Anomaly Detection via Mining Numerical Workflow Relations from Logs

Bo Zhang, Hongyu Zhang, Pablo Moscato
School of Electrical Engineering and Computing, The University of Newcastle, NSW, Australia
c3288930@uon.edu.au, hongyu.zhang@newcastle.edu.au, pablo.moscato@newcastle.edu.au

***Abstract*—Complex software intensive systems, especially distributed systems, generate logs for troubleshooting. The logs are text messages recording system events, which can help engineers determine the system's runtime status. This paper proposes a novel approach named ADR (stands for Anomaly Detection by workflow Relations) that employs matrix nullspace to mine numerical relations from log data. The mined relations can be used for both offline and online anomaly detection and facilitate fault diagnosis. We have evaluated ADR on log data collected from two distributed systems, HDFS (Hadoop Distributed File System) and BGL (IBM Blue Gene/L supercomputers system). ADR successfully mined 87 and 669 numerical relations from the logs and used them to detect anomalies with high precision and recall. For online anomaly detection, ADR employs PSO (Particle Swarm Optimization) to find the optimal sliding windows' size and achieves fast anomaly detection. The experimental results confirm that ADR is effective for both offline and online anomaly detection.**

I. Introduction

Many large-scale distributed systems provide online services to a large number of users from around the world. Therefore, high system availability and reliability are essential. A small problem could affect user experience and even cause significant financial loss. Although a lot of effort has been devoted to quality assurance, in reality, distributed systems still encounter many problems and often fail to satisfy user requests.

To facilitate problem identification and diagnosis, most distributed systems generate console logs. The logs are usually unstructured text messages recording system events, which can help engineers understand the system's internal status and determine whether the status is normal or not. When a system behaves abnormally, the engineers can try to identify the reason for the failure by examining the logs. Clearly, to uncover abnormal system behaviors in an effective manner, manually checking the logs is extremely time-consuming or even infeasible, especially for large-scale distributed systems that provide 24-7 services. Therefore, automated log-based anomaly detection is important.

Over the years, many automated log-based anomaly detection approaches have been proposed. These approaches include supervised machine learning methods (such as Logistic Regression (LR) [1], Decision Tree (DT) [2], Support Vector Machine (SVM) [3]) and unsupervised methods (such as Log Clustering (LC) [4] and Principle Component Analysis (PCA) [5]). However, one drawback of these machine learning based models is that most of them are unexplainable - it is difficult for engineers to explain the results of these models and gain insights into the anomalies. The approach Invariants Miner (IM) [6] is able to identify explainable invariants from the logs. In this model, the mined invariants describe the numerical relations among the log events that should hold true during the normal running of the system. The invariants are human understandable and explainable so that they can help engineers understand the behavior of the system and may point to the reason for the observed anomalies [6]. However, IM has its limitations too. Firstly, it only supports *linear* invariants, therefore other important types of non-linear relations are ignored. Secondly, IM employs a search-based method to look for invariants and it cannot ensure to find all available invariants. Besides, it suffers from performance problems when the number of log events increases. For example, in a comparative study [7], the researchers had to manually set a stopping criteria to speed up the search process by avoiding searching for linear invariants that involved too many events.

In this paper, we propose ADR (an acronym that stands for Anomaly Detection by workflow Relations), a novel approach to mine numerical relations from logs and use the relations for anomaly detection. ADR first transforms the log data into a sequence of log events, counts the occurrences of the events, and constructs an extended event-count-matrix. Then, ADR calculates the nullspace of the matrix, which covers relations in both simple workflows (such as sequential or conditional workflows) and complex workflows (combinations of simple workflows). The observed violated relations can be used to detect anomalies and provide insightful information for explaining the anomalies.

The highlights of our approach include:

- ADR requires a very small size of training data yet it is able to produce comparable results with the state-of-the-art approaches.
- For training, ADR only requires the logs that are produced when the system is running normally. Such normal logs are easy to collect in practice.
- By searching a proper window size using PSO (Particle Swarm Optimization), ADR can group consecutive log entries to sliding windows and perform online anomaly detection. That is, as soon as a new log entry is generated, ADR can determine if an anomaly has emerged.

We have evaluated the proposed approach ADR on log data collected from two industrial distributed system, HDFS (Hadoop Distributed File System) [8] and BGL (IBM Blue

```
1  2016-10-02 12:24:52,337 INFO org.apache.hadoop.hdfs.server.datanode.DataNode.clienttrace: src: /10.10.34.11:58237, dest: /10.10.34.11:50010,
       bytes: 144, op: HDFS_WRITE, cliID: DFSClient_NONMAPREDUCE_1903091109_103, offset: 0, srvID: d9ef1b17-4314-4cd8-91eb-095413c3427f,
       blockid: BP-108841162-10.10.34.11-1440074360971:blk_1074555986_815162, duration: 16128279
2  2016-10-02 12:24:52,337 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: PacketResponder:
       BP-108841162-10.10.34.11-1440074360971:blk_1074555986_815162, type=HAS_DOWNSTREAM_IN_PIPELINE terminating
3  2016-10-02 12:24:52,393 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Receiving
       BP-108841162-10.10.34.11-1440074360971:blk_1074555988_815164 src: /10.10.34.11:58242 dest: /10.10.34.11:50010
4  2016-10-02 12:24:52,394 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Receiving
       BP-108841162-10.10.34.11-1440074360971:blk_1074555989_815165 src: /10.10.34.11:58243 dest: /10.10.34.11:50010
5  2016-10-02 12:24:52,399 INFO org.apache.hadoop.hdfs.server.datanode.DataNode.clienttrace: src: /10.10.34.11:58242, dest: /10.10.34.11:50010,
       bytes: 66, op: HDFS_WRITE, cliID: DFSClient_NONMAPREDUCE_1530486806_104, offset: 0, srvID: d9ef1b17-4314-4cd8-91eb-095413c3427f,
       blockid: BP-108841162-10.10.34.11-1440074360971:blk_1074555988_815164, duration: 2155456
```

Fig. 1. A snippet of HDFS (Hadoop Distributed File System) logs

Gene/L supercomputers system) [9]. ADR discovered 87 and 669 relations from the HDFS and BGL log data. It achieved high precision and recall scores (all greater than 0.9) for offline anomaly detection and outperforms the state-of-the-art methods. For online anomaly detection, ADR takes advantages of the event-count-matrix's rank and employs PSO to find the optimal window size to split the logs. On the BGL dataset, ADR achieves the F1 score of 0.97 and returns the detection result quickly (within 11ms in average).

