# Service Call Chain Analysis for Microservice Systems

Zhiqiang Hao[1], Xufan Zhang[1], Jia Liu[1], Qing Wu[2*]

[1] Software Institute, Nanjing University, China
[2] Business School, Nanjing University, China
leetguai@163.com, xufan.zhang@outlook.com, liujia@nju.edu.cn, wuqing@nju.edu.cn

## Abstract

Industrial practitioners widely adopt the microservice architecture to build applications. An application with microservice architecture can be composed of a set of individual services. Although microservice can improve the scalability of a system by isolating services, the complexity and difficulty of defect detection and analysis grow. High verification cost, a long feedback cycle, and high communication cost pose challenges to the maintenance of microservice systems. To address the problem, we propose a call chain tracing and analysis approach designed for the microservice architecture. To evaluate its effectiveness, we implement our approach as a plugin, namely Cam, to monitor and analyze exceptions by tracing call chains. Currently, Cam is packaged as a maven plugin for applications using Spring Cloud, which is an opensource microservice framework for Java programs. We experiment it with a microservice system to demonstrate its availability. The result shows that Cam can help software engineers understand the workflow of a service call and locate potential defects.

**Keywords:** Call chain tracing, Microservice, Exception analysis

## 1 Introduction

Microservice systems are characterized by implement-ing the service-independent software architecture. Developers can develop and maintain service nodes independently, which realizes the decoupling of the system and improves the scalability. When a node fails, other nodes are relatively independent, improving the reliability of the system. The existing analysis of microservices architecture is based on few nodes [1]. However, in the microservice architecture, with the expansion of the overall server function, the number of service nodes increases, and the call relationship between services becomes more complex. In consequence, it is hard to deal with the relationship and evolution of service parts. The operation status of the whole system will be challenging to grasp because each microservice maintains its log separately. A request often needs to be processed by multiple distributed service nodes, making it challenging to locate the exception. These problems bring significant challenges to the operation and maintenance of the system.

It is difficult to grasp the overall system status under the microservice architecture. Moreover, it is challenging to locate the problem in the complex call relationship. To solve the problems of microservice systems in reality, i.e., for the sake of monitor of the overall system status and management of exceptions, we need a call chain monitoring and analysis approach for the microservice architecture systems [2]. Therefore, existing approaches cannot clearly analyze the tracing process of an approach for the microservice call chain.

To propose a novel approach, we think about detecting exceptions, locating them, and notifying operations and maintenance developers of exceptions. More specifically, this approach should implant span information, trace, and monitor microservice nodes [3]. Then exception handling task requests and push exception handling tasks are created.

This approach is described as using a tracer to solve intercepting and tracing span information [4]. It first intercepts the span information. According to the analysis of span information, the continuous monitoring and exception handling of the system state can be realized. After intercepting and storing the data reasonably, this approach should provide an intuitive interactive interface so that the overall situation of the system and the specific situation of each node in real time can be better understood. Moreover, visualizing those data could help the operation and maintenance developers understand the call relationship and find problems [5]. In addition, when a system exception occurs suddenly, the approach should locate the system exception and notify the operation and maintenance developers. Ultimately, we need to turn the results of our research into a tool. This tool was evaluated and verified to be more available than other tools.

The main contributions are as follows:

1) we propose a call-chain-based analysis approach for microservice systems. This novel approach can monitor the overall status, locate and handle exceptions for microservice systems.

2) the processes of solving problems correspond to each function of the tool. The effectiveness of the approach is verified by verifying the availability of functions.

## 2 Background

### 2.1 Microservice

The approach is aimed at the tracing, monitoring, and processing of microservices. Since there is not a formal definition for microservice, it is easy enough for you to call whatever you do a microservice architecture. What you call your system is relatively unimportant, but there are two crucial feature descriptions: microservices are small, autonomous

*Corresponding Author: Qing Wu; E-mail: wuqing@nju.edu.cn

DOI: 10.53106/160792642022112306004

services that work together; loosely coupled service-oriented architecture with bounded contexts [6].

A microservice is an independently deployable component of bounded scope that supports interoperability through message-based communication. Microservice architecture is a style of engineering highly automated, evolvable software systems made up of capability-aligned microservices [7]. Microservice can improve software delivery speed as functional scope grows, because it has greater agility, higher composability [8], improved comprehensibility, independent service deployability and organizational alignment. And microservice can maintain software system safety as scale increases because it has higher availability and resiliency, better efficiency, independent manageability and replaceability of components, increased runtime scalability, and more simplified testability. The ideal technological environment for microservices features cloud infrastructure [9],which facilitates rapid provisioning and automated deployment. The use of containers is particularly useful to enable portability and heterogeneity. Middleware for data storage, integration, security, and operations should be web API-friendly in order to facilitate automation and discovery, and should also be amenable to dynamic, decentralized distribution. The ideal programming languages for microservices are API friendly as well and should be functional while also matching the skill sets of your organization. It is particularly useful to provide tools for developers that simplify their tasks yet incorporate constraints that encourage good operational behavior of their resulting code.

## 2.2 Remote Procedure Call

To complete the call between microservices, RPC (Remote Procedure Call) Protocol is needed. The mainstream RPC frameworks include Alibaba's Dubbo, Facebook's thrift, Hadoop's Avro, etc.

