A Semantic-aware Representation Framework for Online Log Analysis

Weibin Meng[†¶], Ying Liu[‡¶], Yuheng Huang[‖], Shenglin Zhang[*✉]
Federico Zaiter[†¶],Bingjin Chen[§], Dan Pei[†¶]
[†]Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China
[‡]Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing 100084, China
[‖]School of computer science, Beijing University of Posts and Telecommunications, Beijing 100084, China
[*]College of Software, Nankai University, Tianjin 300071, China
[§]School of Data and Computer Science, Sun Yat-sen University, Guangzhou 510275, China
[¶]Beijing National Research Center for Information Science and Technology (BNRist), Beijing 100084, China
mwb16@mails.tsinghua.edu.cn, liuying@cernet.edu.cn, huangyuheng@bupt.edu.cn, zhangsl@nankai.edu.cn
zaitertf10@mails.tsinghua.edu.cn, chenbjin@gmail.com, peidan@tsinghua.edu.cn

***Abstract*—Logs are one of the most valuable data sources for large-scale service management. Log representation, which converts unstructured texts to structured vectors or matrices, serves as the the first step towards automated log analysis. However, the current log representation methods neither represent domain-specific semantic information of logs, nor handle the out-of-vocabulary (OOV) words of new types of logs at runtime. We propose Log2Vec, a semantic-aware representation framework for log analysis. Log2Vec combines a log-specific word embedding method to accurately extract the semantic information of logs, with an OOV word processor to embed OOV words into vectors at runtime. We present an analysis on the impact of OOV words and evaluate the performance of the OOV word processor. The evaluation experiments on four public production log datasets demonstrate that Log2Vec not only fixes the issue presented by OOV words, but also significantly improves the performance of two popular log-based service management tasks, including log classification and anomaly detection. We have packaged Log2Vec into an open-source toolkit and hope that it can be used for future research.**

***Index Terms*—Log analysis, OOV words, AIOps**

Historical logs:
L_1. Interface ae3, changed state to down
L_2. Interface ae3, changed state to up
L_3. Interface ae1, changed status to down
L_4. Interface ae1, changed status to up
Real-time logs:
L_5. Vlan-interface vlan22, changed state to down
L_6. Vlan-interface vlan22, changed state to up

Out-of-vocabulary	Vlan-interface
Relation triples	(Interface, changed, state)
Antonym pairs	(down, up)
Synonym pairs	(state, status)

Fig. 1: Examples of logs and domain-specific information

I. Introduction

Large-scale services usually generate logs (see the top half of Fig. 1), which describe a vast range of events observed by them and record service runtime information. Therefore, they are crucial for service management [1]. A log message is a line of text printed by logging statements (*e.g.*, printf()) defined by developers. However, a large-scale service is often implemented/maintained by hundreds of developers/operators. Usually, a developer/operator has incomplete information on the overall service, and none of them is familiar with all logs generated by services. Moreover, the volume of logs is growing rapidly, for instance, at a rate of about 50 gigabytes (around 120∼200 million lines) per hour [2]. Therefore, the traditional way of log analysis that largely relies on manual inspection has become a labor-intensive and error-prone task.

Automatic log analysis approaches, which are employed for services management, have been widely studied. These methods are designed for monitoring status [3], understanding events [4], detecting anomalies [1], and predicting failures [5]. Most of the above methods require structured input (*e.g.,* a vector or a matrix) [6]. However, service logs are usually unstructured texts, which require to be properly represented before they can be effectively employed [2]. Therefore, log representation usually serves as the first step towards automated log analysis, and thus it is vitally important to the above methods. Take Figure 1 as a example, in L_1, "ae3" is a *variable* word, whereas the rest, *i.e.*, "Interface ..., changed state to down", is the *constant* field (template). A traditional way to applying logs to management is learning templates from logs, maps logs to templates, the indices or embedding of which are put into those log analysis approaches (*e.g.,* Deeplog [1], PreFix [5], LogAnomaly [7]).

In the literature, many log representation approaches have been proposed [2], [8]. Template2Vec, a state-of-the-art log representation work, presents the first step towards extracting

✉ Shenglin Zhang is the corresponding author.