The main contributions of this paper are as follows:

1) We propose ADR to mine numerical relations from the logs' event-count-matrix and construct extra elements for the matrix to cover more types of relations.
2) We propose a PSO-based method to find the optimal window size to split the logs.
3) We evaluate ADR on public log datasets in both offline and online manner. The results confirm the effectiveness of ADR in log-based anomaly detection.

The organization of the paper is as follows. In Section II, we introduce the background information on log analysis and anomaly detection. In Section III, we describe the relationship between workflows and their numerical relations. Then we describe our proposed approach in detail in Section IV. The experiment design and results are presented in Section V and VI. The threat to validity is summarized in Section VII. We introduce the related work to our research in Section VIII and conclude the paper in Section IX.

II. Background

Many software systems generate logs for troubleshooting purposes. The logs record the system events that can help engineers maintain or diagnose the systems. Figure 1 shows a snippet of raw logs produced by HDFS. It can be seen that the raw logs are often unstructured texts. To facilitate log analysis, several log parsers (e.g. Drain [10], AEL [11], Spell [12], IPLoM [13]) have been proposed to parse the raw logs into structured log events. Then each raw log entry can be regarded as a certain event with specific parameters. For example, in Figure 1, the first and second log entries belong to two different log events while the third and fourth log entries belong to the same event *"Receiving block* $\langle * \rangle$ *src:* $\langle * \rangle$ *dest:* $\langle * \rangle$*"*.

Afterwards the log entries are usually grouped to log sequences. There are three common types of grouping strategies, i.e. sessions, fixed windows, and sliding windows. Grouping by sessions is to classify the log events by certain identifiers, such as *TaskID*, *InstanceID*, or *BlockID* [14], [15], [7]. Grouping by fixed or sliding windows are based on the timestamps of the log entries. For each log sequence (obtained by sessions or fixed/sliding windows), the occurrences of the events are counted, resulting in an event-count-matrix. For example, the following is an event-count-matrix for sessions:

$$\begin{array}{c} \\ \textit{session 1} \\ \textit{session 2} \\ \ldots \\ \textit{session m} \end{array} \begin{array}{c} \begin{array}{cccc} \textit{Event 1} & \textit{Event 2} & \ldots & \textit{Event n} \end{array} \\ \begin{bmatrix} c_{11} & c_{12} & .. & c_{1n} \\ c_{21} & c_{22} & .. & c_{2n} \\ .. & .. & .. & .. \\ c_{m1} & c_{m2} & .. & c_{mn} \end{bmatrix} \end{array} \quad (1)$$

where c_{mn} indicates the number of occurrences of *Event n* in *session m*.

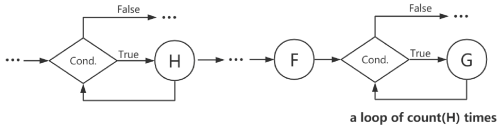
Over the years, many machine learning based methods have been proposed to employ the event-count-matrix to detect system anomalies [14], [16], [17], [18], [19], [20], [21], [22], [23]. Some of these methods use supervised learning techniques such as Logistic Regression [1], Decision Trees [2], and Support Vector Machines (SVMs) [3], while others employ unsupervised approaches such as Log Clustering [4], Principal Component Analysis [5], and the previously mentioned Invariant Miner (IM) [6]. Among them, the Logistic Regression based method [1] uses labelled instances to train a logistic function to estimate the probability of the system being abnormal. SVM based method [3] tries to construct a hyperplane to separate normal and abnormal instances in high-dimensional space. IM [6] employs a greedy search method to discover linear relations between the occurrences of system events and the instances that violate the invariant relationships are considered as the anomalies.

In a comparative study, He et al. [7] found that supervised approaches usually perform better than unsupervised approaches when detecting anomalies. However, the supervised approaches require labelled logs to train the models, for which the labelling work is very time-consuming and requires domain expertise on the subject system. Another concern for the current approaches

is that although they can identify system anomalies, explainable information for follow-up troubleshooting are often hard to be provided because the original events information is usually transformed to a feature space. Among the related approaches, IM is able to find human-understandable invariants, which can help locate the causes of the anomalies [6]. However, IM only supports linear invariants and cannot ensure to find a complete set of available invariants [7]. Besides, it assumes an ad hoc support ratio for the invariants (e.g. in [6], [7] a threshold of 98% was used for the computational experiments, i.e. it is implicitly assuming the proportion of anomalies lower than 2%), which is not always true. Moreover, IM suffers from performance issues when multiple events are involved in the target invariants.

III. Numerical Relations in Logs

As logs are usually generated at critical points during system runtime, different sequences of log events are inherently related to the system's workflows. Table I presents three basic workflows: sequence, condition, and loop. Here we use $count(Event)$ to denote the number of occurrences of $Event$. For Workflow 1, if this workflow is normally executed several times, we will see that the occurrences of the events must comply with $count(A) = count(B)$. Similarly, for Workflow 2, we can infer that $count(C) = count(D) + count(E)$ because every normal execution of this workflow can ensure an event C must be followed by either an event D or an event E. For Workflow 3, which contains a loop, it can be inferred that $count(G) = k \times count(F)$ where k is an integer related to the terminating condition of the loop. Moreover, we can consider some complex workflows which are composed of the basic flows. For example, a combination of Workflows 2 and 3 could result in a relation that $count(C) = count(D) + k \times count(F)$. Another complex combination is shown in Figure 2, which produces the relation that $count(G) = count(H) \times count(F)$, where the number of loops is determined by the occurrence of another event. Note that, even if some intermediate statements are not logged, the other events can still comply with certain relations. Figure 3 shows a combination of Workflow 1 and 2 in which some statements in-between are not logged, but the relation $count(A) = count(D) + count(E)$ still holds if the system is running normally.



$count(G) = count(H) \times count(F)$

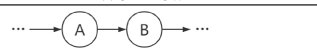
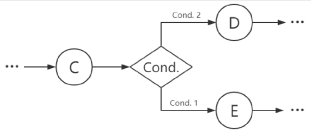
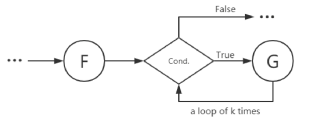
Fig. 2. A workflow that contains a variable-length loop

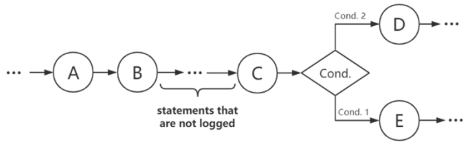
IV. Proposed Approach

A. Overview

In this paper, we propose an anomaly detection approach based on numerical relations mined from logs. The overview workflow of the proposed approach is shown in Figure 4. It starts by parsing the raw logs into structured log events. Several state-of-the-art tools such as Drain [10], Spell [12] are employed to complete this task. Then the parsed log events are grouped into sequences by sessions or sliding windows. In step 3, for each session or window, the occurrences of its log events are counted, resulting in the event-count-matrix. Then the available numerical relations are extracted from the event-count-matrix by evaluating its nullspace. Finally, the extracted relations are used to detect abnormal log sequences in the offline or online manner. We describe the details of offline anomaly detection at session level in Section IV-B and online anomaly detection at entry level in Section IV-C.

