Efficient Unsupervised Latency Culprit Ranking in Distributed Traces with GNN and Critical Path Analysis

Mahsa Panahandeh
Electrical and Computer Engineering department,
University of Alberta
Edmonton, Alberta, Canada
panahand@ualberta.ca

Naser Ezzati-Jivan
Department of Computer Science, Brock University
St. Catharines, Ontario, Canada
nezzatijivan@brocku.ca

Abdelwahab Hamou-Lhadj
Department of Electrical and Computer Engineering,
Concordia University
Montreal, Quebec, Canada
wahab.hamou-lhadj@concordia.ca

James Miller
Department of Electrical and Computer Engineering,
University of Alberta, Canada
Edmonton, Alberta, Canada
jimm@ualberta.ca

ABSTRACT

Microservices offer the benefits of scalable flexibility and rapid deployment, making them a preferred architecture in today's IT industry. However, their dynamic nature increases their susceptibility to failures, highlighting the need for effective troubleshooting strategies. Current methods for pinpointing issues in microservices often depend on impractical supervision or rest on unrealistic assumptions. We propose a novel approach using graph unsupervised neural networks and critical path analysis to address these limitations. Our experiments on four open-source microservice benchmarks show significant results, with top-1 accuracy ranging from 86.4% to 96%, over 6% enhancement compared to existing methods. Moreover, our approach reduces training time by 5.6 times compared to similar works on the same datasets.

CCS CONCEPTS

• **Software and its engineering → Software defect analysis**; • **Computer systems organization → Reliability**.

KEYWORDS

Culprit identification, Graph neural Network, Critical path analysis, Distributed traces, FIRM dataset

ACM Reference Format:
Mahsa Panahandeh, Naser Ezzati-Jivan, Abdelwahab Hamou-Lhadj, and James Miller. 2024. Efficient Unsupervised Latency Culprit Ranking in Distributed Traces with GNN and Critical Path Analysis . In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United K ingdom.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3629527.3651841

ICPE '24, May 7–11, 2024, London, United Kingdom

ACM ISBN 979-8-4007-0445-1/24/05…$15.00
https://doi.org/10.1145/3629527.3651841

1 INTRODUCTION

Microservice architecture is becoming increasingly popular among various systems architectures due to its fast delivery, scalability, and independence. However, managing quality in microservice applications remains a significant challenge [5, 6, 10]. This challenge involves the need to set suitable alert thresholds and filters to alert developers to issues promptly, without overwhelming them with extraneous data [6]. Hassan et al. [5] and Jamshidi et al.[6] advocate that machine learning could significantly mitigate these challenges. As a result, numerous trace-based methods employing machine learning techniques have been developed to quickly identify and address issues as they emerge [8, 10, 12]. However, these methods face challenges, primarily due to idealistic assumptions or relying on supervision methods, reducing their effectiveness and practical applicability [10].

Numerous studies [8, 14, 16, 19] have employed supervised machine learning models for anomaly detection and predicting root causes in microservices. These approaches heavily depend on labelled datasets, which can be costly to acquire [7]. Furthermore, they require comprehensive coverage of all possible fault types to accurately differentiate between anomaly propagation paths. Yet, some of these studies do not incorporate request types or fault types in their approach, and they may also lack this information, e.g., [16].

Although certain studies [11, 12, 16] overlook the variability in anomaly propagation patterns, other research works [10, 14] emphasise the crucial importance of considering this variability to enhance anomaly detection and analysis. For example, FIRM [14] emphasizes how latency anomalies may propagate differently across scenarios, even for identical request types. To investigate variations of anomaly propagation paths, FIRM [14] proposes studying critical paths in request executions. It employs a supervised classification approach, which still makes it rely on labelled datasets including both normal and abnormal requests of different execution paths. Moreover, FIRM [14] assumes that the longest service on a critical path contributes most to the total latency. However, our experiments show that this assumption does not always hold because individual services can show a wide variance in latency, even under normal conditions.

Addressing the need for distinguishing between anomaly propagation paths as well as breaking the need for having a labelled dataset, Li et al.[10] propose a trace analysis approach based on studying historical data of service invocations. Although Li et al. study[10] outperforms other unsupervised methods, it may not be optimal for microservice systems handling extensive requests involving numerous services. The issue arises from their method requiring feature selection for each service invocation within a real-time window, which can result in significant time consumption. Additionally, the effectiveness of Li et al.'s approach[10] might be compromised in scenarios with a scarce number of normal or abnormal traces, or when the test window includes a limited range of request types.

