2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)

It Can Understand the Logs, Literally

Aidi Pi, Wei Chen, Will Zeller, Xiaobo Zhou
Department of Computer Science
University of Colorado, Colorado Springs, CO, USA
{epi, cwei, wzeller, xzhou}@uccs.edu

***Abstract*—Workflow reconstruction through logs is crucial for troubleshooting targeted distributed systems. It is also challenging to extract enough information from logs and keep a concise view, which makes manual log analysis hard to practice. However, currently popular tools rely on identifier-based log parsing, leaving a large amount of workflow information unexploited.**

In this paper, we propose a log extraction approach *NLog*, which utilizes a natural language processing based approach to obtain the key information from log messages and identify the same object in logs generated by different statements without any domain knowledge. We propose to use *keyed message*, a new log storage structure to store the parsed logs. We implement *NLog* and apply it to distributed data analytics frameworks Spark and MapReduce. Evaluation results show that *NLog* can accurately identify the objects in log messages even without explicit identifiers. By using keyed messages, users can have a concise as well as flexible view of the workflows.

***Index Terms*—Distributed Systems, Profiling, Information Extraction, Troubleshooting**

I. Introduction

Logs generated by distributed systems contain rich information including operations done by objects, the amount of processed data, system events, warnings, errors and etc. Effectively extracting and presenting information from raw logs helps users to boost the process of understanding workflows, detecting anomalies and troubleshooting the targeted systems.

Due to the unstructured nature of raw logs, how to effectively utilize raw logs is an important problem. Spell [1] shows that logs can be used to infer the corresponding log statements in the source code so as to group log messages generated by the same statement. Stitch and Iprof [2], [3] focus on reconstructing system workflows from raw logs to accelerate manual log analysis. For automatic anomaly detection, a number of studies apply machine learning methods to logs [4], [5], [6], [7], [8], [9], [10]. However, extracting and reconstructing logs of distributed systems are challenging tasks because of the following two reasons:

- Profiling one log message: one log message usually consists of multiple fields, such as timestamps, identifiers, events. To extract information from a log message, the profiling tool needs to distinguish the fields and their meaning in one log message.
- Profiling multiple log messages: multiple log messages can record the same kind of objects performing various operations. For example, three log messages may record the same task. The first one indicates the start mark of the task. The second one indicates the task is running, and the last one indicates the end mark of the task. It is even more challenging if the object is not tagged with any identifier. An effective profiling tool needs to identify the same kind of objects that appear in different log messages.

TABLE I
Lines and percentages of natural language (NL) logs in three frameworks, respectively.

Platform	NL logs	total logs	% of NL logs
Yarn	84652	88628	99.5%
Spark	106686	106686	100%
MapReduce	85752	92648	92.6%
average	-	-	**97.4%**

There is a study that focuses on leveraging identifiers to reconstruct a hierarchical view of the workflow [2]. However, there is no work identifying the same kind of objects or events that are not tagged with any identifier, which leaves a large amount of recorded information unexploited. Furthermore, the prevalent identifier-recognition based approaches are unable to extract the exact events in the log messages. For example, following is a simplified log message generated by Spark: `"Task 39 force spilling in-memory map to disk and it will release 159.6 MB memory"`. Both Spell and Stitch [2], [1] can identify that this is a log message concerning `Task 39`, but they cannot recognize the `spilling` event.

We propose NLog, a log analysis tool that targets on distributed data analytics systems. It is inspired by the original goal of system logs: *logs are for users to read*. In other words, logs are usually written in natural languages by system designers. To show this fact, we inspect about 20 MB log files with over 100,000 log messages generated by Spark [11], Yarn [12] and Hadoop MapReduce [13], respectively. We convert the log messages to their corresponding message types by Spell [1] and manually inspect whether the resulting message types are written in natural language. The results in Table I show that on average 97.4% of the logs are written in natural languages. The rest logs are mainly about the status of applications and clusters.

In this paper, we propose and develop *NLog* to extract objects in logs and present them to users in a uniformed format called *keyed message*. Our main contributions lie in the following aspects:

First, we propose a natural language processing (NLP) based approach to identify the objects and events even without any identifiers in logs. Since most logs are written in natural languages, the part of speech of each word in a log message

978-1-5386-5555-9/19/$31.00 ©2019 IEEE
DOI 10.1109/IPDPSW.2019.00084

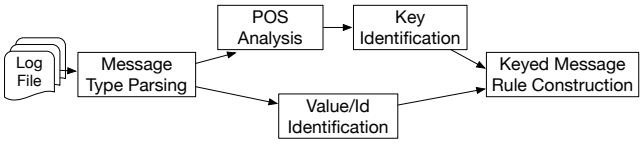


Fig. 1. Overview of the parsing process by *NLog*.

can be used to distinguish objects and tell the operations done by the objects. We further apply a sorting strategy to increase the accuracy of object identification. The NLP procedure takes sample log files as input. It is a one-time off-line procedure for each targeted system. Results are rules that can be utilized to transform raw log messages to keyed message at runtime.

