add new anomaly detection techniques to suit their needs.

## 5.3 Alerting with Visualization

After an anomaly is detected via the above anomaly detection module, an alert would be triggered. Different from existing approaches, *LogAD* provides an interpretable report for this alert, including the time series of template count (help understand the historical pattern of each template), normal template sequence with FSA (help understand the workflow of task execution), the time series of specific variable value, and distribution of log variables. In this way, engineers could easily understand why current logs are abnormal. More importantly, the interpretable report could assist engineers in taking action to diagnose and mitigate the anomaly.

## 6 EVALUATION AND CASE STUDY

## 6.1 Performance and Application

To demonstrate the efficacy of *LogAD*, we compare it with the best performance of baseline methods on ten real-world datasets in §4. The results are displayed in Figure 7. It is clear that the average F1-score of *LogAD* achieves 0.83, outperforming baseline approaches. In comparison, the average F1-score of baselines only achieve 0.48 ~ 0.66 (Table 3). We further investigated the reasons for the superiority of *LogAD*. First of all, through integrating multiple anomaly detection techniques, *LogAD* can deal with various logs and complex abnormal patterns. In comparison, existing work can handle only one or two abnormal patterns, incurring unsatisfactory and unstable performance across all datasets. Second, *LogAD* incorporates a knowledge base accumulating rich expert experience to tackle some special cases. For example, the performance of log parsing could be enhanced, some crucial keywords as prior knowledge are configured in *LogAD* and so on. Third, the anomaly detection techniques used in *LogAD* are effective compared with some existing work, which could enable *LogAD* to identify anomalies accurately. For example, LSTM-based multivariate template count anomaly detection performs better than PCA. Consequently, the results confirm the practical effectiveness of *LogAD*.

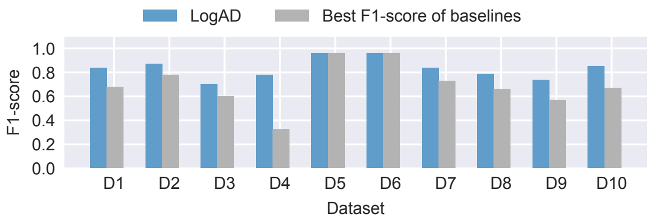


**Figure 7: F1-score comparison between *LogAD* and the best performance of baseline approaches**

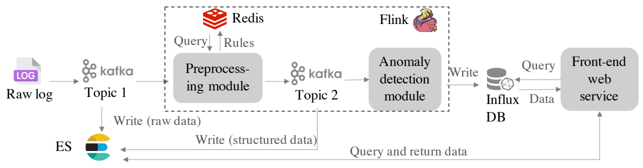


**Figure 8: Architecture of *LogAD* in practice**

Figure 8 presents the deployment architecture of *LogAD*. In detail, raw log data are pushed to Kafka [1]. The log preprocessing module consumes raw logs from Kafka in real time, and the outputted structured data are also pushed to a new Kafka topic for further anomaly detection. We adopt Apache Flink [5] to implement log preprocessing module and anomaly detection algorithms in the computation layer, which can process the stream data with high performance and low latency in a distributed way. All log anomalies detected by the anomaly detection core are stored in a time-series database (InfluxDB [6]) with the format of (timestamp, anomaly). Knowledge base in *LogAD* is maintained in Redis [14] since the rules would not occupy too much space, and Redis as an in-memory data structure store is highly efficient. Besides, all raw logs and structured information are indexed into ElasticSearch (ES) [3] via consuming Kafka, so that data become available for search immediately. Finally, the front-end web service could read data from ES and InfluxDB to provide a visualization for engineers.

We interviewed several engineers and they appreciated LogAD from two aspects. 1) The anomaly detection techniques are white-box, so that engineers could understand the reported anomalies with good interpretability and grasp normal patterns. 2) Engineers could obtain comprehensive information intuitively. For example, engineers only know the appearance of OOM using keywords on GC logs. However, LogAD could present various information like heap space, time cost and GC frequency.

## 6.2 Case Study

We introduce three success cases collected from the real world.

