LogTransfer: Cross-System Log Anomaly Detection for Software Systems with Transfer Learning

Rui Chen §, Shenglin Zhang §, Dongwen Li§, Yuzhe Zhang§, Fangrui Guo§,
Weibin Meng†, Dan Pei†, Yuzhi Zhang * §, Xu Chen §, Yuqing Liu §
§Nankai University, {rzchen, lidongwen, zyzcs, guofangrui, 1612839, 1612932}@mail.nankai.edu.cn,
{zhangsl, zyz}@nankai.edu.cn
†Tsinghua University, mwb16@mails.tsinghua.edu.cn, peidan@tsinghua.edu.cn

***Abstract*—System logs, which describe a variety of events of software systems, are becoming increasingly popular for anomaly detection. However, for a large software system, current unsupervised learning-based methods are suffering from low accuracy due to the high diversity of logs, while the supervised learning methods are nearly infeasible to be used in practice because it is time-consuming and labor-intensive to obtain sufficient labels for different types of software systems. In this paper, we propose a novel framework, LogTransfer, which applies transfer learning to transfer the anomalous knowledge of one type of software system (source system) to another (target system). We represent every template using Glove, which considers both global word co-occurrence and local context information, to address the challenge that different types of software systems are different in log syntax while the semantics of logs should be reserved. We apply an LSTM network to extract the sequential patterns of logs, and propose a novel transfer learning method sharing fully connected networks between source and target systems, to minimize the impact of noises in anomalous log sequences. Extensive experiments have been performed on switch logs of different vendors collected from a top global cloud service provider. LogTransfer achieves an averaged 0.84 F1-score and outperforms the state-of-the-art supervised and unsupervised log-based anomaly detection methods, which are consistent with the experiments conducted on the public HDFS and Hadoop application datasets.**

***Index Terms*—Transfer learning, system log, anomaly detection, word embedding, LSTM**

I. Introduction

As the scale and complexity of software services increase dramatically, a large number of software systems have been deployed on different types of services, *e.g.,* cloud computing, e-commerce, 5G, blockchain. They continuously provide services for millions of users all over the world, and thus reliability and availability are of vital importance to them. However, anomalous behaviors may occur on these software systems occasionally, which can affect system reliability and availability, impact user experience, or even lead to economic losses [1]. These anomalies could be introduced by various types of causes, *e.g.,* software bugs, malicious attacks, memory leaks, disk failures, or misconfigurations. Therefore, operators carefully monitor these software systems in order to rapidly detect anomalies and timely mitigate them.

*Yuzhi Zhang is the correspondence author.

[SIF pica_sif]Interface te-1/1/11, changed state to down
[SIF pica_sif]Interface te-1/1/11, changed state to up
[OSPF]Neighbour(rid:, addr:) on vlan20, changed state from Init to ExStart
[OSPF]Neighbour(rid:, addr:) on vlan20, changed state from ExStart to Exchange
[OSPF]Neighbour(rid:, addr:) on vlan20, changed state from Exchange to Loading
[OSPF]Neighbour(rid:, addr:) on vlan20, changed state from Loading to Full
[OSPF]Neighbour(rid:, addr:) on vlan20, changed state from Full to Down
[SIF]Vlan-interface vlan20, changed state to down
[SIF]Vlan-interface vlan20, changed state to up
%%10IFNET/3/LINK_UPDOWN(l): GigabitEthernet1/0/10 link status is DOWN.
%%10IFNET/3/LINK_UPDOWN(l): GigabitEthernet1/0/10 link status is UP.
%%10OSPF/3/OSPF_NBR_CHG(l): OSPF 1 Neighbor (Vlan-interface20) from Loading to Full.
%%10OSPF/3/OSPF_NBR_CHG(l): OSPF 1 Neighbor (Vlan-interface20) from Full to ExStart.
%%10OSPF/3/OSPF_NBR_CHG(l): OSPF 1 Neighbor (Vlan-interface20) from Full to Down.
%%10OSPF/3/OSPF_NBR_CHG(l): OSPF 1 Neighbor (Vlan-interface20) from Full to Init.
%%10IFNET/3/LINK_UPDOWN(l): Vlan-interface20 link status is DOWN.
%%10IFNET/3/LINK_UPDOWN(l): Vlan-interface20 link status is UP.

Fig. 1. The anomalous log sequences generated by two different types of software systems. The two sequences are very similar in semantics but different in syntax.

System logs have become increasingly popular for anomaly detection recently [2]–[5], because they could not only reflect the status of software systems but also reveal root causes. For example, the log "Interface te-1/1/11, changed state to down" tells us that an anomaly may occur on the system since the state of an interface becomes down. This cannot be obtained from performance counters (*e.g.,* page view count), usage statistics (*e.g.,* CPU utilization), or system metrics (*e.g.,* number of threads), which are usually time series data [6].

Due to the large scale and high diversity of logs in today's software services, it is very challenging to detect anomalies based on logs. Specifically, a large software service typically consists of various types of software systems, and each type of software system would generate a specific type of logs. Different types of logs are usually different in syntax. For example, Fig. 1 shows two log sequences that are generated by two different types of software systems. Both of the two sequences indicate that the software systems are suffering from interface flapping. They are different in syntax, although they are very "similar" in semantics. Therefore, it is nearly infeasible to use a log-based anomaly detection model to fit all types of software systems. In other words, we have to train a specific anomaly detection model for each type of software

system. A large software service usually includes tens to hundreds of different types of software systems. For example, operators can deploy different types of operating systems (say Ubuntu, Redhat, CentOS) on servers, and each type of operating system generates a specific type of logs. In addition, a software service provider usually purchases devices from different vendors with different models for commercial considerations, and logs generated by those switches/routers/firewalls of different vendors/models are typically different.

A large number of machine learning-based log anomaly detection methods have been proposed over the years, and they are either supervised methods [7]–[10] or unsupervised methods [2], [11]–[15]. However, it is still challenging to deploy these methods in a real-world large software service with diverse types of software systems, where labelled data is insufficient while accurate and rapid anomaly detection is required. Although unsupervised learning methods do not need any labels and thus they are applicable for large volumes of logs, they usually suffer from low accuracy [10]. Supervised methods, which usually require sufficient labels, can achieve higher accuracy than those unsupervised ones. Nevertheless, it is time-consuming and labor-intensive to label anomalous and normal logs because of the large volume and high diversity of logs. Therefore, supervised learning methods are very difficult, if not impossible, to be deployed in large cloud systems in practice.

To build an accurate and applicable log-based anomaly detection model, we propose LogTransfer, which enables cross-system anomaly detection for software systems in a large software service. The core idea of the cross-system is to conduct anomaly detection on a software system with insufficient anomaly labels (target system) by learning from a software system with sufficient anomaly labels (source system). The intuition behind LogTransfer is that the anomalous logs of different types of software systems usually share similar patterns. For example, as shown in Fig 1, the two anomalous log sequences generated by two different types of software systems are quite similar. We can improve a target system's anomaly detection performance by "transferring" the "similarity" from a source system to it.

