LOG ANALYSES AND ANOMALY DETECTION USING OPEN SOURCE TOOLS

V.Prathyusha[1], Shaik Sohail[2], Mucharla Praveen Kumar[3], Boddupally Shivamani[4], Chatla Sumanth[5]

[1] Associate Professor, Dept. of CS, Sri Indu College of Engineering and Technology, Hyderabad,

[2 3 4] Research Student, Dept. of CS Sri Indu College of Engineering and Technology, Hyderabad

Abstract—Anomaly detection plays an important role in man- agement of modern large-scale distributed systems. Logs, which record system runtime information, are widely used for anomaly detection. Traditionally, developers (or operators) often inspect the logs manually with keyword search and rule matching. The increasing scale and complexity of modern systems, however, make the volume of logs explode, which renders the infeasibility of manual inspection. To reduce manual effort, many anomaly detection methods based on automated log analysis are proposed. However, developers may still have no idea which anomaly detection methods they should adopt, because there is a lack of a review and comparison among these anomaly detection methods. Moreover, even if developers decide to employ an anomaly detection method, re-implementation requires a non- trivial effort. To address these problems, we provide a detailed review and evaluation of six state-of-the-art log-based anomaly detection methods, including three supervised methods and three unsupervised methods, and also release an open-source toolkit allowing ease of reuse. These methods have been evaluated on two publicly-available production log datasets, with a total of 15,923,592 log messages and 365,298 anomaly instances. We believe that our work, with the evaluation results as well as the corresponding findings, can provide guidelines for adoption of these methods and provide references for future development.

I. Introduction

Modern systems are evolving to large scale, either by scaling out to distributed systems built on thousands of commodity machines (e.g., Hadoop [1], Spark [2]), or by scaling up to high performance computing with supercomputers of thou- sands of processors (e.g., Blue Gene/L [36]). These systems are emerging as the core part of IT industry, supporting a wide variety of online services (such as search engines, social networks, and e-commence) and intelligent applications (such as weather forecasting, business intelligence, and biomedical engineering). Because most of these systems are designed to operate on a 24x7 basis, serving millions of online users globally, high availability and reliability become a must. Any incidents of these systems, including service outage and degradation of quality of service, will break down applications and lead to significant revenue loss.

Anomaly detection, which aims at uncovering abnormal system behaviors in a timely manner, plays an important role in incident management of large-scale systems. Timely anomaly detection allows system developers (or operators) to pinpoint issues promptly and resolve them immediately,

thereby reducing system downtime. Systems routinely gen-erate logs, which record detailed runtime information dur- ing system operation. Such widely-available logs are used as a main data source for system anomaly detection. Log- based anomaly detection (e.g., [27], [38], [47]) has become a research topic of practical importance both in academia and in industry. For traditional standalone systems, developers manually check system logs or write rules to detect anomalies based on their domain knowledge, with additional use of keyword search (e.g., “fail”, “exception”) or regular expression match. However, such anomaly detection that relies heavily on manual inspection of logs have become inadequate for large- scale systems, due to the following reasons:

1) The large-scale and parallel nature of modern systems makes system behaviors too complex to comprehend by each single developer, who is often responsible for sub-components only. For example, many open- source systems (e.g., Hadoop, Spark) are implemented by hundreds of developers. A developer might have only incomplete understanding of the overall system behaviors, thus making it a great challenge to identify issues from huge logs.
2) Modern systems are generating tons of logs, for exam- ple, at a rate of about 50 gigabytes (around 120~200 million lines) per hour [32]. The sheer volume of such logs makes it notoriously difficult, if not infeasible, to manually discern the key information from the noise data for anomaly detection, even with the utility such as search and grep.
3) Large-scale systems are typically built with different fault tolerant mechanisms employed. Systems some- times run the same task with redundancy and even proac- tively kill a speculative task to improve performance. In such a setting, the traditional method using keyword search becomes ineffective for extracting suspicious log messages in these systems, which likely leads to many false positives that are actually log messages unrelated to real failures [27]. This will significantly increase the effort in manual inspection.

As a result, automated log analysis methods for anomaly detection are highly in demand. Log-based anomaly detection has been widely studied in last decades. However, we found that there is a gap between research in academia and practice in industry. On one hand, developers are, in many cases

on this subject. They have to go through a large body of literature to get a comprehensive view of current anomaly detection methods. This is a cumbersome task yet does not guarantee that the most suitable method can be found, because each research work usually focuses specifically on reporting a detailed method towards a target system. The difficulty may be exacerbated if developers have no prior background knowledge on machine learning that is required to understand these methods. On the other hand, to our knowledge, no log-based open-source tools are currently available for anomaly detection. There is also a lack of comparison among existing anomaly detection methods. It is hard for developers to know which is the best method to their practical problem at hand. To compare all candidate methods, they need to try each one with their own implementation. Enormous efforts are often required to reproduce the methods, because no test oracles exist to guarantee correct implementations of the underlying machine learning algorithms.

To bridge this gap, in this paper, we provide a detailed review and evaluation of log-based anomaly detection, as well as release an open-source toolkit[1] for anomaly detection. Our goal is not to improve any specific method, but to portray an overall picture of current research on log analysis for anomaly detection. We believe that our work can benefit researchers and practitioners in two aspects: The review can help them grasp a quick understanding of current anomaly detection methods; while the open-source toolkit allows them to easily reuse existing methods and make further customization or improvement. This helps avoid time-consuming yet redundant efforts for re-implementation.

The process of log analysis for anomaly detection involves four main steps: log collection, log parsing, feature extraction, and anomaly detection. In our last work [24], we have presented a review and evaluation of automatic log parsing methods, where four open-source log parsers are publicly released. In this work, we will focus primarily on the aspects of feature extraction and machine learning models for anomaly detection. According to the type of data involved and the machine learning techniques employed, anomaly detection methods can be classified into two broad categories: supervised anomaly detection and unsupervised anomaly detection. Supervised methods need labeled training data with clear specification on normal instances and abnormal instances. Then classification techniques are utilized to learn a model to maximize the discrimination between normal and abnormal instances. Unsupervised methods, however, do not need labels at all. They work based on the observation that an abnormal instance usually manifests as an outlier point that is distant from other instances. As such, unsupervised learning techniques, such as clustering, can be applied.

More specifically, we have reviewed and implemented six representative anomaly detection methods reported in recent literature, including three supervised methods (i.e., Logistic Regression [12], Decision Tree [15], and SVM [26]) and three unsupervised methods (i.e., Log Clustering [27], PCA [47], and Invariant Mining [28]). We further perform a systematic evaluation of these methods on two publicly-available log datasets, with a total of 15,923,592 log messages and 365,298 anomaly instances. The evaluation results are reported on *precision* (in terms of the percentage of how many reported anomalies are correct), *recall* (in terms of the percentage of how many real anomalies are detected), and *efficiency* (in terms of the running times over different log sizes). Though the data are limited, but we believe that these results, as well as the corresponding findings revealed, can provide guidelines for adoption of these methods and serve as baselines in future development.

In summary, this paper makes the following contributions:

- A detailed review of commonly-used anomaly detection methods based on automated log analysis;
- An open-source toolkit consisting of six representative anomaly detection methods; and
- A systematic evaluation that benchmarks the effectiveness and efficiency of current anomaly detection methods.

