Self-Supervised Anomaly Detection from Distributed Traces

Jasmin Bogatinovski*‡, Sasho Nedelkoski*‡, Jorge Cardoso†§, Odej Kao*
*Complex and Distributed IT-Systems Group, TU Berlin, Berlin, Germany
{jasmin.bogatinovski, nedelkoski, odej.kao}@tu-berlin.de
†Huawei Munich Research Center, Munich, Germany
§CISUC, Dept. of Informatics Engineering, University of Coimbra, Portugal
jorge.cardoso@huawei.com
‡Equal contribution

***Abstract*—Artificial Intelligence for IT Operations (AIOps) combines big data and machine learning to replace a broad range of IT Operations tasks including reliability and performance monitoring of services. By exploiting observability data, AIOps enable detection of faults and issues of services. The focus of this work is on detecting anomalies based on distributed tracing records that contain detailed information of the services of the distributed system. Timely and accurately detecting trace anomalies is very challenging due to the large number of underlying microservices and the complex call relationships between them. We addresses the problem anomaly detection from distributed traces with a novel self-supervised method and a new learning task formulation. The method is able to have high performance even in large traces and capture complex interactions between the services. The evaluation shows that the approach achieves high accuracy and solid performance in the experimental testbed.**

***Index Terms*—anomaly detection; distributed traces; distributed systems; self-supervised learning.**

I. Introduction

Billions of devices and users depend on the availability of large distributed systems such as the cloud. Many applications often have service-level agreement (SLA), where uninterrupted service with low response times guarantees are required. Therefore, loss of control is not allowed for any system or infrastructure.

Successful operation of large-scale systems requires deployment of numerous utilities. These utilities include introducing additional intelligence to the IT-ecosystem, such as employing network reliability engineers (NRE), site reliability engineers (SRE), using automated tools for infrastructure monitoring, and developing tools based on artificial intelligence for load balancing, capacity planning, resource utilization, storage management, and anomaly detection. The techniques and methods from the field of Artificial Intelligence for IT Operations (AIOps) become an immutable part of the monitoring and utility toolboxes to address the challenges imposed by the previous tools. AIOps uses machine learning, data analytics, and monitoring data to improve the operation and maintenance (O&M) of distributed systems.

The first step towards available and reliable systems and services is that an anomaly must be detected and recognized, before it leads to a service or a system failure. Timely detection allows prevention and increasing the opportunity window for conducting a successful reaction from the operator. This is especially important if urgent expertise and/or administration activity is required. These anomalies often develop from performance problems, component and system failures, or security indignant and leave some fingerprints within the monitored data: logs, metrics or distributed traces.

Depending on the origin of the data, the observable system data, describing the state in distributed IT system, are grouped into three categories: metrics, application logs, and distributed traces [1], [2]. The *metrics* are time-series data representing the utilization of the available resources and the status of the infrastructure, typically regarding CPU, memory, disk, network throughput, and service call latency. *Application logs* record which actions were executed at runtime by the software. The metrics and log data sources are limited on a service or resource level. They cannot relate the interactions between the different components within a distributed system. *Distributed traces* are a graph-like abstraction built on top of logs that encode information for the multiple services serving a particular user request. Traces are composed of events or spans. The spans contain information about the execution workflow and performance at a (micro)service level. As such, they preserve the information for the interaction between the services and are better suited for tasks such as anomaly detection.

While the anomaly detection using system log and metric data has been previously investigated [3]–[6], only a few studies make use of tracing data [7], [8] as it is significantly more complex to implement, collect, and handle.

There are three important requirements of anomaly detection from distributed traces; the methods should handle traces under the assumption of existing of noise, traces can be of arbitrary length, and the methods should be unsupervised. Due to the complex nature of the operations within a distributed environment and large noise, it can happen that although the observed sequence of events is not present in the set of observed traces, it still to be normal. Noise occurs because complex systems rely on software patterns, such as caching and load balancing, to increase efficiency and reliability. The existence of noise has a strong implication for trace anomaly detection since methods need to classify traces which were

978-0-7381-2394-3/20/$31.00 ©2020 IEEE
DOI 10.1109/UCC48980.2020.00054

never seen before as normal. The existing approaches, such as [9], use finite state machines (FSM) to model the correct behaviour of systems. These approaches work well when traces do not contain noise. However, the introduction of noise scales the number of potential transitions exponentially.