To overcome these limitations, we developed an unsupervised method that integrates the analysis of critical paths for enhanced culprit prioritization. Our approach starts with establishing a baseline model using the application of graph neural networks (GNN), which learns the expected latency distributions across all services and their interdependencies, effectively mirroring the system's anticipated behaviour. This model is then used for anomaly detection at the granularity of individual requests. To narrow down the context of latency propagation and identify culprits, we study abnormal requests against normal requests, sharing the same critical path. Our method involves clustering historical requests based on their critical paths, thereby creating distinct profiles for each path. Within these profiles, we detail vectors for each service that capture the service's distribution within the critical path. By comparing the observed latency of services in abnormal paths to their standard distributions outlined in the profiles, we effectively isolate and identify the culprits. Services exhibiting greater deviation from their expected distribution are ranked as more suspicious culprits.

To enhance the efficiency of our method compared to existing approaches, We design our GNN model with the assumption that the service invocations graph is static. By this assumption, we effectively reduce the complexity associated with dynamic network models, leading to increased efficiency. To prevent missing any update in service invocations, we propose periodic evolutionary updates as an alternative to ensure the model stays in sync with any changes. Additionally, we integrate a feature sampling technique from services across various layers in our GNN model. This strategy ensures the capture of critical information from services within the GNN, maintaining the model's scalability and efficiency, particularly when dealing with extensive service invocation or service dependency graphs.

Our experiments conducted on four benchmark systems in different sizes and complexity demonstrate that our approach has an accuracy of 86.4%-96% and outperforms similar methods [16] by over 6% while also enhancing the training time by a factor of 5.6. Our main contributions can be summarized as follows:

- Proposing an unsupervised GNN model, eliminating the requirement for labelled datasets which can be expensive to obtain in practice.
- Integration of our latency detection methodology with critical path analysis, refining the focus on potential culprits and enhancing the efficiency of culprit prediction.
- Refinement of the GNN model to tackle scalability challenges inherent in large service invocations or service dependency graphs, as well as improving time efficiency during both training and testing phases.

2 BACKGROUND SUMMARY

Critical Path Analysis involves identifying the longest duration path in a request's journey through a distributed system, which can be complex due to mixed synchronous and asynchronous communications. Distributed tracing helps trace these paths by collecting detailed request flow data [1, 14].

Culprit Identification focuses on pinpointing the service or component causing latency anomalies. While 'culprit' refers to the direct cause of performance slowdowns, 'root cause' delves into the underlying issue. This work prioritizes culprit ranking in abnormal request paths as a step towards root cause analysis [9].

3 METHOD DESIGN

Our approach introduces an end-to-end latency anomaly detection and culprit ranking framework, which comprises three primary phases: I) Data Preparation, II) Anomaly Detection, and III) Culprit Ranking, as detailed in the subsequent subsections.

3.1 Data Preparation

Our approach takes two main inputs: historical data representing the expected behaviour of the system and interdependencies between services. This data is collected from telemetry data provided by any distributed traces monitoring tool. We ensure our data encompasses a wide range of requests, each with a unique execution path. Each request is collected as a trace that can be defined by a feature vector that details the latency of services (spans) involved in the request:

$$V_{R_0} = [l_1, l_2, ..., l_n], \tag{1}$$

Here, n is the number of services in request R_0 and l is the latency value for each service. The historical data comprises numerous such feature vectors. To ensure consistency across the data, we normalize all latency values. Then, we determine the critical path for each request using the method outlined in the literature [14], and accordingly cluster the data based on critical paths. For each cluster, we create a profile that encapsulates the distribution of latencies, including mean and standard deviation, for each service within the critical path. Such a standardized approach allows for a comprehensive comparison of service performance across various service paths.

Figure 1 presents latency distributions for a service (10-contact) in a ticket-booking benchmark [20]. On the left, it shows the latency distribution across all requests in historical data, while on the right, latency distributions across different critical paths are shown. The distribution of data across clusters facilitates a more accurate discernment of patterns. Such clarity is especially beneficial for data lacking obvious service relationships or evident request clustering patterns, as observed in our experiments about the dataset used in this study, where no clustering algorithm, such as k-means or hierarchical clustering, yields meaningful clusters. This limitation makes some existing methods [13, 18] inappropriate for such data.