The remainder of this paper is organized as follows. Section II describes the overall framework of log-based anomaly detection. Section III reviews six representative anomaly detection methods. We report the evaluation results in Section IV, and make some discussions in Section V. Section VI introduces the related work and finally Section VII concludes the paper.

II. Framework Overview

Figure 1 illustrates the overall framework for log-based anomaly detection. The anomaly detection framework mainly involves four steps: log collection, log parsing, feature extraction, and anomaly detection.

Log collection: Large-scale systems routinely generate logs to record system states and runtime information, each comprising a timestamp and a log message indicating what has happened. These valuable information could be utilized for multiple purposes (e.g., anomaly detection), and thereby logs are collected first for further usage. Fore example, Figure 1 depicts 8 log lines extracted from the HDFS logs on Amazon EC2 platform [47], while some fields are omitted here for ease of presentation.

Log parsing: Logs are unstructured, which contain free-form text. The purpose of log parsing is to extract a group of event templates, whereby raw logs can be structured. More specifically, each log message can be parsed into a event template (constant part) with some specific parameters (variable part). As illustrated in Figure 1, the 4th log message (*Log 4*) is parsed as *"Event 2*" with an event template "*Received block * of size * from **".

Feature extraction: After parsing logs into separate events, we need to further encode them into numerical feature vectors, whereby machine learning models can be applied. To do so, we first slice the raw logs into a set of log sequences by using different grouping techniques, including fixed windows, sliding windows, and session windows. Then, for each log sequence, we generate a feature vector (event count vector), which represents the occurence number of each event. All feature vectors together can form a feature matrix, that is, a event count matrix.

[1] Available at https://github.com/cuhk-cse/loglizer

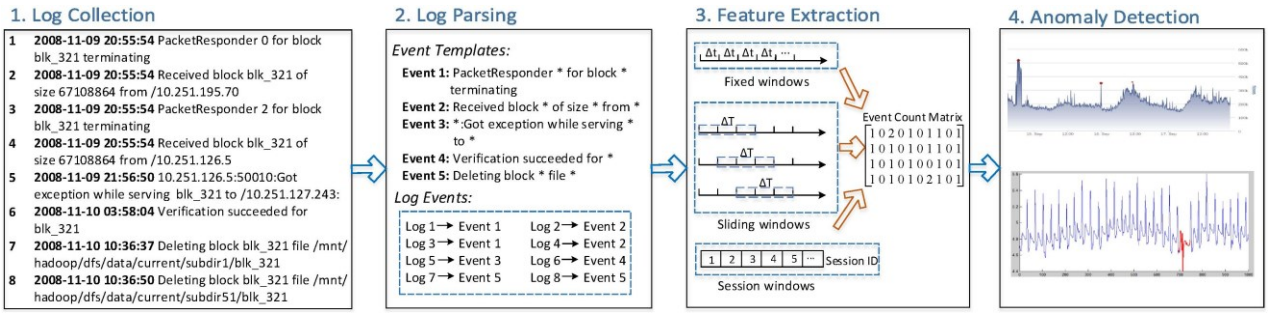


Figure 1: Framework of anomaly detection

Anomaly detection: Finally, the feature matrix can be fed to machine learning models for training, and thus generate a model for anomaly detection. The constructed model can be used to identify whether or not a new incoming log sequence is an anomaly.

III. Methodology

In this section, we give a detailed review on methods for different phases: log parsing, feature extraction and anomaly detection. For log parsing, we briefly give the basic ideas and introduce several typical log parsers. Then, three feature extraction techniques are discussed, which are applied on the parsed log events to generate feature vectors. After obtaining the feature vectors, we focus on six representative anomaly detection approaches, of which three are supervised methods and the other three are unsupervised.

A. Log Parsing

Logs are plain text that consists of constant parts and variable parts, which may vary among different occurrences. For instance, given the logs of "Connection from 10.10.34.12 closed" and "Connection from 10.10.34.13 closed", the words "Connection", "from" and "closed" are considered as constant parts because they always stay the same, while the remaining parts are called variable parts as they are not fixed. Constant parts are predefined in source codes by developers, and variable parts are often generated dynamically (e.g., port number, IP address) that could not be well utilized in anomaly detection. The purpose of log parsing is to separate constant parts from variable parts and form a well-established log event (i.e., "Connection from * closed" in the example).

There are two types of log parsing methods: clustering-based (e.g., LKE [20], LogSig [44]) and heuristic-based (e.g., iPLoM [29], SLCT [45]). In clustering-based log parsers, distances between logs are calculated first, and clustering techniques are often employed to group logs into different clusters in the next step. Finally, event template is generated from each cluster. For heuristic-based approaches, the occurrences of each word on each log position are counted. Next, frequent words are selected and composed as the event candidates. Finally, some candidates are chosen to be the log events. We implemented and compared four log parsers in our previous work [24]. Besides, we published an open-source log parsing toolkit online[2], which is employed to parse raw logs into log events in this paper.

B. Feature Extraction

The main purpose of this step is to extract valuable features from log events that could be fed into anomaly detection models. The input of feature extraction is log events generated in the log parsing step, and the output is an event count matrix. In order to extract features, we firstly need to separate log data into various groups, where each group represents a log sequence. To do so, windowing is applied to divide a log dataset into finite chunks [5]. As illustrated in Figure 1, we use three different types of windows: fixed windows, sliding windows, and session windows.

Fixed window: Both fixed windows and sliding windows are based on timestamp, which records the occurrence time of each log. Each fixed window has its size, which means the time span or time duration. As shown in Figure 1, the window size is Δt, which is a constant value, such as one hour or one day. Thus, the number of fixed windows depends on the predefined window size. Logs that happened in the same window are regarded as a log sequence.

Sliding window: Different from fixed windows, sliding windows consist of two attributes: window size and step size, e.g., hourly windows sliding every five minutes. In general, step size is smaller than window size, therefore causing the overlap of different windows. Figure 1 shows that the window size is ΔT , while the step size is the forwarding distance. The number of sliding windows, which is often larger than fixed windows, mainly depends on both window size and step size. Logs that occurred in the same sliding window are also grouped as a log sequence, though logs may duplicate in multiple sliding windows due to the overlap.

Session window: Compared with the above two windowing types, session windows are based on identifiers instead of the timestamp. Identifiers are utilized to mark different execution paths in some log data. For instance, HDFS logs with block_id record the allocation, writing, replication, deletion of certain block. Thus, we can group logs according to the identifiers, where each session window has a unique identifier.

After constructing the log sequences with windowing techniques, an event count matrix X is generated. In each log sequence, we count the occurence number of each log event

[2]Log parsers available at: https://github.com/cuhk-cse/logparser

to form the event count vector. For example, if the event count vector is [0, 0, 2, 3, 0, 1, 0], it means that event 3 occurred twice and event 4 occurred three times in this log sequence. Finally, plenty of event count vectors are constructed to be an event count matrix X, where entry $X_{i,j}$ records how many times the event j occurred in the i-th log sequence.

C. Supervised Anomaly Detection

Supervised learning (e.g., decision tree) is defined as a machine learning task of deriving a model from labeled training data. Labeled training data, which indicate normal or anomalous state by labels, are the prerequisite of supervised anomaly detection. The more labeled the training data, the more precise the model would be. We will introduce three representative supervised methods: Logistic regression, Decision tree, and Support vector machine (SVM) in the following.