The second challenge is related to the range of lengths of a trace for different operations. Approaches relying on LSTMs [7], [8] can only process traces up to a certain length of k. These approaches are termed autoregressive since they use previous spans of a trace to predict the following span. If the prediction is correct, the trace is classified as normal. Otherwise, it is anomalous. The problem is that as traces become long, the first spans of traces in a behaviour model are forgotten.

Lastly, we only consider the option for unsupervised method, as the labelling by experts or injection of anomalies directly into the cloud platforms to obtain labeled data do not meet the requirements of real-word systems.

Contribution. We address these challenges by proposing a formulation of the problem of next span prediction [8] into a masked event prediction task. Masked span prediction as a learning task is concerned with the correct prediction of a masked event on a random position in the trace utilizing the remaining, non-masked information from the trace. The decision for the normality of a trace is done with a threshold procedure at the top of the masked event prediction procedure. This allows the following benefits. The method opens the ability to learn from the fully observed trace structure. It exploits the information from the context spans in the trace in order to predict the masked span allowing for arbitrary dependency between the spans in the trace. The method does not use labels for learning, hence it is unsupervised. It exploits the overall information of the trace, meaning it is not dependent on the length of the trace. The challenge of noise generated by the underlying distributed system is addressed first with sparse encoding and second with allowing for an arbitrary number of span to be mistaken. This allows for greater flexibility of correctly modeling of unseen normal traces, which results in improving the performance scores. With exhaustive empirical evaluation we show that the method outperforms the previous state of the art on both experimental testbed.

II. Related work

Anomaly detection as a data mining task is important due to its great practical relevance across many diverse areas. As a learning task is concerned with finding observations in a corpus of data that differ from the expected behaviour [10]. Anomalies in large systems such as cloud and high-performance computing (HPC) platforms can impact critical applications and a large number of users [11]. Therefore, a timely and accurate detection is necessary for achieving reliability, stable operation, and mitigation of losses in a complex computer system.

A common approach for data-driven anomaly detection from tracing data is with a one-class classification [12] approaches. They assume that the available data originate from the normal operation of the observed system. Thus, the methods aim to fit a decision boundary around the normal data. The data points that lie outside of the learned decision boundary are classified as anomalies. Recently, the anomaly detection in complex distributed systems (e.g., cloud platforms) using various monitoring data is gaining a lot of popularity [7]–[9].

One approach to model the normal execution of the traces per workload is to construct a Finite Discrete Automata (FDA) to capture the complete trace execution cycle [9]. In this work, the FDA is built per workload from a specialized distributing tracing procedure. The generated FDA's are often complex due to the large size of the traces. This makes the FDA highly sensitive on the monitoring tracing procedure when transiting from one state to another. Since the transition from a current state to the next one utilizes the information up to the current span within the trace, the FDAs are not efficient in the utilization of the whole available information from the trace. This method does not support publicly available implementation making it hard for comparison.

The current systems for anomaly detection using tracing data and deep learning techniques model the normal system behaviour by utilizing history h of recent log/trace events as input, and predict the next event key in the sequence [7], [8]. We refer to this task as the next event/span prediction. They utilize the execution path of the trace and LSTMs [13], [14] to enable the learning from the sequential nature of the traces. The anomaly detection is performed by predicting the next span, if the prediction is successful then the span is normal, otherwise is considered as anomaly.

In contrast to the above anomaly detection systems, we reformulate the problem of next event prediction into mask span prediction. The model opens the ability to learn from the fully observed trace structure. It exploits the information from the context spans in the trace to predict the masked event.

III. Attention Approach for Anomaly Detection in Distributed Traces

In this section, first, we discuss the data preprocessing step for the approaches described in this work. We built an intuition for the definition of the self-supervised task explaining the strong aspects of such problem formulation. We describe the architecture of the main model.