The RPC provider maps an ID for each service. The parameters of the service are sent through serialization and received through deserialization. When starting, it registers services with the registry according to the information configured in the service publishing file server.xml, caches the list of service nodes returned by the registry in local memory, and establishes a connection with RPC server [10].

As shown in Figure 1, RPC client calls the service. When starting, it subscribes the service to registry according to the information configured in the service reference file client.xml, caches the list of service nodes returned by the registry in the local memory, and establishes a connection with RPC client.

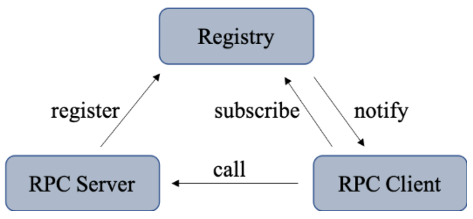


**Figure 1.** Interaction diagram of role relationship of microservice

The RPC client selects an RPC server from the list of local cache service nodes based on the load balancing algorithm to initiate the call.

When the RPC server node changes, the registry will be cached in the local memory after sensing the change.

## 2.3 Call Chain

In the process of a call, the call information (time, interface, level, result) between services is punctured into the log. Then all the data of the points are connected into a tree chain, and a call chain is generated [11]. The whole process of a request call is connected in series through the call chain, and the monitoring of the request call path is realized, which is convenient for rapid fault location.

We often use call chains in microservice because the call chain can well show the relationship between service nodes. However, the call chain is complex, and when services run into problems, it is difficult to locate. The overall system performance and operation need to be clearly reflected to adjust resources according to the actual situation. Therefore, it is necessary to analyze and monitor the call chain of microservices. The relationship data between microservices can be analyzed, and then we can locate exceptions.

The tracing system analyses and processes the log information [12] generated in the process, restores the complete call process of end-to-end business execution, and makes statistical analysis according to different dimensions; In this way, the abnormal service calls can be identified, and the abnormal services can be quickly analyzed and delimited. At the same time, the system performance bottleneck can be analyzed according to the data statistics [13].

All nodes of a service call are concatenated to form a call chain. A request is a trace, and a call in a request is a span. We can give the call chain a TraceID [14]. In addition to TraceID, SpanID is required to record the calling parent-child relationship. Each service records the TraceID and SpanID attached to the request as the ParentID, and also records the SpanID generated by itself. Each microservice records the ParentID and SpanID, through which the parent-child relationship of a complete call chain can be organized. To view a complete call, just find out all call records [15] according to TraceID, and then organize the whole call parent-child relationship through ParentID and SpanID.

## 3 Approach

To realize monitoring the system's overall situation and locate the system exceptions, as shown in Figure 2, our workflow mainly includes the following steps: First of all, we need to intercept call data between node direction information. Then we process the intercepted data into structured data because structured data is easy to analyze exceptions and display visually. Through data structure, entity data is generated. After entity data is generated, microservices are aggregated for granular data. Then we weighted out the indicators needed for diagnoses, such as microservice health and abnormal call risk level. We analyze the data to find out whether the exception exists. If an exception exists, we need to locate it. After pushing, the detected anomalies are handed over to the operation and maintenance personnel for handling.

If the exception does not occur, the target system is continuously monitored.