1) Logistic Regression

Logistic regression is a statistical model that has been widely-used for classification. To decide the state of an instance, logistic regression estimates the probability p of all possible states (normal or anomalous). The probability p is calculated by a logistic function, which is built on labeled training data. When a new instance appears, the logistic function could compute the probability p ($0 < p < 1$) of all possible states. After obtaining the probabilities, the states with the largest probability is the classification output.

To detect anomalies, an event count vector is constructed from each log sequence, and every event count vector together with its label are called an instance. Firstly, we use training instances to establish the logistic regression model, which is actually a logistic function. After obtaining the model, we feed an testing instance X into the logistic function to compute its possibility p of anomaly, the label of X is anomalous when $p \geq 0.5$ and normal otherwise.

2) Decision Tree

Decision Tree is a tree structure diagram that uses branches to illustrate the predicted state for each instance. The decision tree is constructed in a top-down manner using training data. Each tree node is created using the current "best" attribute, which is selected by attribute's information gain [23]. For example, the root node in Figure 2 shows that there are totally 20 instances in our dataset. When splitting the root node, the occurrence number of Event 2 is treated as the "best" attribute. Thus, the entire 20 training instances are split into two subsets according to the value of this attribute, in which one contains 12 instances and the other consists of 8 instances.

Decision Tree was first applied to failure diagnosis for web request log system in [15]. The event count vectors together with their labels described in Section III-B are utilized to build the decision tree. To detect the state of a new instance, it traverses the decision tree according to the predicates of each traversed tree node. In the end of traverse, the instance will arrive one of the leaves, which reflects the state of this instance.

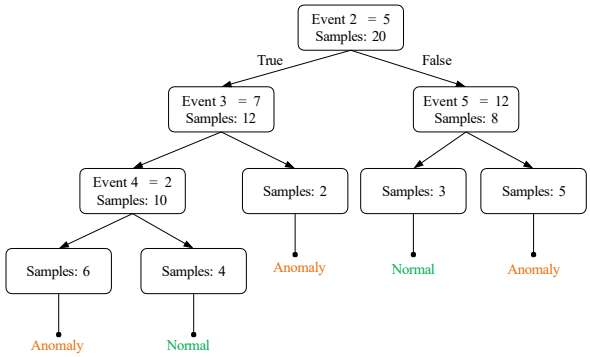


Figure 2: An example of decision tree

3) SVM

Support Vector Machine (SVM) is a supervised learning method for classification. In SVM, a hyperplane is constructed to separate various classes of instances in high-dimension space. Finding the hyperplane is an optimization problem, which maximizes the distance between the hyperplane and the nearest data point in different classes.

In [26], Liang et al. employ SVM to detect failures and compared it with other methods. Similar to Logistic Regression and Decision Tree, the training instances are event count vectors together with their labels. In anomaly detection via SVM, if a new instance is located above the hyperplane, it would be reported as an anomaly, while marked as normal otherwise. There are two kinds of SVM, namely linear SVM and non-linear SVM. In this paper, we only discuss linear SVM, because linear SVM outperforms non-linear SVM in most of our experiments.

D. Unsupervised Anomaly Detection

Unlike supervised methods, unsupervised learning is another common machine learning task but its training data is unlabeled. Unsupervised methods are more applicable in real-world production environment due to the lack of labels. Common unsupervised approaches include various clustering methods, association rule mining, PCA and etc.

1) Log Clustering

In [27], Lin et al. design a clustering-based method called LogCluster to identify online system problems. LogCluster requires two training phases, namely knowledge base initialization phase and online learning phase. Thus, the training instances are divided into two parts for these two phases, respectively.

Knowledge base initialization phase contains three steps: log vectorization, log clustering, representative vectors extraction. Firstly, log sequences are vectorized as event count vectors, which are further revised by Inverse Document Frequency (IDF) [41] and normalization. Secondly, LogCluster clusters normal and abnormal event count vectors separately with agglomerative hierarchical clustering, which generates two sets of vector clusters (i.e., normal clusters and abnormal clusters) as knowledge base. Finally, we select a representative vector for each cluster by computing its centroid.

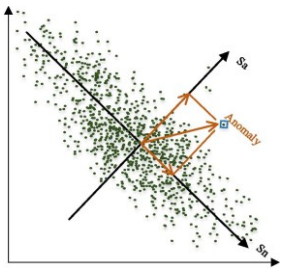


Figure 3: Simplified example of anomaly detection with PCA

Online learning phase is used to further adjust the clusters constructed in knowledge base initialization phase. In online learning phase, event count vectors are added into the knowledge base one by one. Given an event count vector, the distances between it and existing representative vectors are computed. If the smallest distance is less than a threshold, this event count vector will be added to the nearest cluster and the representative vector of this cluster will be updated. Otherwise, LogCluster creates a new cluster using this event count vector.

After constructing the knowledge base and complete the online learning process, LogCluster can be employed to detect anomalies. Specifically, to determine the state of a new log sequence, we compute its distance to representative vectors in knowledge base. If the smallest distance is larger than a threshold, the log sequence is reported as an anomaly. Otherwise, if the nearest cluster is a normal/an abnormal cluster, the log sequence is reported as normal/abnormal.

2) PCA

Principal Component Analysis (PCA) is a statistical method that has been widely used to conduct dimension reduction. The basic idea behind PCA is to project high-dimension data (e.g., high-dimension points) to a new coordinate system composed of k principal components (i.e., k dimensions), where k is set to be less than the original dimension. PCA calculates the k principal components by finding components (i.e., axes) which catch the most variance among the high-dimension data. Thus, the PCA-transformed low-dimension data can preserve the major characteristics (e.g., the similarity between two points) of the original high-dimension data. For example, in Figure 3, PCA attempts to transform two-dimension points to one-dimension points. S_n is selected as the principal component because the distance between points can be best described by mapping them to S_n.

PCA was first applied in log-based anomaly detection by Xu et al. [47]. In their anomaly detection method, each log sequence is vectorized as an event count vector. After that, PCA is employed to find patterns between the dimensions of event count vectors. Employing PCA, two subspace are generated, namely normal space S_n and anomaly space S_a. S_n is constructed by the first k principal components and S_n is constructed by the remaining $(n-k)$, where n is the original dimension. Then, the projection $y_a = (1 - PP^T)y$ of an event count vector y to S_a is calculated, where $P = [v_1, v_2, \ldots, v_k]$ is the first k principal components. If the length of y_a is larger

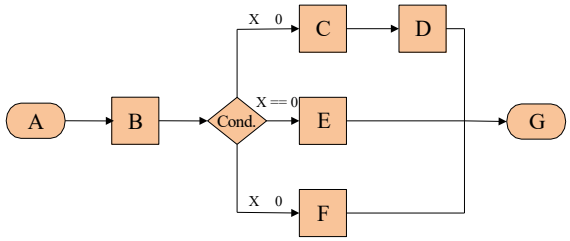


Figure 4: An example of execution flow

than a threshold, the corresponding event count vector will be reported as an anomaly. For example, the selected point in Figure 3 is an anomaly because the length of its projection on S_a is too large. Specifically, an event count vector is regarded as anomaly if

$$SPE \equiv \|y_a\|^2 > Q_\alpha$$

where squared prediction error (i.e., SPE) represents the "length", and Q_α is the threshold providing $(1 - \alpha)$ confidence level. We set $Q = 0.001$ as in the original paper. For k, we calculate it automatically by adjusting the PCA to capture 95% variance of the data, also same as the original paper.