However, the design of a cross-system log anomaly detection faces the following two challenges:
(1) **Different types of systems are different in log syntax.** Cross-system transfer learning should measure the similarities of logs between source and target system. However, the state-of-the-art log representation method, which only considers local context information, cannot robustly measure the similarity of logs across multiple datasets with various syntax.
(2) **Noises in anomalous log sequences.** The anomalous log sequences usually have noises that can degrade the performance of anomaly detection. For example, in an anomalous log sequence, there are usually one or more logs that are irrelevant to this anomaly, which brings a great challenge to identify anomalous log sequences [16].

To address the above two challenges, we design LogTransfer as follows.

(1) An accurate representation construction method. We first extract log templates from raw logs using FT-tree [17], which has been demonstrated to be accurate and efficient on real-world system logs. After that, we employ Glove [18], a popular unsupervised word representation technique, to represent every word in the templates by fixed-dimension vectors. Glove combines the global word co-occurrence and local context information of template words, which are in turn integrated (*e.g.,* averaged) to represent every template. In this way, the representation of templates minimizes the impact of syntax (*i.e.,* the order of words) while preserving the semantics information (*i.e.,* the meaning of templates), and thus robustly measures the similarities of cross-system logs.
(2) A novel transfer learning method. We apply Long Short Term Memory (LSTM) networks to identify the sequential patterns of template sequences. The output of LSTM is then fed into fully connected networks for anomaly detection (classification). LogTransfer uses a novel transfer learning approach where the source and target system share the same fully connected networks instead of the same LSTM networks, which are more robust to the noises in log sequences because the former way is more tolerant to the trivial differences of log sequences than the latter case.

To demonstrate the performance of LogTransfer, we have conducted extensive experiments on switch logs of different vendors collected from a top global cloud service provider. LogTransfer achieves an averaged accuracy of 0.84 (in terms of F1-score) on these switch logs, which outperforms the state-of-the-art supervised and unsupervised log anomaly detection methods. Besides, we also perform experiments on public HDFS and Hadoop application datasets to demonstrate LogTransfer's generality.

The contributions of this paper are summarized as follows:
(1) To the best of our knowledge, we are the first to apply transfer learning for log anomaly detection and identify the challenges lying in that for a large software service.
(2) We propose to use Glove to construct logs' representations to accurately measure the similarities of cross-system logs.
(3) We propose a novel transfer learning approach that shares fully connected networks between source and target systems, addressing the impact induced by the noises in log sequences.
(4) We have conducted extensive evaluation experiments using real-world logs to demonstrate LogTransfer's performance.

The rest of this paper is organized as follows. We introduce some preliminary knowledge of log-based anomaly detection in Section II, and present the design of LogTransfer in Section III. The evaluation experiments are described and discussed in Section IV, followed by the discussion of related works in Section V. Finally, we conclude our work in Section VI.

II. Preliminary

A. System Logs and Templates

Each software system reports, from time to time, the observed condition or (anomalous) event, in a system log. Examples of such conditions or events include state changes

of interfaces, links, or neighbors (*e.g.*, the state of an interface changes from up to down), operational maintenance (*e.g.,* operators log in/out), environmental condition alerts (*e.g.,* high temperature), *etc.* Although logs are designed mainly for debugging software and hardware problems, they can also be used for detecting, diagnosing, and predicting anomalies [2]–[5], [10], [16].

A log entry usually consists of two parts: constant part (*template*) and parameter part. For instance, in the syslog message **Interface te-1/1/11, changed state to down** shown in Fig. 1, **te-1/1/11** is a parameter that varies from one log to another, whereas the rest, *i.e.,* **Interface ..., changed state to down**, sketches out the event, and hence is a *template* that summarizes this and other similar logs. The logs and their templates generated by different types of software systems are different in syntax because they are usually designed by different vendors/developers.

Several automatic template extraction approaches have been proposed, which can be classified into four main categories [19]: longest common subsequence-based methods (*e.g.,* Spell [20]), frequent item mining-based methods (*e.g.*, FT-tree [17]), heuristics-based methods (*e.g.*, IPLoM [21] and Drain [22]), and cluster-based methods (*e.g.,* LogSig [23]). In this work, we apply FT-tree, which has been demonstrated to be accurate and efficient using real-world logs, to extract templates from logs. Note that applying FT-tree is not one of our contributions.

B. System Anomaly

System anomalies can be roughly classified into several different types, including (1) external problems such as malicious attacks, power down; (2) configuration problems such as VPN tunneling errors; (3) hardware failures such as the crash, induced by hardware errors, of a machine; and (4) software crash due to bugs. These anomalies could lead to failures to software systems and in turn impact user experience and/or bring economic loss. Therefore, operators are paying much attention to anomaly detection in order to mitigate anomalies as quickly as possible.

Although a small part of service anomalies can be detected through service metrics (*e.g.,* average response time), a large portion of service anomalies remain undetected. That is because only a small portion of anomalies lead to such service anomalies that can be observed by these service metrics, and most anomalies are underlying ones that deteriorate gradually and degrade the performance of services finally. For example, the memory leaking of a software system does not impact service performance at the beginning, but it will eventually degrade service performance if not fixed. Therefore, it is vitally important for operators to proactively detect all anomalies based on system logs and mitigate them before they impact service performance.

For supervised log-based anomaly detection methods, manually labeling logs indicating system anomalies is labor-intensive, time-consuming and error-prone, because logs are very large in quantity and diverse in represented events. Therefore, in this paper, we try to apply transfer learning to minimize this effort.

C. LSTM in Anomaly detection

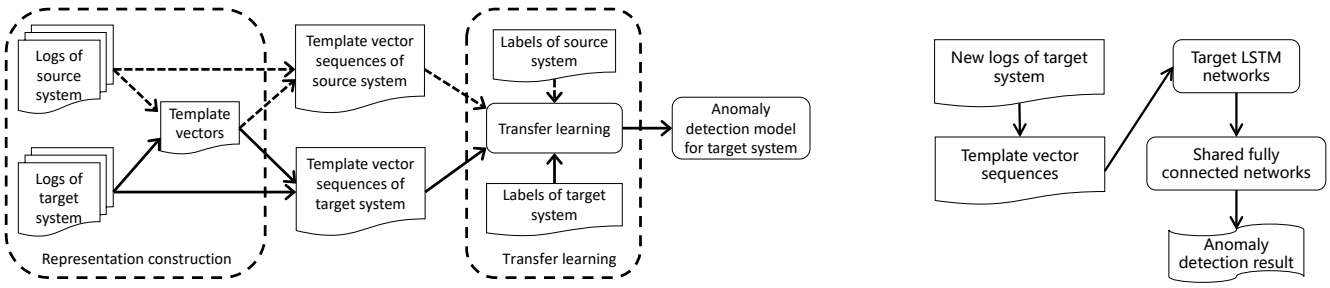
A Recurrent Neural Network (RNN) is an artificial neural network in which nodes are directionally connected. The current internal state of an RNN block relies on both current input and previous states. Long Short-Term Memory (LSTM)) networks are instances of RNNs, and they can solve the long-term dependency issue in RNN (*i.e.,* the current state may be affected by the state a long time ago). Inspired by the fact that system logs are essentially a series of log entries in chronological order, and a log anomaly is usually identified by a log sequence, DeepLog [2] applied LSTM to first extract sequential patterns from log sequences, and then predict the template of the next log given a log sequence. Moreover, LogAnomaly [15] also leveraged LSTM to extract sequential features from log sequences. Both DeepLog and LogAnomaly, which have demonstrated their performance using real-world logs, show LSTM's superior performance in extracting sequential features from logs. Consequently, in this work, we also apply LSTM to learn logs' sequential patterns.