Second, we propose to use *keyed message*, a uniformed log storage structure, to store processed logs. A keyed message is a key-value like tuple but it contains more fields. Database operations such as Count, Sum and GroupBy can be easily applied to keyed messages to help users understand the targeted systems.

We implement *NLog* and apply it to distributed data analytics frameworks Spark [14] and MapReduce [13] on Apache Yarn [12]. Evaluation results show that it can help users to effectively reconstruct the workflows of the targeted systems. By applying database operations, keyed messages provide the flexibility for users to get both a coarse- and fine-grained view of the workflows.

II. Log Message Parsing

In *NLog*, log parsing is divided into three steps: 1) identifying the corresponding message type of each log message, 2) extracting the key objects in log messages, and 3) finding the identifiers that uniquely distinguish key objects from each other and the numeric values in log messages. Step 1 is considered to be a solved problem. In Steps 2 and 3, we use NLP-based approaches. Finally, the parsing results can be directly used to construct keyed message [15]. Figure 1 depicts the overview of the parsing process.

Step 1 We apply the definition of message type from Spell [1]: each log message can be mapped to a message type, and each message type has a one-to-one relationship to the log printing statement in the source code. We utilize *Spell* [1] to find the corresponding message types of logs messages.

Step 2 We introduce an NLP-based approach to identify the objects recorded in each log message concerning the workflows. Our intuition is that if an object is concerning the workflow of an application, it may repeatedly appear in log files and it is usually recorded in multiple message types. For example, `task` may appear in multiple log messages that indicate the events about the task. Thus, we regard `task` as a key object. However, since a message type consists of multiple words, it is challenging to determine which of them actually contains the key object without knowing the literal meaning of the words. In the following, we describe our NLP-based approach.

Step 2.1 POS analysis In order to identify the key objects in message types, we apply part-of-speech analysis (POS analysis) on each log type. We use the tool *OpenNLP* [16] to tag each word with its part-of-speech in the context within a log type.

Step 2.2 Key identification Taking the training log files as the input, we infer the part-of-speech of each word in a log message from the corresponding tagged message type. In order to identify the key objects, we have to decide the importance of the words. Most of objects are recorded as noun in log messages, which indicates that noun words have a higher priority than words of other kinds of part-of-speech.

First, we extract all the noun words appearing in the training log files but filter out all the other words. In this step, we only consider the static words in log messages but ignore the variables. After extracting all the nouns, we further lemmatize each noun to its singular form.

Second, we calculate the frequency of each noun word. The result shows that words concerning the key objects usually have higher frequencies (10X ˜ 1000X) than other words have. We sort the result in descending order and take the top α percent of the words as the candidate keys. A key indicates the key information recorded in a log message.

Finally, we assign message types with keys. In this step, we iterate through the sorted candidate keys and assign a key to the message type if the key appears as a noun in it. Since a message type can indicate multiple objects or events, it is necessary to allow assigning multiple keys to one message type. However, a log type assigned with too many keys may confuse the users in manual analysis, since the users are hard to tell which are most important objects. In our design, we allow as many as four keys to be assigned to one message type. We discard a message type if it is not assigned with any key after the iteration.

Step 3 Identifiers can be recorded as either words (e.g. user name) or numbers (e.g. task id). The numeric identifiers are hard to be distinguished from values in log messages. Previous work [2] requires users' efforts to specify the identifiers in log messages. In our design, we also utilize a NLP-based approach to identify numeric identifiers and numeric values.

Identifier In Step 2, we mark all the noun words in log messages. In our design, if a numeric identifier follows a noun word, we treat the numeric number and the noun word as an identifier and its referring object, respectively. We manually identify the total number of numeric fields and the number of actual id fields from all message types, and compare the results to the identifiers found by *NLog*. Table II shows the results of accuracy of identifier recognition by *NLog*.

Value Numeric values that are not identifiers are all regarded as values. Furthermore, the values in log messages usually indicate the amount of processed data or the elapsed time of operations. Such values are usually followed by units such as `kb` or `ms`.

