Deep Attentive Anomaly Detection for Microservice Systems with Multimodal Time-Series Data

Yufu Chen[1,2], Meng Yan[1,2,*], Dan Yang[2,*], Xiaohong Zhang[1,2], Ziliang Wang[1,2]
[1]Key Laboratory of Dependable Service Computing in Cyber-Physical Society (Chongqing University), Ministry of Education, China
[2]School of Big Data & Software Engineering, Chongqing University, Chongqing, China
Email: {chenyufu, mengy, dyang, xhongz, wangziliang}@cqu.edu.cn

***Abstract*—Software architecture is undergoing a transition from monolithic architectures to microservices to achieve resilience, agility, and scalability in the software life circle. However, microservice architecture is not perfect and suffers from intermittent faults, leading to economic and user losses. Therefore, it is essential to detect anomalies in microservice systems accurately. The key limitation of current approaches lies in a lack of ability to detect multitype anomalies, excessive resource overhead, and requirements of expert knowledge. In this paper, we present a Deep Attentive anomaly detection approach with Multimodal data named DAM. With multimodal fusion, attentive LSTM, and a dynamic threshold selecting algorithm, DAM could detect anomalies accurately and efficiently in an unsupervised manner. We evaluate our approach by injecting six types of anomalies on a widely used microservice system, Train-Ticket. The result shows that DAM could detect multitype anomalies well, with 80.46% F-measure, achieving 16.76% and 29.52% improvement over two state-of-the-art baselines (Donut and DAGMM), respectively.**

***Index Terms*—microservice, anomaly detection, multimodal data, LSTM, attention network**

I. Introduction

There is an increasing number of applications that adopt microservice architecture to replace the traditional monolithic structure in the areas such as IoT (Internet of Things) [1] and Cloud-Native [2]. A large application is decoupled into fine-grained services, and independent programmers could develop them parallelly with unlimited technology stacks. It gives the microservice systems several superior properties, such as CI/CD (Continuous Integration, Continuous Delivery) and language-free.

However, the microservice architecture is not perfect and also suffers from occasional crashes that lead to tremendous losses. A survey reported by Statista recently shows that the average economic loss caused by server outage of microservices for one hour is between $300,000 and $400,000 [3]. Therefore, system anomalies must be detected quickly to ensure microservices are running reliably and with high uptime. Besides, traditional troubleshooting usually requires the maintenance engineers to look through a large number of logs or KPI panels, which could be costly and inefficient. Automatic anomaly detection with a well-trained model could help alleviate this problem.

*Corresponding author.

Anomaly detection for microservice systems is challenging due to the following characteristics [4]: *1) Heterogeneous Multimodal Data:* logs, metrics, and traces are three main kinds of monitoring data in microservice systems that record the runtime status in different ways. It is difficult to design a method to dig out the dispersive information from the heterogeneous data (i.e., texts, numbers, and graphs). *2) Fast Iteration:* microservices are frequently updated to meet customers' requirements (e.g., Netflix updates thousands of times per day [5]). It means that a trained model may be invalid within several hours. It puts forward higher requirements for the training efficiency of anomaly detection models.

Researchers have proposed numerous anomaly detection approaches for microservice systems, including metric-based approaches [6], [7] and log-based approaches [8], [9]. Metric-based approaches detect anomalies by learning the periodic patterns of service KPIs (Key Performance Indicators). And log-based approaches detect anomalies by extracting log templates and pointing out unexpected log events. However, current approaches are limited in the generality of anomaly detection because they use a single type of monitoring data. For instance, faults like asynchronous invocations and API version errors barely affect the metrics. As a result, metric-based approaches cannot detect these faults even though they keep causing service quality degeneration. Similarly, log-based approaches fail to detect faults like resource contentions in metrics. And approaches that use resource-consuming architectures such as GAN (Generative Adversarial Networks) require extensive expert knowledge and take a lot of time to train.