III. Design of LogTransfer

A. Design Overview

The architecture of LogTransfer is shown in Fig. 2. In the offline model training process, for both source system and target system, LogTransfer first matches logs to templates, constructs template vectors to represent templates and generates sequences of template vectors from raw logs and template vectors. With the sufficient labels of source system, the insufficient labels of target system, and the above template vector sequences, LogTransfer transfers the anomalous patterns from the source system to the target system, so as to train an accurate anomaly detection model for the target system. In the online anomaly detection process, LogTransfer matches new logs of the target system to template vector sequences based on the template vectors learned in the offline training process. After that, the LSTM networks and shared fully connected networks trained in the offline learning process extract the sequential features from these template vector sequences and determine whether these sequences are anomalous, respectively.

There are two main components in LogTransfer, namely representation construction and transfer learning. In the representation construction component, in order to simultaneously avoid the impact of syntax and reserve the semantics information, LogTransfer applies Glove [18], a robust word embedding method, to robustly embed template words into vectors. In the transfer learning component, LogTransfer first trains a base model which is composed of *source LSTM networks* and fully connected networks using the log sequences and anomaly labels from the source system. Then, source LSTM networks in the base model will be fine-tuned by the log sequences and anomaly labels from the target system to



(a) Offline model training process

(b) Online anomaly detection process

Fig. 2. The architecture of LogTransfer

Raw system log:
[SIF pica_sif] Interface te-1/1/11, changed state to down

Log template:
Interface *, changed state to down

Word embeddings:
Interface: $V_1 \rightarrow (v_{11}, v_{12}, v_{13},..., v_{1n})$
changed: $V_2 \rightarrow (v_{21}, v_{22}, v_{23},..., v_{2n})$
state: $V_3 \rightarrow (v_{31}, v_{32}, v_{33},..., v_{3n})$
to: $V_4 \rightarrow (v_{41}, v_{42}, v_{43},..., v_{4n})$
down: $V_5 \rightarrow (v_{51}, v_{52}, v_{53},..., v_{5n})$

Template embedding:
Interface *, changed state to down:
t $\rightarrow$ average($V_1 + V_2 + V_3 + V_4 + V_5$)

Fig. 3. Example of how to generate a template embedding. In representation construction component, a single log entry is matched to its template embedding following this procedure.

obtain *target LSTM networks*. Fully connected networks are shared and connected to target LSTM networks. Sharing the fully connected networks rather than the LSTM networks makes LogTransfer more robust to the noises in log sequences because it is more tolerant to the trivial differences of log sequences.

B. Representation Construction

The representation construction component tries to represent logs in order to reserve logs' semantics information and minimize the impact of syntax simultaneously for robustly measuring the similarities of cross-system logs. These representations are the input into machine learning models to extract logs' features and learn anomaly's patterns. We present an example of how to generate the template embedding for a given log entry in the representation construction component in Fig. 3.

It is a common practice to first extract templates from logs and then matches logs to templates [19] for representing logs. FT-tree [17], which is accurate, efficient, and supports incremental learning, has demonstrated its good performance on real-world system logs. Therefore, in this work, we apply FT-tree to extract templates from logs. Note that applying FT-tree is not one of our contributions.

Using embedding rather than indices of templates has been demonstrated to be able to extract the semantics information from logs and facilitate a more accurate anomaly detection model [15]. Although LogAnomaly takes a first step towards applying embedding to represent templates, the word embedding method used in it, *i.e.,* word2Vec, can only extract the local context information of templates, without considering the global information of the whole template set. Ignoring global information can degrade the performance of measuring the similarities of cross-system logs and their templates (more details can be seen in Section IV-C).

Therefore, LogTransfer employs Glove [18], which not only extracts the local context information but also considers the global word co-occurrence information of template set, to represent template words. Formally, in Glove, the objective function is:

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2 \tag{1}$$

where V is the size of the template set, w_i and b_i represent the embedding and bias of word i, respectively, and $\tilde{w}_j$ and $\tilde{b}_j$ represent the vector and bias of a separate context word j. In addition, X_{ij} denotes the number of occurrences of word j within the context of the word i in the global occurrence matrix X). Moreover, $f(x)$ is the weighting function which is defined as follows:

$$f(x) = \begin{cases} (x/x_{\max})^{\alpha} & \text{if } x < x_{\max} \\ 1 & \text{otherwise} \end{cases} \tag{2}$$

we set $\alpha = 3/4$ following [18].

A template vector t is then calculated as:

$$t = \sum_{k=1}^{n} w_k / n \tag{3}$$

where n is the number of words in the template. In this way, a template vector is irrelevant to the syntax (*i.e.,* the order of words) of logs and their templates.

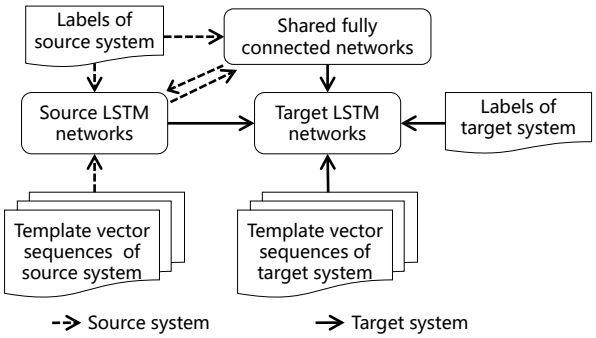


Fig. 4. The architecture of transfer learning component

C. Transfer Learning

Because (1) target systems do not have sufficient anomaly labels to train accurate anomaly detection models, (2) source systems and target systems share the same or similar log patterns when anomalies occur (*e.g.,* the example shown in Fig. 1), and (3) transfer learning enables learning patterns from data across different domains and/or distributions [24], LogTransfer applies transfer learning to "transfer" the learned anomaly patterns from source system to target system.

LogTransfer applies LSTM to extract sequential features from logs and uses fully connected layers to determine whether the output is anomalous or not. The performance of transfer learning depends on the layers used for transferring [25]. Because a source system usually has much more anomaly labels than a target system, and the anomaly patterns of the source system can cover those of the target system, LogTransfer shares the fully connected networks, which are essentially classifiers determining whether a log sequence is anomalous or not, to transfer the learned anomaly patterns from the source system to the target system. We do not share the LSTM networks, which extract the sequential features of logs, because the noises mingling in anomalous log sequences can degrade the similarity between the sequences of the source system and those of the target system. That is to say, if we just copy the parameters of the LSTM networks learned in the source system to the target system without further tuning these parameters based on the logs of the target system, the anomaly detection model will mistakenly determine some anomalous log sequences of the target system as normal (more details can be seen in Section IV-D).