978-1-7281-6607-0/20/$31.00 ©2020 IEEE

semantic information from logs by embedding the words of a template into a vector, which has been demonstrated to achieve a better performance than using template indices [7]. However, extract semantic information from logs for log representation faces two challenges that have not yet been addressed: (1) accurate extraction of domain-specific semantic information and (2) embedding out-of-vocabulary (OOV) words into vectors.

For the first challenge, the current semantic information extraction methods (*e.g.*, template2Vec [7]) embed words and templates into vectors based on the distributional hypothesis, which assumes that the words with a similar context tend to have a similar meaning [9]. However, they usually fail to capture the precise meanings of many words in logs, because some pairs of opposites may have similar contexts. For example, "down" and "up" in Figure 1 are antonyms but they have similar contexts.

For the second challenge, operators continuously conduct software/firmware upgrades on services to introduce new features, fix bugs, or improve performance [10]. These upgrades usually generate new types of logs with OOV words (*e.g.,* "Vlan-interface" in Figure 1). The current word embedding methods cannot embed these OOV words because they are usually offline trained based on the vocabulary of historical logs, which cannot be updated at runtime.

To address the above two challenges, we propose Log2Vec, a semantic-aware representation framework for online log analysis. The original goal of service logs, "*logs are for users to read*", inspires our work. That is, logs are designed by developers and "printf"-ed by services. Moreover, the intuitions and methods in natural language processing can be applied or improved for log representation. Specifically, Log2Vec integrates a **l**og-**s**pecific **w**ord **e**mbedding method, LSWE, to extract log specific semantic information, with an OOV word processor to embed OOV words into vectors at runtime.

The contributions of this paper are as follows.

1) We propose Log2Vec, a framework to convert unstructured logs to distributed representation with log-specific information.
2) Since Log2Vec has a mechanism for generating OOV word embeddings, operators do not need to do redundant retraining on original word embedding corpus when new types of logs appear.
3) Experiments on two popular log-based service management tasks, including log classification and anomaly detection, have both demonstrated that Log2Vec can extract the critical features of logs and significantly improve the performance of log-based service management tasks.
4) We have open-sourced[1] Log2Vec, and hope that it can be used for future research.

The rest of the paper is organized as follows: We discuss related log representation works in Section II and propose our approach in Section III. The evaluation is shown in Section IV.

[1]Log2Vec is available on Github: https://github.com/WeibinMeng/Log2Vec

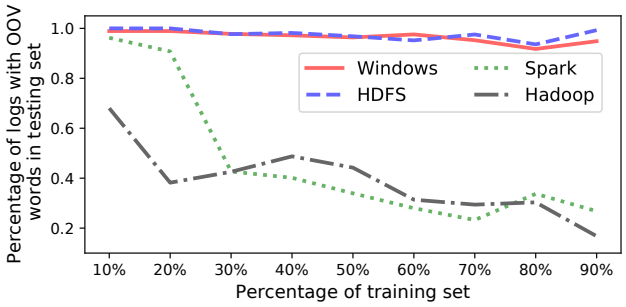

Fig. 2: Analysis of logs with OOV words on four datasets

In Section V, we introduce a case study of Log2Vec. Finally, we conclude our work in Section VII.

II. Related Work

Service logs play an important role in service management. Most of the log-based service management tasks require structured input (*e.g.,* event/template index or matirx). Therefore, log representation usually serves as the the first step towards automated log analysis.

The most popular log representation approach is automatic template extraction. It extracts constant fields (templates) from logs and puts template index into machine learning approaches. There are many categories of template extraction approaches [2], such as LogSig (a cluster-based method) [11], FT-tree (based on frequent items mining) [12], IPLoM (Iterative partitioning) [13] and Spell (longest common subsequence) [8]. However, valuable information could be lost when only log template indexes are used, because they cannot reveal the semantic relations of logs [7].

To be applicable for machine learning algorithms, bag-of-words models are used to convert logs to vectors when detecting anomalous logs [14]. Template2Vec [7] combines templates and Word2vec [15] to represent templates by vectors. Nevertheless, corpus-based methods (*e.g.,* word2Vec) usually fail to capture the domain-specific meanings for many words. For instance, some semantically related but dissimilar words may have similar contexts (*e.g.,* synonyms and antonyms).