To put it all together, we use a log message from Hadoop MapReduce to illustrate the parsing process of *NLog* step by step, as shown in Figure 2.

TABLE II
THE ACCURACY OF *NLog* IDENTIFIER RECOGNITION. TP, FP, TN AND FN STAND FOR 'TRUE POSITIVE', 'FALSE POSITIVE', 'TRUE NEGATIVE' AND 'FALSE NEGATIVE', RESPECTIVELY

Total # of numeric fields	# of actual id fields	TP	FP	TN	FN
112	42	36	4	70	6

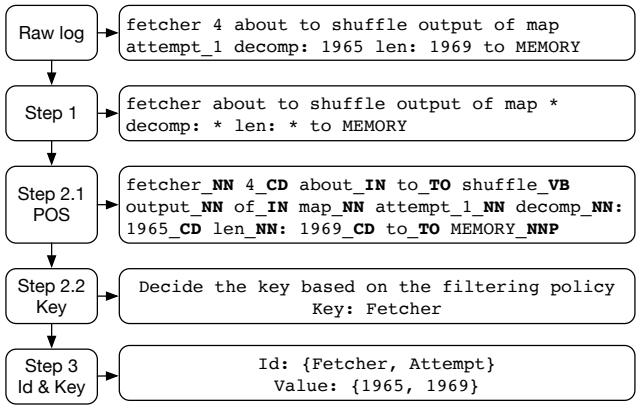


Fig. 2. An example of step-by-step parsing process on a log message by *NLog*.

III. KEYED MESSAGE

Once we have identified the keys of a log message and extracted the values and identifiers, we use them to construct keyed messages, a uniformed log storage structure. Users can reconstruct the workflow by applying database operations on the keyed messages.

A keyed message consists of four fields in order to describe a raw log message. Table III shows the four fields in a keyed message.

Key objects represent the objects or events in a log message. As described in Section II, a message type can be assigned with multiple keys. A log message can be transformed to as many keyed messages as the number of keys assigned to its corresponding message type.

Identifiers are used to uniquely identify the objects or eventsin a message.

Value stores a numeric value recorded in a log message if applicable. It is common that a log message contains a numeric value that usually indicates the amount of processed data or elapsed time.

TABLE III
DESCRIPTION OF FIELDS IN A KEYED MESSAGE.

Field	Description
key objects	the key objects in the message
identifiers	to identify the object
values	a list of numeric variables to store the value in the log message
timestamp	the timestamp in the log message

TABLE IV
ACCURACY OF KEY IDENTIFICATION IN DIFFERENT FRAMEWORKS. THE **TOTAL** COLUMN SHOWS THE TOTAL NUMBER OF MESSAGE TYPES OF EACH FRAMEWORK. THE **CORRECT** COLUMN SHOWS THE MESSAGE TYPES THAT ARE ASSOCIATED WITH CORRECT KEYS.

Platform	Total	Correct	Accuracy
Yarn	115	99	85.3%
Spark	34	32	94.1%
MapReduce	92	86	93.5%

Timestamp is extracted from the timestamp field of one log message.

Taking the advantage of keyed messages, operations such as Groupby, Count and Sum can be easily applied for workflow reconstruction. The procedure is shown in Section IV.

IV. IMPLEMENTATION AND EVALUATION

We implement *NLog* tool with about 2,000 lines of Java code and 100 lines of Python code. Its part-of-speech analysis and lemmatization are implemented by using OpenNLP [16], a natural language processing framework based on the principle of maximum entropy. We evaluate *NLog* using logs generated by Yarn [12], a popular container-based resource allocation framework in data-parallel clusters, and Spark [14] and MapReduce [13], two prevalent BigData application frameworks. In the implementation, we assign the corresponding application ids and the container ids to keyed messages. We use OpenTSDB [17] as the time series database and figure plotting tool.

The log files are generated by the applications on a 25-node cluster managed by Yarn. Each node is installed with Ubuntu Server 16.04 and has four Intel Xeon E5-2640 v3 CPUs, 132 GB RAM. The cluster is connected by 10 Gbps Ethernet. In experiments, we run Spark-2.1.0 and Hadoop MapReduce on Hadoop-3.0.0-alpha. The total amount of the log files is about 2 GB. For each framework (Yarn, Spark and MapReduce), we randomly select 20 MB log files to conduct the object identification and parsing process.