In this paper, we propose a novel approach named DAM. It is an application-agnostic and unsupervised approach designed for container-based microservice environments. The core idea of DAM is to predict the future values of multimodal monitoring data and label samples that deviate from predictions to be anomalies. DAM mainly consists of four parts. First, the data collection, which collects and stores data in chronological order using open-sourced monitoring tools like Prometheus and Logstash. Second, the multimodal fusion, which normalizes and aligns multimodal data (logs and metrics) based on log parsing methods like Drain [10] and standardizing methods like z-score. Third, the predicting model, which captures the temporal and interdimensional dependencies with LSTM

and attention mechanism. Last, the anomaly detection with a dynamic threshold selecting algorithm based on Extreme Value Theory (EVT) [11].

We evaluate our approach on a widely used microservice system, Train-Ticket [12] deployed on Kubernetes. The results show that our approach can effectively detect different kinds of anomalies, with 80.46% F-measure detecting multitype anomalies and 99.53% recall at best, outperforming two state-of-the-art baseline approaches Donut [13] and DAGMM [14].

In conclusion, our contributions could be summarized as follow:

- We propose a Deep Attentive anomaly detection approach with Multimodal data named DAM. DAM grasps a comprehensive picture of the system runtime by fusing multimodal data. With LSTM and attention mechanism, DAM captures the temporal and interdimensional dependencies in multimodal data and predicts future values precisely. And EVT-based dynamic thresholds selecting algorithm makes DAM detect anomalies accurately without any expert knowledge.
- We evaluate our approach on a widely used benchmark system, Train-Ticket. The results show that DAM can detect anomalies effectively in an unsupervised manner, improving Donut and DAGMM by 16.76% and 29.52% in terms of F-measure. Additionally, DAM is more efficient, making it possible to update the model within a short interval.

II. Approach

This section will describe in detail how DAM detects anomalies utilizing both logs and metrics. The overall framework of DAM is illustrated in Fig. 1.

A. Data Collection

To characterize the behaviors of the microservice system, we have to collect some KPIs (e.g., CPU usage, memory usage, and network transmission rate) and log lines to record the historical status of the system and discover the latent relationship across multivariate data. We use Prometheus to collect KPIs with an interval of one second and store them in InfluxDB. Meanwhile, Logs are collected by Logstash and stored in ElasticSearch. We inject six kinds of typical anomalies in microservice systems during data collection, and we record the start and end times of each anomaly. Therefore, logs and KPIs are collected and tagged with service names, timestamps, and labels. However, we only use labels during evaluation, and our model is trained in an unsupervised manner.

B. Multimodal Fusion

We adopt a state-of-the-art log parsing algorithm, Drain, to extract log templates. The template IDs acquired by Drain are consecutively sorted by the frequency of occurrence in descending order. Afterward, template IDs are transferred into template numbers according to their index in the sorted array. It means that log lines printed less frequently will have bigger template numbers. Then we use the maximum value in a one-second time window, which reveals the most uncommon system behavior at this point, to represent the original template number sequence. We do not apply the average value of each window because the anomaly rate of log events is extremely low, and the average value represents the majority of the data.

We align metrics and template numbers by timestamps to deal with the different generation rates between log and metrics (Prometheus samples metrics every second, but the log could print 100 or more lines per second). Whereafter, all dimensions of the aligned data are normalized with z-score standardization to eliminate the influence of different units of measurements. We apply sliding windows of length W (W=10) to get fixed-length sequences which are prerequisites of the LSTM network.

C. Model Building

Since our model is an unsupervised approach, the model is built with the collected raw data without any anomaly labels. DAM aims to capture the temporal patterns and predict future value precisely. Therefore, we apply some widely used network structures in the area of time series forecasting.

LSTM is well known for its formidable ability to prioritize historical information and capture complex temporal dependence between multivariate observations [15]. As a result, LSTM is widespread in domains including speech recognition and time series anomaly detection [16]. Therefore, we apply LSTM to learn the temporal pattern of historical data.

