Graph Based Liability Analysis for the Microservice Architecture

Onur Kalinagac, Wissem Soussi and Gürkan Gür
Zurich University of Applied Sciences (ZHAW) InIT
Winterthur 8401, Switzerland
[name.surname]@zhaw.ch

Abstract—**In this work, we present Graph Based Liability Analysis Framework (GRALAF) for root cause analysis (RCA) of the microservices. In this Proof-of-Concept (PoC) tool, we keep track of the performance metrics of microservices, such as service response time and CPU level values, to detect anomalies. By injecting faults in the services, we construct a Causal Bayesian Network (CBN) which represents the relation between service faults and metrics. The constructed CBN is used to predict the fault probability of services under given metrics which are assigned discrete values according to their anomaly states.**

Keywords*—*Root cause analysis, Causal Bayesian Network, microservices.

I. Context and Motivation

In recent years, the complexity of software systems has increased vastly, and monolithic systems are getting more difficult to build and maintain. Microservice architecture allows a large application to be divided into smaller, independent parts, each with its own set of responsibilities. They can be implemented by different service providers using various technologies or programming languages. A microservices-based application can call on many internal microservices to compose its response in order to fulfill a single user request. As a result, in the event of a failure, the responsible party may be unclear, compromising liability and accountability.

For network and service management, it is an active challenge to support confidence between parties and compliance with regulation for cloud and telecom infrastructures. In that regard, one of the proposed solutions is to use Service Level Agreements (SLAs) which are extensively investigated. They are contracts in which service providers guarantee the quality of the services by defining usage conditions [1]. SLAs provide a basic rule set; however, the cases where no violation occurs due to overly loose commitments or multiple SLA violations occur due to internal dependencies may have an adverse effect on the accuracy of determining the true cause of a failure. To address this issue, we present Graph Based Liability Analysis Framework (GRALAF) which is being developed in the INSPIRE-5Gplus project [2]. It performs near-real time anomaly detection and root cause analysis (RCA) in a microservice environment.

There are many research works dealing with anomaly detection and RCA in the literature [3]. Studies including a causality graph-based analysis mostly use PC algorithm [4] to build causality graph. In one of them, namely Microscope [5], they use standard deviation-based heuristics to detect anomalies in response times on front-end requests. Starting from the application front-end, their algorithm recursively visits the nodes of causality graph in the opposite direction of edges. It checks a node's neighbors if it exhibits abnormal behavior. At the last step, the root cause candidates are ranked according to their anomaly score or the Pearson correlation between their response times and the application front-end. In [6], the MicroRCA algorithm uses topology graph-based analysis and creates a topology graph with vertices representing services and host machines, and oriented arcs representing service interactions and hosting. Each vertex of the topology graph is associated with the time series of KPIs monitored on the relevant service or node. MicroRCA then creates a subgraph by selecting the vertices in the topology graph that correspond to the services where anomalies were discovered. The RCA is carried out by modifying the random walk-based subgraph search suggested in MonitorRank [7].

In GRALAF, we construct a Causal Bayesian Network (CBN) with NOTEARS [8] algorithm from a data set which is obtained by injecting faults to services and consists of the service metrics and service fault states. CBN helps us to learn about the causal relations between the fault state of services and their metrics. Then, we try to detect the meaningful changes in the performance metrics of microservices or an SLA violation. When either of them happens, we predict the corresponding state of services in terms of faultiness. By analyzing our findings, we complete the RCA of the failure.

II. GRALAF

A. Architecture Description

We run all the software elements on a mobile workstation with Intel Core i9-11950H 2.60GHz (2.30GHz $\times$ 16) CPU, 32GB RAM, and 1TB SSD storage. Virtual machines (VMs) are managed with Oracle VM VirtualBox. Our system diagram is shown in Fig. 1. On this block diagram, you may see how the system components interact with each other.

Kubernetes: Kubernetes MicroK8s distribution [9] is deployed in two VMs each with two allocated cores and 8GB RAM, hosted on the mobile workstation.

