HAL
open science

Performance Diagnosis in Cloud Microservices using Deep Learning

Li Wu, Jasmin Bogatinovski, Sasho Nedelkoski, Johan Tordsson, Odej Kao

To cite this version:

Li Wu, Jasmin Bogatinovski, Sasho Nedelkoski, Johan Tordsson, Odej Kao. Performance Diagnosis in Cloud Microservices using Deep Learning. AIOPS 2020 - International Workshop on Artificial Intelligence for IT Operations, Dec 2020, Dubai, United Arab Emirates. hal-02948735

HAL Id: hal-02948735

https://inria.hal.science/hal-02948735v1

Submitted on 25 Sep 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Performance Diagnosis in Cloud Microservices using Deep Learning

Li Wu[1,2], Jasmin Bogatinovski[2], Sasho Nedelkoski[2], Johan Tordsson[1,3], and Odej Kao[2]

[1] Elastisys AB, Umeå, Sweden
[2] Distributed and Operating Systems Group, TU Berlin, Berlin, Germany
[3] Department of Computing Science, Umeå University, Umeå, Sweden
{li.wu, johan.tordsson}@elastisys.com
{jasmin.bogatinovski, nedelkoski, odej.kao}@tu-berlin.de

Abstract. Microservice architectures are increasingly adopted to design large-scale applications. However, the highly distributed nature and complex dependencies of microservices complicate automatic performance diagnosis and make it challenging to guarantee service level agreements (SLAs). In particular, identifying the culprits of a microservice performance issue is extremely difficult as the set of potential root causes is large and issues can manifest themselves in complex ways. This paper presents an application-agnostic system to locate the culprits for microservice performance degradation with fine granularity, including not only the anomalous service from which the performance issue originates but also the culprit metrics that correlate to the service abnormality. Our method first finds potential culprit services by constructing a service dependency graph and next applies an autoencoder to identify abnormal service metrics based on a ranked list of reconstruction errors. Our experimental evaluation based on injection of performance anomalies to a microservice benchmark deployed in the cloud shows that our system achieves a good diagnosis result, with 92% precision in locating culprit service and 85.5% precision in locating culprit metrics.

Keywords: Performance diagnosis · root cause analysis · microservices · cloud computing · autoencoder .

1 Introduction

The microservice architecture design paradigm is becoming a popular choice to design modern large-scale applications [3]. Its main benefits include accelerated development and deployment, simplified fault debugging and recovery, and producing a rich software development technique stacks. With microservices, monolithic application can be decomposed into (up to hundreds of) single-concerned, loosely-coupled services that can be developed and deployed independently [12].

As microservices deployed on cloud platforms are highly-distributed across multiple hosts and dependent on inter-communicating services, they are prone to performance anomalies due to the external or internal issues. Outside factors

include resource contention and hardware failure or other problems e.g., software bugs. To guarantee the promised service level agreements (SLAs), it is crucial to timely pinpoint the root cause of performance problems. Further, to make appropriate decisions, the diagnosis can provide some insights to the operators such as where the bottleneck is located, and suggest mitigation actions. However, it is considerably challenging to conduct performance diagnosis in microservices due to the large scale and complexity of microservices and the wide range of potential causes.

Microservices running in the cloud have monitoring capabilities that capture various application-specific and system-level metrics, and can thus understand the current system state and be used to detect service level objective (SLO) violations. These monitored metrics are externalization of the internal state of the system. Metrics can be used to infer the failure in the system and we thus refer to them as *symptoms* in anomaly scenarios. However, because of the large number of metrics exposed by microservices (e.g., Uber reports 500 million metrics exposed [14]) and that faults tend to propagate among microservices, many metrics can be detected as anomalous, in addition to the true root cause. These additional anomalous metrics make it difficult to diagnose performance issues manually (research problems are stated in Section 2).

To automate performance diagnosis in microservices effectively and efficiently, different approaches have been developed (briefly discussed in Section 6). However, they are limited by either coarse granularity or considerable overhead. Regarding granularity, some work focus on locating the service that initiates the performance degradation instead of identifying the real cause with fine granularity [8, 15, 9] (e.g., resource bottleneck or a configuration mistake). We argue that the coarse-grained fault location is insufficient as it cannot give us more details to the root causes, which makes it difficult to recover the system timely. As for considerable overhead, to narrow down the fault location, several systems can pinpoint the root causes with fine granularity. But they need to instrument application source code or runtime systems, which brings considerable overhead to a production system and/or slows down development [4].