A. Distributed Traces

Distributed traces record the workflows of services executed in response to user requests. These records inside a trace are called spans or events and represent information (e.g. start time, end time, service name, HTTP path, function in RPC calls etc.) about the operations performed when handling an external request. Formally, a trace can be written as $T_i = (S_1^i, \ldots, S_m^i)$, where $i \in \{1, \ldots, N\}$ is a trace from the dataset, and m is the length of T_i or the number of spans in the trace.

Depending on the executed workload, the traces have different lengths and different events having unique description. The

depth of the description of a span depends on multiple factors. For example, a span referring to a RPC call is described with a function call (with respective parameters) while, the span referring to http call can be described with the http method and the endpoint. So, the different nature of the event being executed constraints the set of descriptive features.

Due to the great variability of the endpoints within the distributed system, taking just a raw span description and mapping it into a symbol will result in a very large vocabulary [7]. To mitigate this problem one needs to filter out the parts of the spans that are not relevant. To this end, one can utilize the fact that the result of the filtering step allows for the traces to be treated similarly as the payload of log messages. This opens an opportunity to use a wide toolbox of log-processing methods. To that end, we utilize Drain [15], as being the best overall log-parser [16]. The Drain method constructs a tree based on the frequency counts of the words appearing on particular positions within the text sequence. The spans with reduced description are given as input to Drain. Drain as output generates a set of groups of spans or span templates. Thus the trace can be represented as a sequence of span templates or sequence of symbols.

Due to the complexity arising in the distributed systems it can happen that the traces of successful execution of the same request multiple times, the traces to be of different length. To account for the different length of the traces, the traces are padded up to (max_len) number of spans. This parameter represents the maximum allowed trace length. Additionally, to conserve the information for the length of the traces, two special spans ([START] and [STOP]) are introduced. They are added to the beginning and the end of the trace, prior to padding. The training dataset is composed just of traces of normal execution of the workload. This is a strong requirement since the cost for labeling tracing data is large.

B. Method Design

A common approach for modeling the normal execution of a trace is utilizes the autoregressive modelling concept [9]. It assumes that each trace has a particular number of spans m, where each span $S_m^i \in \mathcal{V}$ is part of the set of span templates $\mathcal{V}$. An autoregressive statistical approach aims to assign probabilities to sequences of symbols using the chain rule of probability:

$$P(S_{1:T}) = \prod_{t=1}^{T} P(S_t|S_{<t}) \tag{1}$$

where $S_{<t}$ denotes the (potentially empty) sequence of spans from S_1 to S_{t-1}. The conditional probabilities on the right-hand side can be modeled with a recurrent neural network e.g LSTM [8]. This approach is autoregressive, since it uses just forward context at time. As such it is limited in the amount of exposed context during the learning phase e.g backwards context is not taken care of.

In our approach, the underlying premise suggests that the appearing of a span on a particular position in a trace is conditioned not only on the previous spans but also on the spans appearing afterwards. With other words, each span within a trace appears within a context of its neighbours. Thus, the representation of a span directly depends on its location in the trace and its relation to the neighbouring context spans. This is a reasonable assumption, as the spans in a trace have causal relationships describing the inter- and intra-service calls. In Fig. 1, we show a trace when the user requests *creation and deletion of a network* as operation in the cloud. At the bottom of the image the *POST /v3/auth/tokens/* span is replaced with the *[MASK]* span. We refer to the *POST /v3/auth/tokens/* as *masked span*, while all of the other spans form its context.

Fig. 1. (Up) An example of a trace as result from *network create and delete trace* user request. (Down) an example of the context of the `POST /v3/auth/tokens/` span as given to the input of self-attention mechanism. *POST /v3/auth/tokens/* is called a masked span.