The core idea of the attention mechanism is to distinguish the task-related importance of different timestamps from a sequence by computing a weight vector. With enough research and practice, the attention mechanism has shown its priority in areas like NLP [17] and image classification [18]. Therefore, we apply an attention network to further extract temporal and interdimensional dependencies from multimodal data.

We divide the aligned data into three groups according to the correlations between dimensions. Group one is load metrics consisting of CPU usage, memory usage, and average load index of the service. When a container faces hardware resource contention, these metrics usually fluctuate dramatically at the same time. Group two is traffic metrics, including network transmission/reception in bytes and file read/write in bytes. These metrics rise simultaneously when a flood of users send requests consecutively and drop to zero immediately when the traffic is gone. Group three is log sequences that represent the events that happened during runtime.

Three groups of data are fed into three LSTM networks to learn the temporal and intraclass relation. The final hidden states and final outputs of LSTMs are then concatenated together to get joint information from the learned hidden space. Taking the concatenated final output as both key (K) and value (V) while taking the concatenated final hidden state as query (Q), we make these joint representations go through an attention network to further learn the inter-dimensional and temporal dependencies. Lastly, attention outputs are fed into a two-layer dense network to predict future values. We apply

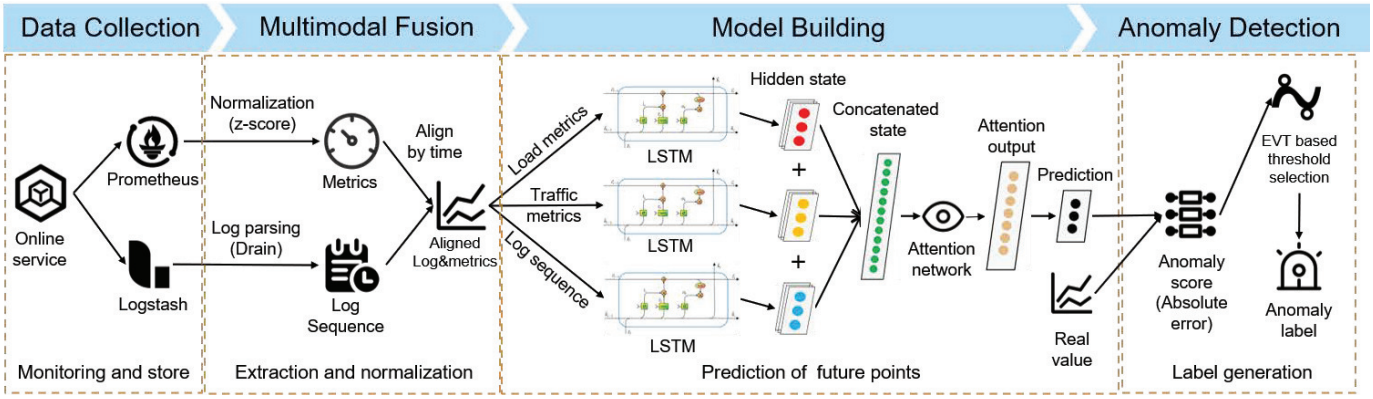

Fig. 1. Overview of proposed anomaly detection method

MSE (Mean Square Error) to be the loss function and the AE (Absolute Error) between the real and predicted values to be the anomaly score.

D. Anomaly Detection

Anomaly detection in microservice systems requires a fast, general, and unsupervised approach to determining whether the observations are anomalous. One simple approach is to make an assumption that the distribution of multivariate data to be diagonal Gaussian $p_\theta(x|z) = \mathcal{N}(\mu_x, \sigma_x^2 I)$, where μ_x and σ_x are the means and standard deviations of independent Gaussian component. But when the data is a stream coming from a rapidly changing environment like microservice systems, such assumptions are no longer safe enough to be made. Meanwhile, manually setting a fixed threshold requires extensive expert knowledge and suffers from a lack of adaptive capacity. To circumvent the hassle of parameter tuning and reduce human impact, we apply the EVT-based threshold selecting method POT (Peaks-Over-Thresholds) to dynamically determine the threshold at a certain point. The key idea of POT is to fit the tail portion of stream data by a Generalized Pareto distribution (GPD) with parameters. A clear view of the theorem can be described as follow:

$$\bar{F}(x) = P(th - x > k \mid X > th) \sim (1 + \frac{\gamma x}{\beta})^{-\frac{1}{\gamma}}$$