In this paper, we adopt a two-stage approach for anomaly detection and root cause analysis (system overview is described in Section 3). In the first stage, we model the service that causes the failure following a graph-based approach [16]. This allows us to pinpoint the potential faulty service that initiates the performance degradation, by identifying the root cause (anomalous metric) that contributes to the performance degradation of the faulty service. The second stage, inference of the potential failure, is based on the assumption that the most important symptoms for the faulty behaviour have a significant deviation from their values during normal operation. Measuring the individual contribution to each of the symptoms at any time point, that leads to the discrepancy between observed and normal behaviour, allows for localization of the most likely symptoms that reflect the fault. Given this assumption, we aim to model the symptoms values under normal system behaviour. To do this we adopt an autoencoder method (Section 4). Assuming a Gaussian distribution of the reconstruction error from

the autoencoder, we can suggest interesting variations in the data points. We then decompose the reconstruction error assuming each of the symptoms as equally important. Further domain and system knowledge can be adopted to re-weight the error contribution. To deduce the set of possible symptoms as a preference rule for the creation of the set of possible failure we consider the symptom with a maximal contribution to the reconstruction error. We evaluate our method in a microservice benchmark named Sock-shop[4], running in a Kubernetes cluster in Google Cloud Engine (GCE)[5], by injecting two types of performance issues (CPU hog and memory leak) into different microservices. The results show that our system can identify the culprit services and metrics well, with 92% and 85.5% in precision separately (Section 5).

2 Problem Description

Given a collection of loosely coupled microservices S, we collect the relevant performance metrics over time for each service $s \in S$. We use $m^{(s,t)}$ to denote the metrics for service s at time t. Furthermore, $m_i^{(s,t)}$ denotes the individual metric (e.g., response time, container cpu utilization, etc.) for service s, collected at time t.

Based on above definition, the performance diagnosis problem is formulated as follows: given metrics m of a cloud microservice, assuming anomalies are detected from metric m_i of a set of services s_a at time t, where i is the index of response time, how can we identify the culprit metrics that cause the anomaly? Furthermore, we break down the research problem as following two sub-problems:

1. How to pinpoint the culprit service s_{rc} that initiates the performance degradation in microservices?;
2. Given the culprit service s_{rc}, how to pinpoint the culprit metric m_{rc} that contributes to its abnormality?

3 System Overview

To address the culprit services and metrics diagnosis problems, we propose a performance diagnosis system shown in Fig. 1. In overall, there are four components in our system, namely data collection, anomaly detection, culprit service localization (CSL) and culprit metric localization (CML). Firstly, we collect metrics from multiple data resources in the microservices, including run-time operating system, the application and the network. In particular, we continuously monitor the response times between all pairs of communicating services. Once the anomaly detection module identifies long response times from services, it triggers the system to localize the culprit service that the anomaly originates from.

[4]Sock-shop - `https://microservices-demo.github.io/`

[5]Google Cloud Engine - `https://cloud.google.com/compute/`

After the culprit service localization, it returns a list of potential culprit services, sorted by probability of being the source of the anomaly. Next, for each potential culprit service, our method identifies the anomalous metrics which contribute to the service abnormality. Finally, it outputs a list of (service, metrics list) pairs, for the possible culprit service, and metrics, respectively. With the help of this list, cloud operators can narrow down the causes and reduce the time and complexity to get the real cause.

3.1 Data Collection

We collect data from multiple data sources, including the application, the operating system and the network, in order to provide culprits for performance issues caused by diverse root causes, such as software bugs, hardware issues, resource contention, etc. Our system is designed to be application-agnostic, requiring no instrumentation to the application to get the data. Instead, we collect the metrics that reported by the application and the run-time system themselves.

3.2 Anomaly Detection

In the system, we detect the performance anomaly on the response times between two interactive services (collected by service mesh) using a unsupervised learning method: BIRCH clustering [6]. When a response time deviates from their normal status, it is detected as an anomaly and trigger the subsequent performance diagnosis procedures. Note that, due to the complex dependency among services and the properties of fault propagation, multiple anomalies could be also detected from services that have no issue.

