Learning Dependencies in Distributed Cloud Applications to Identify and Localize Anomalies

Dominik Scheinert, Alexander Acker, Lauritz Thamsen, Morgan K. Geldenhuys, and Odej Kao
Technische Universität Berlin, Germany, {firstname.lastname}@tu-berlin.de

***Abstract*—Operation and maintenance of large distributed cloud applications can quickly become unmanageably complex, putting human operators under immense stress when problems occur. Utilizing machine learning for identification and localization of anomalies in such systems supports human experts and enables fast mitigation. However, due to the various inter-dependencies of system components, anomalies do not only affect their origin but propagate through the distributed system. Taking this into account, we present *Arvalus* and its variant *D-Arvalus*, a neural graph transformation method that models system components as nodes and their dependencies and placement as edges to improve the identification and localization of anomalies. Given a series of metric KPIs, our method predicts the most likely system state - either normal or an anomaly class - and performs localization when an anomaly is detected. During our experiments, we simulate a distributed cloud application deployment and synthetically inject anomalies. The evaluation shows the generally good prediction performance of Arvalus and reveals the advantage of D-Arvalus which incorporates information about system component dependencies.**

***Index Terms*—Cloud Computing, Root Cause Analysis, Anomaly Detection, Availability, Graph Neural Networks**

I. Introduction

The evolution of IT systems enables rapid innovation in a variety of fields like medicine, autonomous transportation or manufacturing. Often, applications and services are deployed in cloud computing environments, referred to as distributed cloud applications or microservices [1]. Furthermore, they often are decomposed into lightweight, self-contained and independently deployable services which enables high degrees of isolation, fine-grained scaling and fast adaption of individual components to customer requirements. However, these deployments are reaching sizes of hundreds to thousands of interconnected system components [2]. Such complex distributed systems are hard to operate and human experts are increasingly in need of support and automation.

To guarantee high availability and low latency, inevitably occurring problems must be timely identified and localized. Significant effort was done to research the detection of system anomalies by utilizing machine learning (ML) methods [3], [4], [5]. Thereby, monitoring is employed to collect key performance indicator (KPI) metrics such as network latency or resource utilization from all relevant components. Based on the collected data, models are trained to detect anomalies. Although these methods provide support for the operation of monolithic applications, their use in largely distributed and interconnected environments leads to problems. Occurring anomalies in a single component can propagate through the system. In such cases, employing anomaly detection leads to many alarms from the different affected components, leaving the search for the origin to human operators.

Fault localization - sometimes referred to as root cause analysis (RCA) - in distributed systems has been addressed in several works [1], [6], [7]. However, most of these methods are relying on manual rules, selectively constructed metrics, or assume access to source code, input data or configuration parameters. Another line of work explicitly operates on the dependencies of services, modelling them as correlation or causality graphs [8]. Here, the localization of anomalies is done by backtracking graph edges, resulting in a ranked list of possible locations. Yet, the mining of the graphs from existing data and the backtracking when anomalies occur is difficult. During graph mining, the calculation of KPI inter-dependencies usually has a quadratic computational complexity, posing a limit to the scale of systems. Moreover, backtracking with change-based heuristics on calculated KPI inter-dependencies is not generally applicable to all possible system anomalies.

To overcome these limitations, we propose our data-driven anomaly identification and localization approach *Arvalus* and its variant *D-Arvalus* which utilizes a novel graph convolution (GC) method to model system component inter-dependencies. Meta-information about the system deployment are used to build an initial graph structure. A trainable node feature extraction and edge feature transformer are employed to complement the graph. D-Arvalus applies GC to aggregate node features of neighboring nodes. Finally, anomaly identification and localization is done based on the calculated node features. Instead of heuristics and rules, this method introduces a trainable and therefore general anomaly identification and localization. Our approach is evaluated in a simulated environment. The deployment structure of this environment is based on the deployment layout of real cloud applications. To evaluate our proposed graph convolution method, Arvalus and its variant D-Arvalus are compared.