TABLE I
Basic workflows

	Workflow	Relation
1		$count(A) = count(B)$
2		$count(C) = count(D) + count(E)$
3		$count(G) = k \times count(F)$



$count(A) = count(D) + count(E)$

Fig. 3. A relation holds though some intermediate statements are not logged

B. Offline anomaly detection at session level

To perform offline anomaly detection, the first step is to parse the raw logs to structured log events. Several log parsers including Drain [10], AEL [11], Spell [12], IPLoM [13] have been proposed and can be integrated with ADR. For anomaly detection at session level, the log events are grouped by certain session identifiers, such as *BlockID* for HDFS system, *NodeID* for BGL system. Then our approach requires a number of normal session logs to the model and extract the numerical relations among the events. For training, the occurrences of the events in each session are counted and we can obtain its event-count-matrix. In this paper, we propose to extend the original event-count-matrix P by constructing additional elements to cover more relation types. For example, given an event-count-matrix P (as Equation 1), we can construct another

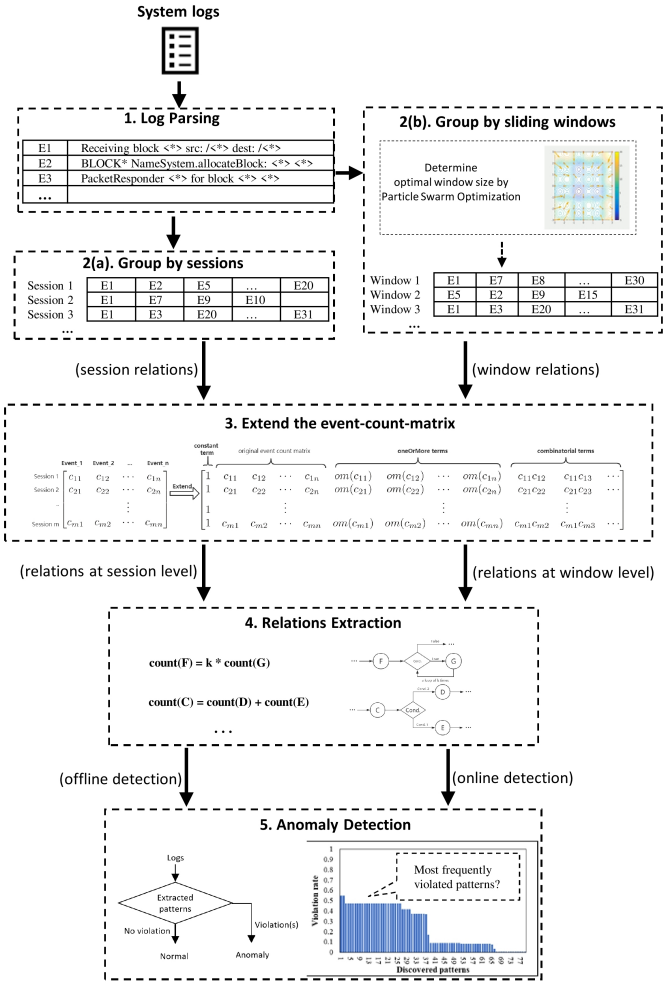


Fig. 4. An overview of ADR

matrix P' as follows:

$$\begin{bmatrix} 1 & c_{11} & c_{12} & .. & \chi(c_{11}) & \chi(c_{12}) & .. & c_{11}c_{12} & c_{11}c_{13} & .. \\ 1 & c_{21} & c_{22} & .. & \chi(c_{21}) & \chi(c_{22}) & .. & c_{21}c_{22} & c_{21}c_{23} & .. \\ .. & .. & .. & .. & .. & .. & .. & .. & .. & .. \\ 1 & c_{m1} & c_{m2} & .. & \chi(c_{m1}) & \chi(c_{m2}) & .. & c_{m1}c_{m2} & c_{m1}c_{m3} & .. \end{bmatrix}$$

In general, P' consists of three more types of extended terms compared with P:

1) The constant terms (the first column in P'), which will capture the events that occur a constant number of times.
2) The $\chi(c_{mn})$ terms, which will equal to 0 when the event count c_{mn} is zero (never occurs) and otherwise 1, meaning the event occurs at least once. The χ terms will enable us to cover more relations in workflows. For example, if a workflow consists of both module A and B but in different sessions module A calls module B different times, the occurrences of *A is loaded* and *B is called* will not comply with a linear relation. But with the help of χ terms, such a workflow can be captured by the relation $\chi(A \textit{ is loaded}) = \chi(B \textit{ is called})$.
3) The combinatorial terms (such as $c_{11}c_{12}$), which consist of the products of the event counts and enable us to capture the relations in some complex workflows, such as the nested loop in Figure 2.

With the extended event-count-matrix, we find that the numerical relations of the events are in essence the relationship among the matrix's linearly dependant columns. Moreover, a matrix's linearly dependant columns are essentially related to its nullspace [24]. The nullspace of the matrix P' consists of all vectors $\mathbf{v}$ such that:

$$P' \cdot \mathbf{v} = \mathbf{0}, \tag{2}$$

Therefore, the complete set of the linearly dependant columns has an one-to-one correspondence with the nullspace of P' (denoted as $ns(P')$).

Based on the analysis of relationship between the extended event-count-matrix's nullspace and the events' relations, each column of $ns(P')$ will depict one numerical workflow relation. For example, assume that we have an event-count-matrix and extended it as:

	1	A	B	C	$\chi(A)$	$\chi(B)$	$\chi(C)$	AB	AC	BC
session 1	1	1	1	0	1	1	0	1	0	0
session 2	1	2	0	2	1	0	1	0	4	0
session 3	1	4	2	2	1	1	1	8	8	4

By using Gaussian elimination [25], we can transform the event-count-matrix to its row echelon form and evaluate its nullspace (presented as its transposed form):

1	A	B	C	$\chi(A)$	$\chi(B)$	$\chi(C)$	AB	AC	BC
0	-1	1	1	0	0	0	0	0	0
-1	0	0	0	1	0	0	0	0	0
-2	1	-3	0	0	4	0	0	0	0
..	..	..	..	..	..	..	..	..	..