3) Invariants Mining

Program Invariants are the linear relationships that always hold during system running even with various inputs and under different workloads. Invariants mining was first applied to log-based anomaly detection in [28]. Logs that have the same session id (e.g., block id in HDFS) often represent the program execution flow of that session. A simplified program execution flow is illustrated in Figure 4.

In this execution flow, the system generates a log message at each stage from A to G. Assuming that there are plenty of instances running in the system and they follow the program execution flow in Figure 4, the following equations would be valid:

$$n(A) = n(B)$$
$$n(B) = n(C) + n(E) + n(F)$$
$$n(C) = n(D)$$
$$n(G) = n(D) + n(E) + n(F)$$

where $n(*)$ represents the number of logs which belong to corresponding event type $*$.

Intuitively, Invariants mining could uncover the linear relationships (e.g., $n(A) = n(B)$) between multiple log events that represent system normal execution behaviors. Linear relationships prevail in real-world system events. For example, normally, a file must be closed after it was opened. Thus, log with phrase "open file" and log with phrase "close file" would appear in pair. If the number of log events "open file" and that of "close file" in an instance are not equal, it will be marked abnormal because it violates the linear relationship.

Invariants mining, which aims at finding invariants (i.e., linear relationships), contains three steps. The input of invariants mining is an event count matrix generated from log sequences, where each row is an event count vector. Firstly, the invariant space is estimated using singular value decomposition, which determines the amount r of invariants that need to be mined in

the next step. Secondly, this method finds out the invariants by a brute force search algorithm. Finally, each mined invariant candidate is validated by comparing its support with a threshold (e.g., supported by 98% of the event count vectors). This step will continue until r independent invariants are obtained.

In anomaly detection based on invariants, when a new log sequence arrives, we check whether it obey the invariants. The log sequence will be reported as an anomaly if at least one invariant is broken.

E. Methods Comparison

To reinforce the understanding of the above six anomaly detection approaches, and help developers better choose anomaly detection methods to use, we discuss the advantages and disadvantages of different methods in this part.

For supervised methods, labels are required for anomaly detection. Decision tree is more interpretable than the other two methods, as developers can detect anomalies with meaningful explanations (i.e., predicates in tree nodes). Logistic regression cannot solve linearly non-separable problems, which can be solved by SVM using kernels. However, parameters of SVM are hard to tune (e.g., penalty parameter), so it often requires much manual effort to establish a model.

Unsupervised methods are more practical and meaningful due to the lack of labels. Log clustering uses the idea of online learning. Therefore, it is suitable for processing large volume of log data. Invariants mining not only can detect anomalies with a high accuracy, but also can provide meaningful and intuitive interpretation for each detected anomaly. However, the invariants mining process is time consuming. PCA is not easy to understand and is sensitive to the data. Thus, its anomaly detection accuracy varies over different datasets.

F. Tool implementation

We implemented six anomaly detection methods in Python with over 4,000 lines of code and packaged them as a toolkit. For supervised methods, we utilize a widely-used machine learning package, scikit-learn [39], to implement the learning models of Logistic Regression, Decision Tree, and SVM. There are plenty of parameters in SVM and logistic regression, and we manually tune these parameters to achieve the best results during training. For SVM, we tried different kernels and related parameters one by one, and we found that SVM with linear kernel obtains the bette r anomaly detection accuracy than other kernels. For logistic regression, different parameters are also explored, and they are carefully tuned to achieve the best performance.

Implementing unsupervised methods, however, is not straightforward. For log clustering, we were not able to directly use the clustering AP I from scikit-learn, because it is not designed for large-scale datasets, where our data cannot fit to the memory. We implemented the clustering algorithm into an online version, whereb y each data instance is grouped into a cluster one by one. There are multiple thresholds to be tuned. We also paid great efforts to implement the invariants mining method, because we built a search space for possible invariants and proposed multiple ways to prune all unnecessary invariants. It is very time-consuming to test different combination of thresholds. We finally implemented PCA method according to the original reference based on the use of an API from scikit-learn. PCA has only two parameters and it is easy to tune.

Table I: Summary of datasets

System	#Time span	#Data size	#Log messages	#Anomalies
BGL	7 months	708 M	4,747,963	348,460
HDFS	38.7 hours	1.55 G	11,175,629	16,838

IV. Evaluation Study

In this section, we will first introduce the datasets we employed and the experiment setup for our evaluation. Then, we provide the evaluation results of supervised and unsupervised anomaly detection methods separately, since these two types of methods are generally applicable in different settings. Finally, the efficiency of all these methods is evaluated.

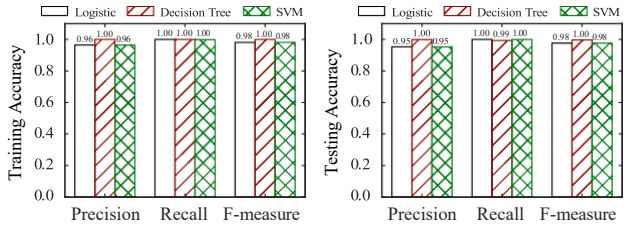
A. Experiments Design

Log Datasets: Publicly available production logs are scarce data because companies rarely publish them due to confidential issues. Fortunately, by exploring an abundance of literature and intensively contacting the corresponding authors, we have successfully obtained two log datasets, HDFS data [47] and BGL data [36], which are suitable for evaluating existing anomaly detection methods. Both datasets are collected from production systems, with a total of 15,923,592 log messages and 365,298 anomaly samples, that are manually labeled by the original domain experts. Thus we take these labels (anomaly or not) as the ground truth for accuracy evaluation purposes. More statistical information of the datasets is provided in Table I.

HDFS data contain 11,175,629 log messages, which were collected from Amazon EC2 platform [47]. HDFS logs record a unique block ID for each block operation such as allocation, writing, replication, deletion. Thus, the operations in logs can be more naturally captured by session windows, as introduced in III-B, because each unique block ID can be utilized to slice the logs into a set of log sequences. Then we extract feature vectors from these log sequences and generate 575,061 event count vectors. Among them, 16,838 samples are marked as anomalies.

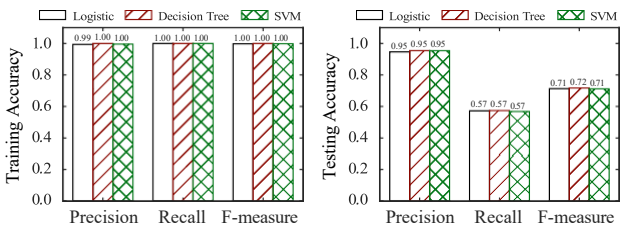
BGL data contain 4,747,963 log messages, which were recorded by the BlueGene/L supercomputer system at Lawrence Livermore National Labs (LLNL) [36]. Unlike HDFS data, BGL logs have no identifier recorded for each job execution. Thus, we have to use fixed windows or sliding windows to slice logs as log sequences, and then extract the corresponding event count vectors. But the number of windows depends on the chosen window size (and step size). In BGL data, 348,460 log messages are labeled as failures, and a log sequence is marked as an anomaly if any failure logs exist in that sequence.