The remainder of the paper is structured as follows: Section II discusses the related work with regards to RCA in distributed systems. Section III describes our approach and explains the idea of exploiting the dependencies of distributed systems for improved anomaly identification and localization. Section IV presents our experimental setup and our evaluation results. Section V concludes the paper.

II. Related Work

Anomaly detection based on statistics and machine learning is a widely studied field. Ahmed et. al [9] provide a taxonomy of anomaly detection methods based on classification, statistics, information theory, and clustering. Due to the propagation of anomalies in distributed and interdependent systems, the application of anomaly detection methods alone eventually leads to a large number of false alarms. Therefore, the related field of root cause analysis (also referred to as fault localization) receives much interest. In [10] the authors propose a method for fault localization using system traces via supervised model training. Unlike Arvalus, their method requires manually selected features. Likewise, the authors of [11] motivate for the use of a path condition time series algorithm in combination with an adapted random walk algorithm. This not only allows to capture sequential relationships in time series data, but also to incorporate causal relationships, temporal order, and priority information of monitoring data. The drawback of this method is its complexity. With Arvalus we simplify the feature extraction by only considering temporal relationships within extracted slices of the respective time series. Additionally, we learn the characteristics of anomalies whereas their approach uses thresholds on root cause metrics which usually requires expert knowledge to be set correctly. In [6], the authors employ a Random Forest algorithm for anomaly detection and RCA in Virtual Network Functions. Similar to Arvalus, they carry out an analysis individually for each system component as well as an ensemble analysis. The latter only considers the global prediction probabilities and neglects the underlying graph structure of system dependencies.

Arvalus is within a class of approaches specific to distributed cloud systems which models system state in the form of a directed graph to store information and identify dependencies between the various nodes. A certain number of these are concerned with diagnosis after an anomaly has been reported and not the detection of anomalies themselves as is the case with our approach. *MonitorRank* [12] is a real time metric collection system used to perform RCA in service-oriented architectures. It proposes an unsupervised and heuristic (clustering) approach for producing a ranked list of possible root causes which aid monitoring teams in investigations.

Further causality graph-based approaches aim to provide a system for end-to-end anomaly detection and root cause analysis. An approach called *LOUD* [4] trains a model with correct executions only and uses graph centrality algorithms to localize faulty resources in cloud systems. Although the authors claim that exclusively relying on positive training is able to detect and localize exceptions to the steady state, this does however prevent the identification of specific anomalies and therefore reduces the ability to respond to them appropriately. *GRANO* [13] is an enterprise level software framework developed at Ebay consisting of a detection layer, an anomaly graph layer, and an application layer which assists fault resolution teams. This extensible framework employs multiple techniques for detecting and localizing anomalous behaviors for cloud-native distributed data platforms. In contrast, the authors of [8] propose to infer root causes by correlating application performance symptoms with corresponding system resource utilization. Here, anomaly detection is realized using *BIRCH* [14], and RCA is implemented based on an attributed graph that models anomaly propagation across systems.

III. Approach

Our method utilizes KPIs of individual cloud system components like hosts, virtual machines or microservices to identify and localize anomalies. Therefore, we model distributed cloud applications as graphs where each node represents a system component and edges represent dependencies between them. The approach consists of three steps. First, a KPI subseries of a component is transformed into a feature vector for the respective graph node. Second, the feature vectors of neighboring nodes together with meta-information are used to calculate edge weights to realize the graph convolution operation. Third, the feature vector of each node is utilized to identify and localize anomalies. Besides a formal problem definition, the subsequent sections provide detailed explanations of these steps.

A. Preliminaries

KPIs such as tracing or metric data are the basis for detecting and localizing anomalies in distributed cloud applications. When continuously collected over time, they provide an abstract representation of the state of each system component.

Metric data KPIs can be defined as multivariate time series, i.e. a temporally ordered sequence of vectors $S = (S_t \in \mathbb{R}^d : t = 1, 2, \ldots, T)$, where d is the number of KPIs and T defines the last sample time stamp. For $S_b^a = (S_a, S_{a+1}, \ldots, S_b)$, we denote indices a and b with $a \leq b$ and $0 \leq a, b \leq T$ as time series boundaries in order to slice a given series S_T^0 and acquire a subseries S_b^a.