Virtual MQTT device: We developed a mock IoT device application built in Python that can be managed through MQTT. It also sends random sensor data periodically to the EdgeX server. These application instances are run directly on the host machine. VMs are assigned to the same NAT network and the connections between the host machine and VMs are provided with individual port proxy configurations.

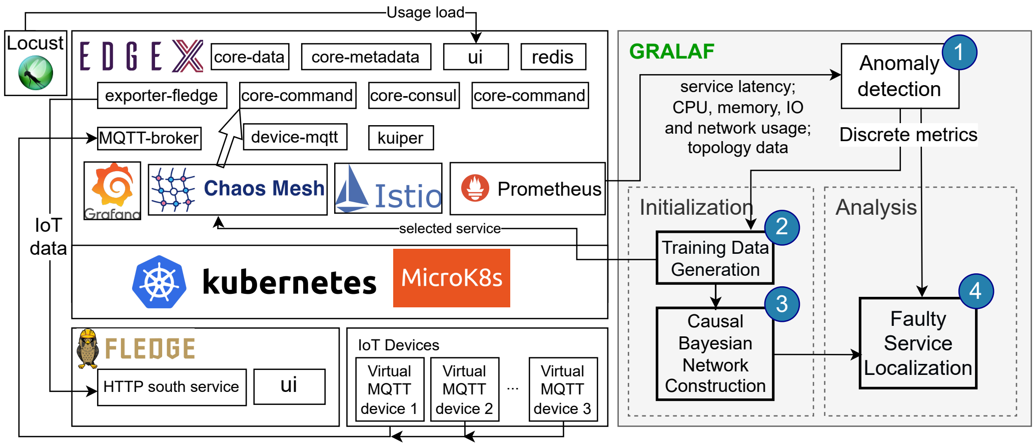


Fig. 1. System block diagram for the demo environment.

Fledge: Fledge [10] is an open source framework and community for the industrial edge focused on IoT devices. It is mainly used for fog computing. In our architecture, it operates as a remote cloud server application where the data exported from EdgeX are stored. It is installed in a VM on the mobile workstation. VM has one allocated core and 2GB RAM.

EdgeX: EdgeX [11] is an open source software framework that provides interoperability between devices and applications at the IoT edge. We deployed its main services on our Kubernetes environment. MQTT-broker indirectly connects MQTT based device service with the virtual MQTT devices. A data exporter app named exporter-fledge is also deployed to send sensor data to the Fledge server.

Istio: Istio is a service mesh that allows operator to transparently add capabilities like observability and traffic management, without adding them to service code [12]. It primarily serves to keep track of intermicroservice response times.

Prometheus: Prometheus is an open-source systems monitoring and alerting toolkit [13]. It periodically scrapes system related data from Kubernetes pods, node exporter and Istio service, and stores it as a time series database.

Chaos Mesh: Chaos Mesh [14] is an open-source fault-generation tool for Kubernetes. We use it to create delay and CPU fault injections to our microservices via its REST API.

Locust: Locust [15] is an open source performance testing tool that can be used to simulate a very large number of concurrent users. We developed a Python script which uses its library to generate multi-threaded load on the REST endpoints of the EdgeX UI.

Grafana: Grafana [16] is a tool that enables you to query, visualize, receive alerts about your metrics and comprehend them. We use it to demonstrate the effect of fault injections on microservices' metrics.

GRALAF: It is developed in Python and performs RCA based on SLA and CBN. The application is run on the host mobile workstation.

B. *Description of the GRALAF workflow*

In order to monitor service metrics, GRALAF periodically queries all the response time, CPU and memory metrics of microservices from Prometheus.

1 We start creating our training data set with the initial 15 readings. We also add columns for the fault state of the services. Services are assumed to be representing their normal working conditions at this initial phase so their fault states are set to 0. The threshold values for each metric are obtained by multiplying the average of the metric readings by a correction coefficient. All the new readings are compared to these thresholds and the ones with higher readings are marked as abnormal. Abnormal values are set to 1; otherwise set to 0.