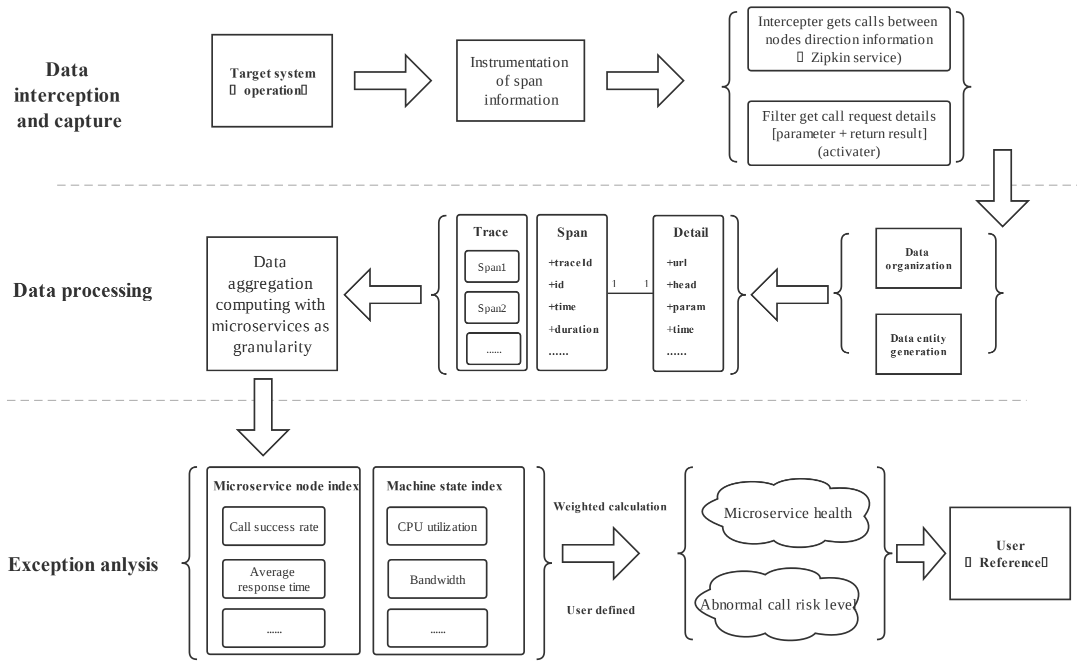


**Figure 2.** Workflow of microservice call chain analysis

## 3.1 Data Interception and Capture

Call relationship data count for much in our analysis, but the call relationship data might be simple or complex in a service call [16]. In addition, the form of the call chain is uncertain, which leads to difficulty in specifying the relationship data. Different data requires different data structures. Therefore, we need to design a reasonable data structure for different scenarios to carry the call relationship data and transfer the call chain information to the front end completely, reasonably, and easily parsed to provide convenience for subsequent data visualization.

So, in our approach, we need to configure some components to intercept the call relation data, including the path of this call, the ID of each calling node, and its parent ID [17]. The root node has no ParentID. The path of this call can be set to TraceID, which can be used to indicate which nodes a call passes through in proper order. The ID of each calling node and its ParentID can be set to SpanID and ParentID. The service of each node will record the ParentID and SpanID, through which the parent-child relationship of a complete call chain can be organized. Span without ParentID becomes root span, and the other span can be expressed recursively.

When an RPC call is initiated, the system puts TraceID into the HTTP request and records the timestamp. When a request reaches a node, the SpanID of the node will be generated as the relative position of the node in the whole call chain. Then, the intercepted call relation data is stored in the database. Finally, we can query according to the API provided by some distributed tracing systems. If there is a TraceID in the log file, you can jump to it directly. Otherwise, you can query the call relationship data based on attributes such as service, operation name, label, and duration.

In the intercepted HTTP request [18], the specific request parameters, request approaches, request results and return values of the request need to be focused on, which are more specific data information with smaller granularity. Trace and other call chain information are recorded in the head of the HTTP request in advance and stamped with a time stamp. After it is intercepted by a program similar to the sensor, the call, duration, and other call chain information are recorded. The tracing function can be realized by comparing the time stamp and other call chain information. But when intercepting these data, we cannot simply take the data from the response object. At the same time, we need to make a backup of data and store it, which does not affect the completion of the request itself.

There is the practical significance of intercepting the specific content of each HTTP request because the operation and maintenance personnel can monitor and analyze the call chain-related functions of the system through the microservice call chain and find that the abnormal rate of each service node is too high, and then check the active call and passive call of the node. It is found that an exception always occurs in a call chain when the data request arrives at the node, resulting in the failure of the full request of the call chain. At this time, the operation and maintenance personnel will view the details of the abnormal request under the node, including the request parameters, the returned abnormal error information, etc., and then judge the cause of the exception or seek the help of the R&D personnel.

Besides, we should record the log file of each node [19]. But we cannot use the traditional log file format. We should structure the log file to a certain extent. Structuralization helps to optimize storage and improve query efficiency. Adding the key value in the structured log file is helpful to distinguish the log information and avoid querying too much log information in the later query.

In brief, when the client sends a request to the server, cam intercepts it and generates span information. After processing, Cam will then hand over the request to the server for processing. The span information includes TraceID, SpanID, span type, response time, etc., and stores the span in the database through other components.

## 3.2 Data Processing

After collecting the call chain relationship data, we need to process and count the collected data. Multiple service nodes cooperate to achieve the function, but some nodes have high average access frequency, while others have low access frequency. Similarly, the average access time of some nodes is longer, and that of some nodes is shorter. Therefore, we pay different attention to each node. For example, we can focus on the node with a longer average access time, focus on the statistics of the load pressure brought by this node, and evaluate its impact on the whole system.

Based on our different attention to different nodes, we rank these nodes to distinguish our attention to each calling node. After sorting each node, the operation and maintenance personnel can assign physical resources to the node according to such characteristics and prioritize solving the performance and abnormal problems of essential nodes. At the same time, they ask the R&D personnel for better peak bearing capacity in R&D design.

The module generates span information according to the obtained records, and the index characteristics of span information generated by each node will be different. The tool can map this span information into symbolic information, which is convenient for later visualization. For example, we can associate node volume with the number of service calls and associate node color with service health [20].

Since each node has its own business after the distributed microservice of the service, we need to split the log file and map it to each node [21]. Each log file corresponds to the microservice of the target system one by one. If the R&D personnel has located the log information to be viewed, they need to obtain all the specific log file lists first and then obtain the specific logs. Moreover, time information is often used as the partition key of the log file, which is the first attribute to filter when locating logs. Therefore, we can divide the log file into the data according to the time information.

When we structurize the call chain data, span information, and log file, we can monitor these data and carry out the operation and maintenance of the system. For example, we can get the call chain data in the past day, then get all the response time of each node by traversing each span, and finally get the average response time of each microservice node.

## 3.3 Exception Analysis

In the running phase of the system, span information is collected into a collection. At the same time, we need to restore the tree call through SpanID and ParentID. Then we can find the specific exception by extracting span information and comparing the exception table. The exception table can be established according to the experience of developers, which records the call parameters of standard exceptions [22].