Distributed cloud applications can be modelled as graphs to represent the dependencies between system components. A directed, weighted and attributed graph $G = (V, E)$ with n nodes consists of a set of vertices $V = \{v_1, \ldots, v_n\}$ and a set of edges $E \subseteq \{(v_i, v_j) | v_i, v_j \in V\}$. Each node v_i has a node feature vector $\vec{x}_i \in \mathbb{R}^F$. An edge $e_{ij} \Leftrightarrow (v_i, v_j) \in E$ describes a directed connection between vertex v_i and v_j. Thus, the node v_j is then called a neighbor of node v_i, formally written as $j \in \mathcal{N}(i)$. Each edge e_{ij} is attributed by a vector $\vec{r}_{ij}$, containing meta information about the connection. The adjacency matrix A of a graph G is an $n \times n$ matrix with entries A_{ij} such that $A_{ij} = \vec{r}_{ij}$ if an edge e_{ij} exists, otherwise 0.

B. Node Feature Extraction

The previously presented preliminaries enable us to describe our approach. First, all replicas of the same application service are grouped. We refer to them as service groups. Other than that, no grouping is applied. Next, our method transforms KPI subseries $S_b^a \in \mathbb{R}^{d \times (b-a)}$ of each node into a node feature vector $\vec{x} \in \mathbb{R}^F$. Intuitively, the resulting feature vector can be

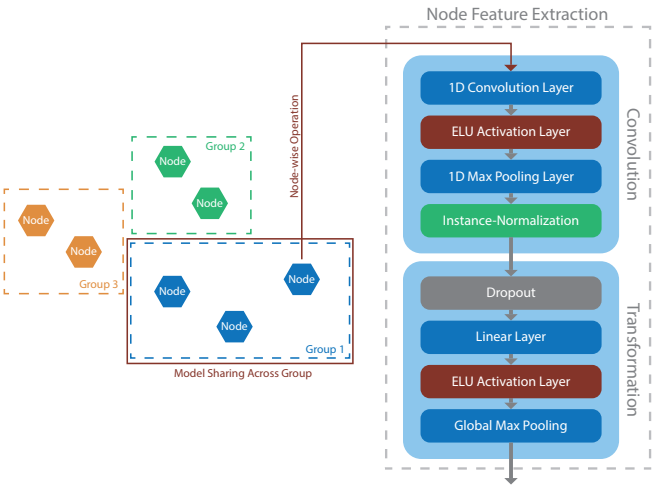

Fig. 1: Node feature extraction consisting of a convolution and transformation block. It transforms KPI subseries of each node into a node feature vector. Hexagons symbolize the extracted KPI subseries.

interpreted as a compact representation of the node state. The steps are further visualized in Figure 1.

Initially, the *Convolution* block aims at the extraction of features from the KPI subseries. A convolution layer with k filters, each of different filter size to capture features at different scale, is applied and the results are concatenated resulting in a feature matrix $X \in \mathbb{R}^{d \times ((b-a) \cdot k)}$. After that we perform an ELU activation, 1D max-pooling and instance normalization. The feature matrix dimensionality is preserved. These steps aim at the extraction of meaningful features from the raw KPI subseries. The *Transformation* block transforms the feature matrix into a node feature vector. During training, a dropout layer is used to mitigate overfitting. A linear transformation is employed to map the feature matrix to a desired dimensionality $X' \in \mathbb{R}^{d \times F}$. Finally, a column-wise global max-pooling results in a compact node representation $\vec{x} \in \mathbb{R}^F$.

These operations are applied for each node. However, every node group (circles of same colour enclosed by the dotted rectangle in Figure 1) shares the same model layer weights.

C. Dependency Model

Distributed cloud system components have diverse inter-dependencies. Virtual machines are hosted on hypervisor nodes, containers which are hosted on virtual machines contain microservices. Thus, the state of a single system component is determined both by local executions and external factors. This is our motivation to model the underlying dependencies of such systems via edge weight learning and graph convolution.