To address the problem of the prediction of a masked span of a trace given its context, we introduce the masked span prediction (MSP) task. As a learning task, it aims to pinpoint what is the most likely span to occur on a particular masked position given its context from the neighbouring spans. A masked span in a trace can be any randomly chosen span that during the learning procedure is labeled with a special *[MASK]* span from the input. During the learning procedure, the true value of the masked token is used as a target and it is predicted by the remaining spans that construct the context (given as input). In such a way, one allows for the masked span to "score" how much the spans from the context are relevant for its prediction.

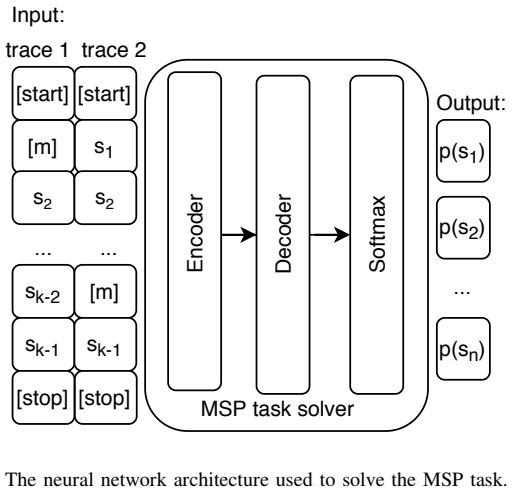

Fig. 2. The neural network architecture used to solve the MSP task. It is an encoder-decoder architecture with residual connections, layer-normalization and dropout at each layer applied as regularization.

Fig. 2 depicts the adopted method. It is an encoder-decoder structure that maps the context of the span in a vector format

to a probability distribution over the vocabulary of spans. The encoder uses a multi-head self-attention neural network learning mechanism as an encoder [17]. This architecture allows for selective prediction of the relevant spans from the context for prediction of the given masked span.

To train the neural network the inputs should be in the appropriate format. They are formed in a way that for each of the traces k randomly chosen spans during the learning phase, are masked. In such a way a single trace is multiple times fed through the encoder neural network, each time with different span being masked. As such it is a key feature that allows the usage of the method for anomaly detection. Additionally, it gives flexibility of the method to build one single model for multiple user requests instead of separate models for different user requests. Encoder neural network implements the self-attention learning mechanism.

Outputs from the self-attention layer are fed through a one-layer network with soft-max at the end that serves as a decoder. The soft-max is used as a function to generate probability estimates over the whole dictionary of span templates $\mathcal{V}$. These probabilities suggest how likely is the current masked span to be associated with a symbol within the vocabulary of symbols conditioned on the context.

The MSP is a proxy task for anomaly detection. As a standalone task cannot be used for anomaly detection. To this end, we introduce additional postprocessing of the predictions of the MSP model to detect anomalies. The output from MSP for a particular trace is an ordered list of spans for each position of the trace. The lists are ordered according to their relevance to fill up the particular position of the trace. During the anomaly detection procedure, each of the ordered lists on the particular position in the trace is examined in the following way. If the real value on a particular position of the trace is not in the first $top - k$ elements of the list generated of the MSP task module for that position, we consider the span as not being correct. We count the number of errors for each trace and divide with the trace length forming the ratio of misclassified examples (span error rate per trace). This span error rate serves as an anomaly score. It is expected that if a model makes many mistakes, the anomaly score to be high, thus the trace is anomalous. Setting a decision threshold on the anomaly score serves to decide if the trace is normal or anomalous.

IV. Results and Discussions

In this section, first, details on experimental testbed used for evaluation of the method and the data are given. Second, a description of the learning scenarios we adopted to evaluate the performance of our method and the corresponding comparative analysis of the three learning scenarios.Finally, we investigate how the attention scores differ between the normal and anomalous traces and how this can be utilized to infer interesting patterns between the normal and anomalous traces.

A. Experimental data

To test and verify our approach and due to the absence of publicly available datasets for evaluation, we first deployed an OpenStack [18] testbed. Fig. 3 depicts the architecture. It is based on a microservice architecture, running in a dockerized environment Kolla-Ansible [19]. There are 4 compute and one control node. Further specifications include deployment on bare-metal nodes, where each node has RAM 16GB, 3x 1TB of disks, and 2x 1Gbit Ethernet NIC. To automate and unify the multi-node OpenStack deployment, cloud verification, testing, profiling the workload generation and anomaly injection we used Rally [20].