In addition, according to the data processing, we get the average response time and average number of calls per node. In order to detect exceptions, we set thresholds for the analyzed data. These thresholds include call times, call success rate, average call depth, and other state information. By comparing the analysis data stored in the database with the threshold [23], we can find the exception in real-time.

After catching the above exception, we need to process the exception data to facilitate the visualization of the interactive interface. We process abnormal data into microservice evaluation indicators. Microservice evaluation indicators mainly include microservice node health and abnormal call risk level. The health degree of the microservice node is weighted by the health degree of node instance and the health degree of the environment [24], representing the current health state of the microservice node. Let H denote the health of the microservice, $H_e$ and $H_n$ denote the health of the environment and the health of the instance. $K_e$ and $K_n$ denote the weights of environment health and instance health, respectively. $K_i$ and $K_j$ denote the weights of the machine state index and the weights of the node state index, respectively. $S_i$ and $S_j$ denote the machine state and the node state index. $H_e$ is calculated by weighting the machine state index. $H_n$ is calculated by weighting the node state index. Index calculation Calculate the metrics of each microservice node every day through scheduled tasks and store them in the database. The operator can be described as:

$$H_e = \sum K_i S_i \quad (K_1 + K_2 + \cdots + K_i = 1) \tag{1}$$

$$H_n = \sum K_j S_j \quad (K_1 + K_2 + \cdots + K_j = 1) \tag{2}$$

$$H = K_e H_e + K_n H_n \quad (K_e + K_n = 1) \tag{3}$$

The risk level of exception call is the same as the health degree of the microservice node, but the indicators used are different. The risk level of the exception call is equal to the importance of the microservice node in the whole system. At present, it is mainly calculated by the node call times, average call depth, and the number of microservices related to the node. It can also be calculated by the user-defined combination of indicators.

Finally, if we catch an exception, we need to set up different exception tasks according to different exceptions. Exception handling tasks must have indicators such as completion status and processing progress to indicate the status of the exception handling task [25]. We will mark the captured exception in advance in the call chain and establish the corresponding exception handling task for the located exception. To view the details of the corresponding exception calls in the exception handling task, you can carry out the operation and maintenance of the system. When the developer completes the repair of the exception, the status of the exception handling task will be changed to complete. At this time, the system will ask the user to fill in the exception handling task feedback report and create the exception task feedback to store in the database.

## 3.4 Exception Feedback

When the tool analysis determines the abnormality, it needs timely feed back to the operation and maintenance personnel. We mainly carry out exception feedback from two aspects: real-time interface update and exception message notification [26].

After processing the relational data, the tool generates visual charts such as the ranking chart and the line chart. These charts together constitute the monitoring page of the system. The monitoring page mainly presents the data chart of the target system to users, mainly describes the real-time data amount and change trend of the service node's access times, the number of call chains, and other information in a period by a data line chart, statistical ranking, and other ways, to provide the data basis for operation and maintenance, data analysis and other work.

The monitoring panel presents the data by the statistical chart, and we can quickly understand the operation of the target system from the perspective of crucial data. At the same time, the front end also presents the data in a directed graph. We can understand the system from the perspective of service nodes. The advantage of this is to show all nodes in a graph, which constitutes the global call graph page of the system. The page shows the global call diagram of all service nodes of the target system in the way of service node as directed graph node and the call relationship as the edge. The number of calls is associated with the node volume. The node color is related to the service health level, which reflects the overall and local operation relationship of the target system.

When a vital exception suddenly occurs, we hope that the tool can push messages timely. These messages should at least be displayed on the interactive interface. At the same time, the relevant person in charge can be informed through the WeChat push or email push. After receiving the exception push, we can create our own exception handling task or directly use the default exception handling task. The default exception handling task should be automatically created by the tool and pushed by the tool.

In order to facilitate maintenance and repair by the operation and maintenance personnel, we hope that the tool can click specific node requests in the expanded call chain. The tool will expand the specific span information and HTTP request details of the node request.

## 4 Threats to Validity

Putting the generated span information into the request will invade the code. First, the intrusive code increases the monitoring resource consumption, resulting in a certain request delay. Secondly, code intrusion may affect the security of code.

Microservice monitoring often needs to generate APM (application performance monitor) reports for report analysis. Cam lacks the generation of APM report, so it lacks the analysis of the time dimension.

## 5 Evaluation

In order to evaluate the approach, this approach is implemented into a tool, then evaluate each module of the tool, and evaluate the whole tool. The problem we want to solve corresponds to the function of the module. Functions that change for the same reason are grouped together to form a microservice, and functions that change for different reasons are placed in different microservices [27]. Through the function division, the modules are divided.

This tool after division contains the call chain information module, the HTTP request management module, the log information module [28], the service evaluation module, the task push module, and the exception management module. Call chain list page loading, call chain details page loading, exception handing task creation request, exception handling task feedback request, microservice instance node creation request, and microservice health index calculation request is realized through these modules [29].

Cam focuses on tracing, monitoring, and exception handling of the microservice call chain. Specifically, it should display the list of service nodes in the call chain, display the details of service nodes, evaluate the health of microservices, create exception handling task requests, and provide push and feedback of exception handling tasks.