*6.2.1 Case 1: Predicting OOM Failure.* A historical OOM failure was discovered by traditional keyword-based ("OutOfMemory") method. With *LogAD*, as presented in Figure 9(a), we could predict the failure in advance via the increase in the number of "FullGC" templates. It is because this OOM failure is not transient, which is induced by frequent FullGC operation. Thus we could predict it through the abnormal behavior of FullGC templates. Meanwhile, the variables also behave abnormally (increase in GC time cost and heap space), which could assist engineers in analyzing the problem. In comparison, existing approaches fail to analyze templates and variables together and provide an interpretable report. Overall, *LogAD* could enable engineers to predict OOM failure in advance and provide key variable analysis for failure diagnosis.

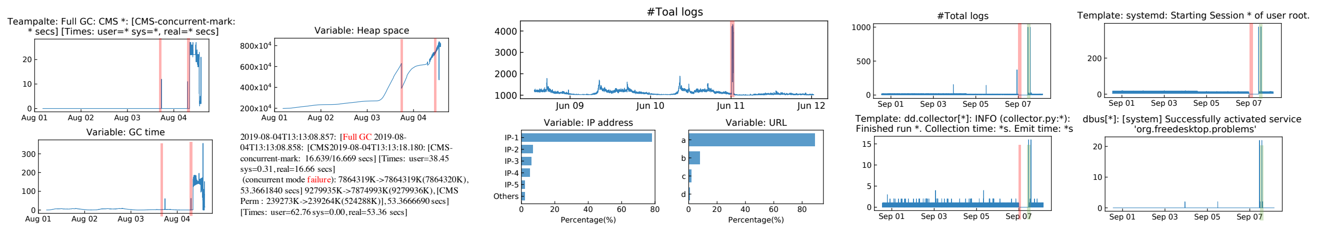
*6.2.2 Case 2: Identifying Illegal Access.* Usually, Nginx access logs record the HTTP requests and contain time, URL, user IP and return code. As a common log type in the real world, access logs can be transformed into the combination of these variables with the help of a knowledge base. As shown in Figure 9(b), *LogAD* detected the illegal access problem via a spike of the number of total logs, achieving more than 4000 requests per minute (pink band). Through the analysis of variable distribution, we find the IP-1 and URL *a* account for a large part. Thus we could easily find the illegal user and the behavior. However, keyword-based method cannot handle access log because it does not have negative keywords (setting error return code, e.g., 404, could cause many false alarms). Besides, existing approaches in academic also fail to detect and analyze the problem, because access log is unsuitable for log parsing, and they cannot analyze variable distribution. Actually, special logs in practice like access logs should be treated specially with human involvement, including processing them with defined manual rules and selecting suitable anomaly detection methods.

*6.2.3 Case 3: Detecting Unavailable Server.* In traditional monitoring, engineers adopt an agent to regularly check the availability of servers (e.g., one hour). We find that the abnormal status of servers could be identified more accurately and timely via the analysis of server system logs. In this case, the server is hung due to the occurrence of deadlock. We could detect the problem via the significant decrease in the number of total logs and some templates (pink bands in Figure 9(c)). It is because server hang will cause the tasks on the server to fail to run, so the number of system logs and some templates (each template generally represents a type of task) will be significantly reduced. For example, sessions cannot be created in the server (Template: "systemd: Starting Session * of user root") and the data collection agent does not work (Template: "dd.collector[*]: INFO (collector.py:*): Finished run *. Collection time: *s."). When engineers notice the problem, they will reboot the server to mitigate the incident. Thus the number of total logs and some templates would increase due to server restart (green bands in Figure 9(c)).