Besides, services can generate new types of logs at runtime [7], however, existing log representation approaches cannot handle out-of-vocabulary (OOV) words in new logs, which also lose semantic information. For example, Figure 2 shows percentages of logs containing OOV words as the percentage of training data increases from 10% to 90% (detailed information is shown in Section IV-B). We observe that OOV words has a big percentage when trained on a smaller sample of the logs.

III. Design of Log2Vec

A. Observation

We design Log2Vec to represent words in logs based on the following observations:

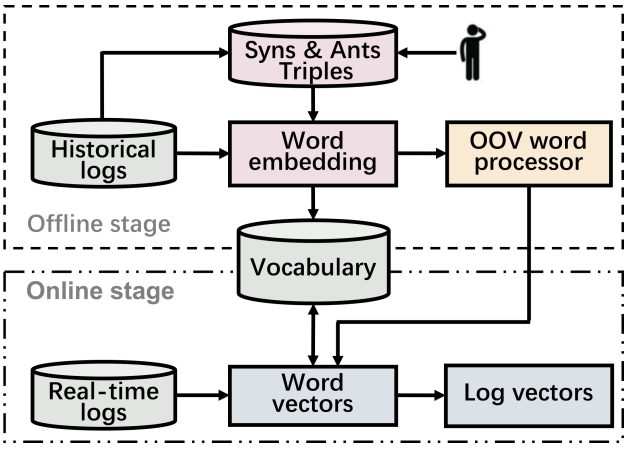

Fig. 3: Log2Vec framework

1) A log is predefined by developers using the "printf" function. It typically characterizes the event that occurs on the system. The text of logs represents the semantic information of the event.
2) The vocabulary across logs is growing continuously because the software/firmware can be upgraded on services due to changes introduced to add new features, to fix bugs or to improve the system's performance.
3) Logs contain lots of domain-specific words. Extracting the semantic information of these domain-specific words can be difficult yet critical for log analysis.

B. Overview of Log2Vec

Motivated by these observations, we propose Log2Vec (as shown in Figure 3), a novel representation framework for service log analysis. Specifically, we propose a log-specific word embedding method, LSWE, to extract log specific semantic information (Section III-C), and adopt an OOV word processor to embed OOV words at runtime (Section III-D). We further introduce the online and offline stages of Log2Vec in this Section.

The framework of Log2Vec consists of two components: the offline stage and the online stage. In the offline stage, Log2Vec constructs the domain-specific information set (*e.g.,* antonyms and triples in this paper) by combining the information extracted from logs or offered by operators. Next, Log2Vec represents words distributedly with domain-specific semantic. Since services will generate OOV words in new types of logs at runtime, Log2Vec trains an OOV word processor for OOV words in the online stage. In the online stage, we first determine whether each word in logs is in vocabulary. If yes, we then convert existing words to word vectors based on offline learning. Otherwise, we will assign a new embedding vector to the OOV word by the OOV word processor. In the end, Log2Vec converts all words in logs into word embedding vectors and calculates the log vector, which is the weighted average of its words vectors.

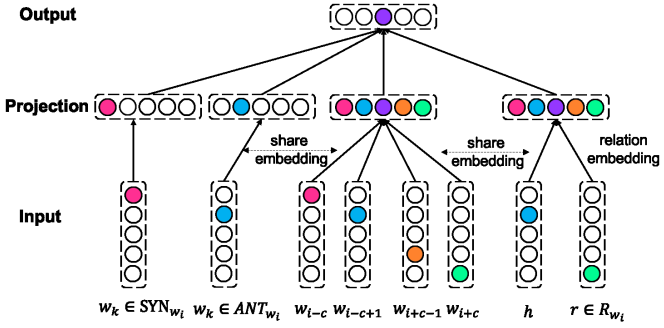

Fig. 4: Architecture of LSWE

C. Log-specific Word Embedding

Logs are designed to facilitate user readability; hence constant parts of logs are defined in natural languages by service designers. Previous methods use natural language process (NLP) methods (*e.g.,* word2vec [15]) to represent the words in text. However, these methods are not designed specifically to deal with the words in logs, which results in several shortcomings. (1) For instance, word2vec [15] uses context words to predict target words, where only local contextual information is considered. As a result, it fails to represent synonyms and antonyms. (2) Word2vec does not take relation information in to consideration, which is also important information of logs.

