Log Anomaly Detection using NLP

Anil Tipirneni & Arun Surendranath

Abstract

System logs have been widely used in software engineering and development to save immense run time information. With the advent of modern computers and software systems, the wealth of information generated by these automated logging systems has immensely increased in size; with such a vast amount of data logs generated, it becomes pertinent for software engineers and scientists to explore the advantages of using an AI-based system to analyze these system logs and make accurate predictions and make the system more efficient. We are planning to use the log hub-based database (we will have to choose the database here) Many existing solutions for log anomaly detection are based on traditional data-mining techniques, such as regular expressions filtering entries containing keywords that may be indicative of anomalies. Such solutions, apart from having a high false-positive rate, are not able to detect some types of anomalies. This is a significant challenge due to the rate at which logs are created in large distributed systems. An efficient way to understand the logs and react can be precious to companies as this will reduce downtimes and improve the overall performance of the systems.

Introduction

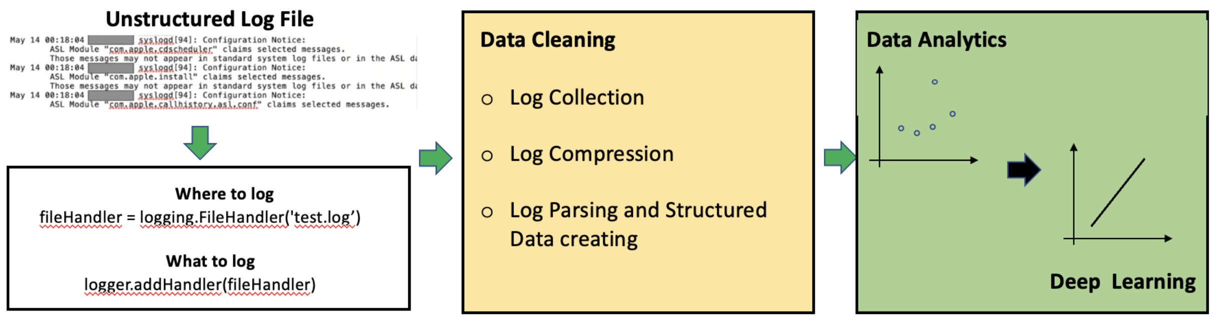
The comprehensive run-time information of all software applications is often recorded into message logs, which allows software developers and engineers to watch behavior and do any necessary analysis to monitor system performance. Message logs have been used extensively in the creation and maintenance of software.

Whenever an event on the network that falls under a certain classification occurs, a computer generates log files automatically. Log files were created because text-based records of the system's activities make it simpler for software and hardware engineers to diagnose and debug their products. Each of the top operating systems is specifically set up to produce and classify event logs in response to particular kinds of occurrences. In

Why use the Log Files

To power essential business services, large IT firms rely on a broad network of IT infrastructure and applications. Monitoring and analyzing log files make this network more observable, increasing transparency and enabling visibility into the cloud computing environment. While it shouldn't be viewed as the final aim, observability should always be viewed as a tool for attaining actual business goals: increasing the systems' dependability for users. Log files include data about system performance that may be used to assess whether more space is required to improve user experience. Analysts may use log files to find slow queries, transaction-delaying problems, and performance-affecting flaws in websites and applications. Maintain cloud computing environments' security posture and stop data breaches.

User behavior analytics is a field of study that examines user activity in log files, which record user behavior within an application. Developers can improve an application's performance to help users reach their goals more quickly, boosting customer happiness and generating income. An example framework for AI-powered log analytics is shown in Figure 1.



```
May 14 00:18:04 [redacted] syslogd[94]: Configuration Notice:
        ASL Module "com.apple.cdscheduler" claims selected messages.
        Those messages may not appear in standard system log files or in the ASL da
May 14 00:18:04 [redacted] syslogd[94]: Configuration Notice:
        ASL Module "com.apple.install" claims selected messages.
        Those messages may not appear in standard system log files or in the ASL da
May 14 00:18:04 [redacted] syslogd[94]: Configuration Notice:
        ASL Module "com.apple.callhistory.asl.conf" claims selected messages.
```

Where to log
fileHandler = logging.FileHandler('test.log')
What to log
logger.addHandler(fileHandler)
Data Cleaning
Log Collection
Log Compression
Log Parsing and Structured Data creating
Data Analytics
Deep Learning

Figure 1: Log analytics framework

Loghub Datasets

System logs are kept in a database by Loghub[29] and are freely available for study. While some logs were gathered from existing systems in by log hub, others were production data from earlier investigations. The logs are not anonymized, sanitized, or altered in any manner when feasible. Over 77 GB of logs are contained in all of these. with requests for the complete dataset being processed through the open dataset-sharing platform Zenodo [30].