Each vector in the nullspace represents a numerical workflow relation. For example, the first vector is $\begin{bmatrix} 0 & -1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$ and along with the corresponding events it shows that $-count(A) + count(B) + count(C) = 0$. This equation indicates the conditional workflow shown ass Workflow 2 in Table I. The second relation is $-1 + \chi(A) = 0$, which indicates that event A should occur at least once. The third relation is $-2 + count(A) - 3count(B) + \chi(B) = 0$, which indicates a more complex relation between that the occurrence of event A and B.

Because the above-mentioned relations are extracted from the log sequences which are produced when the system behaves normally, we deem that they should hold true if the system is in normal status. If something wrong emerges with the system, one or several of the relations would probably be broken. Therefore, we can determine whether an unknown session is normal by verifying its log sequence against each vector of $ns(P')$. If the session's log sequence satisfies the whole nullspace, we deem it a normal session. Otherwise, the session is an anomaly and the violated relations can provide insightful information for diagnosis.

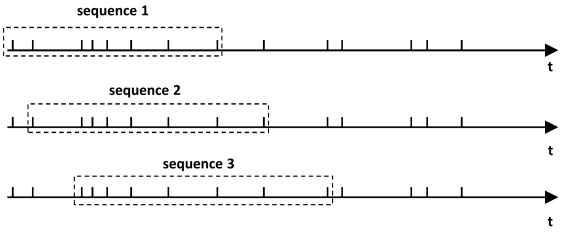


Fig. 5. Online grouping strategy

C. Online anomaly detection at entry level

In the offline manner, each log sequence consists of events from a session's start to its end, so we cannot determine the session's status when it is still running. Such an offline manner is similar to the state-of-the-art tools collected in [7]. In this paper, we further develop our model to enable the anomaly detection in the online manner, i.e. as soon as a new log entry is generated, the system's status can be evaluated. To achieve the goal, we propose a novel grouping strategy which differs from the existing ones. As shown in Figure 5, the log entries in each session are further grouped by windows. Each window contains a fixed number of log entries but has a sliding step of one entry. More specifically, a number of consecutive log entries are grouped as a window. Then the next window is formed by moving the current window forward with the step size of one. The events in each window are counted, forming an event-count-matrix, where its rows represent the windows and columns represent the events' counts. The step size of one entry ensures that as soon as a new log entry is generated by the system, a new window will be formed and evaluated.

Obviously, a proper window size is the key to capture the numerical relations when using the window grouping strategy. In previous studies [7], [17], time windows are often used and the window sizes are chosen in an ad hoc manner, such as 30 minutes, 6 hours, 9 hours. Instead of using ad hoc time windows, in this paper we employ an optimization algorithm, named PSO (Particle Swarm Optimization) [26], to determine the best size of the windows. PSO is a widely-used swarm intelligence optimization algorithm where each candidate solution is called a particle, and multiple particles coexist and optimize cooperatively to search for the optimal solution. To enable the optimization process, we propose a novel fitness function (Formula 3), which can quickly evaluate the performances of different window sizes. As we intend to mine more relations, we have found that the existence of more relations indicates a higher nullity and a lower rank of the event-count-matrix (rank-nullity theorem: nullity + rank = number of columns of the matrix [24]). So, we propose that the rank of the extended event-count-matrix can be used as an indicator for the fitness of the window size. A bigger rank difference between the abnormal and normal event-count-matrices is better because it suggests that the window size will capture more relations in normal logs rather than in abnormal logs. In Formula 3, the matrices I_{AN} and I_N are the event-count-matrix for the abnormal and normal logs when adopting the window size found by the particle.

$$fitness(particle) = rank(I_{AN}) - rank(I_N) \tag{3}$$

The PSO process starts with randomly generating an initial population of particles (each particle is a candidate value that defines the size of the windows). We use s_i^t, v_i^t to denote the location and velocity of i^{th} particle at the moment of t. Generally speaking, the location of a particle is determined by its previous position, the best position it has reached and the global best position all particles have reached. Formula 4 and 5 show how we update the positions of the particles. In the formulas, ω is a positive number referred to as inertia weight and ϵ_1 and ϵ_2 are acceleration factors. r_1 and r_2 are two random numbers between 0 and 1. $s_{i\_best}^t$ is the optimum position s_i has ever reached until moment t, $s_{g\_best}^t$ is the optimum position that all particles have reached until moment t. The initial positions and velocities of the particles are randomly generated and their performances are evaluated to decide the global best position. After that the particles are updated according to Formula 4 and 5 until reaching a termination threshold. In the end of search the best position all particles have ever reached is returned.

$$s_i^{t+1} = s_i^t + v_i^t \tag{4}$$

$$v_i^t = \omega v_i^{t-1} + \epsilon_1 r_1 (s_{i\_best}^{t-1} - s_i^{t-1}) + \epsilon_2 r_2 (s_{g\_best}^{t-1} - s_i^{t-1}) \tag{5}$$

$$\omega_t = \omega_s - (\omega_s - \omega_e)(t/T)^2 \tag{6}$$

After determining the size of the windows, we can obtain the nullspace of I_N, denoted as $ns(I_N)$ and use it to evaluate unknown log sequences. For online detection, our approach will keep collecting log entries until they can form a window. Then the window is checked against each vector of $ns(I_N)$ to determine whether it breaches the relations. Then, as soon as a new log entry is generated, the window will slide to form a new window and it will be checked. The response of the detection can be very fast because the detection is essentially no more than one operation of matrix dot product calculation.

V. Experimental Design

A. Subject datasets

In our experiments, we use two public log datasets that contain a large number of log events to evaluate the proposed approach. One is the HDFS dataset [5], [27], which contains 11,175,629 log entries collected from a Hadoop Distributed File System on Amazon EC2 platform. The other is the BGL dataset [9], [27], which contains 4,747,963 log entries collected from the BlueGene/L supercomputer system. Each log entry in BGL dataset is labelled as normal or abnormal, while the HDFS dataset are labelled at session level identified by *BlockId*. Table II presents the information of the two subject datasets. The experiments are run on a server with Windows Server 2012 R2, Intel Xeon E5-2609 CPU, and 128GB RAM.

TABLE II
SUBJECT DATASETS

Dataset	Size	#Log Entries	#Log Events	#Total Sessions	#Abnormal Sessions
HDFS	1.5 G	11,175,629	48	575,061	16,838
BGL	743 M	4,747,963	384	69,252	31,375

B. Research Questions

We design three research questions (RQs) to evaluate the proposed approach ADR.

1) RQ1: Can the proposed approach extract numerical relations from logs?: For each dataset, firstly we use Drain [27] to parse the raw logs to structured logs. Then the structured logs are grouped by session identifiers, which is *BlockID* for HDFS logs and *NodeID* for BGL logs. Then the event-count-matrix for the sessions is extended to include the constant terms, χ terms and combinatorial terms (quadratic combinations). After that, we evaluate whether ADR can extract the numerical relations from the extended event-count-matrix and compare the results with those of Invariants Miner (IM).