Experimental setup: We ran all our experiments on a Linux server with Intel Xeon E5-2670v2 CPU and 128GB DDR3 1600 RAM, on which 64-bit Ubuntu 14.04.2 with



(a) Training Accuracy (b) Testing Accuracy

Figure 5: Accuracy of supervised methods on HDFS data with session windows



(a) Training Accuracy (b) Testing Accuracy

Figure 6: Accuracy of supervised methods on BGL data with fixed windows

Linux kernel 3.16.0 was running. Unless otherwise stated, each experiment was run for five times and the average result is reported. We use *precision*, *recall* and *F-measure*, which are the most commonly used metrics, to evaluate the accuracy of anomaly detection methods as we already have the ground truth (anomaly or not) for both of the datasets. As shown below, precision measures the percentage of how many reported anomalies are correct, recall measures the percentage of how many real anomalies are detected, and F-measure indicates the harmonic mean of precision and recall.

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

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

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

For all three supervised methods, we choose the first 80% data as the training data, and the remaining 20% as the testing data because only previously happening events could lead to a succeeding anomaly. By default, we set the window size of fixed windows to one hour, and set the window size and step size of sliding windows to be six hours and one hour, respectively.

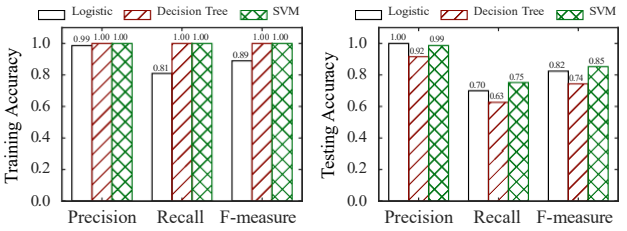
B. Accuracy of Supervised Methods

To explore the accuracy of supervised methods, we use them to detect anomalies on HDFS data and BGL data. We use session windows to slice HDFS data and then generate the event count matrix, while fixed windows and sliding windows are applied to BGL data separately. In order to check the validity of three supervised methods (namely Logistic Regression, Decision Tree, SVM), we first train the models on training data, and then apply them to testing data. We report both training accuracy and testing accuracy in different settings, as illustrated in Figure 7~9. We can observe that all supervised methods achieve high training accuracy (over 0.95), which implies that normal instances and abnormal instances are well separated by using our feature representation. However, their accuracy on testing data varies with different methods and datasets. The overall accuracy on HDFS data is higher than the accuracy on BGL data with both fixed windows and sliding windows. This is mainly because HDFS system records relative simple operations with only 29 event types, which is much less than that in BGL data, which is 385. Besides, HDFS data are grouped by session windows, thereby causing a higher correlation between events in each log sequence. Therefore, anomaly detection methods on HDFS perform better than on BGL.

In particular, Figure 5 shows the accuracy of anomaly detection on HDFS data, and all three approaches have excellent performance on testing data with the F-measure close to 1. When applying supervised approaches on testing data of BGL with fixed windows, they do not achieve high accuracy, although they perform well on training data. As Figure 6 illustrates, all three methods on BGL with fixed windows have the recall of only 0.57, while they obtain high detection precision of 0.95. We found that as the fixed window size is only one hour, thus, it may cause the uneven distribution of anomalies. For example, some anomalies that happened in current window may actually be related to events in the former time window and they are incorrectly divided. As a consequence, anomaly detection methods with one-hour fixed window do not perform well on BGL data.

Finding 1: Supervised anomaly detection methods achieve high precision, while the recall varies over different datasets and window settings.

To address the problem of poor performance with fixed windows, we employed the sliding windows to slice BGL data with window size = 6h and step size = 1h. The results are given in Figure 7. Comparing with the fixed windows, anomaly detection based on sliding windows achieve much higher accuracy on testing data. The reasons are that by using sliding windows, we not only can obtain as many windows (event count vectors) as fixed windows, but also can avoid the problem of uneven distribution because the window size is much larger. Among supervised methods, we observe that SVM achieves the best overall accuracy with F-measure of



(a) Training Accuracy (b) Testing Accuracy

Figure 7: Accuracy of supervised methods on BGL data with sliding windows

Table II: Sliding windows number for different window size and step size

Window Size	1 h	3 h	6 h	9 h	12 h
#Sliding windows	5153	5151	5150	5145	5145

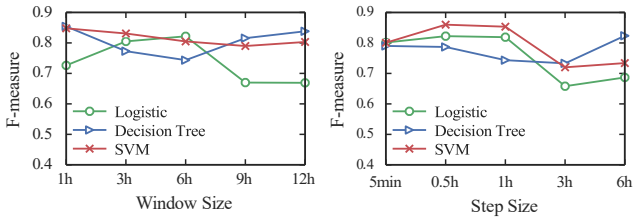
Step Size	5 min	30 min	1 h	3 h	6 h
#Sliding windows	61786	10299	5150	1718	860

0.85. Moreover, decision tree and logistic regression that are based on sliding windows achieve 10.5% and 31.6% improvements in recall compared to the results on the fixed windows.

To further study the influences of different window sizes and various step sizes on anomaly detection accuracy, we conduct experiments by changing one parameter while keeping the other parameter constant. According to the diagram a) of Figure 8, We hold the step size at one hour while changing the window size as shown in Table II. Window sizes larger than 12 hours are not considered as they are not practical in real-world applications. We can observe that the F-measure of SVM slightly decreases when the window size increases, while the accuracy of logistic regression increases slowly first but falls sharply when window sizes increase to nine hours, and then it keeps steady. It is clear that logistic regression achieves the highest accuracy when window size is 6 hours. The variation trend of decision tree accuracy is opposite to the logistic regression, and it reaches the highest accuracy at 12 hours. Therefore, logistic regression is sensitive to the window size while decision tree and SVM remain stable.

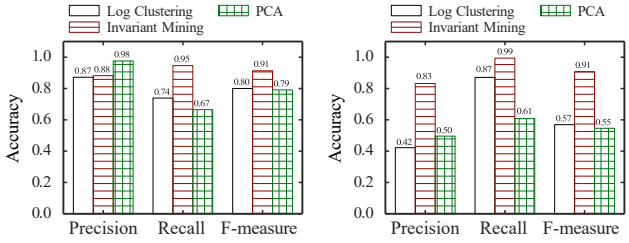
Finding 2: Anomaly detection with sliding windows can achieve higher accuracy than that of fixed windows.

Compared with window size, step size likely has a large effect on anomaly detection accuracy. Table II illustrates that if we reduce the step size while keeping the window size at six hours, the number of sliding windows (data instances) increases dramatically. All three methods show the same trend that the accuracy first increases slightly, then have a drop at around 3 hours. This may be caused by the reason that the number of data instances dramatically decreases when using a large step size, for example, at 3 hours. One exception happened at the step size of six hours: The window size equals to the step size, thus sliding window is the same as fixed window. In this situation, some noise caused by overlapping are removed, which leads to a small increase of detection accuracy.



(a) Different window sizes (b) Different step sizes

Figure 8: Accuracy of supervised methods on BGL data with different window sizes and step sizes



(a) Accuracy on HDFS data (b) Accuracy on BGL data

Figure 9: Accuracy of unsupervised methods on HDFS data and BGL data

C. Accuracy of Unsupervised Methods

Although supervised methods achieve high accuracy, especially on the HDFS data, these methods are not necessarily applicable in a practical setting, where data labels are often not available. Unsupervised anomaly detection methods are proposed to address this problem. To explore the anomaly detection accuracy of unsupervised methods, we evaluate them on the HDFS data and BGL data. As indicated in the last section, sliding window can lead to more accurate anomaly detection. We therefore only report the results of sliding windows on BGL data.