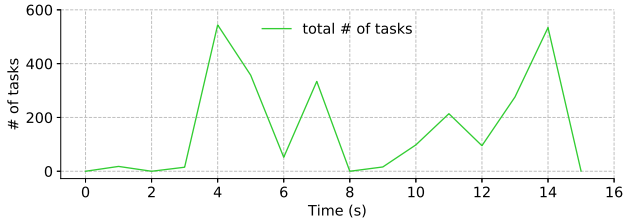
We evaluate *NLog* by answering the following two questions: 1) How accurate do keys reflect the contents of the log messages? 2) How can keyed messages be used to reconstruct the workflow?

A. Accuracy

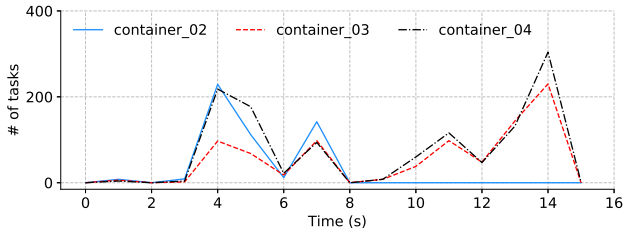
After the object identification process, message types are either assigned with keys or discarded. We evaluate the accuracy by presenting two metrics: 1) the number of message types assigned with keys, and 2) among the message types assigned with keys, the number of message types in which keys actually tell the objects or events recorded. Following the key assignment approach, an assigned key always appears in the corresponding message type. However, these keys might be words that have too general meanings, such as `event` or `service`. We categorize a message type as inaccurate, 1) if all of its keys have too general meanings for users to understand, or 2) if all of its keys do not include the key objects in the log message.

```
Q1: key: task                  Related message types:
    aggregator: count          1. Got assigned task *
                               2. Running task * in stage * (TID *)
                               3. Task * force spilling in-memory map to
Q2: key: task                  disk and it will release * memory
    aggregator: count          4. Finished task * in stage * (TID *)
    groupBy: container         5. Executor killed task * in stage * (TID *)
```

(a) Queries and related message types about tasks.



(b) Total number of tasks at each moment of execution.



(c) Number of tasks in each container.

Fig. 3. Queries, message types, and query results of Spark task objects.

We use Yarn, Spark and MapReduce as the targeted systems. Table IV shows the accuracy results. The accuracy of *NLog* in Yarn is lower than that in the other two systems. The reason is that a Yarn message type is usually written in a fashion of long sentence that records many objects. In such a case, *NLog* has more chances to assign inaccurate keys to a message type. To increase the accuracy, one solution is to increase the number of keys that can be assigned to a message type. However, the trade-off is that a message type may have too many keys such that it may confuse users during manual analysis since users have difficulty in knowing the actually important information. In contrary, the logs of Spark and Hadoop MapReduce are written in short sentences, each of which contains limited number of objects. Thus, *NLog* can more accurately identify the objects in logs.

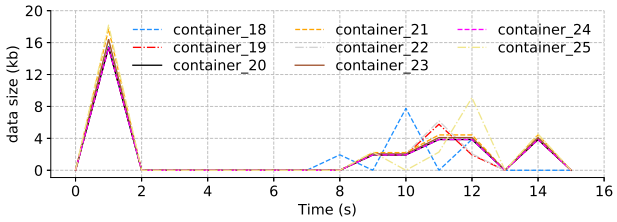
B. Queries on Keyed Message

In this experiment, we use two case studies, one with a Spark application and the other with a MapReduce application, to demonstrate the efficacy of keyed messages.

We run a Spark TPC-H application configured with three containers. By applying database operations on keyed messages, users can flexibly inspect the tasks during the execution. For the ease of presentation, we down sample all queries into one-second granularity. The queries and related message types are shown in Figure 3(a). Figure 3(b) shows that using query Q1, users can have a coarse-grained view of the workflow in terms of the number of running tasks during the execution.

```
Q3: key: fetcher               Related message types:
    aggregator: sum            1. Fetcher* about to shuffle
    identifier: {value=len}    output of * decamp: * len: *
    groupBy: container         to MEMORY
```

(a) Queries and related message types about fetchers.



(b) Data fetched by each container.

Fig. 4. Queries, message types, and query results of MapReduce fetcher objects.

In this figure, we find that the number of concurrently running tasks varies significantly from time to time during execution. After getting the overview of the number of tasks, users can further drill down and inspect the number of tasks in each container using query Q2 in Figure 3(a), which provides a fine-grained view of the workflow. Results are shown in Figure 3(c).