2) RQ2: How effective is the proposed approach for offline anomaly detection?: To answer this RQ, we investigate the capacity of the extracted relations to detect system anomalies. The structured logs are grouped by session identifiers and split into two parts, i.e. training set and testing set. We use different ratios to split the training and testing sets to investigate the impact of training size on the results. For supervised methods, a smaller training size is better because it will reduce the efforts spent on labelling the data. We compare ADR with some state-of-the-art approaches including Invariants Miner (IM) [6], Logistic Regression (LR) [1], Support Vector Machine (SVM) [2]. The detection performance is measured using *precision*, *recall*, *F1*, and *Matthews correlation coefficient (MCC)* [28]. *Precision* measures how many reported anomalies are correct, *Recall* measures how many real anomalies are detected and *F1* is the harmonic mean of *Precision* and *Recall*. *MCC* is in essence a correlation coefficient between the observed and predicted binary classifications.

$$Precision = \frac{TP}{TP + FP}, \quad Recall = \frac{TP}{TP + FN} \tag{7}$$

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{8}$$

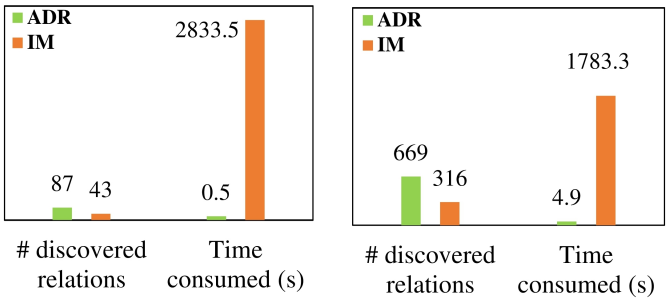
$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{9}$$

3) RQ3: Can PSO find an optimal window size for online anomaly detection?: This RQ is to evaluate whether we can automate the selection of window size by PSO rather than the previous ad hoc manner [7]. As the BGL dataset contains labels for each log entry, we use it to evaluate ADR's capacity for online anomaly detection. Firstly we employ PSO to search for the optimal window size for the dataset. Following some previous work on PSO [29], the two acceleration factors (i.e. ϵ_1 and ϵ_2 in Formula 5) are set to 1.494, the number of particles is set to 20, and the total number of iterations is set to 350. The inertia weight (i.e. ω in Formula 5) is a changing value which is determined by Formula 6. In the formula, ω_t is the inertia weight for the t^{th} iteration, ω_s is the inertia weight in the start of PSO and set to 0.9, and ω_e is the inertia weight in the end and set to 0.4 [29]. After obtaining the optimal window size, the normal sessions from the BGL training set in RQ2 are further split into windows and we obtain the event-count-matrix. Then ADR extracts the numerical relations from the logs by calculating the event-count-matrix's nullspace. To test the anomalies in the online manner, the sessions in testing set are continuously fed to form windows and checked against the nullspace. The metrics including precision, recall, F1 are calculated. As speed is vital to online detection, we also record the time to detect an instance.

VI. EXPERIMENTAL RESULTS AND DISCUSSION

A. Results of RQ1

We employ ADR and IM to mine the relations from HDFS and BGL logs and compare the number of discovered relations and their time performance on the two datasets. The experimental results are shown in Figure 6. It can be seen that ADR discovered more relations in less time than IM. For HDFS logs, the ADR found 87 relations in less than 1 second while IM found 43 relations in more than 47 minutes. Example 1 in Table III is a relation found by ADR. The relation is $count(E1) = \chi(E7) + 3$, which involves two events, E1 and E7. E1 records the event of receiving a block from a source to a destination and E7 shows that a block is transmitted. The corresponding workflow means that if E7 occurs E1 will occur 4 times (1+3), otherwise E1 will occur 3 times (0+3).



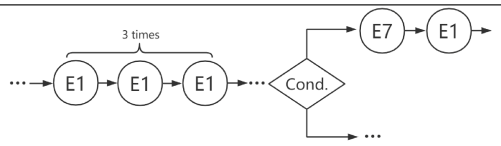
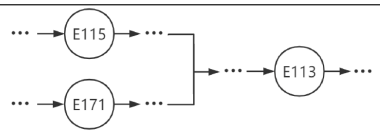
(a) HDFS (b) BGL

Fig. 6. Number of discovered relations and the time spent

For BGL logs that have 384 events, ADR discovered 669 relations in 4.9 seconds while IM discovered 316 relations in 30 minutes. We also sampled and verified some of the relations manually.Example 2 in Table III shows a discovered relation from BGL logs. It is a conditional workflow involving three events. From the relation, we know that $E115$ and $E171$ may be two statements in two branches of a conditional statement

and are executed before $E113$ (e.g. {if ... : E115; else: E171}; E113).

TABLE III
EXAMPLES OF DISCOVERED RELATIONS BY ADR

Example 1 (HDFS)	
Involved Events	E1: Receiving block <*> src: <*> dest: <*> E7: Transmitted block <*> to <*>
Discovered Relation	**count(E1) = χ(E7) + 3**
Likely workflow	
Example 2 (BGL)	
Involved Events	E115: Node card status: …Phy JTAG Reset is asserted. ASIC JTAG Reset is <*> OK. MPGOOD ERROR LATCH IS ACTIVE… E171: Node card status: …ASIC JTAG Reset is asserted. Temperature Mask is not active. No temperature error. Temperature Limit Error Latch is clear… E113: Node card is not fully functional
Discovered Relation	**count(E113) = count(E115) + count(E171)**
Likely workflow	

To check the correctness of the mined relations, we sampled some of the relations and manually verified their correctness. Moreover, the high precision and recall scores achieved by the follow-up anomaly detection experiments (answers to RQ2 and RQ3) also confirmed the correctness of these relations. The reasons why ADR can discover more relations than IM could be as follows: firstly ADR extends the event-count-matrix by including the constant terms, the χ terms, and the combinatorial terms. Therefore more types of relations can be covered. Secondly, IM uses a brute force method combined with greedy search to mine the relations. When dealing with a large number of events (e.g. 384 events in BGL), the search space is very large so that IM is set to ignore some complex relations to speedup the search process [7]. As a comparison, ADR discovers the relations by using the property of the matrix's nullspace, which is a fast operation offered by common mathematical packages [30], [31], [32]. Moreover, the nullspace is a complete set of the available relations for the extended event-count-matrix and it ensures the absence of duplicates.