Specifically, as shown in Fig. 4, for a source system LogTransfer first applies source LSTM networks, which are trained based on the labels of the source system, to extract the sequential features from template vector sequences. The outputs of the LSTM networks are then fed into the shared fully connected networks, which are also trained based on the labels of the source system. The fully connected networks and source LSTM networks continuously train each other until both of them achieve good performance. For the target system, LogTransfer applies target LSTM networks, which are initialized based on source LSTM networks, to learn the sequential patterns of logs. The LSTM networks are fine-tuned based on the labels of the target system and the shared connected networks. Note that the shared connected networks are "transferred" and fixed, and they will not be tuned like LSTM networks, which is the core idea of transfer learning.

Sharing fully connected networks instead of LSTM networks facilitates LogTransfer more robust to the noises in the anomalous log sequences of the target system because the sequential features learned in the target LSTM networks are fine-tuned based on these sequences. LogTransfer is, to the best of our knowledge, the first work to combine LSTM with transfer learning by sharing fully connected networks in the anomaly detection domain.

Typically, LogTransfer can be applied to transfer anomalous log patterns across different types of software systems within the same domain. For example, Red Hat, Cent OS, Ubuntu, and Windows are all operating systems; HDFS, WordCount and PageRank are all Hadoop related application systems; switches, routers, and firewalls are all network devices; search engine, online video, and online shopping are all web services. Therefore, LogTransfer can be used to transfer learning within operating systems, Hadoop related applications, network devices, or web services. It cannot be used to transfer learning across different domains. However, LogTransfer takes the first step towards applying transfer learning in log-based anomaly detection, and it gives several insights for this direction.

IV. Evaluation

A. Experimental Setup

1) Dataset: To evaluate the performance of LogTransfer, we conduct our experiments on switch logs collected from a top global cloud service provider with a 2-year period. These switches belong to three types[1]. Logs of different types of switches are different in syntax. Table I lists the detailed information of the datasets. Since switch type A and type C have much more anomalous chunks than switch type B, we apply the software systems of switch type A and type C as two individual source systems, and that of switch type B as the target system. 200 anomalous chunks of the target system are utilized to fine-tune LogTransfer (more details can be seen in Fig. 11).

To demonstrate the generality of LogTransfer, we further evaluated its performance using two public datasets: the Hadoop application dataset which is collected from Hadoop applications [14], and the HDFS dataset collected from Hadoop file systems [11]. The detailed information of the above two datasets is also listed in Table I. Note that the anomalies of the two Hadoop application datasets, including machine down, network disconnection, and disk full, are manually injected by [14].

The distributions of anomalies in these datasets are shown in Fig. 5. For each dataset, we calculate the percentage of anomalous chunks across different switches/blocks as the

[1]A type of switch is a specific switch model produced by a specific manufacturer.

TABLE I
DATASET OVERVIEW

Source of dataset	Type of system	Source of labels	# chunks	# anomalous chunks	# switches/log files
Switch	Type A	Real-world anomalies	2,345,646	6,406	22
	Type B		49,946	1,096	14
	Type C		525,427	4,939	21
Hadoop application	PageRank & WordCount [14]	Manually injected	121,878	73,936	1008
Hadoop file system	HDFS [11]	Real-world anomalies	3,725,203	108,024	575,061

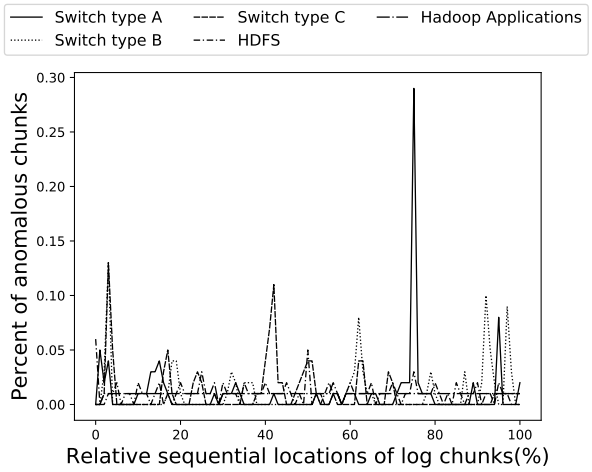


Fig. 5. The distribution of anomalies in the six datasets

relative sequential locations of chunks vary. For example, 28% of switches of type A suffer from anomalies when these switches have generated 78% of total logs. We can see that in general the anomalies are randomly distributed across all log chunks.

2) Metrics: We apply precision, recall, and F1-score to evaluate the performance of LogTransfer following [15]. They are defined as:

$$Precision = \frac{Anomalies\ detected}{Anomalies\ reported} \tag{4}$$

$$Recall = \frac{Anomalies\ detected}{Entire\ anomalies} \tag{5}$$

$$F1\text{-}score = \frac{2(Precision \times Recall)}{(Precision + Recall)} \tag{6}$$

3) Parameters: We investigate three model-related parameters (*i.e.,* α, β, and L), and data-related parameters (*i.e.,* W and S). In LogTransfer, α and L are the number of memory units and that of layers in one LSTM network, respectively. β is the number of memory units in a fully connected layer, and W and S are respectively the length and step size of each sliding window forming a log chunk. As we can see from Fig. 6(a), (b), (c), the performance of LogTransfer remains stable as model-related parameters vary. On the contrary, data-related parameters can impact the performance of LogTransfer. As can be seen from Fig. 6(d), (e), the F1-score of LogTransfter may decrease if W or S is set inappropriately. Intuitively, if W is set too small, the sequential information of a log chunk will be insufficient to detect anomalies. However, if W is set too large, a log chunk may contain many noise log entries which degrade detection accuracy. Similarly, although a larger S may cause some anomalous logs to be undetected, it can also skip some noise logs. Hence, both W and S need to be adjusted carefully so as to achieve good performance. In our evaluation experiments, we set $\alpha = 128$, $L = 2$, $\beta = 192$, $S = 4$, $W = 20$ to achieve a good LogTransfer performance.

4) Environmental Setting: Our experiments are conducted on a server with Intel XeonE5 12 cores CPU with 128GB memory. We have open-sourced LogTransfer[2].

B. Evaluation of Overall Performance

To evaluate the performance of LogTransfer, we compare it with four supervised log-based anomaly detection approaches, including Linear regression [7], SVM [8], Decision tree [9], and CNN-based model [10], as well as six unsupervised approaches, including PCA [11], Isolation forest [12], IM [13], LogCluster [14], DeepLog [2], and LogAnomaly [15]. We implement DeepLog, LogAnomaly, and CNN-based model with Python3.6. PCA, Invariant mining, SVM, Linear regression, LogCluster, Isolation forest and Decision tree are evaluated based on a popular open-source toolkit implemented in [26]. Note that the parameters of all the above methods are set best for accuracy.