Where th is the initial threshold of anomaly scores, γ and β are shape parameters of GPD, X is any value in the stream data $\{X_1, X_2, X_3, \ldots, X_n\}$. The initial threshold is empirically set to a low quantile (e.g., 5%), and the parameters γ and β are estimated by Maximum Likelihood Estimation (MLE). The final threshold Th is then computed by:

$$Th \simeq th - \frac{\hat{\beta}}{\hat{\gamma}}\left(\left(\frac{qn}{N_t}\right)^{-\hat{\gamma}} - 1\right)$$

Where q is the desired probability to observe $X < th$, n is the total number of observations, N_t is the number of peaks, i.e., the number of X_t s.t. $X_t > th$. The algorithm is complicated, but there is only one parameter q that needs to be tuned. In our context, we traverse q from 10^{-3} to 4×10^{-2} with an interval of 10^{-3}, which means the algorithm automatically selects the best parameter from 40 possible candidates without any manual intervention. As a result, the threshold could change dynamically according to the system's behavior.

III. Experimental Setup

In this section, we design experiments to answer the following research questions (RQs):

- How effective is DAM in identifying various kinds of faults in microservice systems?
- What are the impacts of the attention network and multimodal fusion?
- How efficient is DAM compared with the baseline approaches?

Datasets and Setup. We conduct experiments based on a widely-used microservice benchmark system named Train-Ticket. It consists of 41 microservice implemented by Java, Python, and Go. We choose Kubernetes (k8s) to manage containers and configurations.

We conduct our fault injection experiments following the work in [19]. We use ChaosBlade, a chaos engineering tool open-sourced by Alibaba, along with Jmeter, a famous stress test tool, to inject service-level faults and simulate real traffic. We manually modify the source code of Train-Ticket to inject business faults and standardize the log formats. Due to the unmanageable data volume of multiple microservices, we only monitor the most relevant business service during every fault injection experiment (e.g., we monitor *ts-seat-service* when we simulate seat-reservation traffic and inject faults on it). We build models for each business-critical service on account that different services may have different patterns (e.g., load balancing service is more network-intensive and caching service is more memory-intensive). In real systems, we could deploy multiple models on critical nodes simultaneously for comprehensive anomaly detection.

To evaluate the effectiveness of the proposed approach, we inject six types of typical faults into the benchmark system, i.e., CPU contention, memory contention, thread deadlock, network transmission abortion, burst traffic, and business faults. The causes of the faults mentioned above are varied, including server breakdowns that lead to unbalanced traffic and resource contention, fast iterations of code that introduce bugs, changing user behaviors that have the power to overload

TABLE I
DETAILS OF THE DATASET

Samples	LogLines	Anomalies	Anomalies(%)	Anomaly type
79,318	2,076,756	8,058	10.16	6

the services in a short time, et al. By detecting these common faults automatically, DAM could help reduce the manual work of maintenance engineers and improve the robustness of the microservice system.

Metrics are sampled by Prometheus with an interval of one second. Logs are collected and split into lines by Logstash. And we apply Drain to extract log templates. Through repeated experiments and evaluations, we set the parameters of Drain (i.e., the similarity threshold and the depth of leaf nodes) to be 0.5 and 7, respectively. Each fault injecting experiment lasts from one hour to eight hours (3600 to 28800 points with sampling at a one-second interval) in order to get enough training data. And faults (2 to 5 minutes) are injected randomly during data collection. It means that there may be anomalies in the training data, and it is beneficial for verifying the robustness of each approach.