B. Results of RQ2

Table IV shows the precision, recall, F1, and MCC scores achieved by the comparative approaches on HDFS logs (size of training set : size of testing set = 0.05% : 99.95%) and BGL logs (size of training set : size of testing set = 0.5% : 99.5%). The MCC scores of the approaches are in consistency with their F1 scores, which is reasonable because both of them are based on the precision and recall scores. For HDFS logs, ADR obtains the best recall and F1, which are 0.929 and 0.925, respectively. The precision of ADR on HDFS is 0.931, which is just slightly lower than the highest precision obtained by Logistic Regression (0.936). However, the recall of Logistic Regression is only 0.795, which indicates that though its predictions are precise, a large number of anomalies cannot be identified. Our proposed approach also performs well on BGL logs. Its F1 score is as high as 0.967, which is very close to the highest score (0.988 by SVM). It is also worth noting that the recall scores of ADR on both HDFS and BGL logs are the highest (0.929 and 1), which means that it can identify nearly all the anomalies at high precision.

TABLE IV
RESULTS OF LOG-BASED ANOMALY DETECTION

Approach	HDFS				BGL			
	Precision	Recall	F1	MCC	Precision	Recall	F1	MCC
ADR	0.931	**0.929**	**0.925**	**0.925**	0.937	**1.000**	0.967	0.940
IM	0.881	0.634	0.723	0.734	0.658	0.213	0.319	0.170
LR	**0.936**	0.795	0.854	0.856	0.995	0.969	0.982	0.968
SVM	0.919	0.766	0.830	0.832	**0.997**	0.979	**0.988**	**0.979**

Moreover, ADR obtains consistent accuracy on the two different datasets (0.944 for HDFS dataset and 0.967 for BGL dataset) while the metric values for other approaches vary a lot. For example, IM has a high F1 score on HDFS dataset (0.870), but it has a very poor F1 score on BGL logs (0.319). Though SVM achieves the best F1 score on BGL dataset (0.988), it performs much poorly on HDFS dataset. By comparison, ADR achieves good precision and recall scores on both datasets, which could be attributed to the correctness and the completeness of its discovered relations.

We also investigate how the size of the training set affects the anomaly detection results of ADR and other tools. Figure 7 shows the F1 scores on the testing sets when using different percentage of sessions to train the approaches. ADR can achieve high accuracy with small training sets on both HDFS and BGL datasets. For example, the F1 score achieved by ADR on HDFS logs reaches 0.9 when using less than 0.01% of the sessions for training. ADR's F1 score remains the highest among all the compared approaches as the training size increases. For BGL dataset, the F1 score achieved by ADR is also one of the highest when using different training sizes. The other two supervised methods, Logistic Regression (LR) and SVM, also achieve higher F1 scores as more training sessions are provided. As a contrast, the unsupervised method IM has a much lower F1 score (lower than 0.3). This is because the sizes of training sets are far from enough for the unsupervised methods. It is worth noting that IM has the F1 scores around 0.8 on HDFS but quite lower scores on BGL (around 0.4). The reason is that the proportion of abnormal sessions for BGL logs is 45.3%, which makes it very difficult for IM to produce the correct invariants from the search results by using an assumed support ratio [33].

The distribution of the violated relations can also provide insightful information about the anomalies. Figure 8 presents

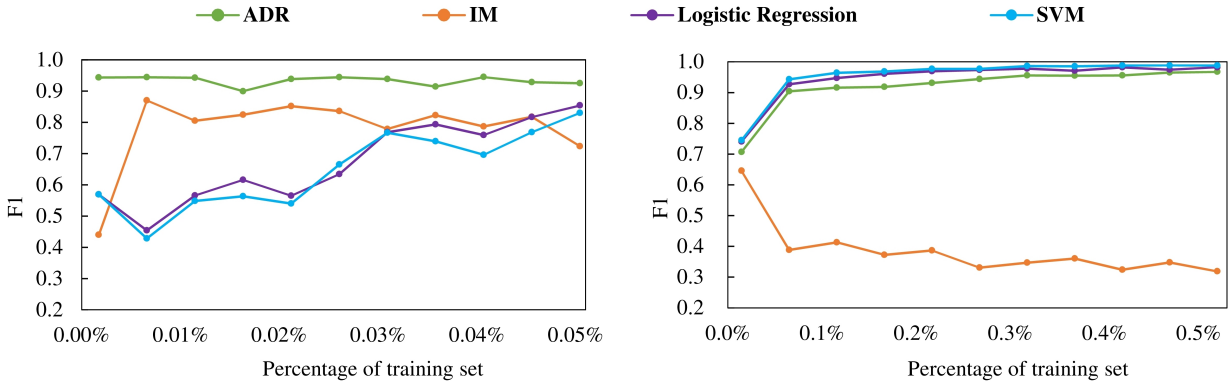


Fig. 7. F1 scores under different training sizes on HDFS and BGL datasets

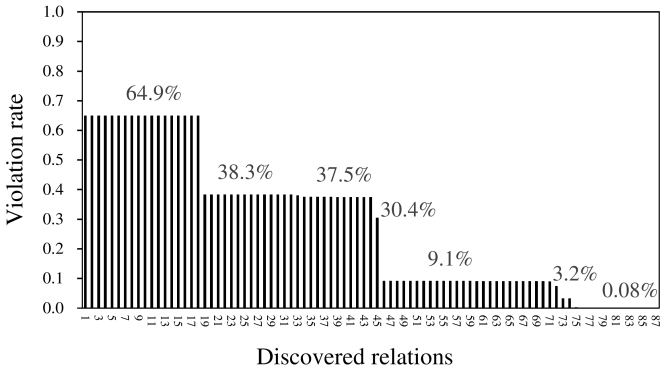


Fig. 8. Distribution of violated relations in HDFS

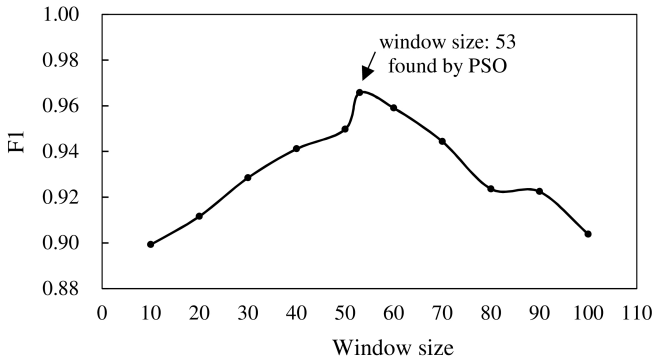


Fig. 9. F1 scores for anomaly detection on testing set when adopting different window sizes

how often each discovered relation from HDFS is violated by the anomalies. The relations are sorted by their violation rates from high to low. As shown in Formula 10, we define a relation's violation rate as the percentage of the anomalies that violate this relation. It can be seen that some relations have much higher violation rates than others. For example, 64.9% of the abnormal sessions violated the same 18 relations. One example of the relation is *c(E1) = c(E2)* where E1 is *"Deleting block* $\langle * \rangle$ *file* $\langle * \rangle$*"* and E2 is *"BLOCK* $\langle * \rangle$ *NameSystem.delete:* $\langle * \rangle$ *is added to invalidSet of* $\langle * \rangle$*"*. Also, some of the discovered relations are only violated by a very few anomalies. The discrepancy between the high and low violation rates suggests that some workflows may be more vulnerable and should be paid more attention to.

$$\text{violation rate} = \frac{\text{\# anomalies violating the relation}}{\text{\# total detected anomalies}} \quad (10)$$

Overall, we have found that ADR has a strong capacity to extract the relations from log data and use them to detect anomalies even when the training set is small. This also increases the applicability of log-based anomaly detection because labelling the log data manually is tedious and time-consuming.