Fig. 7 shows the comparison results among LogTransfer, four supervised methods, and six unsupervised methods, when the software system of switch type A or type C is utilized as source system, and that of type B as target system. As shown in Fig. 7(a), (c), LogTransfer outperforms all the four supervised methods in terms of precision, recall, and F1-score, whether type A or type C as the source system. That is because: (1) the target system does not have sufficient anomaly labels to train these supervised methods, and (2) LogTransfer utilizes template embeddings, which sufficiently extracts the semantics information of logs, rather than template indexes, to represent log entries. When we compare LogTransfer with the six unsupervised methods, it still achieves better F1-score

[2]LogTransfer is available on github:https://github.com/logtransfergit/LogTransfer.

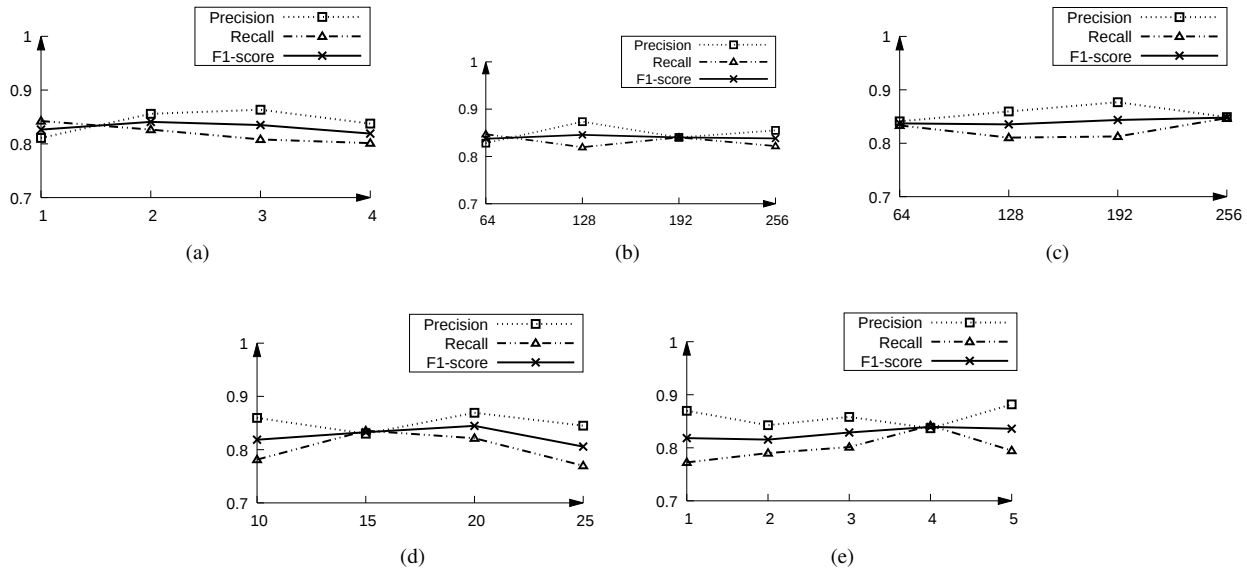


Fig. 6. The performance of LogTransfer as: (a) the number of LSTM layers L; (b) the number of memory units in LSTM network α; (c) the number of memory units in a fully connected network β; (d) window length W; (e) step size S; vary. (switch type A as the source system, and type B as the target system)

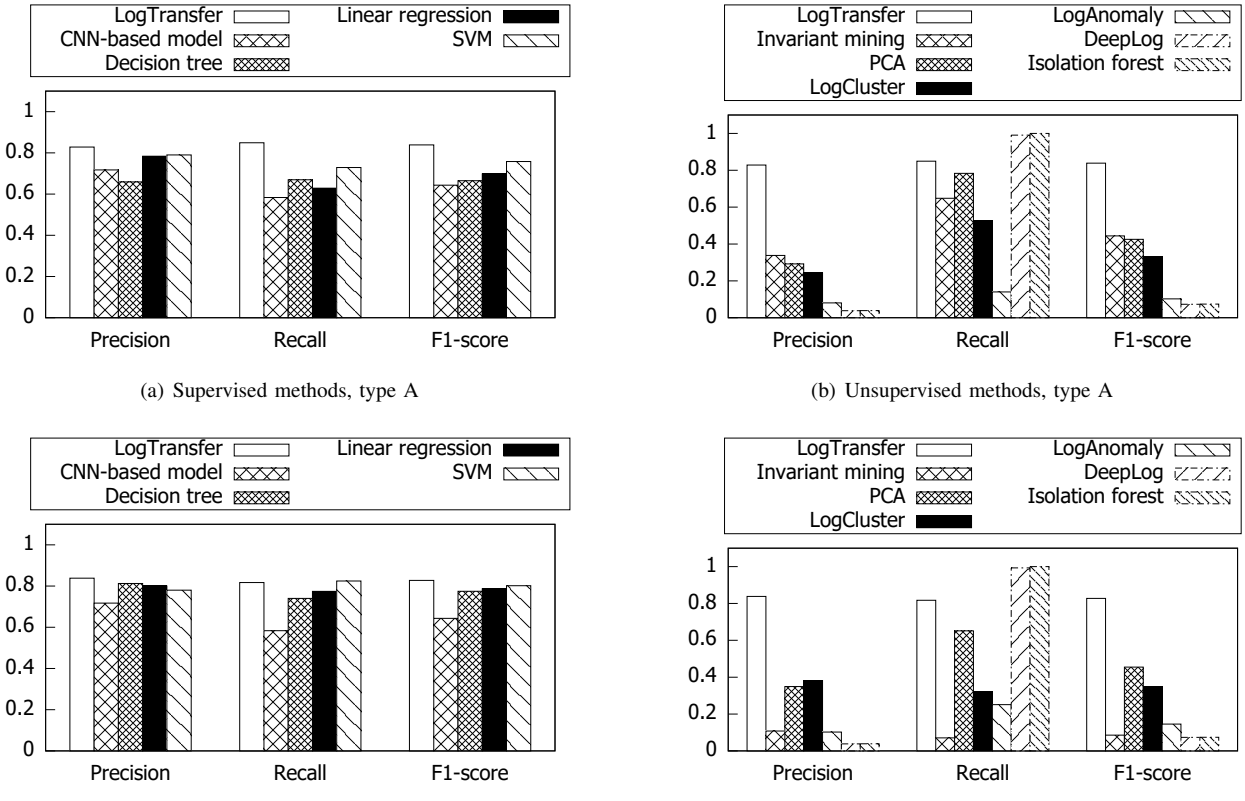


Fig. 7. The overall performance of LogTransfer, four supervised methods, and six unsupervised methods on switch datasets (switch type A or type C as the source system, and type B as the target system)