For this work, we are considering log datasets from the supercomputer logs such as Thunderbird, Blue Gene(BGL), and windows systems logs. All the log datasets are downloaded from the loghub dataset (insert reference here). An overview of the loghub datasets is shown in Table 1, along with some information about the description, time period, #messages, data size, and label information. Time span specifically identifies the time frame over which the logs are gathered. A dataset's total number of log messages is indicated by the number #Messages. Data volume is represented by data size.Labeled and unlabeled log datasets fall into several groups.For certain log analysis tasks, logs in labeled datasets contain labels (e.g., anomaly detection and duplicate issues identification).

Software System	Description	Labeled	Time Span	No. Messages	Data Size
BGL	Blue Gene/L supercomputer log	Yes	214.7 days	4,747,963	708.7MB
Thunderbird	Thunderbird supercomputer log	Yes	244 days	211,212,192	29.6 GB
Windows	Windows event log	No	226.7 days	114,608,388	26.09 GB

Table 1: Log datasets

Log Parsing

The majority of AI-powered log analysis methods call for structured input data, such as a list of system events complete with event IDs or a matrix. On the other hand, software logs are often unstructured documents that include many fields as well as descriptions written by developers in their native language. Thus, a critical step in AI-powered log analytics that converts unstructured log messages into organized system events is log parsing. Traditional parsing systems, which mostly depend on manually creating parsing rules, become labor-intensive and ineffective when the number of logs grows quickly. Recent research has suggested many data-driven log parsers [13, 18, 19, 21, 24, 25, 26, 27, 28, 31, 32] to solve this issue. These parsers automatically identify an unstructured log message with the appropriate system event ID. Log parsing is often treated as a clustering issue. Log messages reporting the same system event should be grouped. The system event or event template is considered the standard tokens included in all log messages belonging to the same group. Log parsers may summarize the related system events and match each log message with an event ID by grouping log messages into clusters. The structured logs (i.e., log messages with event IDs) might be quickly converted into a matrix or used directly by log analysis methods. To help narrow down each unique error in the logs, we introduced an additional step in parsing for each specific error message to a class ID. This information was used in the model training process as part of a multiclass prediction using advanced models such as WAN, DAN and BERT .

Anomaly Detection

Systems in the modern era have grown in size and complexity. Millions of people throughout the globe will be served by an expanding number of these systems that will operate continuously. Any significant downtime might result in a significant loss of income [47,62]. Thus, recent research [7,9,11,20,22,23,33] has concentrated on log-based anomaly detection algorithms that flag probable aberrant system behaviors by examining system runtime logs to increase the dependability of contemporary systems.

A binary classification is often used to simulate the anomaly detection problem. The output is a list of labels indicating whether an instance (such as an event or a time period) is abnormal. The input is a list of structured system events or a matrix. Most log-based anomaly detection methods fall into two categories: unsupervised [22, 23, 33] or supervised [7, 9, 11]. How to correctly identify abnormalities based on system logs is the research challenge of log-based anomaly detection.

Related Work

Learning-based strategies are suggested as harmful assaults get increasingly intricate. Three phases typically make up the pipeline for these techniques [4]. To convert log messages into log keys, a log parser is used first. The next step is to create a feature vector to represent a series of log keys in a sliding window using a feature extraction method like TF-IDF. Finally, an unsupervised technique is often used to identify unusual sequences[12,9]. Numerous deep learning-based techniques to log anomaly identification have recently been put forward [2,15,16,17,8,10]. The majority of current methods use recurrent neural networks, particularly long-short term memories (LSTM) or gated recurrent units (GRU), to model the typical log key sequences and generate anomalous scores to identify the typical log sequences that are abnormal [2,17,10].

