Loghub: A Large Collection of System Log Datasets towards Automated Log Analytics

Shilin He  
The Chinese University of Hong Kong  
slhe@cse.cuhk.edu.hk

Pinjia He  
ETH Zurich, Switzerland  
pinjia.he@inf.ethz.ch

Jiemia Zhu  
Huawei Noah’s Ark Lab  
jmzhu@logpai.com

Michael R. Lyu  
The Chinese University of Hong Kong  
lyu@cse.cuhk.edu.hk

ABSTRACT
Logs have been widely adopted in software system development and maintenance because of the rich system runtime information they contain. In recent years, the increase of software size and complexity leads to the rapid growth of the volume of logs. To handle these large volumes of logs efficiently and effectively, a line of research focuses on intelligent log analytics powered by AI (artificial intelligence) techniques. However, only a small fraction of these techniques have reached successful deployment in industry because of the lack of public log datasets and necessary benchmarking upon them. To fill this significant gap between academia and industry and also facilitate more research on AI-powered log analytics, we have collected and organized loghub, a large collection of log datasets. In particular, loghub provides 17 real-world log datasets collected from a wide range of systems, including distributed systems, supercomputers, operating systems, mobile systems, server applications, and standalone software. In this paper, we summarize the statistics of these datasets, introduce some practical log usage scenarios, and present a case study on anomaly detection to demonstrate how loghub facilitates the research and practice in this field. Up to the time of this paper writing, loghub datasets have been downloaded over 15,000 times by more than 380 organizations from both industry and academia.

CCS CONCEPTS
• Software and its engineering → Software maintenance tools;  
• Computing methodologies → Machine learning approaches.

KEYWORDS
Log management, log dataset, log analysis, anomaly detection

1 INTRODUCTION
Logs have been widely adopted in software system development and maintenance. In industry, it is a common practice to record detailed software runtime information into logs, allowing developers and support engineers to track system behaviors and perform post-mortem analysis.

In general, logs are unstructured text printed by logging statements (e.g., printf(), Console.WriteLine()) in source code. A log message, as illustrated in the following example, records a specific system event with a set of fields: timestamp (the occurrence time of the event), e.g., 2008-11-09 20:46:55.556, verbosity level (the severity level of the event, e.g., INFO), and event description in free text.

The rich information recorded by logs enables developers to conduct a variety of log-based system management tasks, such as anomaly detection [16, 18, 30, 65], duplicate issue identification [14, 37, 53], usage statistics analysis [32], and program verification [6, 55]. For example, developers could inspect the log messages and analyze whether the system behaves as expected. However, software systems are evolving to large in scale and complex in structure. The volume of system logs is growing rapidly as well (e.g., 50 GB/h [45]), making manual log analysis become labor-intensive and time-consuming. To address this problem, a line of research [6, 14, 18, 30, 32, 37, 53, 55, 65] has targeted at automated log analytics powered by AI (artificial intelligence) techniques. These studies demonstrate that the use of AI techniques can greatly facilitate log analysis tasks by extracting critical information of runtime behaviors.

Figure 1 illustrates a typical framework for AI-powered log analytics. In the development phase, developers’ logging decisions can be guided by strategic logging practices (i.e., “where to log” [20, 68] and “what to log” [26, 34]) mined from high-quality software repositories. At system runtime, logs are collected and aggregated in a streaming manner. To reduce the volume of system logs, log compression techniques [41] could be further applied. In the operation and maintenance phase, logs need to be parsed into structured data with log parsing techniques [27], and then drive the AI modeling (e.g., deep learning) for a variety of log analysis tasks (e.g., anomaly detection [30], problem identification [29, 39]).

Along with this framework, many efforts have been devoted to improving AI techniques towards logging, log collection, log compression, log parsing, and log analysis. Many more approaches are being proposed as well. However, there is still a large gap between research and practice. First, researchers in this field often work on their own log data. Logs are scarce data in public for research, because companies are often reluctant to release their production data. Second, it is difficult and time-consuming for developers to implement the approaches and accurately reproduce the results.

To bridge this gap, this paper presents loghub [59], a large collection of system log datasets for AI-powered log analytics. Loghub
contains 17 log datasets (see Table 1 for details) generated by a wide range of systems, including distributed systems, supercomputers, operating systems, mobile systems, server applications, and standalone software. All these logs amount to 77 GB in total. In particular, some of the logs are production data released from previous studies, while some others are collected from real systems in our lab environment. Among these log datasets, five of them are labeled (e.g., normal or abnormal, alerts or not alerts), which are amendable to studies for anomaly detection and duplicate issues identification. Additionally, other datasets could facilitate the research on log parsing, log compression, and unsupervised methods for anomaly detection. Since the first release of these logs, they have been downloaded 15,000+ times by more than 380 organizations from both industry (35%) and academia (65%). We envision that loghub could serve as the open benchmarks towards future research and practice for AI-powered log analytics. In summary, our paper makes the following contributions:

- We collect and organize a large collection of log datasets generated by a wide variety of systems, which are valuable for AI-powered log analytics.
- We introduce some practical usage scenarios of loghub, and demonstrate a case study on anomaly detection.
- Since the release of loghub datasets, they have made a measurable impact to the community, benefiting research in over 380 organizations from both industry and academia.