To resolve these problems, we propose LSWE (Log-specific Word Embedding), a novel word vector representation for logs' words that integrates lexical contrast into distributional vectors and strengthens domain-specific semantic and relation information for determining degrees of word similarity. The architecture of LSWE is shown in Figure 4. LSWE adopts two word embedding methods, Lexical Information Word Embedding (LWE) [16] and Semantic Word Embedding (SWE) [17].

Given the set of synonyms and antonyms corresponding to the target word, we use LWE to predict the target word so that the distance of its vector representation is as close as possible to its synonyms and as far away as possible from its antonyms. Formally, the objective function as follows:

$$\mathcal{L}_{lwe} = \beta\Big(\sum_{u \in SYN_{w_i}} \log p(w_i|u) - \sum_{u \in ANT_{w_i}} \log p(w_i|u)\Big) + \sum_{k=1}^{|C|} (\log p(w_i|w_{i-c}^{i+c})) \tag{1}$$

Where w_i is the target word, u is the antonym. SYN_{w_i} and ANT_{w_i} represent the synonyms and antonyms set of w_i, respectively. $p(w_i|u)$ is the probability of w_i when the antonym is u. The second part is the objective function of CBOW [18] (a model of Word2vec), C is the training corpus (logs). Note that CBOW and LWE share the same word vectors.

Generally, the information in the knowledge graph is organized in the form of relation triples (h, r, t). As used in SWE [17], we construct a sample (h, r, w_i), where r represents

a variety of different w_i association relationship. After getting triples, it is necessary to establish a representation of the relationship of words. TransE model [19] is the most effective representation method for triples. For triples (h, r, t), if the triples are factual information, then $(h + r \approx t)$. That is, the corresponding vector of $h + r$ should be closer to t. SWE [17], which combines relation information and CBOW, with objective function as

$$\mathcal{L}_{swe} = \sum_{k=1}^{|C|}(\log p(w_i|w_{i-c}^{i+c})) + \gamma \sum_{r \in R_{w_i}} \log p(w_i|h + r) \quad (2)$$

Then we obtain the objective function of LSWE as follows:

$$\begin{aligned} L_{lswe} &= \mathcal{L}_{lwe} + \mathcal{L}_{swe} \\ &= \sum_{k=1}^{|C|}(\log p(w_i|w_{i-c}^{i+c})) + \gamma \sum_{r \in R_{w_i}} \log p(w_i|h + r) \\ &+\beta(\sum_{u \in SYN_{w_i}} \log p(w_i|u) - \sum_{u \in ANT_{w_i}} \log p(w_i|u)) \end{aligned} \quad (3)$$

LSWE learns the context word based on the co-occurrence information. Furthermore, it also learns the corresponding semantic and relationship information. LSWE combines antonyms set and relation triples to improve the quality of the word vector. Universal antonyms can be found in WordNet [20], which is a lexical database for English. Relation triples can be extracted from processed logs using dependence trees [21], which is a prevalent semantic parsing method. Some domain-specific antonyms and relations, however, have to be added by operators based on domain knowledge.

The significant advantage of LSWE is that it enables generalization to words, which is achieved by integrating the embedding of lexical and relation features into a low-dimensional Euclidean space. These low-dimensional embeddings can capture distributional similarity, so that information can be shared among words that tend to appear in similar contexts. However, it is not possible to enumerate the entire vocabulary of all logs and will miss words that appear in later services (OOV words). In the next section, we will introduce a new technique to handle OOV words.

D. Out-of-vocabulary Word Processor

To handle OOV words at runtime, we adopt MIMICK [22], an approach to generate OOV word embeddings by learning a function from spellings to distributional embeddings.