## 7 DISCUSSION

### 7.1 Lessons Learned

*There exists a gap between the research of algorithms and real application scenarios.* Log anomaly detection algorithm has been extensively studied in the literature. In the real world, however, it is critical to find suitable application scenarios for algorithms and solve practical problems. We want to answer how to apply log anomaly detection properly to service reliability management and what practical issues could be solved by log anomaly detection. Through our study, we find that log anomaly detection could assist in identifying illegal requests with access logs, detecting network failures with network device logs, detecting database problems with database logs, and so on.



```
2019-08-04T13:13:08.857: [Full GC 2019-08-
04T13:13:08.858: [CMS2019-08-04T13:13:18.180: [CMS-
concurrent-mark: 16.639/16.669 secs] [Times: user=38.45
sys=0.31, real=16.66 secs]
 (concurrent mode failure): 7864319K->7864319K(7864320K),
53.3661840 secs] 9279935K->7874993K(9279936K), [CMS
Perm : 239273K->239264K(524288K)], 53.3666690 secs]
[Times: user=62.76 sys=0.00, real=53.36 secs]
```

(a) Predicting OOM failure (b) Identifying illegal access (c) Detecting unavailable server

**Figure 9: Illustration of three cases**

*A single algorithm is usually not a panacea in practice.* In academic, log anomaly detection algorithms are solely evaluated on several public datasets, and the generality cannot be demonstrated. In real applications, a single algorithm cannot always perform well due to diverse situations, unsatisfactory data quality, and customized demand, which usually require human involvement and a combination of multiple algorithms to achieve promising performance. In particular, this argument can be extended to other tasks. For example, the patterns of time series are also diverse in practice, including seasonal, stationary and variable, which also requires different time series anomaly detection algorithms.

*Choose appropriate logs for anomaly detection.* Although various logs exist in practice, from infrastructure-layer to application-layer, it is unnecessary to detect anomalies on all log files. Monitoring all logs would waste too many computation resources and incur numerous alerts, and some alerts are false alarms or unimportant, resulting in alert fatigue for engineers. In general, we summarize several principles for selecting logs. 1) Select familiar and frequently used logs in daily operation and maintenance, so that the alerts are valuable and actionable for engineers in incident discovery and diagnosis. 2) Critical application logs reflecting the service quality directly and generic component logs (e.g., Oracle, DB2, and Hadoop) with rich expert knowledge are preferred. 3) Select well-formatted logs (e.g., single-line). In some log files, a log message is distributed in multiple lines (e.g., java dump logs), which brings significant challenges to process such logs.

*Human-computer interactive log analysis.* It is challenging to detect log anomalies automatically without any human involvement. Both log parsing and log anomaly detection require domain knowledge from experienced experts. For log parsing, engineers should get involved to transform some special logs into structured data (e.g., Nginx access log) and extracting some important variables for variable anomaly detection. For log anomaly detection, engineers need to get involved to decide what abnormal pattern the log could behave. Consequently, it is a formidable task for an algorithm to make such decisions without the help of domain experts.

## 7.2 Threats to Validity

*Implementation.* The internal threat to validity mainly lying in our study is the implementation of our system and compared approaches. To reduce this threat, two authors have carefully checked the code, and the implementation of compared approaches is based on the open-source library. In particular, we implemented them based on a matured framework, which has been presented in §4.2.

*Diversity of evaluation datasets.* Our experimental study used ten real-world log datasets from different service components, but the results might not represent other types of logs. It is challenging to evaluate the performance on all kinds of logs, and our datasets include various logs, from infrastructure-layer to application-layer. Thus we are confident that the results can demonstrate the efficacy of *LogAD*. In the future, we will reduce this threat by conducting an experimental study on more log datasets.

*Evaluation measurements.* Following existing work, we used precision/recall/F1-score as measurements. However, we did not study the sensitivity of parameters (e.g., window size). In the future, we will conduct parameter analysis and consider more comprehensive metrics such as precision-recall curve, to reduce this threat.

## 8 CONCLUSION

In this paper, we propose several significant practical challenges when applying log anomaly detection approaches in academic to practice, including the inability to deal with various logs and abnormal patterns, poor interpretability, and lack of domain knowledge. To help understand practical log anomaly detection, we conduct the first empirical study and an experimental study based on large-scale real-world data and obtain several key observations, supporting these challenges. Inspired by the findings, we propose a generic log anomaly detection system named *LogAD* to tackle these challenges together. *LogAD* contains three core ideas: incorporating multiple anomaly detection techniques to handle diverse abnormal patterns, importing a knowledge base fusing domain experience, and visually providing an interpretable report. The average F1-score of *LogAD* achieves 0.83, demonstrating its effectiveness. To our best knowledge, we are the first to investigate practical log anomaly detection in industrial scenarios. We believe that our work can provide some inspiration and guidance for practitioners and researchers to apply log anomaly detection to practice.