In this figure, we find that the number of tasks in different containers also varies significantly. From the 3^{th} second to the 6^{th} second, container_02 and container_04 receive much more tasks than container_03 does. From the 8^{th} second to the end of the application, however, container_02 does not receive any task. In practice, we find that the uneven distribution of tasks in containers is actually caused by a bug in Spark scheduler. Due to the limitation of space, we omit the details of how we find the bug and its root cause.

We also run a MapReduce wordcount application that is configured with 16 map tasks and 8 reduce tasks. During the reduce phase, the reduce tasks need to fetch the intermediate results. The amount of the fetched data is recorded in logs. *NLog* is capable to extract such events and the amount of the processed data. Query Q3 in Figure 4(a) obtains the amount of fetched intermediate data at each moment of execution. The results are shown in Figure 4(b).

The amount of fetched data is closely related to the network usage of a container. Such information helps users diagnose network interference in the cluster.

V. Discussions and Limitations

A major advantage of *NLog* is that it does not require the domain knowledge of the targeted system. The only constraint is that logs of the targeted systems must be written in a natural language. The effectiveness of *NLog* depends on the quality of log messages. In practice, we found that *NLog* has higher accuracy if a targeted system generates the log messages in a fine-grained manner where each log message only contains one object or event and its corresponding

information such as identifiers, amount of processed data, etc. Otherwise, the key assignment process may reduce accuracy of key identification process, because there are more objects in the log message than the maximum number of keys of a message type. Increasing the maximum number of keys helps capture the information in a log message, but can also mislead users away from the key information. In the experiments, we show that applying simple NLP based approaches can achieve high accuracy in identifying the key object in log messages. Currently, *NLog* targets log messages in English. The same idea can be extended to other languages by leveraging the corresponding POS taggers.

One drawback of *NLog* is that it only considers noun words in log messages, while some operations are not recorded as noun words. For example, words such as `Abort`, `Timeout` that explicitly indicate anomalies are common in log messages. We argue that users can use pattern matching to extract these specific words. In the future work, we aim to extend *NLog* to identify the importance of words that are not noun.

VI. Related Work

There are extensive studies on log analysis. Applying learning methods on logs generated by the targeted system is a hot topic [5], [18], [19], [7], [6], [20], [9], [21]. The goal of learning approaches is to automatically report potential anomalies to users, which reduces efforts required for users to do manual analysis. One approach [7] extracts fields in log messages by inferring the source code, and applies Principle Component Analysis to detect the anomalies.

Studies [4], [3], [2], [22], [23], [24], [25], [15], [26] focus on using logs to reconstruct the workflow. Our work also falls into this category. By leveraging the reconstructed workflow, users can have a clear view of the system components that serve user requests. Iprof [3] performs source code analysis to infer the relationship between log printing statements. By using such information, it can identify interleaved logs and associates them to requests. A recent work [2] reconstructs the workflow by identifying the hierarchical relationship between object identifiers. However, it requires the logs to follow certain constraints to guarantee the accuracy.

VII. Conclusion and Future Work

This paper presents a NLP-based approach, *NLog*, that identifies the key objects in logs. It proposes to use keyed message, a uniformed structure to store information in raw logs. By applying database operations, users can easily reconstruct the workflow. It also provides the flexibility for users to view the workflow in either coarse-grained or fine-grained level. Experiments show that the NLP-based approach has the ability to accurately identify the key information in logs.

In the future, we plan to use more sophisticated NLP-based approaches to extract more fine-grained information from logs generated by distributed data analytics systems.

VIII. Acknowledgement

We thank the reviewers for their tremendous feedback. This research was supported in part by U.S. NSF award SHF-1816850 and Colorado State Bill 18-086.

References

[1] M. Du and F. Li, "Spell: Streaming parsing of system event logs," in *Proceeding of IEEE Internatinal Conference on Data Mining (ICDM'17)*. IEEE, 2017.

[2] X. Zhao, K. Rodrigues, Y. Luo, D. Yuan, and M. Stumm, "Non-intrusive performance profiling for entire software statcks based on the flow reconstruction principle," in *Proceeding of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI'16)*. USENIX Asscociation, 2016.

[3] X. Zhao, Y. Zhang, D. Lion, M. FaizanUllah, Y. Luo, D. Yuan, and M. Stumm, "Iprof: A non-intrusive request flow profiler for distributed systems," in *Proceeding of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI'14)*. USENIX Asscociation, 2014.