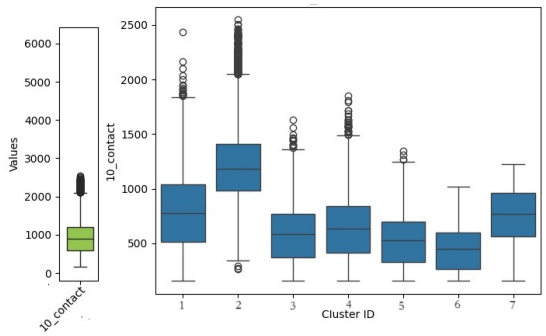

Figure 1: The latency distribution of a service across all execution paths (left) compared to within critical paths (right).

The second input that our approach takes is interdependencies between services such as service invocations, as we believe that these interdependencies primarily govern the propagation of behaviour across the services. This data can be inferred from dependency graphs provided by distributed trace tools or by studying the microservice system itself.

3.2 Anomaly detection

Utilizing the advanced pattern recognition capabilities of deep graph neural networks (GNNs) for structured data analysis, we use a GNN model to establish a baseline for anomaly detection. Our model is constructed based on interdependencies patterns and distribution of latencies, collected from historical data. First, we generate a directed acyclic graph G= (V,E) from service interdependencies, where V represents the set of nodes corresponding to services and E represents the set of edges representing service invocations. The direction of each edge specifies the propagation path of behaviour through services. Each node is associated with a feature array that encapsulates the corresponding latency values from all request feature vectors (V_R vectors) in historical data. To enhance the efficiency of our model compared to existing works [8, 16], we consider the following considerations.

Firstly, we initially assume a static structure for the directed acyclic graph to enhance the GNN's training efficiency and scalability. This assumption is pivotal for managing large call graphs and complex service interaction patterns. To adapt to changes in service invocations, we introduce periodic model updates, allowing our system to reflect changes over time without compromising the initial efficiency gains.

Secondly, we implement batch processing, enabling simultaneous analysis of multiple graphs, each with unique structures and sizes. This diversity is essential in training our model to recognize a wide array of connectivity patterns. Our adaptation employs a neighbourhood sampling strategy, where a fixed-size subset of a node's neighbours is selected for feature aggregation. This method addresses challenges related to variable connectivity and scales effectively for large graphs. Each layer aggregates information from a node's immediate neighbours and subsequent layers aggregate information at increasingly larger distances from the target node. This iterative process ensures that even with sampling, information from the broader neighbourhood can influence the node's representation, albeit indirectly. We mitigate potential data loss from sampling by designing our model to aggregate features across layers. Unlike traditional methods that depend on separate embeddings for each node to produce features, our design consolidates feature generation across the network.

We adopted the GraphSAGE model [4], to construct our model according to these considerations. Our model comprises two graph convolutional layers. The initial layer transforms the features of a node and its neighbours, aggregated through a mean function, into a dimensional hidden space, for abstract representation. The second layer then projects these hidden representations back to the original feature dimensions. This unsupervised learning architecture is adept at encoding and decoding node features, capturing both structural and feature-based graph information to generalize well to unseen nodes or entirely new graphs.

The model's operation during each message-passing iteration is mathematically described as follows:

$$H_v^{(l+1)} = \sigma\left(W^{(l)} \cdot \text{MEAN}\left(\{H_v^{(l)}\} \cup \{H_u^{(l)}, \forall u \in \mathcal{N}(v)\}\right)\right) \quad (2)$$

where $H_v^{(l)}$ is the feature vector of node v at layer l, $W^{(l)}$ is the weight matrix for layer l, σ denotes a non-linear activation function (e.g., ReLU), and $\mathcal{N}(v)$ includes the neighbours of node v. Our optimization goal is to minimize the mean squared error (MSE) between the original node features (X) and their reconstructed counterparts ($\hat{X}$) after processing through the GraphSAGE layers. The optimization goal is succinctly captured by:

$$\text{Minimize } \mathcal{L} = \text{MSE}(X, \hat{X}) \quad (3)$$

Here, X represents the model's input (the original node features), and $\hat{X}$ denotes the output (the reconstructed node features).

To detect anomalies using the trained modes, we convert a test trace into a format that aligns with the request feature vector structure previously outlined. We use our model (f) along with adjacency information encapsulated in the graph (E) to reconstruct the test request feature vector X as $\hat{X} = f(X, E)$. Anomalies are identified by comparing the loss between the original and reconstructed vectors against a threshold, i.e., $MSE(X, \hat{X}) > threshold$. In this paper, we set the threshold at 0.1.