## ACKNOWLEDGMENTS

We thank Junjie Chen for his helpful discussions on this work. This work is supported by the National Key Research and Development Program of China (Grant No.2019YFE0105500), the State Key Program of National Natural Science of China under Grant 62072264, and the Beijing National Research Center for Information Science and Technology (BNRist) key projects.

## REFERENCES

[1] Apache Kafka: A distributed streaming platform.. https://kafka.apache.org. [Online; accessed 04-May-2021].

[2] BGL Dataset. https://www.usenix.org/cfdr-data.

[3] Elasticsearch. https://www.elastic.co/elasticsearch/. [Online; accessed 04-May-2021].

[4] Filebeat. https://www.elastic.co/beats/filebeat. [Online; accessed 04-May-2021].

[5] Flink, Scalable Stream and Batch Data Processing. https://flink.apache.org/. [Online; accessed 04-May-2021].

[6] InfluxDB. https://www.influxdata.com/. [Online; accessed 04-May-2021].

[7] Jensen–Shannon divergence. https://en.wikipedia.org/wiki/Jensen-Shannon_divergence. [Online; accessed 04-May-2021].

[8] loglizer. https://github.com/logpai/loglizer. [Online; accessed 04-May-2021].

[9] logparser. https://github.com/logpai/logparser. [Online; accessed 04-May-2021].

[10] Logstash. https://www.elastic.co/logstash. [Online; accessed 04-May-2021].

[11] NumPy. https://numpy.org/. [Online; accessed 04-May-2020].

[12] pandas. https://pandas.pydata.org/. [Online; accessed 04-May-2020].

[13] PyTorch. https://pytorch.org/. [Online; accessed 40-May-2021].

[14] Redis. https://redis.io/. [Online; accessed 04-May-2021].

[15] Rsyslog. https://www.rsyslog.com/. [Online; accessed 04-May-2021].

[16] scikit-learn. https://scikit-learn.org/.

[17] syslog-ng. https://www.syslog-ng.com/. [Online; accessed 04-May-2021].

[18] Hen Amar, Lingfeng Bao, Nimrod Busany, David Lo, and Shahar Maoz. 2018. Using finite-state models for log differencing. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 49–59. https://doi.org/10.1145/3236024.3236069

[19] Christophe Bertero, Matthieu Roy, Carla Sauvanaud, and Gilles Trédan. 2017. Experience report: Log mining using natural language processing and application to anomaly detection. In *2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 351–360. https://doi.org/10.1109/ISSRE.2017.43

[20] Junjie Chen, Xiaoting He, Qingwei Lin, Yong Xu, Hongyu Zhang, Dan Hao, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. 2019. An empirical investigation of incident triage for online service systems. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice*. 111–120. https://doi.org/10.1109/ICSE-SEIP.2019.00020

[21] Junjie Chen, Shu Zhang, Xiaoting He, Qingwei Lin, Hongyu Zhang, Dan Hao, Yu Kang, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. 2020. How Incidental are the Incidents? Characterizing and Prioritizing Incidents for Large-Scale Online Service Systems. In *35th IEEE/ACM International Conference on Automated Software Engineering*. 373–384. https://doi.org/10.1145/3324884.3416624

[22] Mike Chen, Alice X Zheng, Jim Lloyd, Michael I Jordan, and Eric Brewer. 2004. Failure diagnosis using decision trees. In *International Conference on Autonomic Computing, 2004. Proceedings.* IEEE, 36–43. https://doi.org/10.1109/ICAC.2004.1301345

[23] Rui Chen, Shenglin Zhang, Dongwen Li, Yuzhe Zhang, Fangrui Guo, Weibin Meng, Dan Pei, Yuzhi Zhang, Xu Chen, and Yuqing Liu. 2020. LogTransfer: Cross-System Log Anomaly Detection for Software Systems with Transfer Learning. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 37–47. https://doi.org/10.1109/ISSRE5003.2020.00013

[24] Min Du and Feifei Li. 2016. Spell: Streaming parsing of system event logs. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 859–864. https://doi.org/10.1109/ICDM.2016.0103