TABLE II
THE AUC SCORES OF LOGTRANSFER, FOUR SUPERVISED METHODS, AND SIX UNSUPERVISED METHODS

Type	Method	AUC Score
Transfer learning	LogTransfer	0.892
Supervised	Decision tree [9]	0.745
	SVM [8]	0.754
	CNN-based model [10]	0.682
	Linear regression [7]	0.663
Unsupervised	DeepLog [2]	0.5
	LogAnomaly [15]	0.676
	LogCluster [14]	0.578
	Invariant mining [13]	0.592
	PCA [11]	0.538
	Isolation forest [12]	0.523

than these methods, mainly because these methods are not robust to highly diverse real-world system logs. Some unsupervised methods, *e.g.,* DeepLog, Isolation forest, achieve better Recall than LogTransfer. That is because neither DeepLog nor Isolation forest can utilize the semantics information of logs and they tend to mistakenly determine a large number of normal log chunks as anomalous [15], resulting in that these methods suffer from low precision and generate much more false alarms. These false alarms will bring heavy works to operators, and they are not willing to deploy these methods in practice. More specifically, from Fig. 7(b), (d) we can see that all unsupervised methods suffer from low precision. Generally, the patterns of switch system logs are diverse, and a large number of anomalous logs mingle in the training set, which makes unsupervised methods difficult to precisely learn all the normal patterns.

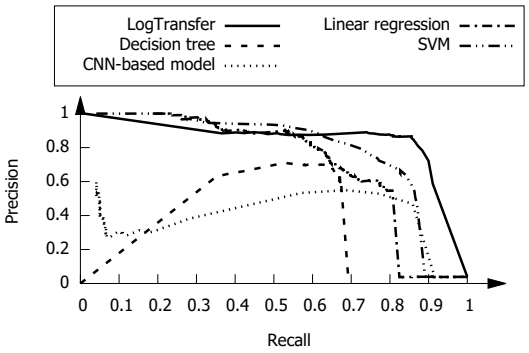
The precision-recall curve (PRC) has been widely used to determine the reliability of a binary classifier. To further demonstrate the robustness of LogTransfer, we plot the PRCs of different methods by varying these methods' thresholds as shown in Fig. 8. We can see that LogTransfer is more robust than both supervised and unsupervised methods.

To further evaluate the overall performance of LogTransfer, we calculate the AUC (area under the curve) scores of it, four supervised methods, and six unsupervised methods, using the software system of switch type A as the source system and that of switch type B as the target system, as listed in Table IV-B. A higher AUC score generally means a better anomaly detection performance. Clearly, LogTransfer achieves the best performance among all the methods.

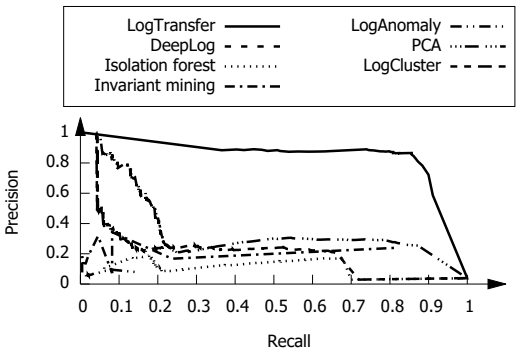
C. Evaluation of Word Embedding Methods

As introduced in Section III-B, LogTransfer applies Glove [18] rather than word2Vec used in LogAnomaly [15], because Glove extracts both local context and global word co-occurrence information, making LogTransfer more robust to measure the similarities of cross-system anomalous logs.

In order to evaluate the performance of Glove in representation construction component, we compare the accuracy of LogTransfer, with that of LogTransfer with word2Vec as



(a) The precision-recall curves of LogTransfer and supervised methods



(b) The precision-recall curves of LogTransfer and unsupervised methods

Fig. 8. Precision-recall curves of LogTransfer, four supervised methods, and six unsupervised methods (the switches of type A as source system and those of type B as target system)

TABLE III
COMPARISON OF WORD EMBEDDING METHODS

Method	F1-score	AUC score	#False positive	#False negative
LogTransfer w/ word2Vec	0.8368	0.8881	88	84
LogTransfer	0.8606	0.9243	80	59

the word embedding method, using the software system of switch type A as the source system and that of switch type B as the target system, as listed in Table III. We can see that Glove helps LogTransfer achieves a higher F1-score and AUC score. That is because, as listed in Table III, Glove successfully misses less anomalous log chunks than word2Vec since it can more accurately measure the similarities of anomalous logs between source system and target system (see Figure 1 for more details).

D. Evaluation of Transfer Learning

LogTransfer shares fully connected networks instead of LSTM networks because the former way is more tolerant to the noises in anomalous log sequences and thus facilitates a more robust anomaly detection model, as described in Section III-C. To demonstrate the benefit of the transfer

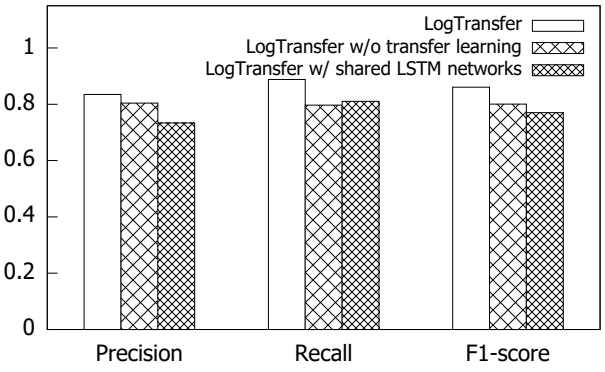


Fig. 9. Evaluation of the transfer learning method

learning design, we compare the performance of LogTransfer, with that of LogTransfer without transfer learning (*i.e.,* the LSTM networks and fully connected networks of the target system are fine-tuned based on their counterparts of source system, respectively), with that of LogTransfer sharing only LSTM networks, as shown in Fig. 9. Apparently, sharing fully connected networks improve accuracy for LogTransfer in terms of both precision and recall. Without transfer learning, the insufficient number of anomaly labels can lead the anomaly detection model to be biased. Moreover, there are usually noises in anomalous log sequences (*e.g.,* the logs indicating operators log in/out of a switch system are irrelevant to anomalies and they are noises as shown in Fig. 10). Sharing LSTM networks only transfer sequential knowledge from the source system to the target system, the performance of which is easily impacted by these noises. Therefore, LogTransfer chooses to share fully connected networks and achieves better performance.