We next introduce the loghub datasets in Section 2, present some typical log usage scenarios in Section 3. Section 4 demonstrates the use of loghub in a case study for anomaly detection. Section 5 reviews the related work and finally Section 6 concludes the paper.

2 LOGHUB DATASETS

Loghub maintains a collection of system logs, which are freely accessible for research purposes. Some of the logs are production data released from previous studies, while some others are collected from real systems in our lab environment. Wherever possible, the logs are not sanitized, anonymized or modified in any way. All these logs amount to over 77 GB in total. Due to its large scale, we thus host a small sample for each dataset in our project website [59], while the full version can be requested through Zenodo [60], an open dataset sharing website.

Table 1 presents an overview of the loghub datasets with some details about the description, time span, #messages, data size, and the label information. Specifically, time span indicates the time range that the logs are collected. #Messages denotes the total number of log messages in a dataset. Data size shows the uncompressed log volume. The “labeled” column indicates whether a dataset is manually labeled with anomaly ground truth information.

There are two categories of log datasets: labeled and unlabeled. Logs in labeled datasets contain labels for specific log analysis tasks (e.g., anomaly detection and duplicate issues identification). For example, in the labeled HDFS datasets, the labels indicate whether the system operations on an HDFS block is abnormal. Thus, developers could utilize the labeled HDFS dataset to evaluate their anomaly detection approaches. In loghub, 5 log datasets are labeled, while 12 log datasets are unlabeled. Note that unlabeled log datasets are also useful for the evaluation of AI-powered log analytics, such as log parsing, log compression, and unsupervised methods towards log analysis. The details of each log dataset in loghub are introduced as follows.

2.1 Distributed Systems

HDFS. HDFS [22] is the Hadoop Distributed File System designed to run on commodity hardware. Due to the popularity of HDFS, it has been widely studied in recent years. We provide two sets of HDFS logs in loghub: HDFS-1 and HDFS-2. HDFS-1 is generated in a 203-nodes HDFS using benchmark workloads, and manually labeled through handcrafted rules to identify the anomalies. The logs are sliced into traces (i.e., log sequences) according to block IDs. Then each trace associated with a specific block ID is assigned a ground truth label: normal or abnormal. Additionally, HDFS-1 also provide the specific anomaly type information, while allows research on duplicate issues identification. HDFS-2 is collected by aggregating logs from the HDFS cluster in our lab at CUHK, which comprises one name node and 32 data nodes. The logs are aggregated at the node level. The logs have a huge size (over 16 GB) and are provided as-is without further modification or labelling, which may involve both normal and abnormal cases.

Hadoop. Hadoop [23] is a big data processing framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. Due to the increasing importance of Hadoop in industry, it has been widely studied in the literature. The logs are generated from a Hadoop cluster with 46 cores across five machines. Each machine has Intel(R) Core(TM) i7-3770 CPU and 16 GB RAM. Two testing applications are executed: WordCount and PageRank. Firstly, the applications are run without injecting any failure. Then, in order to simulate service
### Table 1: Loghub Dataset Details.

| System         | Description                   | Time Span | #Messages     | Data Size   | Labeled |
|----------------|-------------------------------|-----------|---------------|-------------|---------|
| **Distributed systems** |                               |           |               |             |         |
| HDFS           | Hadoop distributed file system log | 38.7 hours | 11,175,629    | 1.47GB      | Yes     |
|                | Hadoop distributed file system log | N.A.      | 71,118,073    | 16.06GB     | No      |
| Spark          | Spark job log                  | N.A.      | 33,236,604    | 2.75GB      | No      |
| Zookeeper      | ZooKeeper service log          | 26.7 days | 74,380        | 9.95MB      | No      |
| OpenStack      | OpenStack infrastructure log   | N.A.      | 207,820       | 58.61MB     | Yes     |
| **Supercomputers** |                               |           |               |             |         |
| BGL            | Blue Gene/L supercomputer log  | 214.7 days| 4,747,963     | 708.76MB    | Yes     |
| HPC            | High performance cluster log   | N.A.      | 433,489       | 32.00MB     | No      |
| Thunderbird    | Thunderbird supercomputer log  | 244 days  | 211,212,192   | 29.60GB     | Yes     |
| **Operating systems** |                               |           |               |             |         |
| Windows        | Windows event log             | 226.7 days| 114,608,388   | 26.09GB     | No      |
| Linux          | Linux system log              | 263.9 days| 25,567        | 2.25MB      | No      |
| Mac            | Mac OS log                    | 7.0 days  | 117,283       | 16.09MB     | No      |
| **Mobile systems** |                               |           |               |             |         |
| Andriod        | Andriod framework log         | N.A.      | 1,555,005     | 183.37MB    | No      |
| HealthApp      | Health app log                | 10.5 days | 253,395       | 22.44MB     | No      |
| **Server applications** |                               |           |               |             |         |
| Apache         | Apache web server error log   | 263.9 days| 56,481        | 4.90MB      | No      |
| OpenSSH        | OpenSSH server log            | 28.4 days | 655,146       | 70.02MB     | No      |
| **Standalone software** |                           |           |               |             |         |
| Proxifier      | Proxifier software log        | N.A.      | 21,329        | 2.42MB      | No      |

Failures in the production environment, the following deployment failures are injected: (1) machine down: during application runtime, turn off one server to simulate the machine failure; (2) network disconnection: disconnect one server from the network to simulate the network connection failure; and (3) disk full: during application runtime, manually fill up one server’s hard disk to simulate the disk full failure. The labels of different failures are provided, making the data amenable to duplicate issues identification research.