3.3 Culprit Service Localization (CSL)

After anomalies are detected, the culprit service localization is triggered to identify the faulty service that initiates the anomalies. To get the faulty services, we use the method proposed by Wu, L., et al. [16]. First, it constructs an attributed graph to capture the anomaly propagation among services through not only the service call paths but also the co-located machines. Next, it extracts an anomalous subgraph based on detected anomalous services to narrow down the root cause analysis scope from the large number of microservices. Finally, it ranks the faulty services based on the personalized PageRank, where it correlates the anomalous symptoms in response times with relevant resource utilization in container and system levels to calculate the transition probability matrix and Personalized PageRank vector. There are two parameters for this method that need tuning: the anomaly detection threshold and the detection confidence. For the detail of the method, please refer to [16].

With the identified faulty services, we further identify the culprit metrics that make the service abnormal, which is detailed in Section 4.

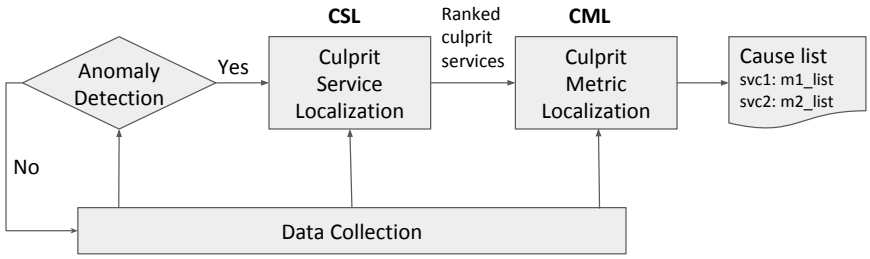

Fig. 1. Overview of the proposed performance diagnosis system.

4 Culprit Metric Localization (CML)

The underlying assumption of our culprit metric localization of the root cause lies in the observation that the underlying symptoms for the faulty behaviour differ from their expected values during normal operation. For example, when there is an anomaly of type "memory leak" it is expected that the memory in the service increases drastically, as compared to the normal operation. In the most general case, it is not known in advance which metric is contributing the most and it is the most relevant for the underlying type of fault in an arbitrary service. Besides, there may exist various inter-relationships between the observed metrics that manifest differently in normal or abnormal scenarios. Successful modelling of this information may improve the anomaly detection procedure and also better pinpoint the potential cause for the anomaly. For example in "CPU hog" we experience not only CPU increase but also a slight memory increase. Thus, some inter-metric relationships may not manifest themselves in same way during anomalies as normal operation.

To tackle these challenges we adopt the autoencoder architecture. An autoencoder is an approach that fits naturally under stressed conditions. The first advantage of the method is that one can add an arbitrary number of input metrics. Thus it can include many potential symptoms as potential faults to be considered at once. The second advantage is that it can correlate arbitrary relationships within the observed metric data with various complexity based on the depth and applied nonlinearities.

An autoencoder [5] is a neural network architecture that learns a mapping from the input to itself. It is composed of an encoder-decoder structure of at least 3 layers: input, hidden and output layer. The encoder provides a mapping from the input to some intermediate (usually lower-dimensional) representation, while the decoder provides an inverse mapping from the intermediate representation back to the input, Thus the cost function being optimized is given as in:

$$\mathcal{L}(X, X) = ||\phi(X) - UU^T\phi(X)||_2^2 \tag{1}$$

where U can be seen as weights of the encoder-decoder structure learned using the backpropagation learning algorithm. While there exist various ways how the mapping from one instance to another can be done, especially interesting is the

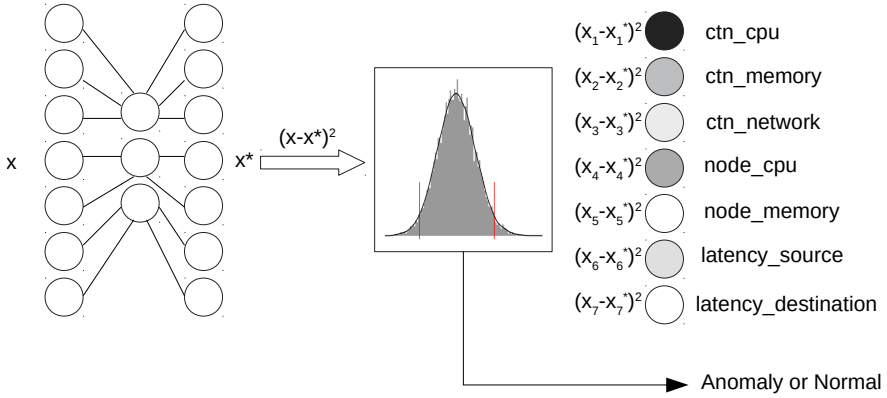

Fig. 2. The inner diagram of the culprit metric localization for anomaly detection and root cause inference. The Gaussian block produces decision that a point is an anomaly if it is below a certain probability threshold. The circle with the greatest intensity of black contributes the most to the error and is pointed as an root cause symptom.