[25] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 1285–1298. https://doi.org/10.1145/3133956.3134015

[26] Qiang Fu, Jian-Guang Lou, Yi Wang, and Jiang Li. 2009. Execution anomaly detection in distributed systems through unstructured log analysis. In *2009 ninth IEEE international conference on data mining*. IEEE, 149–158. https://doi.org/10.1109/ICDM.2009.60

[27] Hossein Hamooni, Biplob Debnath, Jianwu Xu, Hui Zhang, Guofei Jiang, and Abdullah Mueen. 2016. Logmine: Fast pattern recognition for log analytics. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. 1573–1582. https://doi.org/10.1145/2983323.2983358

[28] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R Lyu. 2017. Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE International Conference on Web Services (ICWS)*. IEEE, 33–40. https://doi.org/10.1109/ICWS.2017.13

[29] Shilin He, Pinjia He, Zhuangbin Chen, Tianyi Yang, Yuxin Su, and Michael R Lyu. 2020. A Survey on Automated Log Analysis for Reliability Engineering. *arXiv preprint arXiv:2009.07237* (2020).

[30] Shilin He, Qingwei Lin, Jian-Guang Lou, Hongyu Zhang, Michael R Lyu, and Dongmei Zhang. 2018. Identifying impactful service system problems via log analysis. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 60–70. https://doi.org/10.1145/3236024.3236083

[31] Shilin He, Jieming Zhu, Pinjia He, and Michael R. Lyu. 2016. Experience Report: System Log Analysis for Anomaly Detection. In *27th IEEE International Symposium on Software Reliability Engineering, ISSRE 2016, Ottawa, ON, Canada, October 23-27, 2016*. IEEE Computer Society, 207–218. https://doi.org/10.1109/ISSRE.2016.21

[32] Kyle Hundman, Valentino Constantinou, Christopher Laporte, Ian Colwell, and Tom Soderstrom. 2018. Detecting spacecraft anomalies using lstms and non-parametric dynamic thresholding. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 387–395. https://doi.org/10.1145/3219819.3219845

[33] Xiaoyun Li, Pengfei Chen, Linxiao Jing, Zilong He, and Guangba Yu. 2020. 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, 92–103. https://doi.org/10.1109/ISSRE5003.2020.00018

[34] Yinglung Liang, Yanyong Zhang, Hui Xiong, and Ramendra Sahoo. 2007. Failure prediction in ibm bluegene/l event logs. In *Seventh IEEE International Conference on Data Mining (ICDM 2007)*. IEEE, 583–588. https://doi.org/10.1109/ICDM.2007.46

[35] Qingwei Lin, Hongyu Zhang, Jian-Guang Lou, Yu Zhang, and Xuewei Chen. 2016. Log clustering based problem identification for online service systems. In *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 102–111.

[36] Jinyang Liu, Jieming Zhu, Shilin He, Pinjia He, Zibin Zheng, and Michael R Lyu. 2019. Logzip: Extracting hidden structures via iterative clustering for log compression. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 863–873. https://doi.org/10.1109/ASE.2019.00085

[37] Jian-Guang Lou, Qiang Fu, Shengqi Yang, Ye Xu, and Jiang Li. 2010. Mining Invariants from Console Logs for System Problem Detection.. In *USENIX Annual Technical Conference*. 1–14.

[38] Weibin Meng, Ying Liu, Yichen Zhu, Shenglin Zhang, Dan Pei, Yuqing Liu, Yihao Chen, Ruizhi Zhang, Shimin Tao, Pei Sun, et al. 2019. LogAnomaly: Unsupervised Detection of Sequential and Quantitative Anomalies in Unstructured Logs.. In *IJCAI*. 4739–4745.