**Spark.** Apache Spark [57] is a unified analytics engine for big data processing, with built-in modules for streaming, SQL, machine learning and graph processing. Currently, Spark has been widely deployed in industry. This dataset was collected by aggregating logs from the Spark system in our lab at CUHK, which comprises a total of 32 machines. The logs are aggregated at the machine level. The logs have a huge size (over 2 GB) and are provided as-is without further modification or labelling, which involve both normal and abnormal application runs.

**Zookeeper.** ZooKeeper [69] is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. The log dataset was collected by aggregating logs from the ZooKeeper service in our lab at CUHK, which comprises a total of 32 machines. The logs cover a time period of 26.7 days.

**OpenStack.** OpenStack [49] is a cloud operating system that controls large pools of compute, storage, and networking resources throughout a datacenter. This dataset was generated on CloudLab [12], a flexible, scientific infrastructure for research on cloud computing. Both normal logs and abnormal cases with failure injection are provided, making the data amenable to anomaly detection research.

### 2.2 Supercomputers

**BGL.** BGL is an open dataset of logs collected from a BlueGene/L supercomputer system at Lawrence Livermore National Labs (LLNL) in Livermore, California, with 131,072 processors and 32,768 GB memory [35]. The log contains alert and non-alert messages identified by alert category tags. In the first column of the log, "-" indicates
Windows. This log dataset was collected by aggregating a number of logs from a lab computer running Windows 7. The original logs were located at C:/Windows/Logs/CBS. CBS (Component Based Servicing) is a componentization architecture in Windows, which works at the package/update level. The CBS architecture is far more robust and secure than the installers in previous operating systems. Users benefit from a more complete and controlled installation process that allows updates, drivers and optional components to be added while simultaneously mitigating against instability issues caused by improper or partial installation. The logs have a huge size (over 27 GB) and span a period of 226.7 days.

Linux. Linux logs are usually located at /var/log/. The dataset was collected from /var/log/messages on a Linux server over a period of 263.9 days, as part of the Public Security Log Sharing Site project [11].

Mac. We collected the MacOS logs from /var/log/system.log on a personal Macbook after 7 days of use. The log records the user activities on the Mac OS.

2.4 Mobile Applications

Android. Android [3] is a popular mobile operating system developed by Google and has been used by many smart devices. However, Android logs are rarely available in public for research purposes. We provide a log file, which was generated when we test an instrumented Android smartphone in our lab.

HealthApp. HealthApp is a mobile application for Android devices. We collected the application logs from an Android smartphone after 10+ days of use.

2.5 Server Applications

Apache. Apache HTTP Server [4] is one of the most popular web servers. Apache servers usually generate two types of logs: access logs and error logs. This dataset provides an error log for the purpose of research on anomaly detection and diagnosis. The log file was collected from a Linux system running Apache Web server, as part of the Public Security Log Sharing Site project [11].

OpenSSH. OpenSSH is the premier connectivity tool for remote login with the SSH protocol. We collected the log from an OpenSSH server in our lab over a period of 28+ days.

2.6 Standalone Software

Proxifier. Proxifier [51] is a software program, allowing network applications that do not support working through proxy servers to operate through a SOCKS or HTTPS proxy and chains. We collected the Proxifier logs from a desktop computer in our lab.

3 USAGE OF LOGHUB DATASETS

In this section, we present some common usage scenarios of the loghub datasets.

3.1 Overview

Since its release, loghub has attracted the attention of not only large companies such as IBM, Microsoft, Huawei, Nvidia, MasterCard, Adobe, BMW, and Samsung, but also some startup companies focusing on log-related products, including Elastic.co, Splunk, Rapid7, Element AI, White Ops, Unomaly.com, and Ascend.io [61]. Many top universities has also requested the loghub dataset, including University of Oxford, UC Berkeley, Harvard University, ETH University of Cambridge, etc. To date, loghub has been downloaded 15,000+ times by more than 380 organizations from both industry (35%) and academia (65%). After analyzing the project information collected from about 500 data requests, we manually categorize the log usage scenarios and present the distribution in Figure 2. It is worth noting that due to the limited information provided by users, we could only provide a rough categorization for them. We denote it an unknown category if the user input is not clear. We can see that loghub datasets can potentially facilitate 23 different categories of research and study purposes. The top 5 usage scenarios are anomaly detection, log analysis, security, log parsing, and education. In particular, log analysis may cover the spectrum of log related tasks, but this has not been clearly specified by users. Education indicates the use cases about course projects and thesis projects. Here, we do not intend to expand all usage scenarios shown in the figure, but the variety of them have already confirmed the practical importance of our loghub datasets.

In the following, we present the details of four representative application scenarios that have been widely studied in the literature and describe how loghub can be used in these tasks, including log parsing, log compression, anomaly detection, and duplicate issues identification.