MIMICK [22] regards the problem of out-of-vocabulary (OOV) embeddings as a generation problem: Regardless of how the original embeddings are created, MIMICK assumes that there is a generative wordform-based protocol to create these embeddings. By training a model over the existing vocabulary, MIMICK can later use that model to predict the embedding of an unseen word. Here we introduce the principle of MIMICK briefly: given a language $\mathcal{L}$, a vocabulary $\mathcal{V} \subseteq \mathcal{L}$ of size V, and a pre-trained embeddings table $\mathcal{L} \in \mathbb{R}^{V \times d}$ where each word $\{w_k\}_{k=1}^{V}$ is assigned a vector e_k of dimension d. MIMICK is trained to find the function $f : \mathcal{L} \to \mathbb{R}^d$ such that the projected function $f|v$ approximates the assignments $f(w_k) \approx e_k$. Given such a model, a new word $w_{k*} \in \mathcal{L} \backslash \mathcal{V}$ can now be assigned an embedding $e_{k*} = f(w_{k*})$

As shown in Figure 3, we train an OOV processor in the offline stage with MIMICK [22] and assign a new embedding vector for unseen words when new words appear in the online stage.

Datasets	Description	# of logs
Spark	Spark job log	33,236,604
HDFS	Hadoop distributed file system	11,175,629
Windows	Windows event log	114,608,388
Hadoop	Hadoop MapReduce job	394,308

TABLE I: Detail of the service log datasets

IV. Experiments

In this section, we report all experiments conducted to evaluate the effectiveness of the proposed Log2Vec. Firstly, we describe the experimental setup. Next, we present a measurement study to highlight the challenge of OOV words and show the performance of Log2Vec's OOV processor. Finally, in order to prove the effectiveness of Log2Vec, we compare the performance of three popular log-based service management tasks (with/without Log2Vec), including log classification and anomaly detection.

A. Experimental Setting

1) Datasets: We conduct experiments over four public log datasets from several services, which are Spark logs [2], HDFS logs [2], Windows logs [23], and Hadoop logs [24]. The detailed information of these datasets are listed in Table I.

2) Experimental Setup: We conduct all experiments on a Linux server with Intel Xeon 2.40 GHz CPU. We implement Log2Vec with Python 3.6 and will open source it. As for MIMICK, we use a popular open-source toolkit in [22].

B. OOV Words

1) Measurements of OOV: In this section, we present a large-scale measurement study to highlight the challenge in processing OOV words. We generate training sets with the percentage of original datasets ranging from 10% to 90% and regard the remaining logs as the testing set.

Firstly, we evaluate the number of OOV words in service logs. Figure 5 shows percentages of words on four log datasets, as the percentage of training data increases from 10% to 90%. The results show that, with the growth of training data, the proportion of OOV words in the test set is gradually decreasing, especially in Spark logs. When we use 90% Spark logs to train the model, we find only 2.48% of OOV words in the testing set.

Next, we evaluate the number of logs that contain OOV words. Figure 2 shows percentages of logs containing OOV

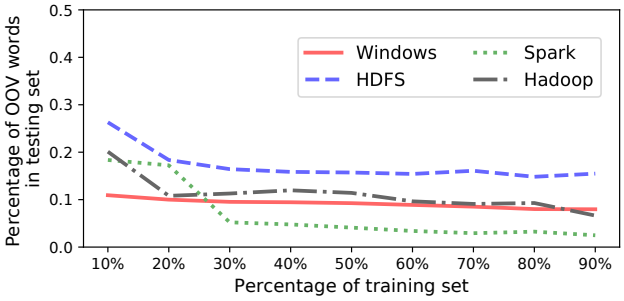

Fig. 5: Measurements of OOV words

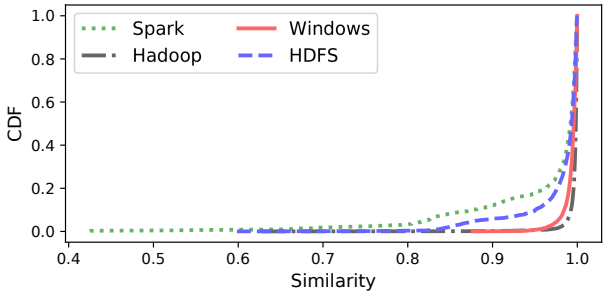

Fig. 6: Distribution of Logs' Similarity in Section IV-B2

words on four log datasets, as the percentage of training data increases from 10% to 90%, respectively. We observe that all test sets have logs with OOV words even when they are trained with 90% of the data, specially Windows and HDFS logs. No matter what proportion of data is used for training, there are always more than 90% logs generated by service online that contain OOV words. Besides, when we train a model based on a 10% training set, more than 70% logs generated by services will contain OOV words, which means that current log analysis models cannot train a model based on a small amount of data (as shown in Fig. 2). The results also show that it's essential to handle OOV words when analyzing logs.