As log clustering is extremely time-consuming on HDFS data with half-a-million instances, tuning parameters become impractical, we then choose the largest log size that we can handle in a reasonable time to represent our HDFS data.

In Figure 9, we can observe that all unsupervised methods show good accuracy on HDFS data, but they obtain relatively low accuracy on BGL data. Among three methods, invariants mining achieves superior performance (with F-measure of 0.91) against other unsupervised anomaly detection methods on both data. Invariants mining automatically constructs linear correlation patterns to detection anomalies, which fit well with the nature BGL data, where failures are marked through some critical events. Log clustering and PCA do not obtain good detection accuracy on BGL data. The poor performance of log clustering is caused by the high-dimensional and sparse characteristics of event count matrix. As such, it is difficult for log clustering to separate anomalies and normal instances, which often leads to a lot of false positives.

We study in depth to further understand why the PCA does not achieve high accuracy on BGL data. The criterion for PCA to detect anomalies is the distance to normal space (squared prediction error). As Figure 10 illustrates, when the distance is larger than a specific threshold (the red dash line represents our current threshold), an instance is identified as an anomaly. However, by using the ground truth labels to plot the distance distribution as shown in Figure 10, we found that both classes (normal and abnormal) cannot be naturally separated by any single threshold. Therefore, PCA does not perform well on the BGL data.

Finding 3: Unsupervised methods generally achieve inferior performance against supervised methods. But invariants mining manifests as a promising method with stable, high performance.

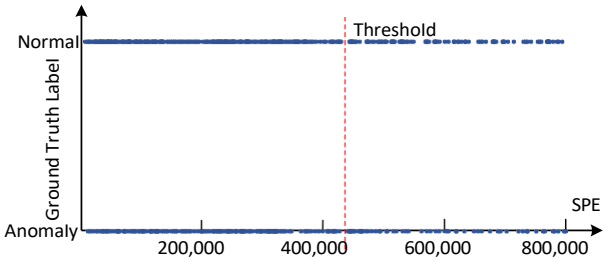


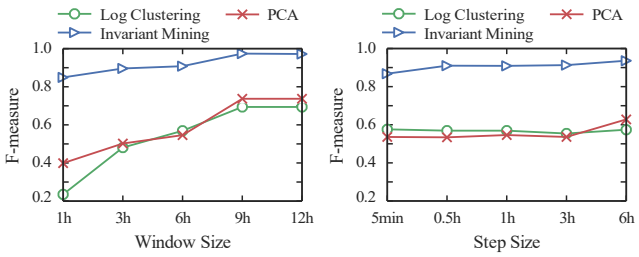
Figure 10: Distance distribution in anomaly space of PCA

Like supervised methods, we also conduct experiments on different settings of window size and step size to explore their effects on accuracy. As shown in Figure 11, we have an interesting observation that the accuracy steadily rises when the window size increases, while the change of step size has little influence on accuracy. This observation is contrary to what we found for supervised methods. As illustrated in Table II, the window number largely decreases when the window size increases. Given a larger window size, more information is covered while more noise can be added as well, but unsupervised methods could discover more accurate patterns for anomaly detection.

Findings 4: The settings of window size and step size have different effects on supervised methods and unsupervised methods.

D. Efficiency of Anomaly Detection Methods

In Figure 12, the efficiency of all these anomaly detection methods is evaluated on both datasets with varying log sizes. As shown in the figure, supervised methods can detect anomalies in a short time (less than one minute) while unsupervised methods are much more time-consuming (except PCA). We can observe that all anomaly detection methods scale linearly as the log size increases, except for the log clustering, whose time complexity is $O(n^2)$. Note that both horizontal and vertical axises are not in linear scale. Furthermore, log clustering cannot handle large-scale datasets in an acceptable time; thus, running time results of log clustering are not fully plotted. It is worth noting that the running time of invariants mining is larger than log clustering on BGL data but not on HDFS data, because there are more event types in BGL data than HDFS data, which increases the time for invariants mining. Besides, it should also be noted that the running time of invariants mining



(a) Different window sizes (b) Different step sizes

Figure 11: Accuracy of unsupervised methods with different window sizes and step sizes on BGL data

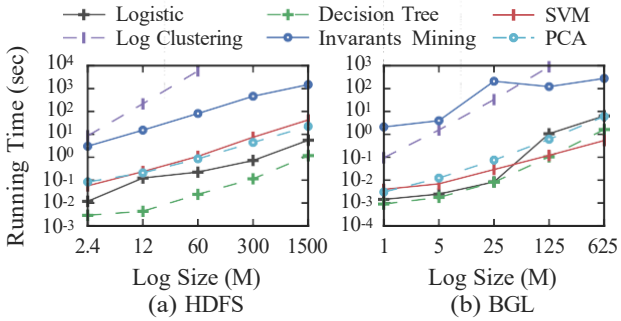


Figure 12: Running time with increasing log size

slightly decreases at the log size of 125 megabytes on BGL data. This is because we set the stopping criteria to control its brute force searching process on large datasets, which could avoid unnecessary search for high-dimensional correlations.

Finding 5: Most anomaly detection methods scale linearly with log size, but the methods of Log Clustering and Invariants Mining need further optimizations for speedup.

V. Discussion

In this section, we discuss some limitations of our work, and further provide some potential directions for future study.

Diversity of datasets. Logs recorded from production systems are invaluable to evaluate anomaly detection methods. However, there publicly-available log dataset are scarce resources, because companies are often unwilling to open their log data due to confidential issues. This is where evaluation becomes difficult. Thanks to the support from the authors in [36], [47], we obtained two production log datasets that have enabled our work. The datasets represent logs from two different types of systems, but the evaluation results and the findings may be still limited by the diversity of the datasets. Clearly, the availability of more log datasets would allow us to generalize our findings and drastically support related research. It is our future plan to collect more log datasets from open platforms.

Feature representation. Typically, different systems usually have quite different logs, as illustrate in HDFS and BGL datasets. To generalize our implementations of different anomaly detection methods, we focus mainly on a feature space denoted by event count matrix, which has been employed in most of existing work (e.g., [28], [47]). There are still some other features that need for further exploration, such as the timestamp of a log message, whereby the temporal duration of two consecutive events and the order information of a log sequence can be extracted. However, as reported in [28], logs generated by modern distributed systems are usually interleaved by different processes. Thus, it becomes a great challenge to extract reliable temporal features from such logs.

Other available methods. We have reviewed and implemented most of the commonly-used, and representative, log analysis methods for anomaly detection. However, there are some other methods employing different models, such as frequent sequence mining [22], finite state machine [20], formal concept analysis [18], and information retrieval [30].

We also believe that more are coming out because of the practical importance of log analysis. It is our ongoing work to implement and maintain a more comprehensive set of open-source tools.

Open-source log analysis tools. There is currently a lack of publicly-available log analysis tools that could be directly utilized for anomaly detection. We also note that a set of new companies (e.g., [3], [4]) are offering log analysis tools as their products. But they are all working as a black box. This would lead to increased difficulty in reproducible research and slow down the overall innovation process. We hope our work makes the first step towards making source code publicly available, and we advocate more efforts in this direction.