3.2 Log Parsing

Most of the AI-powered log analysis approaches require structured input data, such as a list of system events with event IDs or a matrix. However, software logs are often unstructured texts, containing several fields and natural language descriptions written by developers. Thus, log parsing is a crucial step in AI-powered log analytics that transforms unstructured log messages into structured system events.

However, as the volume of logs increases rapidly, traditional parsing approaches that largely rely on manual parsing rules construction becomes labor-intensive and inefficient. To address this problem, recent research has proposed various data-driven log parsers [15, 19, 24, 28, 31, 43, 46, 48, 56, 58, 63, 64], which automatically label an unstructured log message with corresponding system event ID. Typically, log parsing is modeled as a clustering problem,
where log messages describing the same system event should be clustered into the same group. The common tokens in all the log messages in the same group is regarded as the system event or event template. The research problem of log parsing is how to accurately and efficiently separate the unstructured log messages into different clusters by designing similarity metrics for log messages and novel clustering approaches. By clustering log messages into groups, log parsers can summarize the corresponding system events and match each log message with an event ID. The structured logs (i.e., log messages with event ID) could be easily transformed into a matrix or directly utilized by log analysis algorithms.

To evaluate the accuracy and efficiency of log parsing approaches, we need: (1) a large volume of logs and (2) logs generated by a variety of systems. Loghub contains 17 log datasets collected from 6 categories of systems. Besides, all the logs amount to over 77 GB. Thus, datasets in loghub can be employed in the experiments to evaluate the parsing accuracy and efficiency of different log parsing approaches.

### 3.3 Log Compression

Logs can be used in various system maintenance tasks, and thus they often need to be stored for a long time (e.g., a year or more) in practice. As the explosion of log size in recent years, archiving system logs is consuming a large amount of storage space, which leads to high cost of electrical power. General compression approaches do not work well on log compression because they do not consider the inherent structure of log messages. To achieve higher compression ratio, recently, a line of research [13, 17, 25, 38, 52] has proposed compression approaches specialized for system logs.

Log compression can be modeled as a frequent pattern mining problem. Existing approaches focus on finding inherent structure information of log messages (e.g., repetitive text). In particular, these approaches provide different strategies to detect repetitive text, such as utilizing the common format of logs generated by a specific system [13]. The research problem of log compression is how to achieve efficient and lossless compression with high compression rate.

To evaluate the accuracy and efficiency of log compression approaches, similar to the evaluation of log parsers, we need a large volume of logs collected from diverse systems. Thus, all the datasets in loghub can facilitate the evaluation of log compression approaches.

### 3.4 Anomaly Detection

Modern systems have become large-scale in size and complex in structure. An increasing number of these systems are expected to run on a 24 × 7 basis serving millions of users globally. Any non-trivial downtime of them could lead to enormous revenue loss [47, 62]. Thus, to enhance the reliability of modern systems, a line of recent research [7, 16, 18, 30, 39, 42, 65] has focused on log-based anomaly detection approaches that report potential abnormal system behaviors by analyzing system runtime logs.

The anomaly detection problem is usually modeled as a binary classification problem. The input is a list of structured system events or a matrix, while the output is a list of labels indicating whether an instance (e.g., an event or a time period) is abnormal. There are mainly two categories of log-based anomaly detection approaches: *unsupervised* [39, 42, 65] and *supervised* [7, 16, 18]. The research problem of log-based anomaly detection is how to accurately detect the anomalies based on system logs. Towards this end, F-measure (i.e., F1 score) [1, 44], a commonly-used evaluation metric for classification algorithms, is employed as the accuracy metric.

To evaluate the accuracy of log-based anomaly detection approaches, we need log datasets that contain anomaly labels for instances (e.g., whether a time period is regarded as anomaly). Loghub contains 5 labeled log datasets, which can be used in the experiments to evaluate the accuracy of diverse anomaly detection approaches.

### 3.5 Duplicate Issues Identification

To enhance system reliability, one important task for developers is to handle user-reported operational issues efficiently. An operational issue is a system problem reported by users. When a user of Amazon EC2 [2] finds that her node becomes extremely slow, she will report the node slowness as an issue to Amazon. To handle an operational issue, developers need to mainly inspect the runtime logs to understand the system operations, which is time-consuming. Thus, to facilitate the issue handling process, recent research has proposed duplicate issues identification techniques [14, 37, 53] to alleviate unnecessary manual effort.
Table 2: Summary of Log-based Anomaly Detection Tools.

| Anomaly Detection Approach | Technique          | Mode        | Industrial Use |
|-----------------------------|--------------------|-------------|----------------|
| SVM                         | Classification     | Supervised  | IBM            |
| Decision Tree               | Classification     | Supervised  | eBay           |
| LR                          | Classification     | Supervised  | Microsoft      |
| Clustering                  | Clustering         | Unsupervised| Microsoft      |
| Invariants Mining           | Execution Flow     | Unsupervised| Microsoft      |
| PCA                         | Dimension Reduction| Unsupervised| Google         |
| One-Class SVM               | Classification     | Unsupervised| Microsoft      |
| Isolation Forest            | Proper Binary Tree | Unsupervised| H2O.ai         |
| LOF                         | Clustering         | Unsupervised| N.A.           |

The duplicate issue identification problem can be modeled as a clustering problem, while the duplicate issues are clustered into the same group based on the corresponding log messages. If the log messages of two issues demonstrate similar patterns (e.g., occurrence frequency, order), the two issue will be clustered in to the same group. The research problem of log-based duplicate issues identification is how to accurately separate the log messages (i.e., log sequences) of different issues into clusters.