Graph convolutional neural networks (GCNNs) are a class of neural networks which incorporates concepts from graph theory and aim at generalizing the convolution operation to be applied in non Euclidean domains [15]. The convolution operation therefor operates on the neighborhood of each graph node. GCNN methods can be roughly clustered into spectral and spatial methods. In this work, we focus on spacial methods since they are essentially defining the graph convolution in the vertex domain by leveraging the graph structure and aggregating node information from the neighborhoods in a convolutional fashion.

TABLE I: List of defined edge types.

Edge Type	Condition
`Identity`	self-loop, for each node
`Host-Guest`	system i is hosting system j
`Guest-Host`	system i is hosted by system j
`Group`	system i and system j are part of same group
`Dependency`	any other uncovered kind of dependency

We assign the previously calculated node features to the respective nodes and apply an instance normalization, respectively for each graph. After that, Arvalus provides two operation modes, respectively referred to as Arvalus and D-Arvalus and conceptually depicted in Figure 2. It can either use the node features directly to predict and localize system anomalies in the last block (Arvalus) or it utilizes graph convolution to aggregate feature vectors of neighboring nodes, adding this information via a residual connection to the original node features (D-Arvalus).

Inspired by [16], we propose our graph convolution approach designed for distributed cloud applications. First, we construct an initial adjacency matrix A of our graph by fabricating edges e_{ij} between two nodes v_i and v_j using the edge types derived in Table I. As defined in Subsection III-A, directed edges are used. The first three edge types are self-

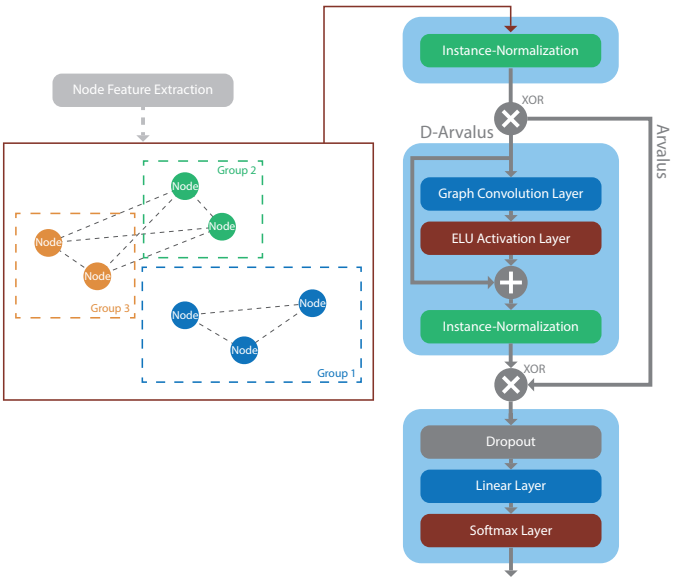

Fig. 2: Dependency model: The graph with its learnable edge weights is used during GC to aggregate features of neighboring nodes. Circles symbolize the computed node feature vectors.

explanatory. The fourth edge type introduces a unique edge tag for each existing component group (e.g. one for $Group1$, one for $Group2$, etc.). To reflect inter-dependencies that are not described by the the first four edge types (e.g. back-end service group depends on the identity service group to authorize access to certain database entries), the fifth edge type is used. Whenever dependencies between two service groups exist, we use a tag for each direction, e.g. $Group1 \rightarrow Group2$ and $Group2 \rightarrow Group1$. We argue that edges between graph nodes can be inferred from information about the respective deployment itself as presented in [17]. Next, each edge is tagged with their respective edge type, as described above. We apply a one-hot-encoding of the tags and attach them to the respective edges as edge attribute vectors. For z unique tags, this results in edge attribute vectors $\vec{r}_{ij} \in \mathbb{N}_0^z$.