Potential directions. *1) Interpretability of methods.* Most of current log-based anomaly detection methods are built on machine learning models (such as PCA). But most of these models work as a "black box". That is, they are hard to interpret to provide intuitive insights, and developers often cannot figure out what the anomalies are. Methods that could reflect natures of anomalies are highly desired. *2) Real-time log analysis.* Current systems and platforms often generate logs in real time and in huge volume. Thus, it becomes a big challenge to deal with big log data in real time. The development of log analysis tools on big data platforms and the functionality of real-time anomaly detection are in demand.

VI. Related Work

Log analysis. Log analysis has been widely employed to improve the reliability of software systems in many aspects [35], such as anomaly detection [10], [28], [47], failure diagnosis [17], [31], [38], program verification [11], [42], and performance prediction [16]. Most of these log analysis methods consist of two steps: log parsing and log mining, which are broadly studied in recent years. He et al. [24] evaluate the effectiveness of four offline log parsing methods, SLCT [45], IPLOM [29], LogSig [44], and LKE [20] , which do not require system source code. Nagappan et al. [34] propose an offline log parsing method that enjoys linear running time and space. Xu et al. [47] design an online log parsing method based on the system source code. For log mining, Xu et al. [47] detect anomalies use PCA, whose input is a matrix generated from logs. Beschastnikh et al. [11] employ system logs to generate a finite state machine, which describes system runtime behaviors. Different from these papers that employ log analysis to solve different problems, we focus on anomaly detection methods based on log analysis.

Anomaly detection. Anomaly detection aims at finding abnormal behaviors, which can be reported to the developers for manual inspection and debugging. Bovenzi et al. [13] propose an anomaly detection method at operating system level, which is effective for mission-critical systems. Venkatakrishnan et al. [46] detect security anomalies to prevent attacks before they compromise a system. Different from these methods that focus on detecting a specific kind of anomalies, this paper evaluates the effectiveness of anomaly detection methods for general anomalies in large scale systems. Babenko et al. [9] design a technique to automatically generate interpretations using of detected failures from anomalies. Alonso et al. [6] detect anomalies by employing different classifiers. Farshchi et al. [19] adopt a regression-based analysis technique to detect anomalies of cloud application operations. Azevedo et al. [8] use clustering algorithms to detect anomalies in satellites. These methods, which utilize performance metrics data collected by different systems, can complement log-based anomaly detection methods evaluated in this paper. Log-based anomaly detection is widely studied [19], [20], [28], [31], [43], [47]. In this paper, we review and evaluate six anomaly detection methods employing log analysis [12], [15], [26], [27], [28], [47] because of their novelty and representativeness.

Empirical study. In recent years, many empirical research investigation on software reliability emerge, because empirical study can usually provide useful and practical insights for both researchers and developers. Yuan et al. [48] study the logging practice of open-source systems and provide improvement suggestions for developers. Fu et al. [21], [49] conduct an empirical study on logging practice in industry. Pecchia et al. [37] study the logging objectives and issues impacting log analysis in industrial projects. Amorim et al. [7] evaluate the effectiveness of using decision tree algorithm to recognize code smells. Lanzaro et al. [25] analyze how software faults in library code manifest as interface errors. Saha et al. [40] study the long lived bugs from five different perspectives. Milenkoski et al. [33] survey and systematize common practices in the evaluation of computer intrusion detection systems. Chandola et al. [14] survey anomaly detection methods that use machine learning techniques in different categories, but this paper aims at reviewing and benchmarking the existing work that applies log analysis techniques to system anomaly detection.

VII. Conclusion

Logs are widely utilized to detection anomalies in modern large-scale distributed systems. However, traditional anomaly detection that relies heavily on manual log inspection becomes impossible due to the sharp increase of log size. To reduce manual effort, automated log analysis and anomaly detection methods have been widely studied in recent years. However, developers are still not aware of the state-of-the-art anomaly detection methods, and often have to re-design a new anomaly detection method by themselves, due to the lack of a comprehensive review and comparison among current methods. In this paper, we fill this gap by providing a detailed review and evaluation of six state-of-the-art anomaly detection methods. We also compare their accuracy and efficiency on two representative production log datasets. Furthermore, we release an open-source toolkit of these anomaly detection methods for easy reuse and further study.

VIII. Acknowledgments

The work described in this paper was fully supported by the National Natural Science Foundation of China (Project No. 61332010), the Research Grants Council of the Hong Kong Special Administrative Region, China (No. CUHK 14205214 of the General Research Fund), and 2015 Microsoft Research Asia Collaborative Research Program (Project No. FY16-RES-THEME-005).

REFERENCES

[1] Apache hadoop (http://hadoop.apache.org/).

[2] Apache spark (http://spark.apache.org/).

[3] Logentries: Log management & analysis software made easy (https://www.loggly.com/docs/anomaly-detection).

[4] Loggly: Cloud log management service (https://www.loggly.com/docs/anomaly-detection).

[5] T. Akidau, R. Bradshaw, C. Chambers, S. Chernyak, R. J. FernÃ¡ndez-Moctezuma, R. Lax, S. McVeety, D. Mills, F. Perry, E. Schmidt, and S. Whittle. The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. In *PVLDB'15: Proc. of the VLDB Endowment*, volume 8, pages 1792–1803. VLDB Endowment, 2015.

[6] J. Alonso, L. Belanche, and Dimiter R. Avresky. Predicting software anomalies using machine learning techniques. In *NCA'11: Proc. of the 10th IEEE International Symposium on Network Computing and Applications*, pages 163–170. IEEE, 2011.

[7] L. Amorim, E. Costa, N. Antunes, B. Fonseca, and M. Ribeiro. Experience report: Evaluating the effectiveness of decision trees for detecting code smells. In *ISSRE'15: Proc. of the 26th IEEE International Symposium on Software Reliability Engineering*, pages 261–269. IEEE, 2015.

[8] D. R. Azevedo, A. M. Ambrósio, and M. Vieira. Applying data mining for detecting anomalies in satellites. In *EDCC'12: Proc. of the Ninth European Dependable Computing Conference*, pages 212–217. IEEE, 2012.

[9] A. Babenko, L. Mariani, and F. Pastore. Ava: automated interpretation of dynamically detected anomalies. In *ISSTA'09: Proc. of the eighteenth international symposium on Software testing and analysis*, pages 237–248. ACM, 2009.

[10] S. Banerjee, H. Srikanth, and B. Cukic. Log-based reliability analysis of software as a service (saas). In *ISSRE'10: Proc. of the 21st International Symposium on Software Reliability Engineering*, 2010.

[11] I. Beschastnikh, Y. Brun, S. Schneider, M. Sloan, and M.D. Ernst. Leveraging existing instrumentation to automatically infer invariant-constrained models. In *ESEC/FSE'11: Proc. of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, 2011.

[12] P. Bodik, M. Goldszmidt, A. Fox, D. B. Woodard, and H. Andersen. Fingerprinting the datacenter: automated classification of performance crises. In *EuroSys'10: Proc. of the 5th European conference on Computer systems*, pages 111–124. ACM, 2010.

[13] A. Bovenzi, F. Brancati, S. Russo, and A. Bondavalli. An os-level framework for anomaly detection in complex software systems. *IEEE Transactions on Dependable and Secure Computing*, 12(3):366–372, 2015.