To evaluate the accuracy of log-based duplicate issues identification approaches, log datasets that contain the issue categories are needed. Loghub contains 2 labeled log datasets (i.e., HDFS-1 and Hadoop) that provide this information. Thus, loghub could be used in this regard.

4 CASE STUDY ON ANOMALY DETECTION

In this section, we demonstrate the usage of loghub by a case study on benchmarking existing log-based anomaly detection approaches.

4.1 Existing Log-based Anomaly Detection Approaches

We have evaluated the performance of 9 log-based anomaly detection tools, which are illustrated in Table 2. There are mainly two categories of anomaly detection approaches: supervised and unsupervised.

Supervised approaches (Decision Tree [10], SVM [36], and LR [7]) require training data that contain labels indicating whether an instance is an anomaly. A classifier is trained based on the labeled data for anomaly detection. Supervised approaches are used when there are decent amount of both normal and abnormal labeled data. Unsupervised approaches are based on different techniques, including classification (LR [54]), isolation via proper binary tree (Isolation Forest [40]), dimension reduction (PCA [65]), execution flow mining (Invariant Mining [42]), and clustering (LOF [8] and Clustering [39]). The core idea of unsupervised approach is to learn the common patterns in logs or log sequences, and report instances the deviating instances as anomalies. In practice, labeled data are often lacking, because (1) anomalies rare occur in real-world systems and (2) data labeling is label-intensive and time-consuming. Thus, unsupervised methods are more applicable in real-world production environment.

4.2 Benchmarking on Loghub

To evaluate the accuracy of anomaly detection approaches, we use precision, recall, and F-measure (i.e., F1 score), which are the most commonly-used metrics. The definitions of the metrics are as follows:

$$\text{Precision} = \frac{\#\text{Anomalies detected}}{\#\text{Anomalies reported}}$$

$$\text{Recall} = \frac{\#\text{Anomalies detected}}{\#\text{All anomalies}}$$

$$F\text{-measure} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

where #Anomalies detected is the number of true anomalies detected; #Anomalies reported is the number of anomalies reported; and #All anomalies is the number of true anomalies in the dataset.

All the anomaly detection approaches are evaluated on the labeled HDFS dataset. This dataset records the system operations on different HDFS blocks. After log parsing and some postprocessing, we can obtain the input: a block-ID-by-event count matrix. Each row of the matrix represents the system operations on a specific block, while each column represents the frequency of occurrence of a system event during runtime.

Figure 3: Accuracy of Supervised Anomaly Detection Approaches

The experimental results are illustrated in Figure 3 and Figure 4. Decision Tree obtains the highest recall (0.99), F-measure (0.99), and the second-highest precision (0.99). The supervised approaches achieve better results. This is because the supervised approaches are trained on labeled data. Additionally, among the unsupervised approaches, this is because the supervised approaches are trained on labeled data. Additionally, among the unsupervised approaches, Invariants Mining approach has the best accuracy. Clustering obtains higher precision (1.00) than Invariants Mining. However, its recall (0.72) is much lower. Although One-Class SVM obtain high precision (0.99), its recall is too low (0.22), leading to low F-measure. This is because One-Class SVM is too conservative on reporting anomalies.
In additional to the labeled HDFS dataset used to evaluate anoamly detection method in this section, loghub also provides other 4 la- beled datasets. Thus, in practice, developers could evaluate an anomaly detection method on different datasets and choose the most suitable one.

4.3 Remaining Questions and Challenges

After benchmarking existing anomaly detection approaches on loghub, we find some remaining questions and challenges, which will be discussed in this session. First, the unsupervised approaches are not as accurate as the supervised approaches. Thus, an accurate unsupervised anomaly detection approach is highly in demand. Second, in practice, the number of anomalies is much less than that of normal instances. In industry, a system may only encounter 1 anomaly in a year, which makes supervised approaches ineffective. Thus, how to design an anomaly detection approach that does not require historical abnormal instances remains an important and challenging problem. Additionally, current anomaly detection approaches are all log sequences-based, which provide limited help to developers on further diagnosis, such as root cause analysis. Lastly, most of the existing approaches will only report whether an instance is an anomaly without other information. Thus, there is a lack of visualization tools that help developers understand the reported anomalies.

5 RELATED WORK

Logging Practice. In practice, there is a lack of rigorous guide and specifications on developer logging behaviors. To address this problem, a line of recent empirical research has focused on studying the logging practice of high-quality software [5, 9, 21, 33, 50, 66, 67]. Additionally, AI-powered strategic logging practice has also been widely studied in recent years with a focus on “where to log” and “what to log”. Where-to-log: In our previous work [68], we proposed a “learning to log” framework, which provides guidance on whether developers should put logging statement in a code snippet. What-to-log: Li et al. [34] developed an log verbosity level suggestion technique based on ordinal regression model. In [26], we characterize the natural language descriptions in logging statements and explore the potential of automated description text generation for logging statements. AI-powered strategic logging practices is an important component in AI-powered log analytics, where different practice will lead to different software logs in runtime.