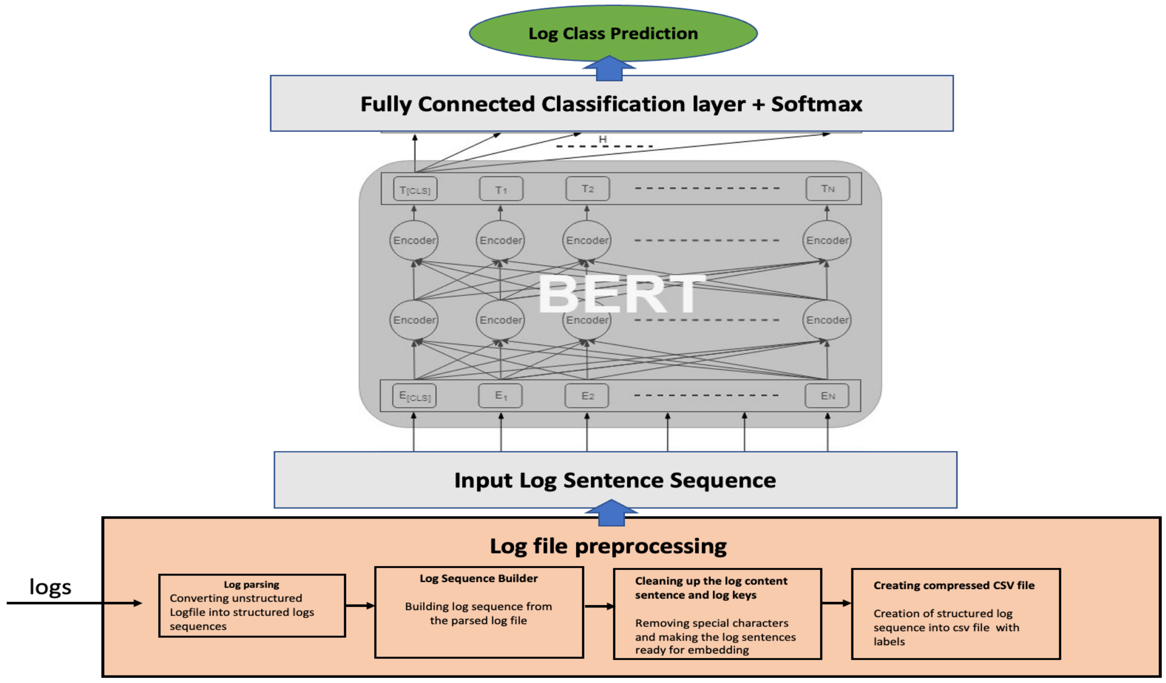
LogBERT, which was developed as an inspiration from BERT [3], uses the Transformer encoder to represent log sequences and is trained using brand-new self-supervised tasks to identify the patterns of typical sequences.LogBERT uses a self-supervised architecture based on bidirectional encoder representations from transformers (BERT) use of BERT to identify patterns in normal log sequences. It builds on the BERT's outstanding performance in modeling sequential text data. LogBERT will encode the knowledge about typical log sequences after training. Then, using LogBerT, it establish a criteria for identifying aberrant log sequences that are worth monitoring.

The approach used in this paper

For the proposed solution, we want to utilize the log hub database for BGL, Thunderbird, and Windows together with a log parser, ideally, the Drain parser as described in the log hub article. As the log message generated during an error message consists mainly of English vocabulary with a few numbers and codes, we will use it as the main input for building the vocabulary instead of log keys as discussed in the logbert (refer appendix) approach. This will allow us to use the BERT vocab builder to create word embeddings and use them to perform binary classification on the labeled datasets mentioned above. for this project; the scope was only to understand if the BERT and other word embedding methods could be used to classify the log messages based on their severity.

Data Cleaning

Most data cleaning was done in the initial phase of gathering the log files and parsing them into the structured dataset for machine learning applications. As fig.2 Below shows, the first step is to convert the unstructured log files into a structured log using the log parser module. The second stage of the process is to create a log sequence construction based on the log file input log; this can vary from different logs; in our approach, we had to use three different approaches for the above three datasets chosen. The third stage in the process is to clean up any special characters in the log messages (this will be the main input for creating the machine learning model), and the final step of the process is to convert the structured and cleaned log file into a .csv file, which can be used in various baseline and BERT based language modeling.