3.3 Culprit Ranking

After detecting an anomaly, we proceed with the Culprit Ranking step by comparing the abnormal test request's feature vector with similar requests from historical data. Our experiments indicate that narrowing these comparisons to critical paths is more efficient. Critical paths encompass services that drive the latency issue in a request, making them a priority for identifying the culprit.

We first, compute the critical path within the test request. Then, we match it with the same critical path cluster in historical data. From this match, we extract the cluster profile that provides detailed insights into the distribution of each service along the critical path. For each service within the test critical path, we hypothesize it is a potential culprit. Then, we use the service distribution details extracted from the critical path profile to choose an appropriate statistical simulation technique, as advised by Forbes [2]. Through

parameter estimation, we generate a new latency value for the service that aligns with its distribution. This value replaces the original one in the test vector, and anomaly detection is performed on this manipulated request vector. If the model identifies the manipulated vector as normal, our hypothesis is validated, and the service is added to the list of culprits; otherwise, it's discarded. Culprits are subsequently ranked based on their deviation from their respective normal distributions.

During the validation of our model, we discovered that checking services individually as potential culprits aids in identifying indirect anomaly propagation paths. We call frequently affected services by a true root cause, even when there are no direct invocations between them, "indirect dependency". Figure 2-**A**, shows direct dependencies, i.e., service invocations, as well as indirect dependencies for the hotel reservation benchmark system. This representation assists in understanding anomaly propagation paths and can be integrated into our model to improve accuracy by accommodating unknown service dependencies.

4 EXPERIMENTS AND EVALUATION

We evaluate our approach on a dataset including four microservice benchmarks provided by Qiu et al.[15] as part of the FIRM research project[14]. These benchmarks cover social networking, hotel reservations, media services, and ticket booking systems. The dataset includes traces representing normal system behaviour, with latency data for all services, which we treat as our historical dataset. Additionally, there are labelled files for each benchmark system, containing traces collected during anomalies in labelled services, treated as test requests for evaluating our approach. The dataset also includes covered execution paths of collected requests, illustrating the interdependency patterns within each benchmark system.

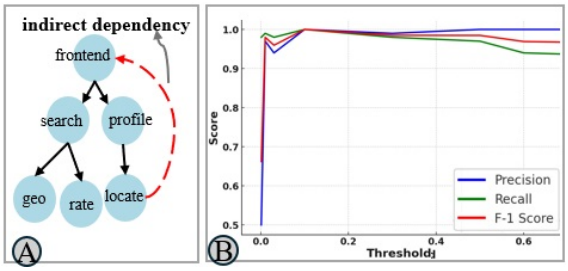

Figure 2: A: Direct and indirect anomaly propagation paths, B: Confusion metrics for benchmark systems

The performed exploratory data analysis (EDA) on the data, combined with in-depth studies and collaborative consultations with the FIRM authors, reveals that each sample in this dataset isn't simply a raw trace, but rather an aggregation of collected requests spanning the system's execution paths within a 1-minute time window. Each sample provides all services' average latency time for handling various requests within this window. Despite being aggregated from numerous requests, there is consistency in workload and covered execution paths for each sample. Therefore, each sample can be treated as a trace covering all services and execution paths,

Benchmarks	ACC	Top-1	Top-3	Top-5
Social-network (MSE=0.005)	87%	83%	86.2%	87%
hotel-reservation (MSE=0.001)	95.4%	94.7%	95.4%	95.4%
media-service (MSE=0.007)	86.4%	85%	86.4%	86.4%
ticket-booking (MSE=0.001)	96%	94.3%	96%	96%

Table 1: Culprit localization results for different benchmarks

as commonly done in the literature [16, 17]. Consequently, within this pre-processed dataset, the critical path of each sample (trace) is a system critical path and can be computed as the execution path with the maximum latency among all other execution paths.

In our study, we constructed datasets comprising 60% training data, with an additional 20% set aside for both validation and testing of our model. To assess the efficacy of the culprit ranking, we employed the same sampling rate used in prior literature [16], selecting 20% of labelled files.

4.1 Results

We conduct evaluations for anomaly detection and culprit ranking separately. Figure 2-**B**, shows averaged results of precision, recall, and F-1 score metrics [3] across all benchmarks. This visualization illustrates the performance of our model in classifying anomalies for various classification threshold values. Based on our findings, we observed that setting the anomaly classification threshold lower than 0.1 results in a decrease in precision. Conversely, for larger threshold values, there is a decrease in recall while precision remains stable. Therefore, we opted to set the threshold at 0.1 to strike a balance in the performance of our model.