Log Compression. Different from general natural language text, software logs have specific inherent structure (e.g., many logs are printed by the same logging statement). Thus, to achieve better compression rate, compression approaches specialized for log files have been well studied [13, 17, 25, 38, 52]. For example, Comprehensive Log Compression (CLC) [25] and Differentiated Semantic Log Compression (DSLIC) [52] identify repetitive items by domain knowledge. Multi-level Log Compression (MLC) [17] detects redundant texts in log files by dividing log messages into groups based on text similarity. All the datasets in loghub can facilitate the evaluation of log compression approaches.

Log Parsing. Most automated and effective log analysis techniques require structured data as input. Therefore, log parsing is crucial in AI-powered log analytics, which transforms unstructured log messages into structured system events. In recent years, log parsers based on various techniques have been proposed. (1) Frequent pattern mining: SLCT [63], LFA [48], and LogCluster [64] regard log event templates as a set of constant tokens that occur frequently in log. (2) Clustering: a line of log parsing studies model it as a clustering problem and design specialized clustering algorithms accordingly. Typical parsers in this category include LKE [19], LogSig [58], LogMine [24], SHISO [46], and LenMa [56]. (3) Heuristics: AEL [31], IPLoM [43], and Drain [28] parse log messages by heuristic rules inspired by unique characteristics of software logs. (4) Others: Some other methods exist, such as Spell [15] that is based on longest common subsequence. All the datasets in loghub can be employed to evaluate log parsing methods.

Log Analysis. Log analysis has been studied for decades to facilitate effective and efficient system maintenance. Typical log analysis tasks include anomaly detection [16, 18, 30, 65], duplicate issue identification [14, 37, 53], usage statistics analysis [32], and program verification [6, 55], most of which design or adopt AI algorithms. For example, Xu et al. [65] employs principal component analysis (PCA), which is a dimension reduction algorithm, to detect potential anomalies in large-scale distributed systems. Du et al. [16] proposed an anomaly detection and diagnosis approach based on a deep neural network model utilizing Long Short-Term Memory (LSTM). Shang et al. [55] design a clustering algorithm to verify the deployment of big data applications. The 5 labeled datasets in loghub can be used to evaluate various log analysis methods.

6 CONCLUSION AND FUTURE WORK

This paper describes loghub, a large collection of log datasets for AI-powered log analytics. Loghub contains 17 log datasets where all the logs amount to over 77 GB. 4 log management or analysis tasks that could be evaluated on loghub are introduced. We envision loghub website acting as a general platform for datasets, benchmarking, and feedbacks from both academia and industry on AI-powered log analysis. Since the first release of loghub, it has been downloaded 15,000+ times by more than 380 organizations.
In the future, we plan to collect more labeled log datasets to facilitate the evaluation of more log analysis tasks. Additionally, we will release open-source log analysis toolkits (e.g., log-based anomaly detection toolkit) and accompanying benchmarking studies using loghub.