2) Evaluation of the OOV Word Processor: As described in Section III-D, Log2Vec contains a OOV processor to handle OOV words at runtime. In this section, we evaluate the performance of the OOV processor. We randomly select 10,000 logs from each of the four datasets as the ground truth. Then we randomly select a word in each log and changed one of the letters to make the word an OOV word. Next, we test the similarity (Cosine Similarity) between the changed log and the original log.

Figure 6 is the CDF of the above experiment, we observe that even when each piece of log contains OOV words, Log2Vec achieves an excellent performance. For example, 99.8% of Windows logs have more than 0.9 similarity with changed logs. Tabel II shows the average similarity when Log2Vec processes the modified logs. All of them achieve high similarity.

TABLE II: Average similarity when Log2Vec processes logs with OOV words

Dataset	Spark	HDFS	Windows	Hadoop
Similarity	0.964	0.984	0.993	0.996

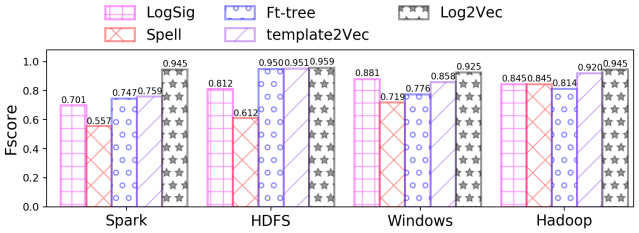

Fig. 7: Comparison of log classification when use 50% training logs

C. Task 1: Log Classification

1) Task description: Generally, operators classify individual logs and focus on the categories of logs they are interested in to monitor the status of services. However, the increasing scale and complexity of modern services make the volume of logs explode, and tens of millions of logs are generated in a large-scale service. Classifying logs manually suffers from inflexibility and labor intensiveness. For this reason, automatic log classification is also an important task in Log Analysis. To evaluate the performance of online log classification, we divide each dataset to 50% training set and 50% testing set in this task.

2) Baselines: The most popular automatic log classification methods are rule-based (*e.g.*, extracting templates automatically). They regard each template as a category and match online logs to extracted templates. Besides, template2Vec [7] also achieves excellent performance on log analysis, they can represent each log as a vector and cluster logs to some categories. In this task, we choose LogSig [11], FT-tree [12], Spell [8], template2Vec [7] as baselines. [2] manually labelled each log's category (in Table I), which serves as the ground truth for classification.

3) Experimental results: Figure 7 shows the experimental results of log classification. We observe that classification with template extraction methods (LogSig, FT-tree, Spell) performs badly because there are many new type of logs in the testing set which cannot be matched to existing templates. Template2Vec performs better than template methods because template2Vec represents the semantic of templates. However, the input of template2Vec are templates, they also cannot handle OOV words. The performance of Log2Vec is stable on four datasets and the average Fscore of Log2Vec is 0.944, which demonstrates that Log2Vec can represent logs more accurately than baselines (average Fscore of 0.745).

D. Task 2: Anomaly Detection

1) Task description: Anomaly detection is a critical step towards building a secure and trustworthy service. The primary

TABLE III: Results of anomaly detection

Methods	Precision	Recall	Fscore
Deeplog	0.898	0.964	0.930
Deeplog w/ Log2Vec	0.941	0.937	0.939

purpose of a service log is to record service states and significant events at various critical points to help debug service failures and perform root cause analysis. Therefore, many methods utilize the various service logs to do anomaly detection. Service anomalies can be inferred based on their log sequences which contain multiple logs violating regular rules. We conduct this experiment on BGL logs [25], where each log was manually labeled as either normal or anomalous.

2) Baselines: Deeplog [1] is the start-of-the-art among log-based sequential anomaly detection methods. However, the input of Deeplog are log template indexes, which already mean a loss of semantic information. To demonstrate the performance of Log2Vec, we change template indexes, the input of Deeplog, to a distributed representation. We compare Deeplog to itself when trained with Log2Vec. The parameters of these methods are all set best for accuracy.