We ended up with a dataset with six types of faults. And we inject multi-type faults (e.g., CPU contention and business fault together) simultaneously for our dataset. The dataset consists of eight dimensions (i.e., CPU usage, memory usage, average load index of a container in 10 seconds, network transmission rate, network reception rate, file write bytes per second, file read bytes per second, and log sequence). In the real world, these dimensions are often used to evaluate service running status and locate root causes after faults occur. Therefore, they have the capacity to provide a comprehensive picture of service health. The basic information of the dataset is reported in Table I. The top 60% of the data is selected as the training data, and the rest is the testing data. We label the data by each fault injection's start and end timestamp. And labels are used only in testing phases.

Baselines. We compare DAM with two state-of-the-art unsupervised baseline approaches: **DAGMM** [13] and **Donut** [14]. We implement our method with PyTorch 1.6 and Python 3.6. For **Donut** and **DAGMM**, we use the code open-sourced on GitHub.

DAGMM is a deep autoencoding network that combines AE (Auto Encoder) and GMM (Gaussian Mixture Model). It adopts a dense network to simulate the EM (Expectation-Maximum) step in a common GMM. Besides, it trains an AE model that aims to reconstruct the multidimensional sequence input precisely. After training, **DAGMM** applies the sum of reconstruction error and sample energy (negative logarithmic probability of the GMM output) as the anomaly score. Lastly, the threshold is set according to the anomaly percentage of the data. However, the anomaly percentage of data is inaccessible in the real world due to a lack of reliable labels. We apply POT to determine the thresholds for **DAGMM** with the same parameters that have been mentioned above in Section II-D.

Donut is a state-of-the-art unsupervised univariate time series anomaly detection appraoch based on VAE (Variational Auto Encoder) and MCMC (Monte Carlo Markov Chain) interpolation. Similar to **DAGMM**, **Donut** trains a model to reconstruct the input sequences and take the reconstruction probability as the anomaly score. Different from the dynamic thresholds applied by us, **Donut** chooses a fixed threshold by traversing all possible values in the range of anomaly scores. To apply **Donut** to multivariate time series anomaly detection, we simply train *M* models for *M-dimensional* data. At time *t*, we compute the average reconstruction probability output by *M* models to be the final anomaly score.

Evaluation Metrics. *FP* is the number of normal samples that are identified as abnormal. *FN* indicates how many abnormal samples are identified as normal. *TP* denotes the number of anomalous samples that are correctly identified. Anomaly detection is a binary classification problem, so precision, recall, and F-measure are usually used as assessment criteria. Precision=$\frac{TP}{TP+FP}$, Recall=$\frac{TP}{TP+FN}$, F-measure=$\frac{2 \cdot Precision \cdot Recall}{Precision+Recall}$. In our context, recall is considered more prominent than precision due to the unbearable consequences of system faults (e.g., economic loss and loss of users). Techniques like alarm aggregation could further handle false alarms, but any omission of fault may be a recipe for disaster.

Inspired by the work in [14], we adjust the anomaly evaluation strategy as follows, if any point in an anomaly segment could be detected as *TP*, we say the segment is detected correctly. In real applications, maintenance engineers do not care about point-wise metrics. If the detection delay is tolerable, faults (corresponding to anomaly segments) can still be handled in time.

IV. RESULTS

In this section, we present our experimental results by answering the RQs mentioned in Section III.

A. RQ1: How effective is DAM in identifying various kinds of faults in microservice systems?

To learn the temporal information in load metrics, traffic metrics, and log sequence separately, we set the input dims of LSTMs in our model to be (3, 4, 1), and the hidden dim of every LSTM to be 32. Corresponding to the input dims of LSTMs, we use 3 three-layers full connected (FC) networks with dims (96, 32, 8, 3), (96, 32, 8, 4) and (96, 32, 8, 1) to predict the future value.