REFERENCES

[1] 2019. F1 score. https://en.wikipedia.org/wiki/F1_score
[2] Amazon. 2019. Amazon EC2. https://aws.amazon.com/tw/ec2/
[3] Android. 2019. Android. https://www.android.com
[4] Apache. 2019. Apache HTTP Server. https://httpd.apache.org
[5] Titus Barik, Robert DeLine, Steven Drucker, and Danyel Fisher. 2016. The bones of Apache. 2019.
[6] Peter Bodik, Moises Goldszmidt, Armando Fox, Dawn B Woodard, and Hans Andersen. 2010. Fingerprinting the datacenter: automatic classification of performance crises. In Proceedings of the 13th European Conference on Computer Systems (EuroSys). ACM, 111–124.
[7] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. 2000. LOF: identifying density-based local outliers. In ACM sigmod record. Vol. 29. ACM, 93–104.
[8] Boyuan Chen and Zhen Ming Jack Jiang. 2017. Characterizing logging practices in Java-based open source software projects: a replication study in Apache Software Foundation. Empirical Software Engineering, 22, 1 (2017), 330–374.
[9] Qiang Fu, Jian-Guang Lou, Yi Wang, and Jiang Li. 2009. Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis. In ICDM'09: Proc. of the 9th International Conference on Data Mining, Miami, Florida, USA, 6-9 December 2009. 149–158.
[10] Qiang Fu, Jian-Guang Lou, Yi Wang, and Jiang Li. 2009. Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis. In ICDM'09: Proc. of the 9th International Conference on Data Mining, Miami, Florida, USA, 6-9 December 2009. 149–158.
[11] Q. Fu, J. Zhu, W. Hu, J. Lou, R. Ding, Q. Lin, D. Zhang, and T. Xie. 2014. Where Do Developers Log? An Empirical Study on Logging Practices in Industry. In ICSF '14: Companion Proc. of the 36th International Conference on Software Engineering Software Engineering, 24–33.
[12] Hadoop. 2019. Hadoop. https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html
[13] Hadoop. 2019. Hadoop. https://hadoop.apache.org
[14] Shilin He, Jieming Zhu, Pinjia He, and Michael R. Lyu. Knowledge Management. ACM, 1573–1582.
[15] Kimmo Hätinen, Jean-François Boulicaut, Mika Klemettinen, Markus Miettinen, and Cyrille Masson. 2003. Comprehensive Log Compression with Frequent Pattern Mining. In Data Warehousing and Knowledge Discovery, 5th International Conference, DaWaK 2003, Prague, Czech Republic, September 3-5,2003, Proceedings. 360–370.
[16] Pinjia He, Zhuangbin Chen, Shilin He, and Michael R. Lyu. 2018. Characterizing the natural language descriptions in software logging statements. In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ACM, 178–189.
[17] Pinjia He, Jieming Zhu, Shilin He, Jian Li, and Michael R. Lyu. 2016. An Evaluation Study on Log Parsing and Its Use in Log Mining. In 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2016, Toulouse, France, June 28 - July 1, 2016. 654–661.
[18] P. He, J. Zhu, Z. Zheng, and M. R. Lyu. 2017. Drain: An Online Log Parsing Approach with Fixed Depth Tree. In ICWS'17: Proc. of the 24th International Conference on Web Services.
[19] 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.
[20] S. He, J. Zhu, P. He, and M.R. Lyu. 2016. Experience Report: System Log Analysis for Anomaly Detection. In ESREF 16: Proc. of the 27th International Symposium on Software Reliability Engineering.
[21] Zhen Ming Jiang, Ahmed E Hassan, Parminder Flora, and Gilbert Hamann. 2008. Abstracting execution logs to execution events for enterprise applications (short paper). In 2008 The Eighth International Conference on Quality Software. IEEE, 184–186. Issue 4.
[22] George Lee, Jimmy J. Lin, Chuang Liu, Andrew Lorek, and Dimitry V. Ryaboy. 2012. The Unified Logging Infrastructure for Data Analytics at Twitter. PVLDB 5, 12 (2012), 1771–1780. https://doi.org/10.4135/2367502367516
[23] H. Li, T. Chen, W. Shang, and A. E. Hassan. 2017. Studying Software Logging Using Topic Models. Empirical Software Engineering (2017).
[24] H. Li, W. Shang, and A. E. Hassan. 2017. Which log level should developers choose for a new logging statement? Empirical Software Engineering 22 (2017), 1684–1716. Issue 4.
[25] Yingliang Liang, Yanyong Zhang, Anand Savasubramaniam, Ramendu K Sahoo, Jose Moreira, and Manish Gupta. 2005. Filtering failure logs for a bluegene/l prototype. In 2005 International Conference on Dependable Systems and Networks (DSN’05). IEEE, 476–485.
[26] Yingliang Liang, Yanyong Zhang, Hui Xiong, and Ramendu Sahoo. 2007. Failure prediction in ibm bluegene/l event logs. In The 7th IEEE International Conference on Data Mining (ICDM). IEEE, 583–588.
[27] Meng-Hui Lim, Jian-Guang Lou, Hongyu Zhang, Qiang Fu, Andrew Beng Jin Teoh, Qingwei Lin, Rui Ding, and Dongmei Zhang. 2014. Identifying recurrent and unknown performance issues. In 2014 IEEE International Conference on Data Mining (ICDM). IEEE, 320–329.
[28] Hao Lin, Jingyu Zhou, Bin Yao, Minyi Guo, and Jie Li. 2015. Cowic: A Column-Wise Independent Compression for Log Stream Analysis. In 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID). 21–30.
[29] Qingwei Lin, Hongyu Zhang, Jian-Guang Lou, Yu Zhang, and Xuewei Chen. 2016. Log clustering based on problem identification for online service systems. In Proceedings of the 38th International Conference on Software Engineering Companion Software Engineering (ASE). IEEE, 863–873.
[30] 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. 1–13.
[31] Adetokunbo Makanju, A. Nur Zincir-Heywood, and Evangelos E. Milios. 2009. Clustering event logs using iterative partitioning. In Proceedings of the 38th International Conference on Software Engineering Companion Software Engineering (ASE). IEEE, 863–873.
[32] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation forest. In Proceedings of the 5th European Conference on Computer Systems (EuroSys). ACM, 102–111.
[33] Zhen Ming Jiang, Ahmed E Hassan, Parminder Flora, and Gilbert Hamann. 2008. Abstracting execution logs to execution events for enterprise applications (short paper). In 2008 The Eighth International Conference on Quality Software. IEEE, 184–186.
[34] Polly Mosendz. 2014. http://log-sharing.dreamhosters.com/
[35] CloudbLab. 2019. CloudbLab. https://cloudblab.us/
[36] Sebastian Decowics and Szymon Grabowski. 2008. Sub-atomic field processing for improved web log compression. In Modern Problems of Radio Engineering, Telecommunications and Computer Science, 2008 Proceedings of International Conference on. IEEE, 551–556.
[37] Meng-Hui Lim, Jian-Guang Lou, Hongyu Zhang, Qiang Fu, Andrew Beng Jin Teoh, Qingwei Lin, Rui Ding, and Dongmei Zhang. 2014. Identifying recurrent and unknown performance issues. In 2014 IEEE International Conference on Data Mining (ICDM). IEEE, 320–329.
[38] Shilin He, Jieming Zhu, Pinjia He, and Michael R. Lyu. 2018. Characterizing the natural language descriptions in software logging statements. In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ACM, 178–189.
[39] Pinjia He, Jieming Zhu, Shilin He, Jian Li, and Michael R. Lyu. 2016. An Evaluation Study on Log Parsing and Its Use in Log Mining. In 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2016, Toulouse, France, June 28 - July 1, 2016. 654–661.
[40] Polly Mosendz. 2014. When It Goes Down, Facebook Loses $24,420 Per Minute. https://www.theatlantic.com/technology/archive/2014/10/facebook-
Loghub: A Large Collection of System Log Datasets
towards Automated Log Analytics