*** logined the switch
*** logouted from the switch
PICALIBCOMM pica_login Fan is plugged in
PICALIBCOMM pica_login RPSU is plugged in serial number ***
Redundancy power supply unit RPSU is plugged in serial number ***
Receive SFP_PRE message module plugged into port ***
10SHELL SHELL_LOGINFAIL TELNET user *** failed to log in from ***
10DEVM POWER REMOVED Trap cPowerRemoved power ID is ***
10SHELL LOGIN Trap cLogIn *** login from ***
10SHELL LOGOUT Trap cLogOut *** logout from ***
10SHELL SHELL_CMD Task IPAddr User *** Command is ***
10LLDP LLDP_CREATE_NEIGHBOR New neighbor created on Port *** ID is ***

Fig. 10. Examples of noises in the anomalous log sequences generated by different types of switches. The logs indicating operators log in/out a switch system is irrelevant to anomalies and they are noises

In order to figure out how many anomalous chunks of target system are needed to train LogTransfer, Fig. 11 shows its accuracy as the number of anomalous chunks increases. We can see that when the number approaches to 200, the accuracy of LogTransfer tends to be stable. Comparing with the total number of anomalous chunks of the target system

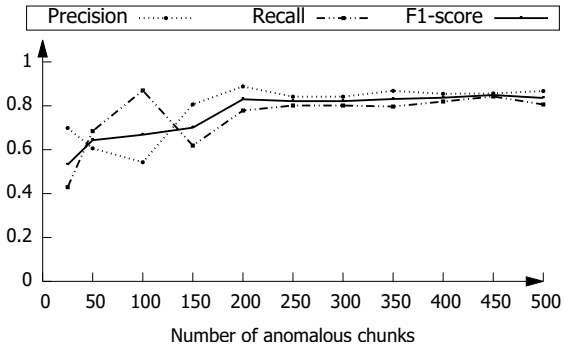


Fig. 11. The accuracy of LogTransfer when the number of anomalous chunks of target system used for training increases

(1000+), what LogTransfer needs for training is much less. Consequently, LogTransfer significantly reduces the labelling efforts for operators.

E. Performance on Hadoop Datasets

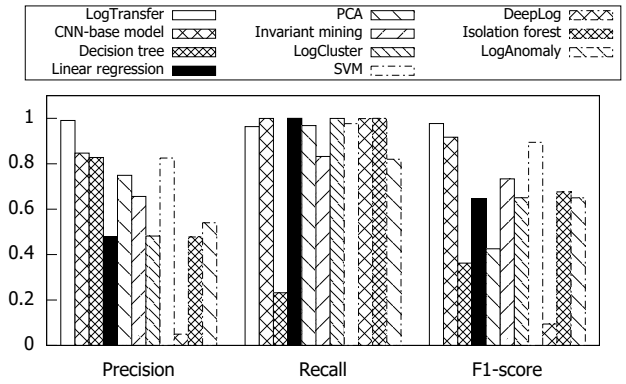


Fig. 12. The performance of LogTransfer, four supervised methods, and six unsupervised methods when HDFS serves as the source system and Hadoop application is the target system.

To demonstrate the generality of LogTransfer, we compare LogTransfer with the above four supervised methods and six unsupervised methods on the HDFS and Hadoop datasets, as shown in Fig. 12. We apply HDFS as the source system and Hadoop application as the target system. LogTransfer achieves the best F1-score (0.977) among all the methods, further demonstrating its superior performance in anomaly detection. More specifically, LogTransfer achieves better precision than the four supervised methods and six unsupervised ones. It is mainly because the novel transfer learning method in LogTransfer enables LogTransfer to be more robust to noises in system logs. Moreover, although some approaches such as Deeplog, Invariant mining, and CNN-base model achieve better recall than LogTransfer, a larger number of normal log chunks are mistakenly determined as anomalous ones by these approaches, resulting in much more false alarms and wasting operators' too much time.

We further evaluate how the novel transfer learning method performs as follows. We gradually increase the number of anomalous chunks of the target system in the training set, and calculate the Precision, Recall and, F1-score of LogTransfer. Fig. 13 shows the result. From the figure, we can conclude that 100 anomalous logs of the target system are enough to fine-tune LogTransfer and obtain a relatively high anomaly detection accuracy. In this way, the manual labelling effort of operators is significantly reduced, which enables LogTransfer easily to be deployed for large-scale service systems.

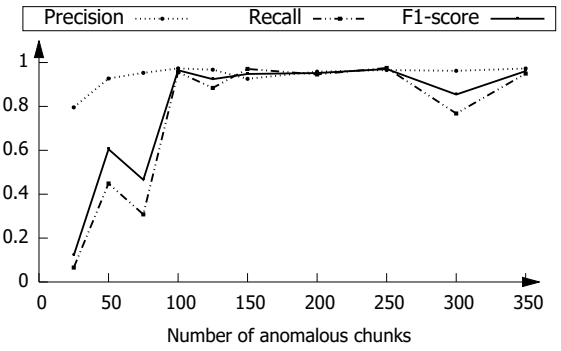


Fig. 13. The accuracy of LogTransfer when the number of anomalous chunks of target system used for training increase on HDFS and Hadoop datasets

V. Related Works

TABLE IV
SUMMARY OF CURRENT LOG BASED ANOMALY DETECTION METHODS

Type	Input of Template	Method
Supervised	Template index	Decision tree [9]
		SVM [8]
		CNN-based model [10]
		Linear regression [7]
Unsupervised	Template count	PCA [11]
	Template index	DeepLog [2]
		LogCluster [14]
		Isolation forest [12]
		Invariant mining [13]
	Template embedding	LogAnomaly [15]

Anomaly detection for software services has attracted lots of attention and a collection of methods have been proposed for this purpose [27]–[34]. Since logs are able to describe a vast range of events for software systems, recently many machine learning methods have been presented to detect anomaly based on logs, which can be classified into two categories: supervised and unsupervised learning-based methods. These methods are summarized in Table IV.

For supervised learning-based methods, Bodik *et al.* trained a logistic regression model based on the event count vectors of logs [7], and Liang *et al.* proposed an SVM based log anomaly detection approach [8]. Moreover, Chen *et al.* classified the predefined features using decision tree [9], and Lu *et al.* utilized a convolutional neural network (CNN) to capture the sequential features of logs [10]. These methods, however, generally need lots of labelled anomalous logs for model training. Because it is labor-intensive and time-consuming to manually labelling anomalies, these methods are very difficult to be applied in practice for large-scale software services. In addition, since these methods consider only the indexes of log templates, they fail to capture the semantics information of logs.

For unsupervised learning-based methods, Xu *et al.* first applied PCA to detect anomalies based on logs [11], and Liu *et al.* presented isolation forest-based method which explicitly isolated anomalies rather than profiled normal points [12]. In addition, Lou *et al.* applied an invariant mining approach to find the inherent relationships of logs [13], and Vaarandi *et al.* tried to automatically mine the anomalous patterns of logs based on LogCluster [35], [36]. Although no labelled data is required in these methods, they usually suffer from low accuracy in real-world service systems [2], [10], [15]. Du *et al.* proposed DeepLog, an unsupervised LSTM-based method which extracted the normal patterns of "log key" (*an index of a log template*) sequences, and determined that a new-coming log was anomalous if it violated the normal patterns learned in the offline procedure [2]. LogAnomaly applied word2Vec to embed templates and extracted the sequential and quantitative features of normal logs [15]. Nevertheless, the word2Vec based word embedding method is not as robust as Glove [18] (more details can be seen in Section IV-C). In addition, both DeepLog and LogAnomaly need a large number of normal logs to train their models. Obtaining such a large scale normal dataset still consumes operators tremendous amounts of time. Consequently, neither DeepLog nor LogAnomaly is suitable in our scenario.