mapping when the hidden layer is of reduced size. This allows to compress the information from the input and enforce it to learn various dependencies. During the training procedure, the parameters of the autoencoder are trained using just normal data from the metrics. This allows us to learn the normal behaviour of the system. In our approach, we further penalize the autoencoder to enforce sparse weights and discourage propagation of information that is not relevant via the L_1 regularization technique. This acts in discouraging the modeling of non-relevant dependancies between the metrics.

Fig. 2 depicts the overall culprit metric localization block. The approach consists of three units: the autoencoder, anomaly detection and root-cause localization part. The root-cause localization part produces an ordered list of most likely cause given the current values of the input metrics. There are two phases of operation: the offline and online phase. During the offline phase, the parameters of the autoencoder and the gaussian distribution part are tuned. During the online phase, the input data is presented to the method one point at the time. The input is propagated through the autoencoder and the anomaly detection part. The output of the latter is propagated to the root-cause localization part that outputs the most likely root-cause.

After training the autoencoder, the second step is to learn the parameters of a Gaussian distribution of the reconstruction error. The underlying assumption is that the data points that are very similar (e.g., lie within 3σ (standard deviations) from the mean) are likely to come from a Gaussian distribution with the estimated parameters. As such they do not violate the expected values for metrics. The parameters of the distribution are calculated on a held-out validation set from normal data points. As each of the services in the system is run in a separate container and we have the metric for each of them, the autoencoder

can be utilized as an additional anomaly detection method on a service level. As the culprit service localization module exploits the global dependency graph of the overall architecture, it suffers from the eminent noise propagated among the services. While unable to exploit the structure of the architecture, the locality property of the autoencoder can be used to fine-tune the results from the culprit service localization module. Thus, with a combination of the strengths of the two methods, we can produce better results for anomaly detection.

The decision for the most likely symptom is done such that we calculate the individual errors between the input and the corresponding reconstructed output. As the autoencoder is constrained to learn normal state, we hypothesize change of the underlying symptom when an anomaly arises to occur. Hence, for a given anomaly as a most likely cause, we report the symptom that contributes to the final error the most.

5 Experimental Evaluation

In this section, we present the experimental setup and evaluate the performance of our system in identifying the culprit metrics and services.

5.1 Testbed and Evaluation Metrics

To evaluate our system, we set up a testbed on Google Cloud Engine [5], where we run the Sock-shop [4] microservice benchmark consisting of seven microservices in a Kubernetes cluster, and the monitoring infrastructures, including the Istio service mesh[6], node-exporter[7], Cadvisor[8], Prometheus[9]. Each worker node in the cluster has 4 virtual CPUs, 15 GB of memory with Container-Optimized OS. We also developed a workload generator to send requests to different services.

To inject the performance issues in microservices, we customize the Docker images of the services by installing the fault injection tools. We inject two types of faults: CPU hog and memory leak, by exhausting the resource CPU and memory in the container, with stress-ng[10], into four different microservices. For each anomaly, we repeated the experiments 6 times in the duration of at least 3 minutes. To train the autoencoder, we collect data of 2 hours in normal status.

To quantify the performance of our system, we use the following two metrics:

- *Precision at top k* denotes the probability that the root causes are included in the top k of the results. For a set of anomalies A, $PR@k$ is defined as:

$$PR@k = \frac{1}{|A|} \sum_{a \in A} \frac{\sum_{i<k}(R[i] \in v_{rc})}{(min(k, |v_{rc}|))} \tag{2}$$

  where $R[i]$ is the rank of each cause and v_{rc} is the set of root causes.