Before applying the graph convolution, we transform each edge attribute vector into a scalar weight value. More precisely, we formulate a trainable transformation function f that uses the node features of two neighbor nodes $\vec{x}_i$ and $\vec{x}_j$ combined with the one-hot encoded edge vector $\vec{r}_{ij}$. Applying an ELU activation and posterior normalization results in an adjusted adjacency matrix $\tilde{A}$ with

$$\tilde{A}_{ij} = \frac{\exp\big(\mathrm{ELU}\big(f(\vec{x}_i, \vec{r}_{ij}, \vec{x}_j)\big)\big)}{\sum_{k \in \mathcal{N}(i)} \exp\big(\mathrm{ELU}\big(f(\vec{x}_i, \vec{r}_{ik}, \vec{x}_k)\big)\big)} \tag{1}$$

where $f : \mathbb{R}^{2F+z} \rightarrow \mathbb{R}$. We implement f as a linear neural network. The output of f is subject to a non-linear activation function and subsequently normalized using softmax. This transformation of A allows for the learning of weights in the range $(0, 1)$. Since our method identifies and localizes anomalies, an edge weight e_{ij} can be intuitively interpreted as the importance of a node v_j to identify and localize an anomaly in node v_i due to the fact that the node vector $\vec{x}_j$ is scaled by the edge weight during convolution. Thus, the actual convolution operation is defined as

$$\vec{x}'_i = \sum_{j \in \mathcal{N}(i)} \tilde{A}_{ij} \cdot \vec{x}_j. \tag{2}$$

This spatial convolution allows for information transfer between neighboring nodes. After a subsequent ELU activation and residual connection, a final instance normalization is applied.

D. Identification and Localization of Anomalies

The calculated node feature vectors of the graph are used for the final anomaly identification and localization block. This is visualized in the last block of Figure 2. During training, another dropout layer is used to mitigate overfitting. After that, a linear transformation maps from the node feature dimension to the output dimension of the model, where the output layer size equals the number of anomaly classes and the normal state class. This effectively leads to a scalar value for each class. A subsequent softmax activation calculates a probability distribution over all classes, indicating the classification probability for each anomaly. By investigating all nodes at the same time, we effectively combine anomaly identification and localization.

TABLE II: List of artificial states (normal, anomalies), their parametrization and adversaries.

State	*Standard*	*Miscellaneous*	*Adversary*
Anomaly 1	$\mu = 0, \sigma = 0.1$	$\mu = 0, \sigma = 0.17$	Anomaly 2
Normal	$\mu = 0, \sigma = 0.2$	-	-
Anomaly 2	$\mu = 0, \sigma = 0.3$	$\mu = 0, \sigma = 0.23$	Anomaly 1

IV. Evaluation

During evaluation we want to investigate the effect of modeling neighborhood inter-dependencies of cloud application components with graph convolution and its ability to identify and localize anomalies. To evaluate our approach we simulate two cloud applications hosted within virtual machines of a cloud system and synthetically inject anomalies. The initial graph represents the structure of two real cloud-hosted applications, a virtual IMS[1] and a RTMP content streaming service (CS)[2], as well as their deployment on cloud virtual machines. The KPIs of each system component are realized as samples drawn from differently parametrized distributions. In our synthetic setup, this leads to 51 simulated system components for which we synthesize 10 KPIs each. Subsequently, we simulate the injection of two anomaly types into the KPIs in order to obtain labeled data. A summary of all system component states is summarized in Table II. The absolute value of each drawn sample is used. The anomaly is injected into exactly one randomly selected KPI of a target component i. Furthermore, there is no temporal overlapping of injections, meaning that anomalies are expected to not run simultaneously. Every anomaly is injected five times into each application service. Taking into account Table II and selecting a random subset of $3 \leq k \leq 7$ neighbors of a target system component i, for injection we randomly choose from three identified scenarios with probabilities 70%, 20% and 10%:

- `Local`: While system component i experiences the anomaly in its *standard* form, all other nodes are normal.
- `Neighborhood`: While the anomaly injected into system component i reveals *miscellaneous* characteristics, its respective k neighbors are subject to an injection of the same anomaly in *standard* form.
- `Adversary`: While injecting an anomaly into system component i, its respective k neighbors are subject to an injection of the corresponding *adversary* anomaly. Both anomalies are in their respective *standard* form.