For Donut, we use the recommended parameters illustrated in their paper. For DAGMM, we set the number of assumed Gaussian distributions in the estimation network to be 4 (half of the input dims). The singular matrix that arises during training will immediately interrupt the program. Therefore, we set the λ_{diag}, a parameter in the loss function which aims to eliminate the singular matrix, to be 0.2 (higher than default). The AE part of DAGMM consists of an eight-layers FC network with dims (8, 60, 30, 10, 1) for encoder and (1, 10, 30, 60 ,8) for decoder.

After model building, we traverse POT's risk parameter q from 10^{-3} to 4×10^{-2} with an interval of 10^{-3} to get the q_{best}. We record the average performance among repeated

TABLE II
AVERAGE RESULTS ON THE FAULT INJECTION DATASET

Approaches	Precision(%)	Recall(%)	F-measure(%)
DAGMM	**77.37**	82.05	68.91
Donut	47.27	98.96	62.12
DAM-A	53.15	75.79	62.50
DAM-M	31.57	91.11	40.58
DAM-L	52.41	81.54	56.34
DAM	71.69	**99.53**	**80.46**

[a]DAM-A: without attention.
[b]DAM-M: without metrics.
[c]DAM-L: without logs.

model building and evaluation procedures. Table II shows the experimental results. DAM achieves the best F-measure on our dataset, outperforming the suboptimal baseline by 11.55%. Meanwhile, the recall of DAM is high, which indicates that it could detect almost every anomaly segment in time with a little loss in precision. DAGMM has the best precision, although it is a trade-off between reducing false alarms and detecting all faults. As mentioned above, faults could cause huge economic loss or user dissatisfaction. Therefore, DAGMM is not quite qualified as an anomaly detector in the microservices context. Donut performs relatively worse than the others since it is a univariate approach and could not capture the complex dependencies hidden in multimodal data. Indeed, Donut is more influenced by the noise in dimensions. As a result, Donut acts too sensitive to fluctuations in a single dimension but ignores the system's overall state, which leads to too many FP predictions.

In summary, DAM can detect various kinds of faults in microservice systems effectively. DAM improves Donut and DAGMM by 16.76% and 29.52% in terms of F-measure.

B. RQ2: What are the impacts of the attention network and multimodal fusion?

We build models with/without attention networks to evaluate the impact of attention networks. And the model built without an attention network is denoted as DAM-A. Table II shows the impact of the attention network in terms of anomaly detection accuracy. Proven by our experiments, the attention network could significantly improve the accuracy while consuming little computing resources (training time increases by 6%).

Wrongly predicted points may introduce additional noises to the anomaly scores. Therefore, accurate prediction is the foundation of anomaly detection. Fig. 2 shows the predicted results when we build our model with/without attention network to fit 12 hours of failure-free monitoring data (6 hours of data for training). Due to limited space, we only show three out of eight dimensions with 36-minutes data in the picture. Black, red, and blue lines denote ground truth, predictions with attention, and predictions without attention, respectively. And it is obvious that the attention network contributes a lot to the reduction of prediction error.

To evaluate the effectiveness of multimodal fusion, we train our model without logs or metrics (denoted as DAM-L and DAM-M). As shown in Table II, both models fail to detect anomalies precisely since the multitype faults we inject into the benchmark system cannot be easily detected with a single data source. And it is consistent with the circumstances that happen in a real system — maintenance engineers usually focus on both logs and metrics rather than either one.

TABLE III
TIME EFFICIENCY OF EACH MODEL

Approaches	DAGMM	Donut	DAM
Time Cost (s)	5,545	1,644	1,154

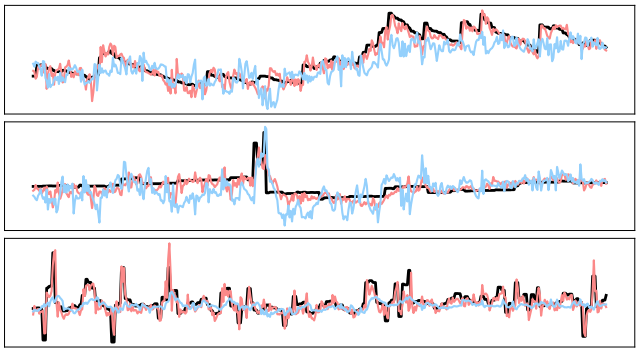

Fig. 2. Effectiveness of attention network. Black lines denote the ground truth, red lines denote predictions with attention network, and blue lines denote predictions without attention network.