To evaluate the effectiveness of our approach in identifying the true culprit, we employ two key metrics: the overall percentage of accurately identified true culprits (referred to as "ACC") and the Top-k metric, which measures the likelihood of ranking the true culprit within the top k locations among all ranked identified culprits. Table 1 displays the performance results for each benchmark. The numerical values provided alongside the name of each benchmark indicate the average loss value incurred by our trained model for that particular benchmark. Our experiments across four case studies indicate that the true culprit is accurately identified in 86.4% to 96% of cases, and is also ranked within the first five culprits in 83% to 94.7% of cases. Our investigation unveiled that having a balanced representation of requests related to various critical paths in the historical data significantly enhances the ability to distinguish between service distributions of the test and historical critical paths which leads to higher accuracy in our analysis.

In comparison with existing works, we create a dataset with a similar train-test split ratio used by B-MEG [16], that proposes a supervised GNN for the same dataset. We conducted multiple iterations of experiments and reported the average result. Our experiments yielded accuracy improvements of approximately 3-8%, along with a reduction in time complexity to over one-fifth of the model training. Due to the need for adjustments in B-MEG, we were unable to compare execution times for processing test requests in the current paper. However, we compared two variations of our approach: v1 solely utilises the main request path in culprit identification, and v2 incorporates the critical paths to narrow down the

test request context. Our results demonstrated that v2 significantly improves processing time for identifying the culprits in different benchmarks from 1.5-13 seconds in v1 to 0.8-8.3 seconds, marking a 58.33% enhancement on average. Figure 3 summarises the average performance of two versions of our approach against Somashekar et al. work [16].

Despite improvements in v2 of our approach, occasional slight accuracy decreases were observed, although accuracy increased for ticket-booking and social-network benchmarks. Further investigation of our dataset revealed that the stated computed critical path method for this dataset may overshadow the true culprit. This happens for systems with sparse connectivity and shorter request paths since they have limited service diversity in different execution paths. Therefore, the computed critical path can be affected if the true culprit exhibits a relatively small latency distribution range compared to other services.

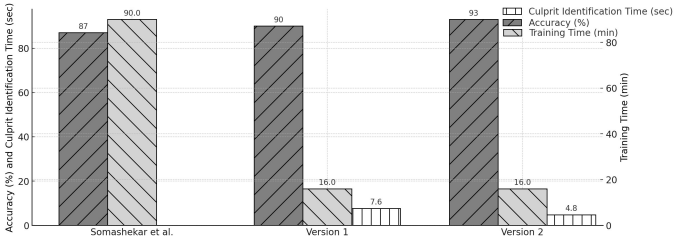

Figure 3: Comparison of two different versions of our approach with B-MEG

5 CONCLUSION

In this paper, we propose an efficient unsupervised GNN model integrated with critical path analysis for culprit ranking in microservice architectures. Our approach eliminates the need for labelled datasets, incorporates critical path analysis for efficient culprit detection, and introduces enhancements for the scalability and speed of the GNN model. Conducted experiments on four benchmarks our model achieves an accuracy in culprit identification ranging from 86.4% to 96%, representing a notable improvement of over 6% compared to existing methods. Furthermore, our approach reduces training time by 5.6 times compared to similar works. Additionally, leveraging critical path analysis results in a 58.33% enhancement in culprit identification speed. Looking ahead, we aim to refine our methodology through additional experiments aimed at broadening its applicability and improving detection capabilities.
The scripts of our model are accessible via the following link: https://anonymous.4open.science/r/ICPE2024-4281/v2.py.

REFERENCES

[1] Brian Eaton, Jeff Stewart, Jon Tedesco, and N. Cihan Tas. 2022. Distributed Latency Profiling through Critical Path Tracing: CPT can provide actionable and precise latency analysis. *Queue* 20, 1 (mar 2022), 40–79. https://doi.org/10.1145/3526967

[2] Catherine Forbes, Merran Evans, Nicholas Hastings, and Brian Peacock. 2011. *Statistical distributions.* John Wiley & Sons.

[3] Cyril Goutte and Eric Gaussier. 2005. A probabilistic interpretation of precision, recall and F-score, with implication for evaluation. In *European conference on information retrieval.* Springer, 345–359.

[4] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems* 30 (2017).

