Network Analysis of Microservices: A Case Study on Alibaba Production Clusters

Ghazal Khodabandeh
Brock University
St. Catharines, Ontario, Canada
gkhodobandeh@brocku.ca

Alireza Ezaz
Brock University
St. Catharines, Ontario, Canada
sezaz@brocku.ca

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

ABSTRACT

Having an observation of the microservices connections complexities within a service is essential for system management and optimization. In this study, we analyzed a dataset of microservice traces from Alibaba's production clusters, segmenting call graphs based on services. Using a community detection model, we uncovered the connections between microservices within each service by finding collaborative patterns and dependencies. Expanding our analysis, we identified similarities among service graphs using clustering techniques. These findings provide detailed insights for system optimization and decision-making, offering a roadmap for using the constructed runtime microservices network behavior to improve overall system efficiency and performance.

CCS CONCEPTS

• **Computing methodologies** → **Classification and regression trees**; • **Applied computing** → **Service-oriented architectures**.

KEYWORDS

Service, Microservice, Community Detection, Graph.

ACM Reference Format:
Ghazal Khodabandeh, Alireza Ezaz, and Naser Ezzati-Jivan. 2024. Network Analysis of Microservices: A Case Study on Alibaba Production Clusters . In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3629527.3651842

1 INTRODUCTION

On the topic of microservices architecture, challenges come from the relationships microservices and services have with each other. A service is a self-contained unit of functionality within a software system. Microservices inside a service have relations with each other to make that service function. Also, services can communicate with each other through well-defined interfaces. This will cause complexities in deployment, scaling, and monitoring in a multi-service framework. Each microservice has distinct functions for user handling, business logic, and backend operations. This format of application is different and more complicated than the monolithic application architecture[2]. To address these challenges we should have a deep and wide insight into this kind of system.

Despite extensive research on microservices [2, 7, 11], the application of social network analysis in large-scale industrial settings is uncommon. This oversight is significant in environments needing detailed analysis of microservices interactions. Our study addresses this gap by exploring these interactions within a major industrial context, aiming to enhance system performance and reliability through novel insights and methodologies.

To highlight the dynamics within microservices relationships, our study relies on a dataset containing over 260 million records of call requests across twenty thousand distinct microservices. Collected from Alibaba's production clusters within a one-hour time frame[1], these records span a network of ten thousand bare-metal nodes [9]. This dataset serves as raw data for our study to find the relations of microservices categorization within individual services and extracting similarities between service's call graphs.

Our theoretical framework draws on social network analysis (SNA) to examine microservices architectures, utilizing methods to analyze relationships among interacting units and understand the complex web of service interactions. By employing community detection algorithms, and graph similarity techniques, we identify closely interconnected groups of microservices and categorize the relationships between different services. This approach allows us to uncover patterns of collaboration and dependency within microservices networks, offering insights into system optimization and performance enhancement in industrial contexts.

The main contribution of this paper is the application of social network analysis techniques to analyze the complex interactions within microservices architectures at an industrial scale. We employ community detection algorithms and graph similarity measures to reveal patterns of microservice interconnections. This method enhances our understanding of microservices dynamics, offering a framework for improving system design, performance, and fault tolerance. Our research provides actionable insights for system managers and developers, aiming to improve software architectures' resilience and efficiency in real-world applications.

2 RELATED WORKS AND BACKGROUND

Exploring the architecture of services and microservices, along with discovering patterns and new observations, can help in different aspects of system application. Studies [3, 10] have explored the structural complexities of microservices, employing graph-based techniques to map the complex web of service interactions. Gaidels et al. [6] emphasizes the significance of employing graph-based

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

ACM ISBN 979-8-4007-0445-1/24/05…$15.00
https://doi.org/10.1145/3629527.3651842

[1] https://github.com/alibaba/clusterdata/tree/master/cluster-trace-microservices-v2022

techniques in system analysis. Similarly, S. Luo's examination of the same dataset [8] indicates that identifying similarities within service call graphs can offer valuable insights into the characterization of microservice dependencies and their runtime performance.

These investigations underscore the potential of network analysis in identifying performance bottlenecks and optimizing service orchestration. Furthermore, study [5] highlights the significance of community detection in uncovering latent patterns within microservices networks, suggesting that such methodologies can facilitate a deeper understanding of service dependencies and interactions.