In summary, the attention network improves the overall performance of our model while increasing negligible training time. And multimodal fusion gives our model the capacity to detect multitype faults that cannot be revealed with a single data source.

C. RQ3: How efficient is DAM compared with the baseline approaches?

Detecting the online faults in microservice systems requires short training time and adaptive capacity to changing environments. Therefore, time efficiency is of vital importance in our context. If the models cannot update in time, they may fail to detect incoming anomalies due to the rapid iteration of code or the varying user behavior. So we investigate the time efficiency of our approach compared to baselines. We train each model on the same GPU server that is equipped with NVIDIA TITAN V 12GB GPU, 256GB memory, and 48-core Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz. With the same batch size and window length, we train each model using 12 hours of failure-free data for 50 epochs and record their training time. Besides, we train Donut for each dimension parallelly and record the average time cost. As illustrated in Table III, DAGMM takes more time than the others on account of the tanglesome model architecture and complex loss function. And Donut's probability computing step acts as a drag on fast training.

Since the testing time is negligible compared to training time and the time efficiency of the POT algorithm has been investigated clearly in [11]. We do not evaluate the time efficiency during testing.

In summary, DAM is more efficient in terms of training time despite LSTM being used. Compared to baseline approaches,

it is more suitable for a fast-changing environment like microservice systems.

V. Related Works

in the microservices context, by the type of data sources, anomaly detectors can be classified into three main categories, metric-based, log-based, and trace-based models. In this section, we will introduce some related approaches proposed in recent years for each category.

Metric-Based Models. Gulenko et al. [6] enable detecting performance anomalies in multi-service applications. It applies the BIRCH online clustering algorithm to cluster the metrics vectors and learn the normal subspaces. Vectors that do not pertain to these spaces will be classified as outliers. Samir et al. proposed DLA [7]. Based on the Spearman's rank correlation coefficient, the model analyses whether a variation in a newly monitored response time corresponds to an expected fluctuation due to an increase/decrease of users' transactions, or whether it actually denotes an anomaly.

Log-Based Models. Du et al. [8] proposed Deeplog, a deep neural network model utilizing LSTM to model system logs as natural language sequences. Logs are parsed into template sequences and fed into LSTM to predict possible incoming templates. Real values that do not fall into the top-k predictions will be labeled as anomalies. Yin et al. proposed LogC [9]. Similar to Deeplog, LogC extracts templates with Drain and matches components in log lines using regular expressions. Components and templates are fed into two bidirectional LSTM models separately to learn the normal pattern and detect the anomalies.

Trace-Based Models. Liu et al. proposed TraceAnomaly [20]. It defines a novel trace representation called STV (Service Trace Vector) by combining span information with response time. STVs are grouped by time, and a deep Bayesian model is used to reconstruct these vectors. The STVs that cannot be reconstructed well will be labeled as anomalies, and the corresponding spans are considered the root-cause invocations.

VI. Conclusion and future work

This paper proposes DAM, an unsupervised anomaly detection approach utilizing multimodal monitoring data. Firstly, DAM transforms the raw log lines into template sequences and aligns them with metrics by timestamps. Then the multivariate data are divided into three groups according to the correlations between dimensions. With LSTMs and attention networks, the model is built to capture the normal pattern of service runtime and predict future values. Anomaly scores are later acquired by computing the absolute error between predictions and real values. Lastly, POT is used to generate dynamic thresholds and anomaly labels. In the future, we will try to combine DAM with some graph-based methods such as graph convolutional neural networks to extract the inter-service dependencies using traces and locate the root cause service after anomaly detection.