This information is used used for labelling the components. During identification and localization, the respective models will be trained to predict the node labels.

A. Training Setup

In the following, we will describe the different aspects of our training setup. First, we assign IDs to each injected

[1]https://www.projectclearwater.org

[2]https://github.com/arut/nginx-rtmp-module

anomaly. We use a sliding window of size 20 and stride 20 on the simulated multivariate time series, slicing the dataset into subseries. The subseries of all components are grouped to have identical logical start and end times. We assign respective anomaly labels only if all samples in a series are drawn from the anomaly distribution. Otherwise they are labeled as normal. We employ a Leave-One-Group-Out (LOGO) cross-validator based on the assigned injection IDs. Since every anomaly was injected five times into each application service, this results in five LOGO groups. For each KPI, its extracted subseries are preprocessed by rescaling to the range $(0, 1)$ with min-max normalization. The min and max boundaries are determined per service group within the training dataset.

We parametrize our model as follows. For the node feature extraction, we employ three convolution filters for each service group with filter sizes $\{3, 5, 7\}$. By setting padding accordingly, the filters are configured to produce an output of the same length as the input subseries, i.e. 20 samples. The same applies to the subsequent max-pooling operation, where a window of size 3 is used. With regards to biases, all linear layers in our architecture except the last one waive a learnable bias. The same applies to the convolution layers. The graph convolution layer of D-Arvalus introduces a limited number of additional parameters, dependent on the number of unique edge attribute vectors and the desired hidden model dimension. For the model training, the improved softmax cross entropy loss function [18] with $\gamma = 2$ is used to down-weight well-classified samples, such that more emphasis is put on the correct classification of hard samples. Further parameters are summarized in Table III.

TABLE III: Overview of model

Aspect	Configuration
Optimizer Training	Adam, lr $= 10^{-2}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ Epochs = 100, batch size = 64
Model	In-Dim. = 20, Hidden-Dim. = 32, Out-Dim. = 3 Weight initialization: Xavier #Parameters: Arvalus (46.539), D-Arvalus (46.651)
Other	weight decay = 10^{-5}; Dropout (probability=50%) Instance normalization (without γ and β)

B. Anomaly Identification

To identify anomalies, classification on graph node features with (D-Arvalus) and without (Arvalus) graph convolution is applied. We investigate accuracy, precision, recall and F1 scores. For both, Arvalus and D-Arvalus, the validation result with the highest macro F1 score was selected for each split. The results are depicted in Figure 3 and Figure 4, where we present the average scores over all splits with corresponding standard deviation. Figure 3 shows in general small standard deviation across splits. While the differences in accuracy appear insignificantly, the remaining evaluation scores are in general higher for D-Arvalus. For anomaly detection, which is a binary classification task, it achieves an average macro

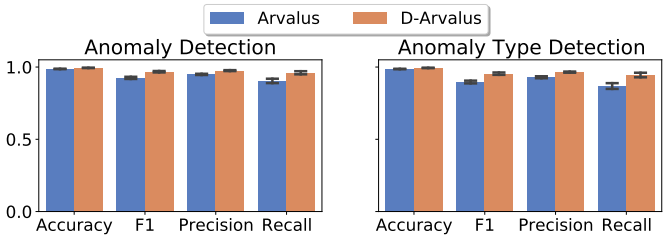

Fig. 3: Identification of Anomalies via classification based on graph node features with (D-Arvalus) and without (Arvalus) graph convolution.