Community detection, a fundamental challenge in network analysis, involves categorizing nodes into distinct groups based on connections, typically focusing on structural aspects [4]. The Louvain method is an algorithm for identifying aggregate connected groups of nodes within a graph. Its primary strength lies in its ability to reveal the underlying modular structure within a network, emphasizing that nodes within the same module share more connections with each other than with nodes outside the module.

Building upon these foundations, our research introduces a novel approach by integrating community detection with clustering algorithms to analyze microservices at an industrial scale. Our study leverages the Louvain method for community detection and the K-means algorithm for clustering, aiming to provide actionable insights into microservices categorization and the optimization of service call graphs.

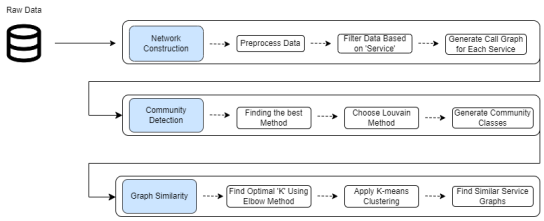

Figure 1: Our system design diagram

3 METHODOLOGY

In this section, we explain our methodology, outlined through three sequential steps as depicted in Figure 1. The main idea is to analyze the relationships among microservices within a singular service architecture by applying the community detection methods as well as analysing the similarities by clustering service graphs, which is the graphs of the microservices inside a unique service.

We start this process by making an accurate dataset for our methodology. This initial phase involves the gathering and preparation of data, ensuring its preparation and conducting subsequent analysis. In the second step, we apply a community detection algorithm to the prepared data. This critical phase plays a key role in categorizing each microservice within a given service into its designated community class, exposing the interconnections. The algorithm identifies patterns and relationships, providing a structured framework for understanding the complex web of associations among microservices.

In the last step, we apply a method that involves an examination of similarities within various service graphs. To find these similarities, we use a clustering method on different service graphs to group them. By analyzing these clusters, we gain insights into the dynamics of different services. Having the results of the second and third steps together can give us an internal observation of the whole system.

3.1 Network Construction

In the present investigation, the Alibaba production clusters data was employed; however, it is necessary to apply preprocessing to construct the network we needed for the rest of our methodology. Our investigation centered on a 1-hour snapshot of this dataset, consisting of over 260 million records and involving more than 28,000 microservices. As depicted in Figure1, this stage includes three sub-steps: preprocessing, filtering, and generating call graphs. Within this phase, the raw data serves as input, and the resulting output consists of graph datasets suitable for the application of community detection and graph similarity methods.

In the preprocessing step, we reduced the unnecessary attributes in the raw data and kept just three columns, focusing on the specific data points crucial for our analysis. We will do this reduction due to the efficiency we will gain from our method with this modified dataset as it would make the methods functioning faster and easier. The resulting dataset included 'um' and 'dm' representing the 'upper microservice' and 'down-stream microservice' in the call request, serving as the foundation for constructing call graph nodes and edges. The 'service' attribute was also retained, providing valuable information for generating the call graph of each service. Rows containing attributes with unacceptable values were omitted.

Following the preprocessing phase, we organize datasets for each service independently by implementing data filtering based on the 'service' attribute. Considering the vast size of our dataset, we opted to focus on a representative sample of all services. As services with a higher number of call requests have a more complex and interesting call graph, we applied a filter to include only service call graphs with a minimum of 50 call requests and randomly selected 300 of them for further analysis.

Following the filtering step, we obtained 300 dataframes, each representing a call graph for a unique service. Within these service datasets, individual rows contain the names of two microservices, with one calling the other on a specific service. Each microservice is treated as a node in the call graph, and if two nodes are present in a single row, a connection is established, resulting in an edge between the corresponding microservice nodes in the call graph. These output datasets facilitate an analysis of the interconnections and dependencies within each service.