The normal and anomalous data is generated from the execution of the three workloads. *Create and delete server* uses a task from Rally to create and delete a virtual machine. The fault is injected in a compute node which restarts the API container that runs on the compute nodes. This simulates a failure of a service. *Create and delete image* uses the *glance* project of OpenStack to create and delete an image. The faults are injected via restarting of the glance-API which runs on the controller node. *Create and delete network* is a an operation that provides network interface. The anomaly is injected with disturbing of one of the neutron services: (e.g. neutron metadata agent, neutron server) during the creation of a network.

To represent a scenario as close to the real-world, the operations are executed concurrently. Furthermore, the operations are scheduled to last for an equal period. Since some of the operations are faster than others (e.g, we need greater time to boot a machine compared to the time needed to create a network) the operations are started with different repetitions. Specifically, 2000, 3000, and 6000 iterations for create and delete server, create and delete image and create and delete network were conducted, respectively. The injection of the faults happens at different rate - 250 for *create and delete server* and *create and delete image* and 500 iterations for *create and delete network*. After the execution of the sequence of workloads, reports for the conducted experiments are generated. The reports have details for the successful execution of the workload. They are used to induce the ground truth label for a particular user-request. This is needed to separate the normal from anomalous traces to perform the evaluation.

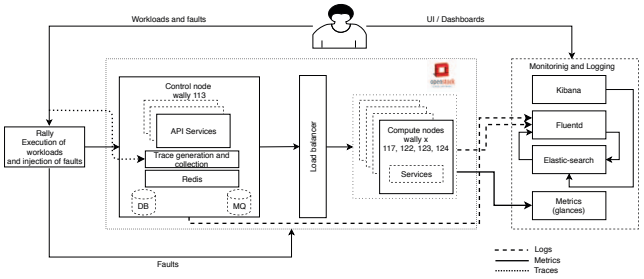

Fig. 3. Illustration of the infrastructure from where the data was generated.

To test and compare the performance and robustness of the proposed attention method and the state of the art in anomaly detection from tracing data, we consider three learning scenarios. To test our hypothesis of the ability of our method to preserve the global and local properties within the traces we grouped our experiments into three groups: short, long, and combined (short and long) traces. The definition of this categories is data-driven, meaning that for the short traces are considered the once that are concentrated around the lower values of the median (≤ 14), while the remaining fall into the category of long traces.

Real anomalies (LS1). In this learning scenario, in the test set, the anomalies as generated by the deployed testbed were used. The goal of this scenario is to inspect how well the methods for anomaly detection from tracing data perform under the presence of anomalies as generated by the system. We tested both of the methods on 9 different parameter configurations. We select the best parameter configuration in this scenario and used it to test its generalizations of both of the approaches in the remaining of the experiments.

Artificial anomalies (LS2). Observation of the anomalies as generated by the system, usually result in a shorten trace. Restarting a service results in interrupting the trace at particular span. The anomalies can be injected at every span in the trace since every span represents a service that can be malfunctioning. To test the generalization and the robustness of how the methods scale to a different type of anomaly a set of artificial anomaly test is created. The creation of this set is done in a way that L traces from the normal test set are selected at random. The selected traces are truncated at a random position. These traces are labeled as an anomalous and are joint with the remaining normal traces to create the test set.

For the LS2 scenario, the best-selected method from the optimization procedure as in LS1 is selected. This allows to directly test the change of the performance of the methods to novel anomalies.

B. Implementation Details

The anomaly detection methods were implemented in Python using Pytorch [21]. The experiments were collected on a personal computer with GPU-NVIDIA GTX 1660. To obtain the best models the number of training epochs and the batch sizes were varied in the following range batch size$= 16, 64, 256$ and epochs$= 10, 25, 50$, accordingly. The number of the layers for the attention model was set to 1 and the number of recurrent layers to 2. The hidden sizes for both models were set to 256. In the anomaly detection phase, the threshold was varied from 0 to 1 with a step of 0.05.