[39] Animesh Nandi, Atri Mandal, Shubham Atreja, Gargi B Dasgupta, and Subhrajit Bhattacharya. 2016. Anomaly detection using program control flow graph mining from execution logs. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 215–224. https://doi.org/10.1145/2939672.2939712

[40] Adam Oliner and Jon Stearley. 2007. What supercomputers say: A study of five system logs. In *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*. IEEE, 575–584. https://doi.org/10.1109/DSN.2007.103

[41] Tongqing Qiu, Zihui Ge, Dan Pei, Jia Wang, and Jun Xu. 2010. What happened in my network: mining network events from router syslogs. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. 472–484. https://doi.org/10.1145/1879141.1879202

[42] Ruben Sipos, Dmitriy Fradkin, Fabian Moerchen, and Zhuang Wang. 2014. Log-based predictive maintenance. In *Proceedings of the 20th ACM SIGKDD international conference on knowledge discovery and data mining*. 1867–1876. https://doi.org/10.1145/2623330.2623340

[43] Liang Tang, Tao Li, and Chang-Shing Perng. 2011. LogSig: Generating system events from raw textual logs. In *Proceedings of the 20th ACM international conference on Information and knowledge management*. 785–794. https://doi.org/10.1145/2063576.2063690

[44] Michail Vlachos, Philip Yu, and Vittorio Castelli. 2005. On Periodicity Detection and Structural Periodic Similarity. In *Proceedings of the 2005 SIAM International Conference on Data Mining, SDM 2005*. https://doi.org/10.1137/1.9781611972757.40

[45] X. Xia, W. Zhang, and J. Jiang. 2019. Ensemble Methods for Anomaly Detection Based on System Log. In *2019 IEEE 24th Pacific Rim International Symposium on Dependable Computing (PRDC)*. https://doi.org/10.1109/PRDC47002.2019.00034

[46] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael Jordan. 2009. Online system problem detection by mining patterns of console logs. In *2009 Ninth IEEE International Conference on Data Mining*. IEEE, 588–597. https://doi.org/10.1109/ICDM.2009.19

[47] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I Jordan. 2009. Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. 117–132. https://doi.org/10.1145/1629575.1629587

[48] R. B. Yadav, P. S. Kumar, and S. V. Dhavale. 2020. A Survey on Log Anomaly Detection using Deep Learning. In *2020 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*. 1215–1220. https://doi.org/10.1109/ICRITO48877.2020.9197818

[49] He Yan, Ashley Flavel, Zihui Ge, Alexandre Gerber, Dan Massey, Christos Papadopoulos, Hiren Shah, and Jennifer Yates. 2012. Argus: End-to-end service anomaly detection and localization from an isp's point of view. In *2012 Proceedings IEEE INFOCOM*. IEEE, 2756–2760.

[50] Lin Yang, Junjie Chen, Zan Wang, Weijing Wang, Jiajun Jiang, Xuyuan Dong, and Wenbin Zhang. 2021. Semi-supervised log-based anomaly detection via probabilistic label estimation. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 1448–1460. https://doi.org/10.1109/ICSE43902.2021.00130

[51] Kun Yin, Meng Yan, Ling Xu, Zhou Xu, Zhao Li, Dan Yang, and Xiaohong Zhang. 2020. Improving Log-Based Anomaly Detection with Component-Aware Analysis. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 667–671. https://doi.org/10.1109/ICSME46990.2020.00069

[52] Xiao Yu, Pallavi Joshi, Jianwu Xu, Guoliang Jin, Hui Zhang, and Guofei Jiang. 2016. Cloudseer: Workflow monitoring of cloud infrastructures via interleaved logs. *ACM SIGARCH Computer Architecture News* 44, 2 (2016), 489–502. https://doi.org/10.1145/2980024.2872407

[53] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, et al. 2019. Robust log-based anomaly detection on unstable log data. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 807–817. https://doi.org/10.1145/3338906.3338931

[54] Nengwen Zhao, Jing Zhu, Yao Wang, Minghua Ma, Wenchi Zhang, Dapeng Liu, Ming Zhang, and Dan Pei. 2019. Automatic and Generic Periodicity Adaptation for KPI Anomaly Detection. *IEEE Transactions on Network and Service Management* 16, 3 (2019), 1170–1183. https://doi.org/10.1109/INFOCOM41043.2020.9155219

[55] Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R. Lyu. 2019. Tools and Benchmarks for Automated Log Parsing. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP '19)*. IEEE Press, 121–130. https://doi.org/10.1109/ICSE-SEIP.2019.00021