3) Experimental results: As shown in Table III, Deeplog has a much lower precision on the BGL dataset (0.898) compared to Deeplog with (w/) Log2Vec (0.941). Generally, large services produce tens of millions of logs everyday. If a log anomaly detection method generates too many false alarms everyday, it will add a large amount of unnecessary work to operators. Therefore, Log2Vec could improve the performance of current log-based anomaly detection methods. Log2Vec achieves better performance, because Log2Vec is able to represent semantic information and handle OOV problems when analyzing logs.

V. Case Study

To further evaluate the performance of Log2Vec, we show a case study on online log clustering, where logs are generated by switches deployed in a top cloud service provider during a month. Log clustering aims to make similar logs come closer together with logs representing the same events in a given space, where logs in the same category have similar semantics. Log2Vec, template2Vec [7] and FT-tree [12] are deployed to do logs clustering. Log2Vec generates 775 categories, Template2Vec generates 862 categories, FT-tree generates 3212 categories. Confirmed by operators, the clustering results of Log2Vec are the best for management. Most categories generated by FT-tree are too fine-grained. Logs belonging to the same event are divided into several categories because FT-tree [12] (and other template methods) cannot bring two logs containing different words together, even they have the same semantics. Although Template2Vec can represent the semantics of logs, it cannot handle OOV words, which will generate more categories with similar semantics.

For example, Figure 8 shows four logs from above case study. Log2Vec classifies them to two categories ($\{L_1, L_2\}$, $\{L_3, L_4\}$), Template2Vec generates three categories ($\{L_1, L_2\}$, $\{L_3\}$, $\{L_4\}$), while FT-tree generates four categories ($\{L_1\}$, $\{L_2\}$, $\{L_3\}$, $\{L_4\}$). In L_1 and L_2, "status" and "state" are synonyms. In L_4, "mainIfname" is a OOV word, but it is similar to "mainName" in L_3. Log2Vec recognized them and classified them into the same categories.

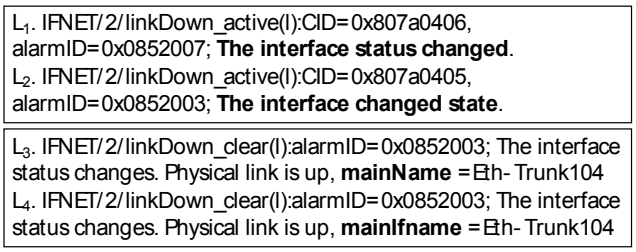

Fig. 8: Case study on log clustering

VI. Discussion and Future Work

The aim of this paper is to highlight the promise of NLP-powered methods for logs, and discuss the challenges (e.g., log-specific information and OOV words) in the online log analysis scenario that must be overcome to realize this vision. Log2Vec is a framework, operators can replace any model or add new models for log analysis.

Log2Vec can serve further downstream purposes, which we consider for future work. We will use Log2Vec to improve log-based service management tasks (*e.g.*, failure prediction, root cause analysis) in the future.

VII. Conclusion

Log representation is the first step of automated log analysis. We propose a semantic-aware representation framework, Log2Vec, to improve the performance of online log analysis. We present a measurement on OOV words and evaluate the performance of OOV word processor in Log2Vec. In addition, we apply Log2Vec to two popular log-based service management tasks, including log classification and anomaly detection, both demonstrating that Log2Vec can extract the crucial features of logs and significantly improve the performance of them. Traditional log representation methods limit many log analysis applications, which require to have a representation for any log. Log2Vec is able to assign a "soft" representation to every log at runtime, in order to avoid false alarms induced by new types of logs.

VIII. Acknowledgment

The work was supported by National Key R&D Program of China (Grant No. 2019YFB1802504, 2018YFB1800405), the National Natural Science Foundation of China (Grant Nos. 61772307, 61902200 and 61402257), the China Postdoctoral Science Foundation (2019M651015) and the Beijing National Research Center for Information Science and Technology (BNRist). We thank the anonymous reviewers for their valuable feedback. We thank Yuzhe Zhang, Yichen Zhu, Yixuan Zhang, Ya Su, Tianke Zhang for their helpful suggestions.