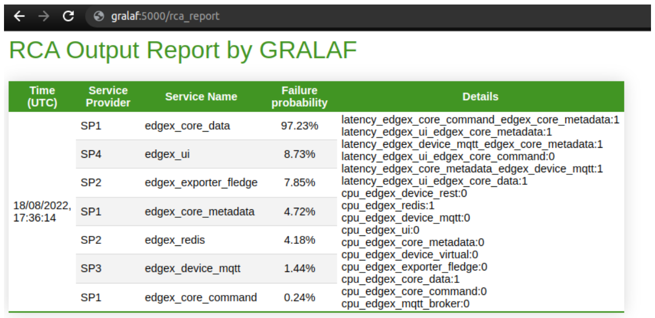
2 In the second phase, we start injecting CPU and delay faults into randomly selected pods through Chaos Mesh and save the corresponding values. The fault state of the selected pods is set to 1 while the rest is set to 0 on each reading. This phase continues until we get enough amount of data depending on the number of microservices.

3 In the third phase, we train a structure model with causalnex Python package which uses NOTEARS algorithm. From the structure model, we construct the CBN by using the same training data set. In order to increase prediction speed, we remove weak links from the graph. That completes our preparation steps.

4 During the operation phase, when an anomaly is detected in the metric readings, it is called an *incident*. When that occurs, GRALAF estimates the probability of fault state for each service by using the prediction feature of the existing CBN. Then an output report with the non-negligible estimated probability values is generated.

III. Planned Demonstration

We present a live demo where we trigger fault injection to one of the microservices, and then show the metric changes on Grafana. We expect GRALAF to detect the anomaly and give us an RCA report in the UI about the newly performed root cause analysis. This report includes information such as timestamp, service name with failure probability and anomalous metrics as shown in Fig. 2.



gralaf:5000/rca_report

RCA Output Report by GRALAF

Time (UTC)	Service Provider	Service Name	Failure probability	Details
18/08/2022, 17:36:14	SP1	edgex_core_data	97.23%	latency_edgex_core_command_edgex_core_metadata:1 latency_edgex_ui_edgex_core_metadata:1 latency_edgex_device_mqtt_edgex_core_metadata:1 latency_edgex_ui_edgex_core_command:0 latency_edgex_core_metadata_edgex_device_mqtt:1 latency_edgex_ui_edgex_core_data:1 cpu_edgex_device_rest:0 cpu_edgex_redis:1 cpu_edgex_device_mqtt:0 cpu_edgex_ui:0 cpu_edgex_core_metadata:0 cpu_edgex_device_virtual:0 cpu_edgex_exporter_fledge:0 cpu_edgex_core_data:1 cpu_edgex_core_command:0 cpu_edgex_mqtt_broker:0
	SP4	edgex_ui	8.73%	
	SP2	edgex_exporter_fledge	7.85%	
	SP1	edgex_core_metadata	4.72%	
	SP2	edgex_redis	4.18%	
	SP3	edgex_device_mqtt	1.44%	
	SP1	edgex_core_command	0.24%	

Fig. 2. RCA output report.

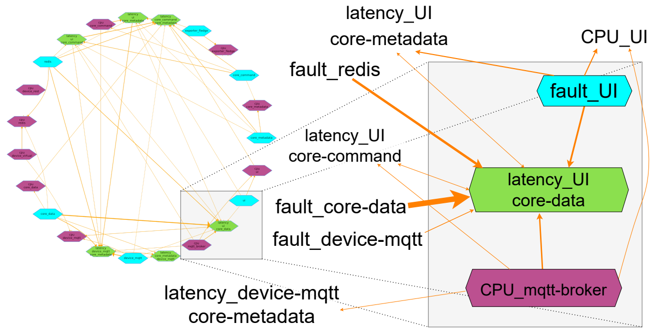


Fig. 3. Constructed CBN after GRALAF training.

A. Considered Demonstration Scenarios

We demonstrate two scenarios for two different RCA methods.