To determine the timeliness and availability of the tool, each module is tested. Some test cases are created to test functions. Intercepting the test case of calling data tests whether the system can accurately intercept and store the data into the database when the service nodes of the target system-call each other. Visual data real-time display of test cases, the main test target system after the service node call, the system can immediately provide the latest call relationship information to users. The HTTP request data accuracy test case mainly tests whether the time information of the intercepted HTTP request data is consistent with the time information of the intercepted call chain information data when the request call occurs in the target system, and provides us with the query function of the specific request data of the call chain. Log information storage and acquisition test cases, the main test is that the tool can store the log information correctly as log files in MongoDB through the provided information storage interface [30]. All users can query the log information accurately through the log query page. The microservice node test cases test whether we can adequately perform the creation of the microservice node instance. Service evaluation information query test case is used to verify whether users can normally query the health-related indicators of microservice nodes and the risk level related indicators of exceptional calls. Modifying push related information test case is used to verify that users can modify personal push-related information properly. Creating push task test cases is used to verify whether users can create push tasks typically. The exception call information query test case is used to verify whether the system can accurately load the corresponding exception call information list when enter various query conditions. The exception handling task creates a test case to verify that we can create an exception handling task based on a specified exception call, assign it to a specified developer, and push it as we choose.

By using Postman and Fiddler interfaces to debug the packet capture tool [31], the testers conducted 50 request operations for specific functional interfaces such as system call chain information management, HTTP request management, log information management, service evaluation, task pushing management, exception management. Test whether the function of the interfaces is available, record maximum and minimum response time, and count the average response time of requests. Finally, these data are counted into a table.

Through the evaluation of push functions, compare whether the results of these functions are consistent with the actual situation, verifying that Cam can monitor the system conditions such as the call relationship of microservice system; Through the evaluation of push function, compare whether the results of these functions are consistent with the actual situation, verifying that Cam can trace, locate and deal with the abnormal conditions of a microservice system.

In this test, the Alibaba Cloud ECS cloud host instance is deployed, and the target monitoring system is also deployed and run on the Alibaba Cloud host. The server uses the CentOS 8.0 system, 2-core CPU, 4G memory, and 40G solid-state drive. Table 1 is the hardware configuration for the test environment.

**Table 1.** Cloud server hardware and environment configuration

| Hardware item | Remark |
|---|---|
| Operating system | CentOS 8.0 |
| CPU | 2 cores |
| RAM | 4G |
| SSD | 40G |
| Quantity | 4 |

Each server is installed with MySQL as the database, and the target monitoring system uses three servers to distribute microservice nodes. Table 2 is the software configuration of the test environment.

**Table 2.** Software configuration of the test environment

| Software item | Remark |
|---|---|
| JDK | Java8 |
| Container service | Docker |
| Database | MySQL |

The experimental results are shown in Table 3. Through the test of those modules, the functions are evaluated, verifying the effectiveness of the method.

The feedback time of the functions is within 3 seconds. 80% of the users participating in the test are satisfied with the feedback time, and 10% of the users say that the feedback time of the tool still needs to be improved [32]. The test results show that Cam can reasonably complete the purpose of microservice tracing. Moreover, Cam’s feedback time is neither too long for users to wait and feel bored nor too short for users to have no response time.

To sum up, Cam can realize more functions and meet more affluent requirements than other tools. This approach effectively processes the call chain data, solves the problems of monitoring the microservice call chain and the feedback and processing of exceptions, and has achieved more effectiveness. Cam is also within the tolerance of most users in terms of efficiency and provides reasonable feedback time.

**Table 3.** Performance requirement test

| Function | Average response time | Maximum response time | Minimum response time | Test result |
|---|---|---|---|---|
| Call chain list page loading | 1223.5 ms | 2145.8 ms | 821.3 ms | success |
| Call chain details page loading | 1434.2 ms | 1613.1 ms | 1042.16 ms | success |
| Exception handling task creation request | 634.1 ms | 803.7 ms | 512.4 ms | success |
| Push task creation request | 541.3 ms | 843.1 ms | 453.6 ms | success |
| Exception handling task feedback request | 624.8 ms | 813.7 ms | 422.4 ms | success |
| Microservice instance node creation request | 536.4 ms | 764.9 ms | 403.9 ms | success |
| Microservice health index calculation request | 1743.7 ms | 2143.7 ms | 1466.5 ms | success |

Compare Cam with the current mainstream call chain monitoring product - SkyWalking, and compare their instrumentation methods, performance loss, data types collected, and service dependency graph display. Cam uses invasive instrumentation, while SkyWalking uses bytecode-enhanced non-invasive instrumentation. But SkyWalking uses non-intrusive instrumentation to run additional monitoring applications, which increases performance loss. Using intrusive instrumentation can customize the type of data collected, which is more flexible than non-intrusive instrumentation. At the same time, SkyWalking needs to generate APM reports, which also increases the performance loss. Cam’s service dependency graph is simple and intuitive, while SkyWalking displays all the collected information, which is relatively complex. The comparison between Cam and SkyWalking is shown in Table 4.