For the parameters of Drain, the values for the similarity and depth are 0.4 and 4 respectively. For training the model, the parameter P is set to 70 % of the number of normal traces. The value for window size parameter for LSTM is set to 3 since it is the maximal number that covers all of the traces in the training set. For the attention method for the parameter max_len the value of 90 is set since it covers the traces of all the lengths within the data.

C. Predictive Performance

In the following, the results from the three learning scenarios are presented. We present the precision, recall and F1 score. For the positive class, we consider the anomalous traces. Thus, the true positives are defined as a correct prediction of true anomalies. True negatives are defined as the correct prediction of normal traces. False positives are predicted anomalies when the trace is normal, while the false negatives are traces that are predicted as normal when their ground truth label is an anomaly.

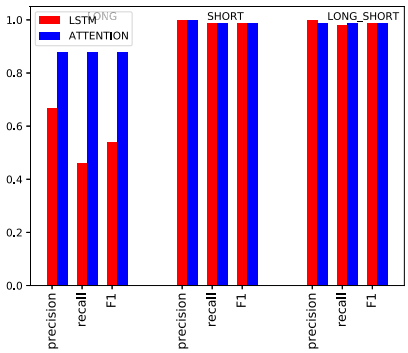

Fig. 4. Results from the experiments for LS1.

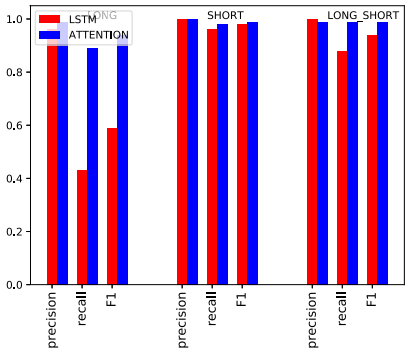

Fig. 5. Results from the experiments for LS2.

Fig. 4 depicts the results of LS1 for the best-selected models from the optimization procedure. When the long traces are considered, one can observe high scores for the attention mechanism compared to the previous state-of-the-art deep learning approaches, the LSTM approach. The attention mechanism can put importance on specific spans of the trace which it learns to be important for a particular workload due to the sparsity property of the attention. Thus the attention mechanism exploits the global properties. Opposed to it, the LSTM-based exploits local properties of a trace due to the autoregressive assumption. The results that both methods show for both the short and the combination of short and long traces

are comparable. The good comparable performance on the short traces results from the ability of both of the models to exploit the locality in the traces, which given the smaller size of the trace does not distinguish from the global properties of the trace. Similarly, the attention model can exploit the whole information existing in the traces. The good performance on the combination of the *long short* traces is due to the over-representation of the short traces in the test dataset.

Fig. 5 depicts the results from the LS2 scenario. It is interesting to observe that the results in this scenario are inline with the previously discussed, with a notable decrease of the recall for both the small and long traces for the LSTM-based approach. This means that the autoregressive model is predicting more of the anomalies as normal. This behaviour can be explained with the fact that the anomalous trace differs from the normal just in its length. The forward blindness of the autoregressive approaches prohibits to infer information from all of the traces implicitly including the information from the length of the trace. Hence, the LSTM method while is still good in modeling of the sequence of spans is shortsighted and it is expected to have worse results especially in scenarios where the size of the traces is much longer. Long traces are more common in a real-world distributed system . Executing of one operation in a micro-service architecture includes invoking of multiple services not necessarily just HTTP calls for the communication between the services. Hence the ability to handle long traces is imperative for a method to be applicable in a real world tracing data. As observed by the results, the strongest point of the attention mechanism is that it can provide good results on long traces since it exploits the information of the whole trace at once. Furthermore, this means that the attention mechanism does not suffer from reduced recall as the LSTM method.