[14] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

[15] M. Chen, A. X. Zheng, J. Lloyd, M. I. Jordan, and E. Brewer. Failure diagnosis using decision trees. In *ICAC'04: Proc. of the 1st International Conference on Autonomic Computing*, pages 36–43. IEEE, 2004.

[16] X. Chen, C. Lu, and K. Pattabiraman. Predicting job completion times using system logs in supercomputing clusters. In *in DSN-W'13: Proc. of the 43rd Annual IEEE/IFIP Conference on Dependable Systems and Networks Workshop (DSN-W)*, pages 1–8. IEEE, 2013.

[17] M. Cinque, D. Cotroneo, R. Della Crte, and A. Pecchia. What logs should you look at when an application fails? insights from an industrial case study. In *DSN'14: Proc. of the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 690–695. IEEE, 2014.

[18] R. Ding, Q. Fu, J. Lou, Q. Lin, D. Zhang, J. Shen, and T. Xie. Healing online service systems via mining historical issue repositories. In *ASE'12: Proc. of the 27th IEEE/ACM International Conference on Automated Software Engineering*, pages 318–321. IEEE, 2012.

[19] M. Farshchi, J. Schneider, I. Weber, and J. Grundy. Experience report: Anomaly detection of cloud application operations using log and cloud metric correlation analysis. In *ISSRE'15: Proc. of the 26th International Symposium on Software Reliability Engineering*. IEEE, 2015.

[20] Q. Fu, J. Lou, Y. Wang, and J. Li. Execution anomaly detection in distributed systems through unstructured log analysis. In *ICDM'09: Proc. of International Conference on Data Mining*, 2009.

[21] Q. Fu, J. Zhu, W. Hu, J. Lou, R. Ding, Q. Lin, D. Zhang, and T. Xie. Where do developers log? an empirical study on logging practices in industry. In *ICSE'14: Companion Proc. of the 36th International Conference on Software Engineering*, pages 24–33, 2014.

[22] X. Fu, R. Ren, J. Zhan, W. Zhou, Z. Jia, and G. Lu. Logmaster: mining event correlations in logs of large-scale cluster systems. In *SRDS'12: Proc. of the 31st IEEE Symposium on Reliable Distributed Systems*, pages 71–80. IEEE, 2012.

[23] J. Han, M. Kamber, and J. Pei. *Data mining: concepts and techniques*. Elsevier, 2011.

[24] P. He, J. Zhu, S. He, J. Li, and R. Lyu. An evaluation study on log parsing and its use in log mining. In *DSN'16: Proc. of the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2016.

[25] A. Lanzaro, R. Natella, S. Winter, D. Cotroneo, and N. Suri. An empirical study of injected versus actual interface errors. In *ISSTA'14: Proc. of the 2014 International Symposium on Software Testing and Analysis*, pages 397–408. ACM, 2014.

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

[27] Q. Lin, H. Zhang, J.G. Lou, Y. Zhang, and X. Chen. Log clustering based problem identification for online service systems. In *ICSE'16: Proc. of the 38th International Conference on Software Engineering*, 2016.

[28] J. Lou, Q. Fu, S. Yang, Y Xu, and J. Li. Mining invariants from console logs for system problem detection. In *ATC'10: Proc. of the USENIX Annual Technical Conference*, 2010.

[29] A. Makanju, A. Zincir-Heywood, and E. Milios. Clustering event logs using iterative partitioning. In *KDD'09: Proc. of International Conference on Knowledge Discovery and Data Mining*, 2009.

[30] C. Manning, P. Raghavan, and H. SchÃŒtze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[31] L. Mariani and F. Pastore. Automated identification of failure causes in system logs. In *Software Reliability Engineering, 2008. ISSRE 2008. 19th International Symposium on*, pages 117–126. IEEE, 2008.

[32] H. Mi, H. Wang, Y. Zhou, R. Lyu, and H. Cai. Toward fine-grained, unsupervised, scalable performance diagnosis for production cloud computing systems. *IEEE Transactions on Parallel and Distributed Systems*, 24:1245–1255, 2013.

[33] A. Milenkoski, M. Vieira, S. Kounev, A. Avritzer, and B.D. Payne. Evaluating computer intrusion detection systems: A survey of common practices. *ACM Computing Surveys (CSUR)*, 48(1):12, 2015.

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

[35] A. Oliner, A. Ganapathi, and W. Xu. Advances and challenges in log analysis. *Communications of the ACM*, 55(2):55–61, 2012.

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

[37] A. Pecchia, M. Cinque, G. Carrozza, and D. Cotroneo. Industry practices and event logging: assessment of a critical software development process. In *ICSE'15: Proc. of the 37th International Conference on Software Engineering*, pages 169–178, 2015.

[38] A. Pecchia, D. Cotroneo, Z. Kalbarczyk, and R.K. Iyer. Improving log-based field failure data analysis of multi-node computing systems. In *DSN'11: Proc. of the 41st IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 97–108. IEEE, 2011.

[39] F. Pedregosa, G. Varoquaux, A. Gramfort, et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[40] R. K. Saha, S. Khurshid, and D.E. Perry. An empirical study of long lived bugs. In *CSMR-WCRE'14: Proc. of the 2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE),*, pages 144–153. IEEE, 2014.

[41] G. Salton and C Buckley. Term weighting approaches in automatic text retrival. Technical report, Cornell, 1987.

[42] W. Shang, Z. Jiang, H. Hemmati, B. Adams, A.E. Hassan, and P. Martin. Assisting developers of big data analytics applications when deploying on hadoop clouds. In *ICSE'13: Proc. of the 35th International Conference on Software Engineering*, pages 402–411, 2013.

[43] J. Tan, X. Pan, S. Kavulya, R. Gandhi, and P. Narasimhan. Salsa: Analyzing logs as state machines. *WASL*, 8:6–6, 2008.

[44] L. Tang, T. Li, and C. Perng. LogSig: generating system events from raw textual logs. In *CIKM'11: Proc. of ACM International Conference on Information and Knowledge Management*, pages 785–794, 2011.

[45] R. Vaarandi. A data clustering algorithm for mining patterns from event logs. In *IPOM'03: Proc. of the 3rd Workshop on IP Operations and Management*, 2003.

[46] R. Venkatakrishnan and M. A. Vouk. Diversity-based detection of security anomalies. In *HotSoS'14: Proc. of the 2014 Symposium and Bootcamp on the Science of Security*, page 29. ACM, 2014.

[47] W. Xu, L. Huang, A. Fox, D. Patterson, and M.I. Jordon. Detecting large-scale system problems by mining console logs. In *SOSP'09: Proc. of the ACM Symposium on Operating Systems Principles*, 2009.

[48] D. Yuan, S. Park, P. Huang, Y. Liu, M. Lee, X. Tang, Y. Zhou, and S Savage. Be conservative: enhancing failure diagnosis with proactive logging. In *OSDI'12: Proc. of the 10th USENIX Conference on Operating Systems Design and Implementation*, pages 293–306, 2012.

[49] J. Zhu, P. He, Q. Fu, H. Zhang, R. Lyu, and D. Zhang. Learning to log: Helping developers make informed logging decisions. In *ICSE'15: Proc. of the 37th International Conference on Software Engineering*, 2015.