[6]Istio - `https://istio.io/`
[7]Node-exporter - `https://github.com/prometheus/node_exporter`
[8]Cadvisor - `https://github.com/google/cadvisor`
[9]Prometheus - `https://prometheus.io/`
[10]stress-ng - `https://kernel.ubuntu.com/~cking/stress-ng/`

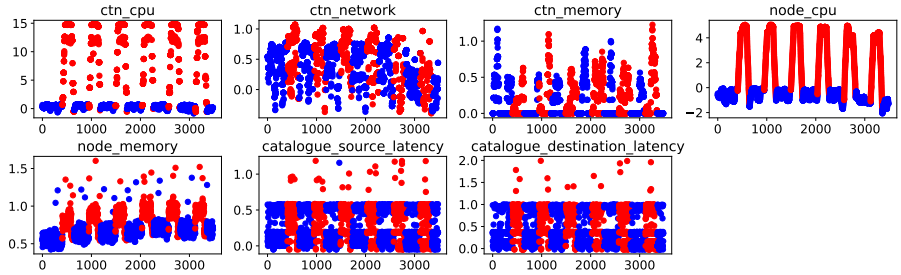

Fig. 3. Collected metrics when CPU hog is injected to microservice catalogue.

– *Mean Average Precision* (MAP) quantifies the overall performance of a method, where N is the number of microservices:

$$MAP = \frac{1}{|A|} \sum_{a \in A} \sum_{1 \leq k \leq N} PR@k. \tag{3}$$

5.2 Effectiveness evaluation

For each anomaly case, we collect the performance metrics from the application (suffixed with latency) and run-time system, including containers (prefixed with ctn) and worker nodes (prefixed with node). Fig. 3 gives an example of the collected metrics when the "CPU hog" anomaly fault is injected to the catalogue microservice, repeated six times within one hour. The data collected during the fault injection is marked in red. The CPU hog fault is expected to be reflected by the ctn_cpu metric. We can see that (i) there are obvious spikes in metrics ctn_cpu and node_cpu. The spike of node_cpu is caused by the spike of ctn_cpu as container resource usage is correlated to node resource usage; (ii) metrics ctn_memory and node_memory also have some deviations; (iii) the fault CPU hog causes spikes in service latency. Therefore, we can conclude that the fault injected to the service manifests itself with a significant deviation from normal status. Meanwhile, it also affects some other metrics.

For each fault injected service, we train the autoencoder with normal data and test with the anomalous data. Fig. 4 shows the reconstruction errors from autoencoder for each metric. We can see that the metric ctn_cpu has a large error comparing with other metrics, which indicates it has a higher probability to be the cause of the anomaly of service catalogue. The second highest reconstruction error is in the node_cpu metric, which is due to its strong correlation with the container resource usage. Hence, we conclude that ctn_cpu is the culprit metric.

Table 1 demonstrates the results of our method on different microservices and faults, in terms of PR@1, PR@3 and MAP. We observe that our method achieve a good performance with 100% in PR@1 in different services and faults, except for the service orders and carts with the fault memory leak. This is because (i) orders and carts are computation-intensive services; (ii) we exhaust their resource memory heavily in our fault injection; (iii) fault memory leak issues manifest as

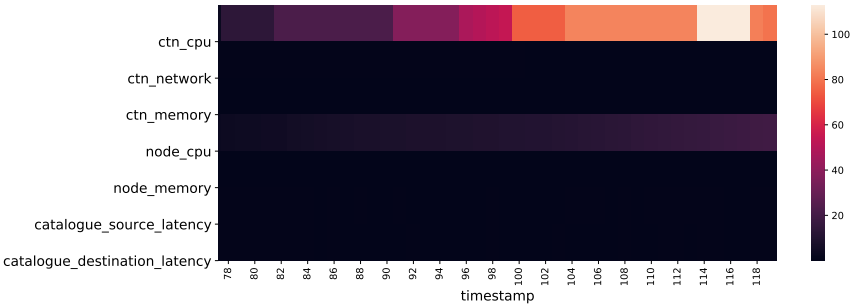

Fig. 4. Reconstruction errors for each metric when CPU hog is injected to microservice catalogue.

Table 1. Performance of identifying culprit metrics.

service	orders	catalogue	carts	user	average
CPU Hog					
PR@1	1.0	1.0	1.0	1.0	1.0
PR@3	1.0	1.0	1.0	1.0	1.0
MAP	1.0	1.0	1.0	1.0	1.0
Memory Leak					
PR@1	0.83	1.0	0	1.0	0.71
PR@3	0.83	1.0	1.0	1.0	0.96
MAP	0.88	1.0	0.83	1.0	0.93

both high memory usage and high CPU usage. As our method target root cause that manifests itself with a significant deviation of causal metric, the accuracy decreases when the root cause manifests in multiple metrics. On average, our system achieves 85.5% in precision and 96.5% in MAP.