**Table 4.** Cam vs SkyWalking

| Item \ Product | Cam | SkyWalking |
|---|---|---|
| Instrumentation method | invasive | bytecode-enhanced |
| Performance loss | low | high |
| Data types collected | customized | stable |
| Service dependency graph display | intuitive | complex |

## 6 Related Work

### 6.1 Call Chain Analysis

Google was the first to discuss call chain tracing techniques [33], and it released the first distributed tracing tool, namely Dapper [34]. The tracing system for Dapper's distributed services needs to record all the work done in the system after a specific request. Dapper's tracing architecture is like a tree embedded in RPC calls, but not limited to this. In the Dapper tracing tree structure [35], the tree node is the basic unit of the entire architecture, and each node represents a reference to span in the log file. Dapper uses trace to represent a complete request call chain, and a single remote procedure call is represented by span. At start and end times, Dapper uses spans to represent parent-child relationships between nodes. Tracing is done by implanting information such as SpanID into threads and collecting span information.

Zipkin is an open-source project for Twitter based on Google Dapper [36]. Zipkin also uses span and trace to implement Dapper's core functions. As an open-source distributed tracing tool, Zipkin uses the approach of intercepting requests to collect real-time call data from various heterogeneous modules of the distributed system [37]. Zipkin can only trace and monitor call chains, and it cannot complete the functions of exception handling task creation and push, so Zipkin has some limitations. But Zipkin supports tracing and has high scalability, so our approach is extended based on Zipkin.

Lan et al. [38] proposed a novel approach to obtain the local composite service dependencies [39] and their discontinuous dependencies through call chain analysis. This approach divided service mining into four steps: data aggregation, service dependency set aggregation counting, service local dependency mining, and discontinuous dependency mining. They thought these complex dependencies could provide primary supporting data for the dynamic deployment and adjustment of microservices.

### 6.2 Microservice Analysis

Ma et al. [40] pointed out three main tasks in microservice testing, i.e., visualizing the dependency relationships between microservices, detecting cyclic dependency references, and improving the coverage of service tests. In order to achieve these goals, construct a service dependency graph (SDG) to collect all service invocation links [41]. Visual display SDG so that users can collect all service dependencies for analysis.

Cortellessa et al. [42] aim at detecting and resolving performance antipatterns by leveraging the traceable relationship between monitoring data and architectural models. They propose some approaches to identify and solve the performance of microservice systems. These approaches include collecting a more extensive set of metrics and performance measures from the running systems, translating refactoring actions into refactorings applied to a running system.

Tianrui et al. [43] attempt to build an application of microservice architecture in a battery monitoring system. They made the following conclusions about microservices: When calling each other within the service, the registry provided by Eureka to the local area can complete load balancing through feign component, which reduces the pressure of a single server and ensures the stability of the system. Therefore, the coordination relationship between microservices is significant for the rational use of microservice resources.

## 7 Conclusion

This paper proposes a call chain tracing approach designed for the microservice architecture. We implement the approach, namely Cam, to monitor and analyze exceptions by tracing call chains. We experiment with a microservice system to demonstrate its availability.

In our approach, we first need to intercept and capture data. We mainly need to obtain two kinds of data: the HTTP request, the log file. Second, we will deal with the intercepted data to achieve structurally. Structured data is conducive to visual display and conducive to monitoring the overall situation of the system and finding abnormal positioning. After structuring the data, we need to locate the exception. When we find and locate the exception, we give the exception feedback from two aspects: real-time interface update and exception message notification. Then the operation and maintenance personnel and developers can handle the exception handling task.

Our approach analyzes multi-dimensional data, such as target system call relation data, specific request data, log information data, etc. Data analysis obtains the overall running status of all service nodes in the target distributed microservice architecture system, the call chain information of specific nodes, and the specific node request response information. With the help of this approach, more complex system operation and maintenance and exception handling can be carried out for the target system of microservice architecture.

## References

[1] P. D. Francesco, I. Malavolta, P. Lago, Research on Architecting Microservices: Trends, Focus, and Potential for Industrial Adoption, *2017 IEEE International Conference on Software Architecture (ICSA)*, Gothenburg, Sweden, 2017, pp. 21-30.

[2] Q. Huang, D. Zeng, Design of Distributed Microservice Architecture Based on Spring Cloud and Docker, *Microcomputer Applications*, Vol. 35, No. 6, pp. 98-101, June, 2019.

[3] X. Guo, X. Peng, H. Wang, W. Li, H. Jiang, D. Ding, L. Su, Graph-based Trace Analysis for Microservice Architecture Understanding and Problem Diagnosis,

*Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2020)*, New York, United States, 2020, pp. 1387-1397.

[4] H. Mi, L. Du, X. Jin, S. Wei, X. Sun, Q. Li, Research on Tracking Technology of Service Call Chain Based on Microservice Architecture, *2020 International Conference on Virtual Reality and Intelligent Systems (ICVRIS)*, Zhangjiajie, China, 2020, pp. 839-843.

[5] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, W. Li, D. Ding, Fault Analysis and Debugging of Microservice Systems: Industrial Survey, Benchmark System, and Empirical study, *IEEE Transactions on Software Engineering*, Vol. 47, No. 2, pp. 243-260, February, 2021.

[6] I. Nadareishvili, R. Mitra, M. McLarty, M. Amundsen, *Microservice Architecture*, O'Reilly Media, 2016.