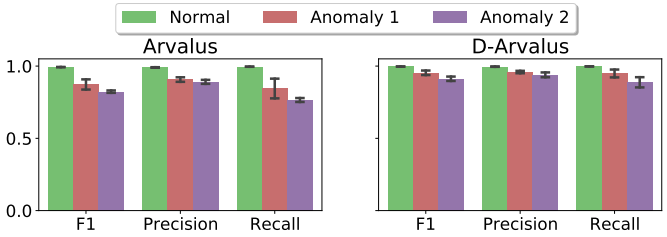

Fig. 4: Insights into Detection of Anomaly Types: D-Arvalus allows for improved detection of anomaly classes

F1 score of 0.967 compared to 0.925 of Arvalus. For anomaly identification, which is a multiclass classification task, a bigger gap between the F1 scores can be observed, i.e. 0.954 compared to 0.896. Due to the imbalance of classes in our scenario, we further unravel the results per class for more insights. Figure 4 demonstrates that on average, D-Arvalus outperforms Arvalus in all evaluation metrics for all classes. Furthermore, the results are less volatile, as indicated by the small standard deviation across splits. While high scores are to be expected for the class representing the normal state, a performance increase for the count-wise underrepresented anomaly classes can be observed as well. From that, we conclude that learning the importance of dependencies between interconnected system components via our GC increases the evaluation scores in this evaluation setup.

C. Anomaly Localization

To quantify the model performance on the task of anomaly localization based on a set of anomalies $|A|$ we use an adapted versions of the $PR@k$ and MAP metrics [8]. $PR@k$ defines the probability that the real anomaly location is among k locations identified as potential locations. A high $PR@k$ score represents the algorithm's ability to correctly localize the anomaly in the system. Smaller values of k require the algorithm to be more precise to achieve high scores. In practise, k can be mapped to the worst case number of manual checks a human expert needs to perform in order to verify the localization of the anomaly. Given L as a set of predicted locations, $PR@k$ is defined as

$$PR@k = \frac{1}{|A|} \sum_{a \in A} \begin{cases} 1, & \text{if } |L| \leq k \\ 0, & \text{otherwise} \end{cases}. \tag{3}$$

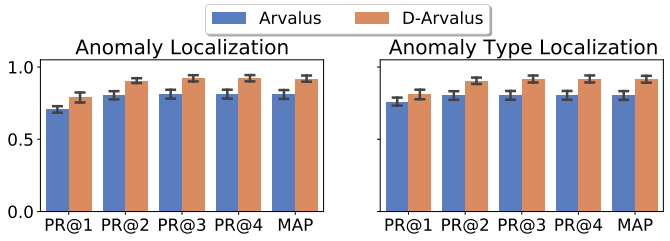

Fig. 5: Localization of Anomalies: Higher localization precision due to D-Arvalus, as indicated by MAP metric

Consequently, for N system components, MAP is defined as

$$MAP = \frac{1}{N} \sum_{1 \leq j \leq N} PR@j \tag{4}$$

Again, we compare the scores of Arvalus and D-Arvalus. The anomaly localization results are presented in Figure 5. Anomaly localization measures the ability to localize the anomaly independent of of its concrete class label. Anomaly type localization refers to the ability to localize the anomaly and predict the correct anomaly type. Obviously, it can be observed that D-Arvalus outperforms Arvalus. Its scores are higher even for small values of k. This performance gap is constant, as indicated by the MAP results. On average, the MAP score of D-Arvalus is 11% higher compared to Arvalus, and this can be shown both for anomaly localization and anomaly type localization. We further evaluate our previously defined scenarios and examine the accuracy of correctly localizing respective target nodes. Arvalus is on average able to correctly localize 97.6% of `Local`, 97.7% of `Adversary`, and 56.6% of `Neighborhood` target nodes. In contrast, D-Arvalus achieves results of 99.9%, 99.0% and 97.6%. This demonstrates increased anomaly localization capabilities when incorporating neighborhood information.

In general, our approach leads to promising results in our simulated scenario, which motivates for future experiments using realistic distributed system testbeds [19].

V. Conclusion

In this paper we presented *Arvalus*, a data-driven and graph-based approach for identifying and localizing anomalies in distributed cloud applications. It allows to aggregate KPIs of interdependent application services by applying graph convolution (D-Arvalus) eventually resulting in node graph features that improve the identification and localization results. Therefore, we developed a graph convolution method to model cloud application inter-dependencies. Our evaluation experiments on a set of simulated cloud application KPIs shows that applying graph convolution consistently leads to better identification (Macro F1 score of 0.95 compared to 0.9) and localization (D-Arvalus has 11% higher MAP than Arvalus) results. Thus, the graph convolution of D-Arvalus has the potential to improve many existing solutions. In the future, we are planning to evaluate our method with data from real-world cloud deployments.