Furthermore, we apply the autoencoder to all of the pinpointed faulty services by the culprit service localization (CSL) module and analyze its performance of identifying the culprit services. For example, in an anomaly case where we inject a CPU hog into service catalogue, the CSL module returns a ranked list and the real cause service catalogue is ranked as the third. The other two services with higher rank are service orders and front-end. We leverage autoencoder to these three services, and the results show (i) autoencoder of service order returns Normal, which means it is a false positive and can be removed from the ranked list; (ii) autoencoder of service front-end returns Anomaly, and the highest ranked metric is the latency, which indicates that the abnormality of front-end is caused by an external factor, which is the downstream service catalogue. In this case, we conclude that it is not a culprit service and remove it from the ranked list; (iii) autoencoder of service catalogue returns Anomaly and the top-ranked metric is

Table 2. Comparisons of identifying culprit services.

Metrics	CSL	CSL + CML	Improvement(%)
PR@1	0.57	0.92	61.4
PR@3	0.83	0.98	18.1
MAP	0.85	0.97	14.1

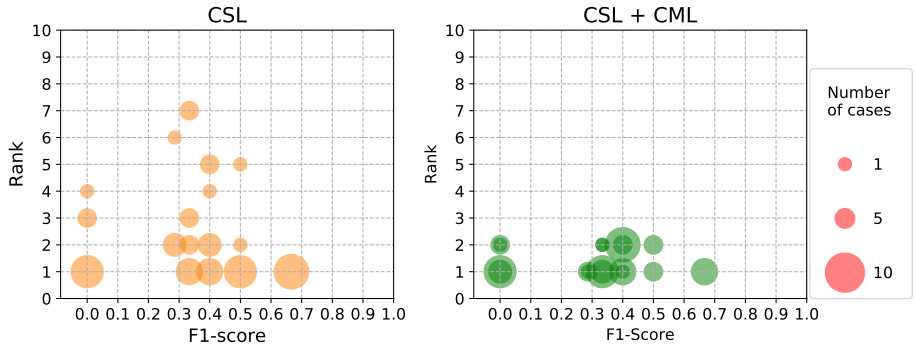

Fig. 5. Calibration of culprit service localization with autoencoder.

ctn_cpu. Therefore, with autoencoder, we can reduce the number of potential faulty services from 3 to 1.

Fig. 5 shows the rank of culprit services identified by CSL and the calibration results with CSL and culprit metric localization module (CSL + CML) against the F1-score (the harmonic mean of precision and recall) of anomaly detection for all anomaly cases. We observe that applying autoencoder on the service relevant metrics can significantly improve the accuracy of culprit service localization by ranking the faulty service within the top two. Table 2 shows the overall performance of the above two methods for all anomaly cases. It shows that complementing culprit service localization with autoencoder can achieve a precision of 92%, which outperforms 61.4% than the results of CSL only.

6 Related Work

To diagnose the root causes of an issue, various approaches have been proposed in the literature. Methods and techniques for root cause analysis have been extensively studied in complex system [13] and computer networks [7].

Recent approaches for cloud services typically focus on identifying coarse-grained root causes, such as the faulty services that initiate service performance degradation [8, 15, 9]. In general, they are graph-based methods that construct a dependency graph of services with knowledge discovery from metrics or provided service call graph, to show the spatial propagation of faults among services; then they infer the potential root cause node which results in the abnormality of other nodes in the graph. For example, Microscope [8] locates the faulty service

by building a service causality graph with the service dependency and service interference in the same machine. Then it returns a ranked list of potential culprit services by traversing the causality graph. These approaches can help operators narrow down the services for investigation. However, the causes set for an abnormal service are of a wide range, hence it is still time-consuming to get the real cause of faulty service, especially when the faulty service is low-ranked in the results of the diagnosis.