[7] D. Salikhov, K. Khanda, K. Gusmanov, M. Mazzara, N. Mavridis, Microservice-based Iot for Smart Buildings, *2017 31st International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, Taipei, Taiwan, 2017, pp. 302-308.

[8] T. Cerny, M. J. Donahoo, M. Trnka, Contextual Understanding of Microservice Architecture: Current and Future Directions, *ACM SIGAPP Applied Computing Review*, Vol. 17, No. 4, pp. 29-45, December, 2017.

[9] D. Lu, D. Huang, A. Walenstein, D. Medhi, A Secure Microservice Framework for Iot, *2017 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, San Francisco, CA, USA, 2017, pp. 9-18.

[10] M. Schroeder, M. Burrows, Performance of Firefly RPC, *ACM SIGOPS Operating Systems Review*, Vol. 23, No. 5, pp. 83-90, December, 1989.

[11] X. T. Jiang, Z. G. Hu, J. B. He, Call Chain Analysis for Low Power Compile Optimization, *Journal of Jilin University (Engineering and Technology Edition)*, Vol. 39, No. 1, pp. 143-147, January, 2009.

[12] M. Macak, D. Kruzelova, S. Chren, B. Buhnova, Using Process Mining for Git Log Analysis of Projects in a Software Development Course, *Education and Information Technologies*, Vol. 26, No. 5, pp. 5939-5969, September, 2021.

[13] A. Husain, B. Kumar, A. Doegar, Performance Evaluation of Routing Protocols in Vehicular Ad Hoc Networks, *International Journal of Internet Protocol Technology*, Vol. 6, No. 1-2, pp. 38-45, June, 2011.

[14] T. Wang, W. Zhang, J. Xu, Z. Gu, Workflow-Aware Automatic Fault Diagnosis for Microservice-Based Applications with Statistics, *IEEE Transactions on Network and Service Management*, Vol. 17, No. 4, pp. 2350-2363, December, 2020.

[15] T. H. Buscher, T. J. Coutre, M. J. Franklin, B. D. Freeman, W. E. Relyea, E. N. Shipley, *Telecommunication Network Arrangement for Providing Real Time Access to Call Records*, United States patent US 5,506,893, April, 1996.

[16] D. Crié, A. Micheaux, From Customer Data to Value: What is Lacking in the Information Chain? *Journal of Database Marketing & Customer Strategy Management*, Vol. 13, No. 4, pp. 282-299, July, 2006.

[17] J. J. Rodrigues, P. A. Neves, A Survey on IP-based Wireless Sensor Network Solutions, *International Journal of Communication Systems*, Vol. 23, No. 8, pp. 963-981, August, 2010.

[18] B. S. Rawal, R. K. Karne, A. L. Wijesinha, Mini Web Server Clusters for HTTP Request Splitting, *2011 IEEE International Conference on High Performance Computing and Communications*, Banff, AB, Canada, 2011, pp. 94-100.

[19] J. H. Andrews, Y. Zhang, Broad-spectrum Studies of Log File Analysis, *Proceedings of the 2000 International Conference on Software Engineering, ICSE 2000 the New Millennium*, Limerick, Ireland, 2000, pp. 105-114.

[20] C. Esposito, A. Castiglione, C. A. Tudorica, F. Pop, Security and Privacy for Cloud-based Data Management in the Health Network Service Chain: a Microservice Approach, *IEEE Communications Magazine*, Vol. 55, No. 9, pp. 102-108, September, 2017.

[21] S. He, J. Zhu, P. He, M. R. Lyu, Experience Report: System Log Analysis for Anomaly Detection, *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, Ottawa, ON, Canada, 2016, pp. 207-218.

[22] A. Brogi, D. Neri, J. Soldani, A Microservice-based Architecture for (Customisable) Analyses of Docker Images, *Software: Practice and experience*, Vol. 48, No. 8, pp. 1461-1474, August, 2018.

[23] L. Qi, S. Meng, X. Zhang, R. Wang, X. Xu, Z. Zhou, W. Dou, An Exception Handling Approach for Privacy-preserving Service Recommendation Failure in a Cloud Environment, *Sensors*, Vol. 18, No. 7, Article No. 2037, July, 2018.

[24] J. Kang, J. Zhang, J. Gao, Improving Performance Evaluation of Health, Safety and environment Management System by combining Fuzzy Cognitive Maps and Relative Degree Analysis, *Safety Science*, Vol. 87, pp. 92-100, August, 2016.

[25] B. S. Lerner, S. Christov, L. J. Osterweil, R. Bendraou, U. Kannengiesser, A. Wise, Exception Handling Patterns for Process Modeling, *IEEE Transactions on Software Engineering*, Vol. 36, No. 2, pp. 162-183, March-April, 2010.

[26] J. Wang, Y. Yang, T. Wang, R. S. Sherratt, J. Zhang, Big Data Service Architecture: a Survey, *Journal of Internet Technology*, Vol. 21, No. 2, pp. 393-405, March, 2020.

[27] L. Gou, Q. Chen, J. Liang, X. Liao, Technical Research and Application Analysis of Microservice Architecture, *2019 International Conference on Computation and Information Sciences (ICCIS 2019)*, Saudi Arabia, 2019, pp. 806-813.