Acknowledgment

This work was supported in part by the National Key Research and Development Project (No. 2021YFB1714200), the Fundamental Research Funds for the Central Universities (No. 2022CDJKYJH001), and the Natural Science Foundation of Chongqing (No. cstc2021jcyj-msxmX0538).

References

[1] B. Butzin, F. Golatowski, and D. Timmermann, "Microservices approach for the internet of things," in *ETFA*. IEEE, 2016, pp. 1–6.

[2] P. Di Francesco, I. Malavolta, and P. Lago, "Research on architecting microservices: Trends, focus, and potential for industrial adoption," in *ICSA*. IEEE, 2017, pp. 21–30.

[3] "Average cost per hour of enterprise server downtime worldwide in 2019." https://www.statista.com/statistics/753938/worldwide-enterprise-server-hourly-downtime-cost/, accessed: 04 January 2022.

[4] L. Wu, J. Tordsson, E. Elmroth, and O. Kao, "Microrca: Root cause localization of performance issues in microservices," in *NOMS*. IEEE, 2020, pp. 1–9.

[5] "Why netflix, amazon, and apple care about microservices." https://www.leanix.net/en/blog/why-netflix-amazon-and-apple-care-about-microservices, accessed: 09 January 2022.

[6] 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 *CLOUD*. IEEE, 2018, pp. 912–915.

[7] A. Samir and C. Pahl, "Dla: Detecting and localizing anomalies in containerized microservice architectures using markov models," in *FiCloud*. IEEE, 2019, pp. 205–213.

[8] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *CCS*, 2017, pp. 1285–1298.

[9] K. Yin, M. Yan, L. Xu, Z. Xu, Z. Li, D. Yang, and X. Zhang, "Improving log-based anomaly detection with component-aware analysis," in *ICSME*. IEEE, 2020, pp. 667–671.

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

[11] A. Siffer, P.-A. Fouque, A. Termier, and C. Largouet, "Anomaly detection in streams with extreme value theory," in *KDD*, 2017, pp. 1067–1075.

[12] X. Zhou, X. Peng, T. Xie, J. Sun, C. Xu, C. Ji, and W. Zhao, "Poster: Benchmarking microservice systems for software engineering research," in *ICSE*. IEEE, 2018, pp. 323–324.

[13] B. Zong, Q. Song, M. R. Min, W. Cheng, C. Lumezanu, D. Cho, and H. Chen, "Deep autoencoding gaussian mixture model for unsupervised anomaly detection," in *ICLR*, 2018.

[14] H. Xu, W. Chen, N. Zhao, Z. Li, J. Bu, Z. Li, Y. Liu, Y. Zhao, D. Pei, Y. Feng *et al.*, "Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications," in *WWW*, 2018, pp. 187–196.

[15] Y. Yu, X. Si, C. Hu, and J. Zhang, "A review of recurrent neural networks: Lstm cells and network architectures," *Neural computation*, vol. 31, no. 7, pp. 1235–1270, 2019.

[16] K. Hundman, V. Constantinou, C. Laporte, I. Colwell, and T. Soderstrom, "Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding," in *KDD*, 2018, pp. 387–395.

[17] D. Hu, "An introductory survey on attention mechanisms in nlp problems," in *Proceedings of SAI Intelligent Systems Conference*. Springer, 2019, pp. 432–448.

[18] F. Wang, M. Jiang, C. Qian, S. Yang, C. Li, H. Zhang, X. Wang, and X. Tang, "Residual attention network for image classification," in *ICPR*, 2017, pp. 3156–3164.

[19] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, W. Li, and D. Ding, "Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study," *TSE*, vol. 47, no. 2, pp. 243–260, 2018.

[20] P. Liu, H. Xu, Q. Ouyang, R. Jiao, Z. Chen, S. Zhang, J. Yang, L. Mo, J. Zeng, W. Xue *et al.*, "Unsupervised detection of microservice trace anomalies through service-level deep bayesian networks," in *ISSRE*. IEEE, 2020, pp. 48–58.