3.2 Community Detection

Utilizing the provided call graph dataset, the subsequent step involves creating community classes and assigning each microservice of a service to the appropriate community class. Community detection methods are mainly designed to recognize groups or communities within a network, focusing on the patterns of connections between nodes. In our case, the goal is to demonstrate the relationships between nodes based on their services, we applied the

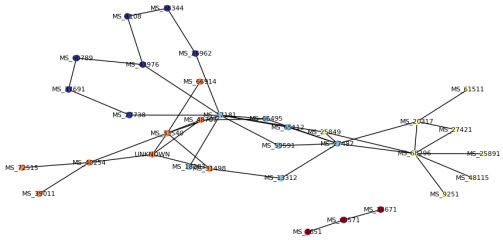

Figure 2: Community classes on a service call graph

community detection method to each service dataset. This process facilitated the identification and categorization of microservices into distinct community classes.

As shown in Table 1, different community detection methods were available for application. To identify the most suitable approach for our dataset, we conducted an evaluation using a sample exceeding 13 million records from the main dataset. The four methods employed for evaluation included the Greedy Modularity method, the Louvain method, the Info-map method, and the Label-propagation method. The assessment was based on key metrics, namely the Coverage, Modularity score, and Silhouette score. Coverage measures the comprehensiveness of community detection methods. In this study, all of the chosen methods have a coverage metric of 1. Modularity score measures the quality of identified community structure. The Silhouette Score, on the other hand, evaluates the cohesion and separation of clusters. These metrics collectively enable an evaluation of the efficacy and quality of community detection algorithms in network analysis. The detailed results of this evaluation are presented in Table 1.

Methods	Silhouette Score	Modularity Score
Greedy Modularity	0.65	0.60
Louvain	0.71	0.67
Info-map	0.58	0.61
Label-propagation	0.63	0.58

Table 1: Evaluation scores for different methods.

As evident in Table 1, the Louvain method has a better performance compared to the other methods under consideration. The Louvain method serves as a robust technique for detecting communities or clusters in complex networks. Its primary goal is revealing the underlying modular structure within a network, emphasizing that nodes within the same module share more connections with each other than with nodes outside the module. As microservices architectures often involve numerous interconnected components, the Louvain method's scalability becomes crucial in handling large-scale networks. Recognized as an enhanced version of the Greedy Modularity algorithm, the Louvain method utilizes a random seed as the foundation for its calculations. To ensure consistent results across different code iterations, we opted to fix the random seed value. To determine the optimal random seed, we systematically varied the seed value from 0 to 1000, selecting the value that yielded the highest Modularity score based on our evaluation criteria. This approach ensured stability and reliability in our community detection results.

To achieve the final results, we applied the Louvain algorithm to the refined dataset obtained in the previous step. The output consists of community classes for each service graph, and Figure 2 displays graphical representations for one selected service call graph. In this plot, nodes represent microservices, and the color of each node signifies its assigned community class. In this picture, nodes in close proximity often share the same community class, reflecting the method's ability to capture patterns in network connections. The color-coded representation facilitates the clear identification of distinct community classes within each service graph. Furthermore, the number of callings between nodes emerges as an essential factor influencing node assignments, providing insights into complex network dynamics and inter-service dependencies captured by the Louvain method.

3.3 Graph Similarity

Building on the previous steps about how microservices relate to each other inside a service, exploring similar service graphs can provide another viewpoint into the entire system. We applied the K-means method to a bunch of service datasets to pinpoint service graphs with similar characteristics. The K-means algorithm uses distinctive features from each graph and then organizes them into 'k' clusters based on these features. This approach helps the categorization of service graphs, providing a deep understanding of the system's diverse patterns and structures. K-Means can handle large datasets and is computationally efficient, making it scalable for systems with a significant number of services. Also, microservices communication patterns may not follow a strict shape, and K-Means can be robust in identifying clusters with irregular shapes. So, in general, this method brings us well to our intended goals for this study.

Within this study, our approach involves clustering graphs based on their structure. Specifically, we extract nodes and edges from each graph, considering them as features important for the clustering process. An additional challenge is in determining the most suitable value for 'k' in this dataset. To address this, we employed the Elbow method [1], which involves testing various values of 'k' within the K-means method and plotting the distribution. Analyzing the plot obtained from the Elbow method, we identified a distinct bend or "elbow," and based on this observation, we determined that setting 'k' to 5 yielded optimal results for our dataset.

Following the execution of K-means on the dataset, we obtained 5 clusters, each including a certain number of graphs. Due to space constraints, we present four out of the five clusters identified in our analysis in Figure 3. To assess the effectiveness of this method, we utilized the Silhouette score. The Silhouette score obtained for our method was 0.6141, indicating a desirable performance for this type of clustering.