1) SLA based RCA: In this baseline scenario, we intend to show the demo environment. GRALAF tracks metrics and compare them with the given SLA data. In the case of SLA violation, GRALAF informs us about the corresponding service which failed to sustain its commitments such as max service delay or service availability.

2) CBN based RCA: The anomaly detection is performed continuously in the live system and provides discretized values to the rest of the flow (1). Since the training data generation (2) takes around one hour, we perform it and construct the CBN(3) beforehand. In Fig. 3, you can see the visualisation of the CBN graph where the thickness of the edges indicates the causal probability between the nodes. A node with a turquoise color represents the fault state of a service, while a node with purple or green coloring represents a service's CPU metric or response time between a service pair. This graph indicates that if an abnormal response time between UI and core-data services is observed, core-data fault is much more likely to be the root cause than UI service fault.

After the fault injection, GRALAF detects the anomaly and gives us the estimated probability of each service being responsible for the incident as an RCA output report (4). The report is accessible from GRALAF UI service as shown in Fig. 2.

B. Demonstrated Aspects

In our demonstration, we present four main aspects:

- A technical overview of the developed system is given.
- Our RCA algorithm is presented to illustrate how the PoC system operates.
- The GRALAF output report is explained to depict the utility of the system from the RCA perspective.
- From anomaly detection to CBN generation, the technical challenges we have faced and our solutions for them are discussed.

IV. Conclusion

We proposed a graph liability analysis framework where we perform anomaly detection and CBN based RCA in a microservice environment. Also, we deployed a realistic IoT edge network framework to develop and test our algorithms. As future work, we will work on further development and performance evaluation of GRALAF.

Acknowledgment

The research leading to these results partly received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 871808 (5G PPP project INSPIRE-5Gplus). The paper reflects only the authors' views. The Commission is not responsible for any use that may be made of the information it contains.

References

[1] C.-Y. Lee, K. M. Kavi, R. A. Paul, and M. Gomathisankaran, "Ontology of secure service level agreement," in *2015 IEEE 16th International Symposium on High Assurance Systems Engineering*, 2015, pp. 166–172.

[2] J. Ortiz *et al.*, "INSPIRE-5Gplus: Intelligent security and pervasive trust for 5G and beyond networks," in *Proceedings of the 15th International Conference on Availability, Reliability and Security*, ser. ARES '20. New York, NY, USA: Association for Computing Machinery, 2020.

[3] J. Soldani and A. Brogi, "Anomaly detection and failure root cause analysis in (micro) service-based cloud applications: A survey," *ACM Comput. Surv.*, vol. 55, no. 3, Feb 2022.

[4] P. Spirtes, C. N. Glymour, R. Scheines, and D. Heckerman, *Causation, prediction, and search*. MIT press, 2000.

[5] J. Lin, P. Chen, and Z. Zheng, "Microscope: Pinpoint performance issues with causal graphs in micro-service environments," in *International Conference on Service-Oriented Computing*. Springer, 2018, pp. 3–20.

[6] L. Wu, J. Tordsson, E. Elmroth, and O. Kao, "MicroRCA: Root cause localization of performance issues in microservices," in *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE, Apr. 2020, pp. 1–9.

[7] M. Kim, R. Sumbaly, and S. Shah, "Root cause detection in a service-oriented architecture," *ACM SIGMETRICS Performance Evaluation Review*, vol. 41, no. 1, pp. 93–104, 2013.

[8] X. Zheng, B. Aragam, P. Ravikumar, and E. P. Xing, "DAGs with NO TEARS: Continuous optimization for structure learning," 2018.

[9] "MicroK8s homepage," https://microk8s.io, 2022.

[10] "Fledge project page," https://www.lfedge.org/projects/fledge, 2022.

[11] "EdgeX Foundry homepage," https://www.edgexfoundry.org, 2022.

[12] "Istio homepage," https://istio.io, 2022.

[13] "Prometheus homepage," https://prometheus.io, 2022.

[14] "Chaos Mesh homepage," https://www.chaos-mesh.org, 2022.

[15] "Locust homepage," https://locust.io, 2022.

[16] "Grafana homepage," https://grafana.com, 2022.