Log Class Prediction
Fully Connected Classification layer + Softmax
H
T[CLS]
T1
T2
TN
Encoder
BERT
E[CLS]
E1
E2
EN
Input Log Sentence Sequence
Log file preprocessing
logs
Log parsing
Converting unstructured Logfile into structured logs sequences
Log Sequence Builder
Building log sequence from the parsed log file
Cleaning up the log content sentence and log keys
Removing special characters and making the log sentences ready for embedding
Creating compressed CSV file
Creation of structured log sequence into csv file with labels

Figure 2: Overall Flow

Model Building

For the model building for the log anomaly detection, we will be using a baseline model as shown below; all the baseline modes will be compared to more advanced models based on word embeddings and

BERT to compare for improvement in overall classification tasks, the performance of the model will be evaluated based on Precision, Recall and F1 score

1.) PCA- principal component analysis,
2.) One-Class SVM- commonly used one-class classification model and commonplace in log anomaly detection [5,16]
3.) Isolation Forest - this algorithm uses an unsupervised methodology for anomaly detection using features as tree structure [6]

Method	BGL			Thunderbird			Windows		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score	Precision	Recall	F1-Score
PCA	10.15	94.78	18.33	37.35	100	54.39	6.91	100	12.93
Isolation Forest	100	13.91	24.42	34.59	1.61	3.09	53.173	100	69.429
One class SVM	1.57	15.44	2.83	17.14	34.74	22.96	-*	-*	-*
Log Clustering	99.06	60.58	75.19	98.08	45.39	62.06	100	49.36	66.10

Table 2: Baseline evaluation metrics(*- One class SVM is not appropriate for a multiclass windows log)

The baseline models, in general, give poor recall, F-1 scores, and precision numbers because these algorithms fail to gain enough insight from the log parameters. As a result, we will need more advanced word embedding techniques to extract features from the logs that can result in accurate classification of the log classes. Table 1 summarizes the baseline analysis of the log dataset from various sources using different algorithms.

For the advanced model building, we will be using the following algorithms for classification tasks.

1.) DAN-Deep average network takes the average of the embeddings and Passes it through 1 or more feed-forward layers. Intuition is that each layer will increasingly magnify small but meaningful differences in the word embedding average.
2.) WAN-An improvement on a model-averaging ensemble is a weighted ensemble, where each member's contribution to the outcome is weighed according to the model's performance.
3.) BERT-CLS - Bidirectional Encoder Representations from Transformers without pretraining of the embedded layer and using the CLS token to make the binary classification
4.) BERT-AVG - It combines by averaging the token-level embeddings' element-wise arithmetic means.
5.) BERT-CNN- Using the above BERT CLS with a CNN layer built on top of the dense layer,

All of the BERT models were developed without pretraining. Although pretraining was used in certain studies, there was no discernible difference between the trained and untrained BERT models.

Experimentation

We utilize the whole dataset for BGL and a portion of the dataset for Thunderbird and windows for model development (total dataset: 20M lines of logs); because of the size of this dataset, we randomize the

Thunderbird dataset and used just 5M log lines. The same is true with Windows dataset. The datasets were divided into train, test, and validation sets, each comprising 80%, 10%, and 10% of the entire dataset. Precision, Recall, and F-1 scores will be used as the primary metrics to evaluate the model's performance discussed in the sections above and to compare it to the baseline result to evaluate the effectiveness of advanced word embedding techniques on the classification of the logs.

Batch size and learning rate were optimized for the models in general; the learning rate of 0.0005 and batch size of 8 was utilized for all the advanced learning models to get the best performance for each of the hyperparameters.