As the conclusion of our efforts, the final results yield a series of graphs, providing enhanced insights into the diverse services within the system and their respective structures. In Figure 3, services are organized into distinct clusters based on their overall structure. As a sample, graphs '3a' and '3b' form one cluster, '3c' and '3d' another,

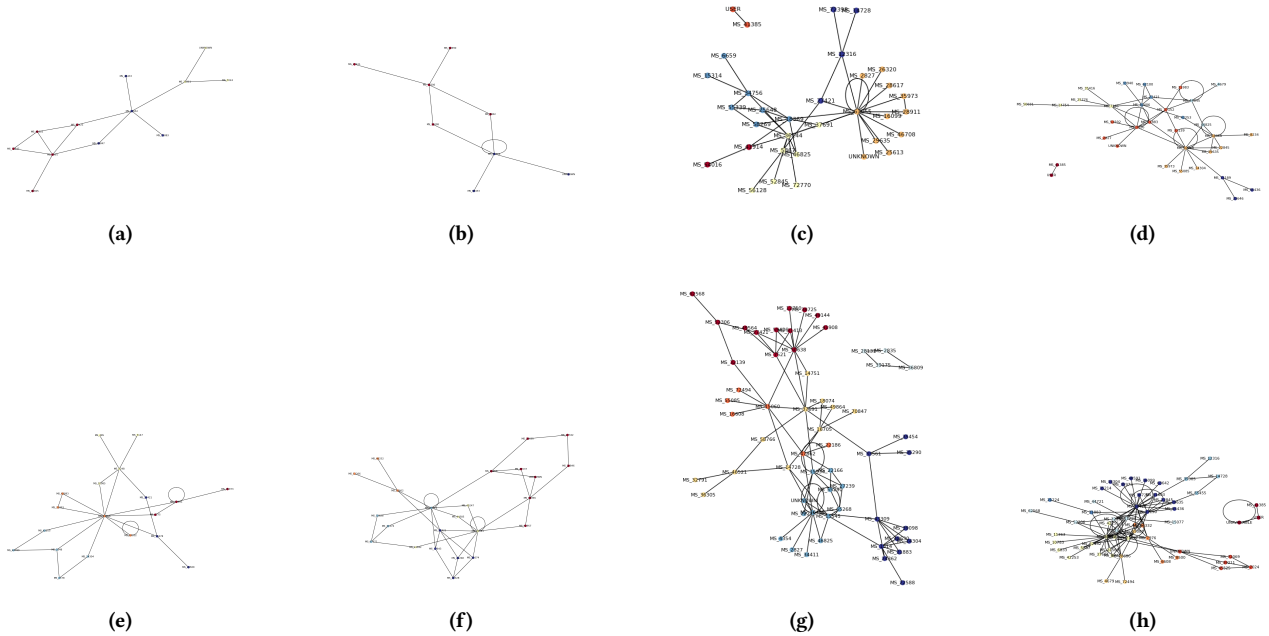

Figure 3: Similarity Clusters on Service Graphs

'3e' and '3f' a third, and '3g' and '3h' constitute a fourth cluster. This analysis contributes to a deeper understanding of the actual runtime configurations present in the system's services. The source codes of the implementations are available from our anonymized GitHub repository [2].

4 DISCUSSION

Upon applying community detection to the randomly selected 300 services, we observed instances where certain services contained only two microservices. In these cases, both microservices would be assigned to the same community class. However, as the service graph complexity increased, there would be more microservices and community classes, and finding a strong connection among microservices within the same class would be worth more. As you can observe in Figure 3, more complex graphs, will cause a heightened frequency of calls and more direct communication between microservices within a class. By identifying these relationships, we can pinpoint areas of latency or bottlenecks in the system. In case of issues, detecting the faulty microservice and its collaborators becomes more feasible. Additionally, this knowledge enables effective isolation of issues, preventing them from affecting other parts of the system. A clear understanding of how different microservices interact during development and debugging is essential. Developers require insights into data flow, dependencies, and communication protocols between microservices to write effective code and troubleshoot efficiently.