V. Conclusion

This paper addresses the problem of anomaly detection in large-scale distributed systems, as an essential task for their security and reliability. We addressed the problem by introducing a new learning task – masked span prediction for the problem of execution-path anomaly detection from tracing data. The novel definition of the problem allows to include information from the entire trace, directly exploiting the existing service relations. It results in better predictive performance on the problem of structural anomaly detection, especially for the long traces when compared to other existing approaches for tracing data based on LSTM. Empirically, we show that the proposed approach is more robust to small permutation in the normal traces, a scenario frequently occurring in practice. The experiments showed that the method has high performance on experimental testbed data .

The proposed approach opens a new possibility for anomaly detection not just from tracing data, but from other sources that have the notion of a distributed representation of an event e.g., log data. We believe that the method will motivate further group of research in the direction of utilizing the full trace information for anomaly detection.

References

[1] J. Kaldor, J. Mace, M. Bejda, E. Gao, W. Kuropatwa, J. O'Neill, K. W. Ong, B. Schaller, P. Shan, B. Viscomi *et al.*, "Canopy: an end-to-end performance tracing and analysis system," in *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 2017, pp. 34–50.

[2] C. Sridharan, *Distributed Systems Observability: A Guide to Building Robust Systems*. O'Reilly Media, 2018.

[3] A. Gulenko, F. Schmidt, A. Acker, M. Wallschläger, O. Kao, and F. Liu, "Detecting anomalous behavior of black-box services modeled with distance-based online clustering," in *2018 IEEE 11th International Conference on Cloud Computing*. San Francisco, CA, USA: IEEE, 2018, pp. 912–915.

[4] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. New York, NY, USA: Association for Computing Machinery, 2017, p. 1285–1298.

[5] F. Schmidt, S.-P. F., A. Gulenko, M. Wallschläger, A. Acker, and O. Kao, "Unsupervised anomaly event detection for cloud monitoring using online arima," in *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion*. Zurich: IEEE, 2018, pp. 71–76.

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

[7] S. Nedelkoski, J. Cardoso, and O. Kao, "Anomaly detection and classification using distributed tracing and deep learning," in *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. Larnaca, Cyprus: IEEE, 2019, pp. 241–250.

[8] ——, "Anomaly detection from system tracing data using multimodal deep learning," in *2019 IEEE 12th International Conference on Cloud Computing*. Milan, Italy: IEEE, 2019, pp. 179–186.

[9] Y. Yang, L. Wang, J. Gu, and Y. Li, "Transparently capturing request execution path for anomaly detection," 2020.

[10] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys*, vol. 41, 2009.

[11] S. Zhang, Y. Liu, D. Pei, Y. Chen, X. Qu, S. Tao, and Z. Zang, "Rapid and robust impact assessment of software changes in large internet-based services," in *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, 2015, pp. 1–13.

[12] M. M. Moya, M. W. Koch, and L. D. Hostetler, "One-class classifier networks for target recognition applications," *NASA STI/Recon Technical Report N*, vol. 93, 1993.

[13] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–1780, 1997.

[14] J. Chung, K. Kastner, L. Dinh, K. Goel, A. C. Courville, and Y. Bengio, "A recurrent latent variable model for sequential data," in *Advances in neural information processing systems*, 2015, pp. 2980–2988.

[15] P. He, J. Zhu, Z. Zheng, and M. Lyu, "Drain: An online log parsing approach with fixed depth tree." IEEE, 06 2017, pp. 33–40.

[16] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. 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)*. Quebec, Canada: IEEE Press, 2019, pp. 121–130.

[17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*. Red Hook, NY, US: Curran Associates, 2017, pp. 5998–6008.

[18] A. Shrivastwa, S. Sarat, K. Jackson, C. Bunch, E. Sigler, and T. Campbell, *OpenStack: Building a Cloud Environment*. Packt Publishing, 2016.

[19] (2020) Kolla-ansible's documentation. [Online]. Available: https://docs.openstack.org/kolla-ansible/latest/

[20] (2020) Rally documentation. [Online]. Available: https://rally.readthedocs.io/en/latest/

[21] A. Paszke, S. Gross, and et. al., "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035.