[5] Ahmed Hassan and Tamilselvan Arjunan. 2024. A Comparative Study of Deep Neural Networks and Support Vector Machines for Unsupervised Anomaly Detection in Cloud Computing Environments. *Quarterly Journal of Emerging Technologies and Innovations* 9, 1 (2024), 15–24.

[6] Pooyan Jamshidi, Claus Pahl, Nabor C Mendonça, James Lewis, and Stefan Tilkov. 2018. Microservices: The journey so far and challenges ahead. *IEEE Software* 35, 3 (2018), 24–35.

[7] Iman Kohyarnejadfard, Daniel Aloise, Michel R Dagenais, and Mahsa Shakeri. 2021. A framework for detecting system performance anomalies using tracing data analysis. *Entropy* 23, 8 (2021), 1011.

[8] Cheryl Lee, Tianyi Yang, Zhuangbin Chen, Yuxin Su, and Michael R Lyu. 2023. Eadro: An End-to-End Troubleshooting Framework for Microservices on Multi-source Data. *arXiv preprint arXiv:2302.05092* (2023).

[9] Richard Li, Min Du, Zheng Wang, Hyunseok Chang, Sarit Mukherjee, and Eric Eide. 2022. LongTale: Toward Automatic Performance Anomaly Explanation in Microservices. In *Proceedings of the 2022 ACM/SPEC on International Conference on Performance Engineering.* 5–16.

[10] Zeyan Li, Junjie Chen, Rui Jiao, Nengwen Zhao, Zhijun Wang, Shuwei Zhang, Yanjun Wu, Long Jiang, Leiqin Yan, Zikai Wang, et al. 2021. Practical root cause localization for microservice systems via trace analysis. In *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS).* IEEE, 1–10.

[11] JinJin Lin, Pengfei Chen, and Zibin Zheng. 2018. Microscope: Pinpoint performance issues with causal graphs in micro-service environments. In *Service-Oriented Computing: 16th International Conference, ICSOC 2018, Hangzhou, China, November 12-15, 2018, Proceedings 16.* Springer, 3–20.

[12] Ping Liu, Haowen Xu, Qianyu Ouyang, Rui Jiao, Zhekang Chen, Shenglin Zhang, Jiahai Yang, Linlin Mo, Jice Zeng, Wenman Xue, et al. 2020. Unsupervised detection of microservice trace anomalies through service-level deep bayesian networks. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE).* IEEE, 48–58.

[13] Mahsa Panahandeh, Abdelwahab Hamou-Lhadj, Mohammad Hamdaqa, and James Miller. 2024. ServiceAnomaly: An anomaly detection approach in microservices using distributed traces and profiling metrics. *Journal of Systems and Software* 209 (2024), 111917.

[14] Haoran Qiu, Subho S Banerjee, Saurabh Jha, Zbigniew T Kalbarczyk, and Ravishankar K Iyer. 2020. {FIRM}: An intelligent fine-grained resource management framework for {SLO-Oriented} microservices. In *14th USENIX symposium on operating systems design and implementation (OSDI 20).* 805–825.

[15] Haoran Qiu, Subho S. Banerjee, Saurabh Jha, Zbigniew T. Kalbarczyk, and Ravishankar K. Iyer. 2020. Pre-processed Tracing Data for Popular Microservice Benchmarks. https://doi.org/10.13012/B2IDB-6738796_V1

[16] Gagan Somashekar, Anurag Dutt, Rohith Vaddavalli, Sai Bhargav Varanasi, and Anshul Gandhi. 2022. B-MEG: Bottlenecked-microservices extraction using graph neural networks. In *Companion of the 2022 ACM/SPEC International Conference on Performance Engineering.* 7–11.

[17] G Somashekar, A Suresh, S Tyagi, V Dhyani, K Donkada, A Pradhan, and A Gandhi. 2022. Reducing the Tail Latency of Microservices Applications via Optimal Configuration Tuning. In *2022 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS).* IEEE, 111–120.

[18] Guangba Yu, Zicheng Huang, and Pengfei Chen. 2023. TraceRank: Abnormal service localization with dis-aggregated end-to-end tracing data in cloud native systems. *Journal of Software: Evolution and Process* 35, 10 (2023), e2413.

[19] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chao Ji, Dewei Liu, Qilin Xiang, and Chuan He. 2019. Latent error prediction and fault localization for microservice applications by learning from system trace logs. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering.* 683–694.

[20] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chenjie Xu, Chao Ji, and Wenyun Zhao. 2018. Benchmarking microservice systems for software engineering research. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings.* 323–324.