In this study, we pursued the identification of similar service graphs based on their microservice connections. This approach provides several advantages for system management and decision-making. One such benefit is evident in resource allocation and scaling, where similar services often exhibit comparable patterns in resource usage and demand. Efficient resource allocation becomes possible by recognizing these similarities. Performance benchmarking is another advantage, allowing for comparisons of microservice relations and performance metrics among similar services. This benchmarking process helps us continually improve and make services work better in the whole system. Importantly, this perspective enhances decision-making by highlighting the similarities among services, playing a critical role in strategic considerations for the system's development.

[2]https://github.com/ghazalkhb/ICPE2024_DataChallenge

5 CONCLUSIONS AND FUTURE WORK

We conducted an analysis of the relationships and communications among microservices within a service, as well as the similarities between different services. This achievement was realized through the application of community detection on microservices within a service, followed by clustering the resulting service graphs. The outcomes of this investigation provide numerous advantages for system performance and management.

While the current study focused on existing attributes, the inclusion of additional attributes for each node or service could further enhance our insights. Future works could involve incorporating response times for each connection, offering a more delicate understanding of the system's dynamics. Furthermore, scaling up the study to a larger dataset would provide a broader observation of system functionality. Another avenue for exploration is the examination of additional methods for community detection and clustering algorithms or even the combination of multiple methods. Integrating machine learning techniques to predict future dynamic network behaviors and community formations represents another promising direction.

REFERENCES

[1] Purnima Bholowalia and Arvind Kumar. 2014. EBK-means: A clustering technique based on elbow method and k-means in WSN. *International Journal of Computer Applications* 105, 9 (2014).

[2] Grzegorz Blinowski, Anna Ojdowska, and Adam Przybyłek. 2022. Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation. *IEEE Access* 10 (2022), 20357–20374. https://doi.org/10.1109/ACCESS.2022.3152803

[3] Álvaro Brandón, Marc Solé, Alberto Huélamo, David Solans, María S Pérez, and Victor Muntés-Mulero. 2020. Graph-based root cause analysis for service-oriented and microservice architectures. *Journal of Systems and Software* 159 (2020), 110432.

[4] Petr Chunaev. 2020. Community detection in node-attributed social networks: a survey. *Computer Science Review* 37 (2020), 100286.

[5] Ruomeng Ding, Chaoyun Zhang, Lu Wang, Yong Xu, Minghua Ma, Xiaomin Wu, Meng Zhang, Qingjun Chen, Xin Gao, Xuedong Gao, et al. 2023. TraceDiag: Adaptive, Interpretable, and Efficient Root Cause Analysis on Large-Scale Microservice Systems. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1762–1773.

[6] Edgars Gaidels and Marite Kirikova. 2020. Service dependency graph analysis in microservice architecture. In *Perspectives in Business Informatics Research: 19th International Conference on Business Informatics Research, BIR 2020, Vienna, Austria, September 21–23, 2020, Proceedings 19*. Springer, 128–139.

[7] Shanshan Li, He Zhang, Zijia Jia, Chenxing Zhong, Cheng Zhang, Zhihao Shan, Jinfeng Shen, and Muhammad Ali Babar. 2021. Understanding and addressing quality attributes of microservices architecture: A Systematic literature review. *Information and Software Technology* 131 (2021), 106449. https://doi.org/10.1016/j.infsof.2020.106449

[8] Shutian Luo, Huanle Xu, Chengzhi Lu, Kejiang Ye, Guoyao Xu, Liping Zhang, Yu Ding, Jian He, and Chengzhong Xu. 2021. Characterizing microservice dependency and performance: Alibaba trace analysis. In *Proceedings of the ACM Symposium on Cloud Computing*. 412–426.

[9] Shutian Luo, Huanle Xu, Kejiang Ye, Guoyao Xu, Liping Zhang, Guodong Yang, and Chengzhong Xu. 2022. The power of prediction: microservice auto scaling via workload learning. In *Proceedings of the 13th Symposium on Cloud Computing* (San Francisco, California) *(SoCC '22)*. Association for Computing Machinery, New York, NY, USA, 355–369. https://doi.org/10.1145/3542929.3563477

[10] Vinay Raj and Ravichandra Sadam. 2021. Evaluation of SOA-based web services and microservices architecture using complexity metrics. *SN Computer Science* 2 (2021), 1–10.

[11] Victor Velepucha and Pamela Flores. 2023. A survey on microservices architecture: Principles, patterns and migration challenges. *IEEE Access* (2023).