VI. Conclusion

In this paper, we present a transfer learning-based log anomaly detection framework, LogTransfer. It applies Glove to accurately extract the global word co-occurrence and local context information to robustly measure the similarities of cross-system logs. Moreover, we propose to share fully connected networks between source and target systems to facilitate a more robust anomaly detection model to noises mingling in logs. Experiments conducted on logs generated by switch software systems demonstrate that LogTransfer outperforms the existing supervised and unsupervised methods. In addition, these experiments also show that both Glove and the novel transfer learning method improve the performance of LogTransfer. We also experiment with LogTransfer on public Hadoop datasets, showing its superior generality in anomaly detection. In the future, we will test LogTransfer on more types of log dataset, and improve it for the cross-domain transfer learning scenarios.

VII. Acknowledgment

This work was supported by the the National Key R&D Program of China under Grant No. 2018YFB0204304.

REFERENCES

[1] X. Zhang, Q. Lin, Y. Xu, S. Qin, H. Zhang, B. Qiao, Y. Dang, X. Yang, Q. Cheng, M. Chintalapati *et al.*, "Cross-dataset time series anomaly detection for cloud systems," in *2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19)*, 2019, pp. 1063–1076.

[2] 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*. ACM, 2017, pp. 1285–1298.

[3] B. Debnath, M. Solaimani, M. A. G. Gulzar, N. Arora, C. Lumezanu, J. Xu, B. Zong, H. Zhang, G. Jiang, and L. Khan, "Loglens: A real-time log analysis system," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2018, pp. 1052–1062.

[4] 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*. ACM, 2018, pp. 60–70.

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

[6] H. Xu, W. Chen, N. Zhao, Z. Li, J. Bu, Z. Li, Y. Liu, Y. Zhao, D. Pei, Y. Feng *et al.*, "Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications," in *Proceedings of the 2018 World Wide Web Conference*. International World Wide Web Conferences Steering Committee, 2018, pp. 187–196.

[7] 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*. ACM, 2010, pp. 111–124.

[8] 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)*. IEEE, 2007, pp. 583–588.

[9] 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.* IEEE, 2004, pp. 36–43.

[10] S. Lu, X. Wei, Y. Li, and L. Wang, "Detecting anomaly in big data system logs using convolutional neural network," in *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*. IEEE, 2018, pp. 151–158.

[11] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan, "Online system problem detection by mining patterns of console logs," in *2009 Ninth IEEE International Conference on Data Mining*. IEEE, 2009, pp. 588–597.

[12] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 Eighth IEEE International Conference on Data Mining*. IEEE, 2008, pp. 413–422.

[13] J.-G. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li, "Mining invariants from console logs for system problem detection." in *USENIX Annual Technical Conference*, 2010, pp. 23–25.

[14] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang, and X. Chen, "Log clustering based problem identification for online service systems," in *Proceedings of the 38th International Conference on Software Engineering Companion*. ACM, 2016, pp. 102–111.

[15] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun, and R. Zhou, "Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, 7 2019, pp. 4739–4745. [Online]. Available: https://doi.org/10.24963/ijcai.2019/658

[16] S. Zhang, Y. Liu, W. Meng, Z. Luo, J. Bu, S. Yang, P. Liang, D. Pei, J. Xu, Y. Zhang *et al.*, "Prefix: Switch failure prediction in datacenter networks," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 2, no. 1, p. 2, 2018.

[17] S. Zhang, W. Meng, J. Bu, S. Yang, Y. Liu, D. Pei, J. Xu, Y. Chen, H. Dong, X. Qu *et al.*, "Syslog processing for switch failure diagnosis and prediction in datacenter networks," in *2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS)*. IEEE, 2017, pp. 1–10.

[18] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[19] 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*. IEEE Press, 2019, pp. 121–130.

[20] M. Du and F. Li, "Spell: Online streaming parsing of large unstructured system logs," *IEEE Transactions on Knowledge and Data Engineering*, 2018.

[21] 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*. ACM, 2009, pp. 1255–1264.

[22] 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)*. IEEE, 2017, pp. 33–40.

[23] L. Tang, T. Li, and C.-S. Perng, "Logsig: Generating system events from raw textual logs," in *Proceedings of the 20th ACM international conference on Information and knowledge management*. ACM, 2011, pp. 785–794.

[24] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[25] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in *Advances in neural information processing systems*, 2014, pp. 3320–3328.

[26] 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)*. IEEE, 2016, pp. 207–218.

[27] L. Wang, Y. Du, and L. Qi, "Efficient deviation detection between a process model and event logs," *IEEE/CAA Journal of Automatica Sinica*, vol. 6, no. 6, pp. 1352–1364, 2019.

[28] P. Zhang, S. Shu, and M. Zhou, "An online fault detection model and strategies based on svm-grid in clouds," *IEEE/CAA Journal of Automatica Sinica*, vol. 5, no. 2, pp. 445–456, 2018.

[29] R. Chalapathy, A. K. Menon, and S. Chawla, "Anomaly detection using one-class neural networks," *arXiv preprint arXiv:1802.06360*, 2018.

[30] A. Brown, A. Tuor, B. Hutchinson, and N. Nichols, "Recurrent neural network attention mechanisms for interpretable system log anomaly detection," in *Proceedings of the First Workshop on Machine Learning for Computing Systems*, 2018, pp. 1–8.

[31] R. Abdulhammed, M. Faezipour, A. Abuzneid, and A. AbuMallouh, "Deep and machine learning approaches for anomaly-based intrusion detection of imbalanced network traffic," *IEEE sensors letters*, vol. 3, no. 1, pp. 1–4, 2018.

[32] S. Zhang, Y. Liu, W. Meng, J. Bu, S. Yang, Y. Sun, D. Pei, J. Xu, Y. Zhang, L. Song *et al.*, "Efficient and robust syslog parsing for network devices in datacenter networks," *IEEE Access*, vol. 8, pp. 30 245–30 261, 2020.

[33] Z. Shenglin, L. Dongwen, S. Yongqian, M. Weibin, Z. Yuzhe, Z. Yuzhi, L. Ying, and P. Dan, "Unified anomaly detection for syntactically diverse logs in cloud datacenter," *Journal of Computer Research and Development*, vol. 57, no. 4, p. 778, 2020.

[34] W. Meng, Y. Liu, Y. Huang, S. Zhang, F. Zaiter, B. Chen, and D. Pei, "A semantic-aware representation framework for online log analysis," *IEEE International Conference on Computer Communications (ICCCN)*, 2020.

[35] R. Vaarandi and M. Pihelgas, "Logcluster-a data clustering and pattern mining algorithm for event logs," in *2015 11th International conference on network and service management (CNSM)*. IEEE, 2015, pp. 1–7.

[36] R. Vaarandi, B. Blumbergs, and M. Kont, "An unsupervised framework for detecting anomalous messages from syslog log files," in *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2018, pp. 1–6.