[48] Meiyappan Nagappan and Mladen A Vouk. 2010. Abstracting log lines to log event types for mining software system logs. In 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010). IEEE, 114–117.

[49] OpenStack. 2019. OpenStack. https://www.openstack.org

[50] Antonio Pecchia, Marcello Cinque, Gabriella Carrozza, and Domenico Cotroneo. 2015. Industry practices and event logging: Assessment of a critical software development process. In Proceedings of the 37th International Conference on Software Engineering-Volume 2. IEEE Press, 169–178.

[51] Proxifier. 2019. Proxifier. https://www.proxifier.com

[52] Balazs Raz and András Lukacs. 2004. High Density Compression of Log Files. In 2004 Data Compression Conference (DCC 2004), 23-25 March 2004, Snowbird, UT, USA, 557.

[53] Mohamed Sami Rakha, Cor-Paul Bezemer, and Ahmed E Hassan. 2018. Revisiting the performance evaluation of automated approaches for the retrieval of duplicate issue reports. IEEE Transactions on Software Engineering 44, 12 (2018), 1245–1268.

[54] Bernhard Schölkopf, John C Platt, John Shawe-Taylor, Alex J Smola, and Robert C Williamson. 2001. Estimating the support of a high-dimensional distribution. Neural Computation 13, 7 (2001), 1443–1471.

[55] Weiyi Shang, Zhen Ming Jiang, Hadi Hemmati, Bram Adams, Ahmed E. Hassan, and Patrick Martin. 2013. Assisting developers of big data analytics applications when deploying on hadoop clouds. In 35th International Conference on Software Engineering, ICSE ’13, San Francisco, CA, USA, May 18-26, 2013. 402–411.

[56] Keiichi Shima. 2016. Length matters: Clustering system log messages using length of words. arXiv preprint arXiv:1611.03213 (2016).

[57] Spark. 2019. Spark. https://spark.apache.org

[58] Liang Tang, Tao Li, and Chang-Shing Perng. 2011. LogSig: generating system events from raw textual logs. In Proceedings of the 20th ACM Conference on Information and Knowledge Management, CIKM 2011, Glasgow, United Kingdom, October 24-28, 2011. 785–794.

[59] LogPAI Team. 2019. Loghub: A large collection of system log datasets for AI-powered log analytics. https://github.com/logpai/loghub

[60] LogPAI Team. 2019. Loghub: A large collection of system log datasets for AI-powered log analytics. https://zenodo.org/record/1596245#.XMMZ1dv7S-Y

[61] LogPAI Team. 2019. Organizations that request loghub datasets. https://github.com/logpai/loghub/wiki/Loghub

[62] UpGuard. 2016. The cost of downtime at the world’s biggest online retailer. https://www.upguard.com/blog/the-cost-of-downtime-at-the-worlds-biggest-online-retailer

[63] Risto Vaarandi. 2003. A data clustering algorithm for mining patterns from event logs. In IP Operations & Management, 2003.(IPOM 2003). 3rd IEEE Workshop on. IEEE, 119–126.

[64] Risto Vaarandi and Mauno Pihelgas. 2015. LogCluster-A data clustering and pattern mining algorithm for event logs. In 2015 11th International Conference on Network and Service Management (CNSM). IEEE, 1–7.

[65] Wei Xu, Ling Huang, Armando Fox, David A. Patterson, and Michael I. Jordan. 2009. Detecting large-scale system problems by mining console logs. In SOSP. 117–132.

[66] Ding Yuan, Soyeon Park, Peng Huang, Yang Liu, Michael M Lee, Xiaoming Tang, Yuanyuan Zhou, and Stefan Savage. 2012. Be conservative: enhancing failure diagnosis with proactive logging. In Presented as part of the 10th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 12). 293–306.

[67] Ding Yuan, Soyeon Park, and Yuanyuan Zhou. 2012. Characterizing logging practices in open-source software. In Proceedings of the 34th International Conference on Software Engineering. IEEE Press, 102–112.

[68] J. Zhu, P. He, Q. Fu, H. Zhang, M. R. Lyu, and D. Zhang. 2015. Learning to Log: Helping Developers Make Informed Logging Decisions. In ICSE’15.

[69] Zookeeper. 2019. Zookeeper. https://zookeeper.apache.org