C. Results of RQ3

We used BGL log entries for the PSO-based optimization process. PSO found that the optimal window size to split the BGL logs was 53. In other words, the rank difference between the abnormal and normal event-count-matrices is the biggest when each sliding window consists of 53 log entries. To investigate whether this window size improves the anomaly detection efficiency, we used the PSO returned window size as well as several other window sizes to split the log entries and detect the anomalies in the remaining testing set. The F1 scores achieved by different window sizes are shown in Figure 9. It can be seen that there does exist an optimal window size for which the anomalies can be detected more efficiently. This can be attributed to the reason that several numerical relations are missed when adopting an improper window size. In a previous study [7], the authors manually chose several time window sizes and also found that some of them resulted in better anomaly detection results. Therefore, using the optimization approaches such as PSO could be a promising idea to tune the parameters in log analysis.

As the sliding windows are used for online anomaly detection, the response time for detecting a new window is

very important. For online detection, as soon as a new log entry is generated by the system, the current window will slide forward to include the new log entry and will be checked against the numerical relations. In the experiments, we observed that the average time to determine whether a new window was abnormal or not was only 11ms, which was nearly a real-time response. The reason is that checking the new window against the numerical relations is actually to calculate the product of two matrices, which is a very fast operation in common mathematical packages [34], [30].

VII. Threats to Validity

We have identified the following threats to validity:

- Subject datasets. In this paper we use the logs collected from the Hadoop Distributed File System (HDFS) and a supercomputer system (BGL). Although both of the datasets come from real systems and contain millions of logs, the number of the subjects is still limited. In the future, we will try to collect the logs from various systems and conduct experiments on them.
- Noise in log data. In this study, we use matrix nullspace to extract the event relations, which achieved high precision and recall scores in the experiments. However, the nullspace strictly satisfies the training set. Therefore, if the training set contains some errors (e.g. log parsing errors and label errors), the nullspace will shrink dramatically. In our future work, we will introduce a tolerance measure so that the relations are not strictly required to fulfil all the training set.
- Correspondence between relations and workflows. In this paper, each discovered relation is considered to correspond to a workflow, but there exists a possibility that several workflows share the same relation. Though the relations have shown strong capacity to identify the anomalies, they could provide more information for diagnosis if the workflows can be precisely determined. In future work we will utilize more information in the logs (e.g. the timestamps of events) to precisely determine the workflows.

VIII. Related Work

The research on automatic log analysis mainly focuses on two problems: the first is how to accurately parse the unstructured logs to structured logs and the second is to automatically determine the system status from logs. There have been several log parsers addressing the first problem, such as IPLoM [35], LKE [23], Spell [12], Drain [10]. A comprehensive comparison of their mechanisms and performances are systematically summarized in an ICSE'19 paper [27].

There are many automatic log-based anomaly detection methods, including supervised approaches (such as Logistic Regression, Decision Tree [2], Support Vector Machine [2], Deep Learning [15]) and unsupervised approaches (such as Log Clustering [4], Principle Component Analysis [5], Invariants Mining [6]). Logistic Regression based method [1] uses labelled instances to train a logistic function to estimates the probability of the system being abnormal. SVM-based method [3] tries to construct a hyperplane to separate normal and abnormal instances in high-dimensional space. PCA-based method [5] generates two sub-spaces, named normal space and anomaly space, and the distance to normal space is used to identify the anomalies. IM [6] employs a brute force combined with greedy search method to discover linear relations between the occurrences of system events and the instances which violate the invariants are considered as anomalies. Log Clustering [4] groups normal log instances into clusters and new instances that do not fall into any clusters are considered as anomalies. He et al. [21] proposed Log3C to identify impactful system problems by utilizing both log sequences and system KPIs (Key Performance Indicators). It designs a fast cascading clustering algorithm for iteratively sampling, clustering, and matching log sequences and then identifies the problems by correlating the clusters with system KPIs. To identify and handle unstable log events and sequences, LogRobust [14] was proposed, which can extract semantic information of log events and then detects anomalies by utilizing an attention-based Bi-LSTM model.

Though He et al. [7] has found that supervised approaches usually achieve better performances than unsupervised approaches, the biggest concern for the supervised approaches is that they require labelled logs to train the models. However, labelling the logs requires expertise of the subject system and tedious manual work, especially if the approach needs a lot of training data. Another drawback of the current approaches is that the approaches can identify the anomalies but cannot provide extra information for troubleshooting, because the original logs are usually transformed in the models. The approach Invariants Mining could find meaningful invariants in the logs, which are human understandable so as to help operators to find the cause of the anomalies [6]. But Invariant Mining suffers from performance problems when the size of the logs grows because of the expansion of the search space [7]. ADR proposed in this paper requires a small number of logs produced by a normally running system, so it alleviates the pain of labelling. Moreover, the numerical relations mined by ADR correspond to certain workflows of the system so that they could provide hints for diagnosing the anomalies.

IX. Conclusion

In this paper, we propose ADR, which can automatically extract relations among log events and detect system anomalies in both offline and online manners. The experimental results on two public log datasets show that ADR can discover 87 and 669 relations from HDFS and BGL logs within just a few seconds. It can achieve high precision and recall scores (all greater than 0.9) in anomaly detection using a small training set and outperforms the state-of-the-art methods. Moreover, ADR employs an optimization approach, PSO, to find the optimal window size to split the log entries. By examining the violated relations of the log sequences, engineers can achieve a better understanding of the system anomalies.

Our experimental tool and data are publicly available at https://github.com/bolzzzz/ADR.

In the future, we will evaluate ADR on more log datasets from a variety of systems. We will also extend the proposed approach to support anomaly detection with noisy log data.

ACKNOWLEDGMENT

This work is supported by Australian Research Council (ARC) Discovery Project DP200102940.