REFERENCES

[1] Min Du, Feifei Li, Guineng Zheng, and Vivek 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*, pages 1285–1298, 2017.

[2] Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R Lyu. Tools and benchmarks for automated log parsing. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 121–130. IEEE, 2019.

[3] Subhendu Khatuya, Niloy Ganguly, Jayanta Basak, Madhumita Bharde, and Bivas Mitra. Adele: Anomaly detection from event log empiricism. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 2114–2122. IEEE, 2018.

[4] Shilin He, Qingwei Lin, et al. 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*, pages 60–70. ACM, 2018.

[5] Shenglin Zhang, Ying Liu, Weibin Meng, Zhiling Luo, Jiahao Bu, Sen Yang, Peixian Liang, Dan Pei, Jun Xu, Yuzhi Zhang, et al. Prefix: Switch failure prediction in datacenter networks. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 2(1):1–29, 2018.

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

[7] Weibin Meng, Ying Liu, Yichen Zhu, Shenglin Zhang, Dan Pei, Yuqing Liu, Yihao Chen, Ruizhi Zhang, Shimin Tao, Pei Sun, et al. 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*, volume 7, pages 4739–4745, 2019.

[8] Min Du and Feifei Li. Spell: Streaming parsing of system event logs. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 859–864. IEEE, 2016.

[9] Michael Roth. Combining word patterns and discourse markers for paradigmatic relation classification. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics(ACL)*, pages 524–530, June 2014.

[10] Shenglin Zhang, Ying Liu, Dan Pei, Yu Chen, Xianping Qu, Shimin Tao, Zhi Zang, Xiaowei Jing, and Mei Feng. Funnel: Assessing software changes in web-based services. *IEEE Transactions on Services Computing*, 11(1):34–48, 2016.

[11] Liang Tang, Tao Li, and Chang-Shing Perng. Logsig: Generating system events from raw textual logs. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 785–794. ACM, 2011.

[12] Shenglin Zhang, Weibin Meng, Jiahao Bu, Sen Yang, Ying Liu, Dan Pei, Jun Xu, Yu Chen, Hui Dong, Xianping 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)*, pages 1–10. IEEE, 2017.

[13] Adetokunbo AO Makanju, A Nur Zincir-Heywood, and Evangelos E Milios. Clustering event logs using iterative partitioning. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1255–1264, 2009.

[14] Weibin Meng, Ying Liu, Shenglin Zhang, Dan Pei, Hui Dong, Lei Song, and Xulong Luo. Device-agnostic log anomaly classification with partial labels. In *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, pages 1–6. IEEE, 2018.

[15] Kim Anh Nguyen, Sabine Schulte im Walde, and Ngoc Thang Vu. Integrating distributional lexical contrast into word embeddings for antonym-synonym distinction. *arXiv preprint arXiv:1605.07766*, 2016.

[16] Luchen Tan, Haotian Zhang, Charles Clarke, and Mark Smucker. Lexical comparison between wikipedia and twitter corpora by using word embeddings. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 657–661, 2015.

[17] Quan Liu, Hui Jiang, Si Wei, Zhen-Hua Ling, and Yu Hu. Learning semantic word embeddings based on ordinal knowledge constraints. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1501–1511, 2015.

[18] Tomas Mikolov, Quoc V Le, and Ilya Sutskever. Exploiting similarities among languages for machine translation. *arXiv:1309.4168*, 2013.

[19] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph and text jointly embedding. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1591–1601, 2014.

[20] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[21] Katrin Fundel, Robert Küffner, and Ralf Zimmer. Relex—relation extraction using dependency parse trees. *Bioinformatics*, 23(3):365–371, 2007.

[22] Yuval Pinter, Robert Guthrie, and Jacob Eisenstein. Mimicking word embeddings using subword rnns. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 102–112, 2017.

[23] Shilin He, Jieming Zhu, et al. Experience report: System log analysis for anomaly detection. In *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, pages 207–218. IEEE, 2016.

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

[25] Adam J. Oliner and Jon Stearley. What supercomputers say: A study of five system logs. *IEEE International Conference on Dependable Systems and Networks (DSN'07)*, pages 575–584, 2007.