Method	**BGL**			**Thunderbird**			**Windows**		
	Precision	**Recall**	**F1-Score**	**Precision**	**Recall**	**F1-Score**	**Precision**	**Recall**	**F1-Score**
DAN	92.6	92.56	96.13	95.54	95.59	97.72	100	93.1	96.42
WAN	92.56	92.56	96.14	95.545	95.545	97.72	100	93.1	96.42
BERT-CLS-no pretraining	99.51	99.51	99.75	95.6	95.6651	97.7845	100	99.91	99.95
BERT-CLS-Trained	92.566	92.50	96.10	95.58	95.855	97.7426	100	93.08	96.41
BERT-Avg	99.49	99.49	99.74	95.589	95.5882	97.7443	100	99.95	99.95
BERT-CNN	99.49	99.46	99.78	95.588	95.59	97.74	100	99.95	99.97

Table 3: NLP Algorithm analysis

To increase the credibility of the anomaly detection, during the parsing phase, HRESULT were searched inside the log and each unique code next to “HRESULT=” was defined in a different class. This information was trained to the BERT models and the model performed very similarly to the single class BERT model and accomplished a high precision, recall and F1-scores. Overall across all the logs, BERT-Avg and BERT-CNN provided the best results.

Conclusion

Log anomaly detection has become a critical need for most technological companies. This project gave us an opportunity to exercise different generic and NLP models against both trained and untrained log data. Overall, experimental results have proven that encoding algorithms performed the best in predicting the anomalies in the log data.

Potential next steps

- Expand the overall training to the larger datasets.
- Increases the labels to identify additional classes.
- Speed up parsing functions(it takes a couple of days to parse 27G of Windows data).
- Fine tune CNN hyperparameters to achieve closer to 100% recall rate
- Increase the overall Epochs(currently limited to 2 for most BERT cases)
- Predicting the sequence of the eventide using the BERT methodology
- Clustering the machines IDs to understand better the root cause of the logs events

Reference:

[1] 2019. F1 score. https://en.wikipedia.org/wiki/F1_score

[2] Du, M., Li, F., Zheng, G., Srikumar, V.: DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. (2017).

[3] Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs] (2018)

[4] He, S., Zhu, J., He, P., Lyu, M.R.: Experience Report: System Log Analysis for Anomaly Detection. In: 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE). (2016).

[5]Haixuan Guo 1 , Shuhan Yuan 1 , and Xintao Wu 2,LogBERT: Log Anomaly Detection via BERT,arXiv:2103.04475v1 [cs.CR] 7 Mar 2021

[6] Peter Bodik, Moises Goldszmidt, Armando Fox, Dawn B Woodard, and Hans Andersen. 2010. Fingerprinting the datacenter: automated classification of performance crises. In Proceedings of the 5th European Conference on Computer Systems (EuroSys). ACM, 111–124.

[7] Liu, F., Wen, Y., Zhang, D., Jiang, X., Xing, X., Meng, D.: Log2vec: A Heterogeneous Graph Embedding Based Approach for Detecting Cyber Threats within Enterprise. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. (2019).

[8] Pecchia, A., Cinque, M., Cotroneo, D.: Event logs for the analysis of software failures: A rule-based approach. IEEE Transactions on Software Engineering 39(06), 806–821 (2013).

[9] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. 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, 1285–1298

[10] Wang, Z., Chen, Z., Ni, J., Liu, H., Chen, H., Tang, J.: Multi-Scale One-Class Recurrent Neural Networks for Discrete Event Sequence Anomaly Detection. In: WSDM (2021)

[11] Qiang Fu, Jian-Guang Lou, Yi Wang, and Jiang Li. 2009. Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis. In ICDM 2009, The Ninth IEEE International Conference on Data Mining, Miami, Florida, USA, 6-9 December 2009. 149–158

[12] Xu, W., Huang, L., Fox, A., Patterson, D., Jordan, M.: Online system problem detection by mining patterns of console logs. In: 2009 Ninth IEEE International Conference on Data Mining. pp. 588–597. IEEE (2009)

[13] M. Du and F. Li. 2016. Spell: Streaming Parsing of System Event Logs. In ICDM'16 Proc. of the 16th International Conference on Data Mining.