[4] I. Beschastnikh, Y. Brun, S. Schneider, M. Sloan, and M. D. Ernst, "Leveraging existing instrumentation to automatically infer invariant-constrained models," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering (ESEC/FSE'11)*. ACM, 2011.

[5] D. J. Dean, H. Nguyen, X. Gu, H. Zhang, J. Rhee, Nipun, Arora, and G. Jiang, "Perfscope: Practical online server performance bug inference in production cloud computing infrastructures," in *Proceeding of 5th ACM Symposium on Cloud Computing (SoCC'14)*. ACM, 2014.

[6] K. Nagaraj, C. Killian, and J. Naville, "Structured comparative analysis of systems logs to diagnose performance problems," in *Proceeding of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI'07)*. USENIX Association, 2012.

[7] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Procceeding of the 22th ACM Symposium on Operating Systems Principles (SOSP'09)*. ACM, 2009.

[8] Z. Yin, D. Yuan, Y. Zhou, S. Pasupathy, and L. Bairavasundaram, "How do fixes become bugs?" in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering (ESEC/FSE'11)*. ACM, 2011.

[9] X. Yu, P. Joshi, J. Xu, and G. Jin, "CloudSeer: Workflow monitoring of cloud infrastructures via interleaved logs," in *Proceeding of the 21th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'16)*. ACM, 2016.

[10] D. Ippoliti, C. Jiang, Z. Ding, and X. Zhou, "Online adaptive anomaly detection for augmented network flows," *ACM Transaction on Autonomous and Adaptive Systems*, vol. 11, no. 3, pp. 17:1–17:28, Sep. 2016.

[11] "Spark," https://spark.apache.org/.

[12] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth *et al.*, "Apache Hadoop YARN: Yet another resource negotiator," in *Proceeding of The ACM Symposium on Cloud Computing (SoCC'13)*. ACM, 2013.

[13] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, 2008.

[14] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets." in *Proceeding of 2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud'10)*. USENIX Asscociation, 2010.

[15] A. Pi, W. Chen, X. Zhou, and M. Ji, "Profiling distributed systems in lightweight virtualized environments with logs and resource metrics," in *Proceeding of the The 27th International ACM Symposium on High-Performance Parallel and Distributed Computing (HPDC'18)*. ACM, 2018.

[16] "OpenNLP," https://opennlp.apache.org/.

[17] "OpenTSDB," http://opentsdb.net/.

[18] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceeding of the ACM Conference on Computer and Communications Security (CCS'17)*. ACM, 2017.

[19] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang, and X. Chen, "Log clustering based problem identification for online service systems," in *Proceeding of IEEE/ACM 38th IEEE International Conference on Software Engineering Companion (ICSE'16)*. IEEE/ACM, 2016.

[20] L. Luo, S. Nath, L. R. Sivalingam, M. Musuvathi, and L. Ceze, "Troubleshooting transiently-recurring problems in production systems with blame-proportional logging," in *Proceeding of the 2018 USENIX Annual Technical Conference. (ATC'18)*. USENIX Association, 2018.

[21] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan, "Online system problem detection by mining patterns of console logs," in *Proceeding of IEEE International Conference on Data Mining (ICDM'09)*. IEEE, 2009.

[22] I. Beschastnikh, Y. Brun, M. D. Ernst, and A. Krishnamurthy, "Inferring models of concurrent systems from logs of their behavior with csight," in *Proceeding of the 36th International Conference on Software Engineering (ICSE'14)*. IEEE/ACM, 2014.

[23] Z. Chothia, J. Liagouris, D. Dimitrova, and T. Roscoe, "Online reconstruction of structural information from datacenter logs," in *Proceeding of the 12th ACM European Conference on Computer Systems (Eurosys'17)*. ACM, 2017.

[24] J.-G. Lou, Q. Fu, S. Yang, J. Li, and B. Wu, "Mining program workflow from interleaved traces," in *Proceeding of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD'10)*. ACM, 2010.

[25] D. Yuan, H. Mai, W. Xiong, L. Tan, Y. Zhou, and S. Pasupathy, "Sherlog: error diagnosis by connecting clues from run-time logs," in *Proceeding of the 15th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'10)*. ACM, 2010.

[26] W. Chen, A. Pi, S. Wang, and X. Zhou, "Characterizing scheduling delay for low-latency data analytic workloads," in *Proceeding of the The 32nd IEEE International Parallel and Distributed Processing Symposium (IPDPS'18)*. IEEE, 2018.