Data Availability

We are pleased to make further technical details, experiment and evaluation scripts, as well as the used data publicly available[3] to the community.

Acknowledgments

This work has been supported through grants by the German Ministry for Education and Research (BMBF) as BIFOLD (funding mark 01IS18025A).

References

[1] P. Wang, J. Xu, M. Ma, W. Lin, D. Pan, Y. Wang, and P. Chen, "Cloudranger: Root cause identification for cloud native systems," in *CCGRID*. IEEE, 2018.

[2] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: yesterday, today, and tomorrow," in *Present and ulterior software engineering*. Springer, 2017.

[3] A. Gulenko, M. Wallschläger, F. Schmidt, O. Kao, and F. Liu, "A system architecture for real-time anomaly detection in large-scale nfv systems," *Procedia Computer Science*, 2016.

[4] L. Mariani, C. Monni, M. Pezzé, O. Riganelli, and R. Xin, "Localizing faults in cloud systems," in *ICST*. IEEE, 2018.

[5] S. Nedelkoski, J. Cardoso, and O. Kao, "Anomaly detection from system tracing data using multimodal deep learning," in *CLOUD*. IEEE, 2019.

[6] C. Sauvanaud, K. Lazri, M. Kaâniche, and K. Kanoun, "Anomaly detection and root cause localization in virtual network functions," in *ISSRE*. IEEE, 2016.

[7] Z. Li, C. Luo, Y. Zhao, Y. Sun, K. Sui, X. Wang, D. Liu, X. Jin, Q. Wang, and D. Pei, "Generic and robust localization of multi-dimensional root causes," in *ISSRE*. IEEE, 2019.

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

[9] M. Ahmed, A. Mahmood, and J. Hu, "A survey of network anomaly detection techniques," *J. Netw. Comput. Appl.*, vol. 60, pp. 19–31, 2016.

[10] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, D. Liu, Q. Xiang, and C. He, "Latent error prediction and fault localization for microservice applications by learning from system trace logs," in *ESEC/FSE*. ACM, 2019.

[11] Y. Meng, S. Zhang, Y. Sun, R. Zhang, Z. Hu, Y. Zhang, C. Jia, Z. Wang, and D. Pei, "Localizing failure root causes in a microservice through causality inference," in *IWQoS*. IEEE, 2020.

[12] M. Kim, R. Sumbaly, and S. Shah, "Root cause detection in a service-oriented architecture," *SIGMETRICS Perform. Eval. Rev.*, vol. 41, no. 1, p. 93–104, Jun. 2013.

[13] H. Wang, P. Nguyen, J. Li, S. Kopru, G. Zhang, S. Katariya, and S. Ben-Romdhane, "Grano: Interactive graph-based root cause analysis for cloud-native distributed data platform," *Proceedings of the VLDB Endowment*, vol. 12, no. 12, pp. 1942–1945, 2019.

[14] 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.

[15] S. Zhang, H. Tong, J. Xu, and R. Maciejewski, "Graph convolutional networks: a comprehensive review," *Computational Social Networks*, vol. 6, no. 1, p. 11, 2019.

[16] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *ICLR*. OpenReview.net, 2018.

[17] L. Wu, J. Tordsson, A. Acker, and O. Kao, "MicroRAS: Automatic Recovery in the Absence of Historical Failure Data for Microservice Systems," in *UCC*. IEEE, 2020.

[18] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 2, 2020.

[19] S. Nedelkoski, J. Bogatinovski, A. K. Mandapati, S. Becker, J. Cardoso, and O. Kao, "Multi-source distributed system data for ai-powered analytics," in *ESOCC*. Springer, 2020.

[3] https://doi.org/10.5281/zenodo.4589255