[14] Yen, T.F., Oprea, A., Onarlioglu, K., Leetham, T., Robertson, W., Juels, A., Kirda, E.: Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks. In: Proceedings of the 29th Annual Computer Security Applications Conference. pp. 199–208 (2013)

[15] Zhang, K., Xu, J., Min, M.R., Jiang, G., Pelechrinis, K., Zhang, H.: Automated IT system failure prediction: A deep learning approach. In: 2016 IEEE International Conference on Big Data (Big Data). pp. 1291–1300 (2016).

[16] Zhang, X., Xu, Y., Lin, Q., Qiao, B., Zhang, H., Dang, Y., Xie, C., Yang, X., Cheng, Q., Li, Z., Chen, J., He, X., Yao, R., Lou, J.G., Chintalapati, M., Shen, F., Zhang, D.: 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. (2019).

[17] Zhou, R., Sun, P., Tao, S., Zhang, R., Meng, W., Liu, Y., Zhu, Y., Liu, Y., Pei, D., Zhang, S., Chen, Y.: LogAnomaly: Unsupervised Detection of Sequential and Quantitative Anomalies in Unstructured Logs. In: IJCAI. pp. 4739–4745 (2019)

[18] Hossein Hamooni, Biplob Debnath, Jianwu Xu, Hui Zhang, Guofei Jiang, and Abdullah Mueen. 2016. Logmine: Fast pattern recognition for log analytics. In Proceedings of the 25th ACM International on Conference on Information and Knowledge Management. ACM, 1573–1582

[19] P. He, J. Zhu, Z. Zheng, and M. R. Lyu. 2017. Drain: An Online Log Parsing Approach with Fixed Depth Tree. In ICWS'17: Proc. of the 24th International Conference on Web Services.

[20] S. He, J. Zhu, P. He, and M.R. Lyu. 2016. Experience Report: System Log Analysis for Anomaly Detection. In ISSRE'16: Proc. of the 27th International Symposium on Software Reliability Engineering

[21] Zhen Ming Jiang, Ahmed E Hassan, Parminder Flora, and Gilbert Hamann. 2008. Abstracting execution logs to execution events for enterprise applications (short paper). In 2008 The Eighth International Conference on Quality Software. IEEE, 181–186.

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

[23] Jian-Guang Lou, Qiang Fu, Shengqi Yang, Ye Xu, and Jiang Li. 2010. Mining Invariants from Console Logs for System Problem Detection.. In USENIX Annual Technical Conference. 1–14.

[24] Adetokunbo Makanju, A. Nur Zincir-Heywood, and Evangelos E. Milios. 2009. Clustering event logs using iterative partitioning. In Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009. 1255–1264.

[25] Masayoshi Mizutani. 2013. Incremental Mining of System Log Format. In 2013 IEEE International Conference on Services Computing, Santa Clara, CA, USA, June 28 - July 3, 2013. 595–602.

[26] Meiyappan Nagappan and Mladen A Vouk. 2010. Abstracting log lines to log event types for mining software system logs. In 2010 7th IEEE

[27] Keiichi Shima. 2016. Length matters: Clustering system log messages using length of words. arXiv preprint arXiv:1611.03213 (2016).

[28] Liang Tang, Tao Li, and Chang-Shing Perng. 2011. LogSig: generating system events from raw textual logs. In Proceedings of the 20th ACM Conference on Information and Knowledge Management, CIKM 2011, Glasgow, United Kingdom, October 24-28, 2011. 785–794.

[29] LogPAI Team. 2019. Loghub: A large collection of system log datasets for AI-powered log analytics. https://github.com/logpai/loghub

[30] LogPAI Team. 2019. Loghub: A large collection of system log datasets for AI-powered log analytics. https://zenodo.org/record/1596245#.XMMZ1dv7S-Y

[31] Risto Vaarandi. 2003. A data clustering algorithm for mining patterns from event logs. In IP Operations & Management, 2003.(IPOM 2003). 3rd IEEE Workshop on. IEEE, 119–126.

[32] Risto Vaarandi and Mauno Pihelgas. 2015. LogCluster-A data clustering and pattern mining algorithm for event logs. In 2015 11th International Conference on Network and Service Management (CNSM). IEEE, 1–7.

[33] Wei Xu, Ling Huang, Armando Fox, David A. Patterson, and Michael I. Jordan. 2009. Detecting large-scale system problems by mining console logs. In SOSP. 117–132.