[28] H. Rubira, R. Voivodic, The Effective Field Theory and Perturbative Analysis for Log-density Fields, *Journal of Cosmology and Astroparticle Physics*, Vol. 2021, Article No. 070, March, 2021.

[29] R. Gupta, S. Agrawal, Y. Yang, W. Dec, S. B. Ahmed, *Intuitive Approach to Visualize Health of Microservice Policies*, United States patent application US 15/298,102. April, 2018.

[30] C. Győrödi, R. Győrödi, G. Pecherle, A. Olah, A comparative study: MongoDB vs. MySQL, *2015 13th International Conference on Engineering of Modern Electric Systems (EMES)*, Oradea, Romania, 2015, pp. 1-6.

[31] A. Soni, V. Ranga, API Features Individualizing of Web Services: REST and SOAP, *International Journal of Innovative Technology and Exploring Engineering*, Vol. 8, No. 9S, pp. 664-671, July, 2019.

[32] Y. G. Hong, J. Huang, Y. S. Xu, On an Output Feedback Finite-time Stabilisation Problem, *Proceedings of the 38th IEEE Conference on Decision and Control*, Phoenix, AZ, USA, 1999, pp. 1302-1307.

[33] S. D. Mallanna, M. Devika, Distributed Request Tracing using Zipkin and Spring Boot Sleuth, *International Journal of Computer Applications*, Vol. 175, No. 12, pp. 35-37, August, 2020.

[34] M. Ghasemi, T. Benson, J. Rexford, Dapper: Data Plane Performance Diagnosis of Tcp, *Proceedings of the Symposium on SDN Research*, Santa Clara, CA, USA, 2017, pp. 61-74.

[35] Y. Tanaka, H. Nakada, S. Sekiguchi, T. Suzumura, S. Matsuoka, Ninf-G: A Reference Implementation of RPC-based Programming Middleware for Grid Computing, *Journal of Grid Computing*, Vol. 1, No. 1, pp. 41-51, March, 2003.

[36] H. Wang, W. Fang, A Trace Agent with Code No-invasion Based on Byte Code Enhancement Technology, *2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS)*, Beijing, China, 2019, pp. 405-409.

[37] S. Nedelkoski, J. Cardoso, O. Kao, Anomaly Detection and Classification using Distributed Tracing and Deep Learning, *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, Larnaca, Cyprus, 2019, pp. 241-250.

[38] Y. Lan, L. Fang, M. Zhang, J. Su, Z. Yang, H. Li, Service Dependency Mining Method Based on Service Call Chain Analysis, *2021 International Conference on Service Science (ICSS)*, Xi'an, China, 2021, pp. 84-89.

[39] M. R. Namjoo, A. Keramati, Analysing Causal Dependencies of Composite Service Resilience in Cloud Manufacturing using Resource-based Theory and Dematel Method, *International Journal of Computer Integrated Manufacturing*, Vol. 31, No. 10, pp. 942-960, July, 2018.

[40] S. P. Ma, C. Y. Fan, Y. Chuang, W. T. Lee, S. J. Lee, N. L. Hsueh, Using Service Dependency Graph to Analyze and Test Microservices, *2018 42nd IEEE International Conference on Computer Software & Applications*, Tokyo, Japan, 2018, pp. 81-86.

[41] S. Kaffash, A. T. Nguyen, J. Zhu, Big Data Algorithms and Applications in Intelligent Transportation System: A Review and Bibliometric Analysis, *International Journal of Production Economics*, Vol. 231, Article No. 107868, January, 2021.

[42] V. Cortellessa, D. D. Pompeo, R. Eramo, M. Tucci, A Model-driven Approach for Continuous Performance Engineering in Microservice-based Systems, *Journal of Systems and Software*, Vol. 183, Article No. 111084, January, 2022.

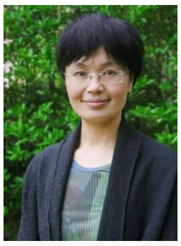
[43] T. Zhao, D. Li, Application of Microservice Architecture in Battery Monitoring System, *Proceedings of 2019 3rd International Conference on Mechanical and Electronics Engineering (ICMEE 2019)*, Chongqing City, China, 2019, pp. 32-36.

## Biographies

**Zhiqiang Hao** received the B.E. degrees in process equipment and control engineering from Nanjing Forestry University, Jiangsu, China, in 2020. He is currently working toward the M.E. degrees in software engineering from Nanjing University, Jiangsu, China.

**Xufan Zhang** received the B.E. and M.E. degrees in software engineering from Nanjing University, Jiangsu, China in 2015 and 2017, respectively. He is currently working toward the Ph.D. degree in software engineering at Nanjing University.

**Jia Liu** received the B.S. degree in computational mathematics, the M.S. degree in information science, and the Ph.D. degree in systems engineering from Nanjing University, Jiangsu, China, in 1998, 2005, and 2012, respectively. He is an Associate Professor with Software Institute, Nanjing University.



**Qing Wu** received the B.S. degree in science, the M.S. and Ph.D. degrees in economy from Nanjing University, Jiangsu, China, in 1998, 2003, and 2009, respectively. She is an Associate Professor with Business School, Nanjing University.