Some approaches identify root causes with fine granularity, including not only the culprit services but also the culprit metrics. Seer [4] is a proactive online performance debugging system that can identify the faulty services and the problematic resource that causes service performance degradation. However, it requires instrumentation to the source code; Meanwhile, its performance may decrease when re-training is frequently required to follow up the updates in microservices. Loud [10] and MicroCause [11] identify the culprit metrics by constructing the causality graph of the key performance metrics. However, they require anomaly detection to be performed on all gathered metrics, which might introduce many false positives and decrease the accuracy of causes localization. Álvaro Brandón, et al. [1] propose to identify the root cause by matching the anomalous graphs labeled by an expert. However, the anomalous patterns are supervised by expert knowledge, which means it can only detect previously known anomaly types. Besides, the computation complexity of graph matching is exponential to the size of the previous anomalous patterns. Causeinfer [2] pinpoints both the faulty services and culprit metrics by constructing a two-layer hierarchical causality graph. However, this system uses a lag correlation method to decide the causal relationship between services, which requires the lag is obviously included in the data. Compared to these methods, our proposed system leverages the spatial propagation of the service degradation to identify the culprit service and the deep learning method, which can adapt to arbitrary relationships among metrics, to pinpoint the culprit metrics.

7 Conclusion and Future Work

In this paper, we propose a system to help cloud operators to narrow down the potential causes for a performance issue in microservices. The localized causes are in a fine-granularity, including not only the faulty services but also the culprit metrics that cause the service anomaly. Our system first pinpoints a ranked list of potential faulty services by analyzing the service dependencies. Given a faulty service, it applies autoencoder to its relevant performance metrics and leverages the reconstruction errors to rank the metrics. The evaluation shows that our system can identify the culprit services and metrics with high precision.

The culprit metric localization method is limited to identify the root cause that reflects itself with a significant deviation from normal values. In the future, we would like to develop methods to cover more diverse root causes by analyzing the spatial and temporal fault propagation.

Acknowledgment

This work is part of the FogGuru project which has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 765452. The information and views set out in this publication are those of the author(s) and do not necessarily reflect the official opinion of the European Union. Neither the European Union institutions and bodies nor any person acting on their behalf may be held responsible for the use which may be made of the information contained therein.

References

1. Álvaro Brandón, et al.: Graph-based root cause analysis for service-oriented and microservice architectures. Journal of Systems and Software **159**, 110432 (2020)
2. Chen, P., Qi, Y., Hou, D.: Causeinfer: Automated end-to-end performance diagnosis with hierarchical causality graph in cloud environment. IEEE Transactions on Services Computing **12**(02), 214–230 (2019)
3. Di Francesco, P., Lago, P., Malavolta, I.: Migrating towards microservice architectures: An industrial survey. In: ICSA. pp. 29–2909 (2018)
4. Gan, Y., et al.: Seer: Leveraging big data to navigate the complexity of performance debugging in cloud microservices. In: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems. p. 19–33. ASPLOS '19 (2019)
5. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016), `http://www.deeplearningbook.org`
6. Gulenko, A., et al.: Detecting anomalous behavior of black-box services modeled with distance-based online clustering. In: 2018 IEEE 11th International Conference on Cloud Computing (CLOUD). pp. 912–915 (2018)
7. łgorzata Steinder, M., Sethi, A.S.: A survey of fault localization techniques in computer networks. Science of Computer Programming **53**(2), 165–194 (2004)
8. Lin, J., et al.: Microscope: Pinpoint performance issues with causal graphs in microservice environments. In: Service-Oriented Computing. pp. 3–20 (2018)
9. Ma, M., et al.: Automap: Diagnose your microservice-based web applications automatically. In: Proceedings of The Web Conference 2020. p. 246–258. WWW '20 (2020)
10. Mariani, L., et al.: Localizing faults in cloud systems. In: ICST. pp. 262–273 (2018)
11. Meng, Y., et al.: Localizing failure root causes in a microservice through causality inference. In: 2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS). pp. 1–10. IEEE (2020)
12. Newman, S.: Building Microservices. O'Reilly Media, Inc, USA (2015)
13. Solé, M., Muntés-Mulero, V., Rana, A.I., Estrada, G.: Survey on models and techniques for root-cause analysis (2017)
14. Thalheim, J., et al.: Sieve: Actionable insights from monitored metrics in distributed systems. In: Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference. p. 14–27 (2017)
15. Wang, P., et al.: Cloudranger: Root cause identification for cloud native systems. In: CCGRID. pp. 492–502 (2018)
16. Wu, L., et al.: MicroRCA: Root cause localization of performance issues in microservices. In: NOMS 2020 IEEE/IFIP Network Operations and Management Symposium (2020)