REFERENCES

[1] P. Bodik, M. Goldszmidt, A. Fox, D. B. Woodard, and H. Andersen, "Fingerprinting the Datacenter: Automated Classification of Performance Crises," in *Proceedings of the 5th European Conference on Computer Systems*, ser. EuroSys '10. Paris, France: ACM, 2010, pp. 111–124.

[2] M. Chen, A. X. Zheng, J. Lloyd, M. I. Jordan, and E. Brewer, "Failure diagnosis using decision trees," in *International Conference on Autonomic Computing, 2004. Proceedings.*, May 2004, pp. 36–43.

[3] Y. Liang, Y. Zhang, H. Xiong, and R. Sahoo, "Failure Prediction in IBM BlueGene/L Event Logs," in *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, Oct. 2007, pp. 583–588.

[4] Q. Lin, H. Zhang, J. Lou, Y. Zhang, and X. Chen, "Log Clustering Based Problem Identification for Online Service Systems," in *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*, May 2016, pp. 102–111.

[5] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting Large-scale System Problems by Mining Console Logs," in *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, ser. SOSP '09. Big Sky, Montana, USA: ACM, 2009, pp. 117–132.

[6] J.-G. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li, "Mining Invariants from Console Logs for System Problem Detection," in *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, ser. USENIXATC'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 24–24.

[7] S. He, J. Zhu, P. He, and M. R. Lyu, "Experience Report: System Log Analysis for Anomaly Detection," in *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, Oct. 2016, pp. 207–218.

[8] A. S. Foundation, "Apache Hadoop," https://hadoop.apache.org/, 2019.

[9] A. Oliner and J. Stearley, "What Supercomputers Say: A Study of Five System Logs," in *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, Jun. 2007, pp. 575–584.

[10] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An Online Log Parsing Approach with Fixed Depth Tree," in *2017 IEEE International Conference on Web Services (ICWS)*, Jun. 2017, pp. 33–40.

[11] Z. M. Jiang, A. E. Hassan, G. Hamann, and P. Flora, "An automated approach for abstracting execution logs to execution events," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 20, no. 4, pp. 249–267, 2008.

[12] M. Du and F. Li, "Spell: Streaming Parsing of System Event Logs," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*, Dec. 2016, pp. 859–864.

[13] A. Makanju, A. N. Zincir-Heywood, and E. E. Milios, "A Lightweight Algorithm for Message Type Extraction in System Application Logs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 11, pp. 1921–1936, Nov. 2012.

[14] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li, J. Chen, X. He, R. Yao, J.-G. Lou, M. Chintalapati, F. Shen, and D. Zhang, "Robust Log-based Anomaly Detection on Unstable Log Data," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2019. Tallinn, Estonia: ACM, 2019, pp. 807–817.

[15] M. Du, F. Li, G. Zheng, and V. Srikumar, "DeepLog: Anomaly Detection and Diagnosis from System Logs Through Deep Learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. Dallas, Texas, USA: ACM, 2017, pp. 1285–1298.

[16] L. Mariani and F. Pastore, "Automated Identification of Failure Causes in System Logs," in *2008 19th International Symposium on Software Reliability Engineering (ISSRE)*, Nov. 2008, pp. 117–126.

[17] T. Li, Y. Jiang, C. Zeng, B. Xia, Z. Liu, W. Zhou, X. Zhu, W. Wang, L. Zhang, J. Wu, L. Xue, and D. Bao, "FLAP: An End-to-End Event Log Analysis Platform for System Management," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '17. Halifax, NS, Canada: ACM, 2017, pp. 1547–1556.

[18] C. Kruegel and G. Vigna, "Anomaly Detection of Web-based Attacks," in *Proceedings of the 10th ACM Conference on Computer and Communications Security*, ser. CCS '03. Washington D.C., USA: ACM, 2003, pp. 251–261.

[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 *2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)*, Nov. 2015, pp. 24–34.

[20] J. Breier and J. Branišová, "A Dynamic Rule Creation Based Anomaly Detection Method for Identifying Security Breaches in Log Records," *Wireless Personal Communications*, vol. 94, no. 3, pp. 497–511, Jun. 2017.

[21] S. He, Q. Lin, J.-G. Lou, H. Zhang, M. R. Lyu, and D. Zhang, "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*, ser. ESEC/FSE 2018. Lake Buena Vista, FL, USA: ACM, 2018, pp. 60–70.

[22] H. Hamooni, B. Debnath, J. Xu, H. Zhang, G. Jiang, and A. Mueen, "LogMine: Fast Pattern Recognition for Log Analytics," in *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, ser. CIKM '16. Indianapolis, Indiana, USA: ACM, 2016, pp. 1573–1582.

[23] Q. Fu, J.-G. Lou, Y. Wang, and J. Li, "Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis," in *International Conference on Data Mining (Full Paper)*. IEEE, Dec. 2009.

[24] G. Strang, *Linear Algebra and Its Applications, 4th Edition*, 4th ed. Belmont, CA: Cengage Learning, 2006.

[25] I. Wikimedia Foundation, "Gaussian elimination," *Wikipedia*, Oct. 2019.

[26] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of the 1995 IEEE International Conference on Neural Networks*, vol. 4. IEEE, 1995, pp. 1942–1948.

[27] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, "Tools and Benchmarks for Automated Log Parsing," in *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice*, ser. ICSE-SEIP '10. Montreal, Quebec, Canada: IEEE Press, May 2019, pp. 121–130.

[28] B. Matthews, "Comparison of the predicted and observed secondary structure of T4 phage lysozyme," *Biochimica et Biophysica Acta (BBA) - Protein Structure*, vol. 405, no. 2, pp. 442–451, 1975.

[29] J. Zhang, J. Chen, D. Hao, Y. Xiong, B. Xie, L. Zhang, and H. Mei, "Search-based Inference of Polynomial Metamorphic Relations," in *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, ser. ASE '14. New York, NY, USA: ACM, 2014, pp. 701–712.

[30] The SciPy community, "SciPy," https://www.scipy.org/, 2019.

[31] The SymPy Development Team, "SymPy," https://www.sympy.org, 2018.

[32] I. The MathWorks, "MATLAB 2019b," https://www.mathworks.com.

[33] J.-G. Lou, Q. Fu, S. Yang, J. Li, and B. Wu, "Mining Program Workflow from Interleaved Traces," in *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '10. Washington, DC, USA: ACM, 2010, pp. 613–622.

[34] N. D. Team, "NumPy," http://www.numpy.org, 2019.

[35] A. A. Makanju, A. N. Zincir-Heywood, and E. E. Milios, "Clustering Event Logs Using Iterative Partitioning," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '09. Paris, France: ACM, 2009, pp. 1255–1264.