基于日志可视化分析的微服务系统调试方法

李文海 彭 鑫 丁 丹 向麒麟 郭晓峰 周 翔 赵文耘

(复旦大学计算机科学技术学院 上海 201203) (上海市数据科学重点实验室(复旦大学) 上海 201203)

摘 要 云计算时代,越来越多的企业开始采用微服务架构进行软件开发或者传统巨石应用改造。然而,微服务系统具有较高的复杂性和动态性,当系统出现故障时,目前没有方法或者工具能够有效支持对故障根源的定位。为此,文中首次提出通过调用链信息关联单次业务请求在所有服务上产生的业务日志,并在此基础上研究基于日志可视化分析的微服务系统调试方法。首先定义了微服务的日志模型,规范化微服务日志可视化分析所需要的数据信息;然后针对 4 种典型的微服务故障(有异常抛出的普通故障、无异常抛出的逻辑故障、服务异步调用序列未控制导致的故障以及服务多实例版本或状态不一致导致的故障)总结出 5 种可视化调试策略,用于支持对故障根源的定位,5 种策略包括:单条调用链日志查看、不同调用链对比、服务异步调用分析、服务多实例分析以及调用链分段。为了实现服务异步调用分析和服务多实例分析,文中设计了两个算法,同时,设计并实现了一个原型工具 LogVisualization。LogVisualization 可以收集微服务系统运行时产生的日志信息、调用链数据以及集群的节点和服务实例信息,能够以较小的代码侵入性,实现通过调用链信息关联所有业务日志,支持用户使用 5 种策略进行可视化调试。最后,将该原型工具应用于实际的微服务系统,通过与现有工具(Zipkin+ELK)的实验对比,验证了该原型工具在 4 种微服务故障根源定位上的有用性和高效性。

关键词 微服务,调用链,日志,可视化,故障,调试

中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/jsjkx.181102210

Method of Microservice System Debugging Based on Log Visualization Analysis

LI Wen-hai PENG Xin DING DAN XIANG Qi-lin GUO Xiao-feng ZHOU Xiang ZHAO Wen-yun

(School of Computer Science, Fudan University, Shanghai 201203, China)

(Shanghai Key Laboratory of Data Science, Fudan University, Shanghai 201203, China)

Abstract In the era of cloud computing, more and more enterprises are adopting microservice architecture for software development or traditional monolithic application transformation. However, microservice system has high complexity and dynamism. When microservice system fails, there is currently no method or tool that can effectively support the location of the root cause of failure. To this end, the paper first proposed that all business log generated on all of the services by a single request can be associated by the trace information. And on this basis, this paper studied the method of microservice system debugging based on log visualization analysis. Firstly, the model of microservice log is defined. So the data information required for log visualization analysis can be specified. Then five kinds of visual debug strategies are summarized to support the location of four kinds of typical microservice fault's root cause. The four kinds of microservice faults are ordinary fault with exceptions, logical fault with no exceptions, fault caused by unexpected service asynchronous invocation sequences and faults caused by service multi-instances. The strategies include single trace with log information, comparison of different traces, service asynchronous invocation analysis, service multi-instances analysis and trace segmentation. Among them, in order to realize service asynchronous invocation analysis and service multi-instances analysis, this paper designed two algorithms. At the same time, a prototype tool named LogVisualization was designed and implemented. LogVisualization can collect log information, trace data, nodes information and service instance information of the cluster, generated by the microservice system runtime. It can associate the business log with trace information by less code intrusion. And it supports users to use five strategies for visual debug. Finally, the prototype tool is applied to the actual micro-service system. Compared with the existing tools (Zipkin+ELK), the usefulness and effectiveness of prototype tool in the root location of four micro-service faults are verified.

Keywords Microservice, Trace, Log, Visualization, Fault, Debugging

到稿日期:2018-11-29 返修日期:2019-03-28 本文受国家重点研发计划项目(2018YFB1004803)资助。

李文海(1994-),男,硕士生,主要研究方向为移动众包、微服务,E-mail:16212010016@fudan.edu.cn;**彭 鑫**(1979-),男,教授,博士生导师,CCF 高级会员,主要研究方向为代码大数据、智能化软件开发、移动云计算,E-mail:pengxin@fudan.edu.cn(通信作者);**丁 丹**(1994-),女,硕士生,主要研究方向为软件工程;**向麒麟**(1994-),男,硕士生,主要研究方向为软件工程;**郭晓峰**(1996-),男,硕士生,主要研究方向为软件工程;**周 翔**(1983-),男,博士生,主要研究方向为软件工程;**赵文耘**(1967-),男,硕士,教授,博士生导师,主要研究方向为软件工程、软件开发工具及其环境、企业应用集成(EAI)。

1 引言

微服务(Microservices)是一种软件架构风格,它将一个大型且复杂的软件应用分解成多个服务,并将每个服务都作为一个小而独立的系统进行实现和运行,通过定义良好的网络接口提供对其内部逻辑和数据的访问。服务之间的相对独立性,使得软件应用具有更快的交付速度、更好的扩展性以及更强的自治性[1]。微服务是软件服务设计、开发和交付的最新趋势,越来越多的企业选择采用微服务架构进行软件开发或者传统巨石应用的改造[2-4]。例如,腾讯的微信系统由2000多个微服务组成,这些微服务运行在位于多个数据中心的40000多台服务器上[5];Netflix公司花了7年时间(2008年8月—2016年1月)完成从传统的巨石体系到微服务架构的迁移[6]。

尽管微服务有许多优点,而且在工业界得到了广泛使用,但是这一领域依然面临不少的挑战,例如:如何合理地划分服务;如何实现有效的资源监控和管理;如何进行微服务故障调试等[7]。在微服务系统中,可能导致故障的原因是多方面的,除了单个服务本身的实现可能存在问题以外,服务运行时的环境、服务间的交互和协作也存在问题[8]。一般情况下,每个微服务都会包含一些环境配置项,比如 cpu 和 memory,不当的配置可能导致服务在实际运行时出错;同时,服务一般都是多实例部署,理想状态下,微服务应该是无状态的,但是在实际的微服务系统中,一些服务不可避免地需要携带状态[9],如果在系统运行过程中,没有机制实现服务之间的状态同步,也会出现问题。微服务系统在提供服务时,处理单个请求可能会涉及到成百上千个服务,例如:亚马逊为了渲染一个页面,一般会进行100～150次的服务调用[10],任意一个服务异常都可能导致系统故障。而且,服务与服务之间的交互可能很复杂,例如不同服务需要跨节点通信以及服务之间存在异步调用,如果节点与节点之间的网络通信存在问题或者异步调用的某些返回序列存在问题,那么系统也会出现故障。因此,可能导致微服务系统故障的因素较多。故要对故障进行调试,实现根源定位,需要有相应的策略或者工具辅助。

在微服务故障调试的策略方面,Zhou 等[11]从微服务运行时的环境出发,定义了5个维度,利用 Delta Debugging 技术[12]辅助开发人员进行故障调试。但是该策略局限于已定义的维度,无法处理微服务本身实现存在问题而导致测试用例一直失败的情况。在工具支持方面,目前主要有3类工具:第一类是分布式系统的调用链追踪工具,如 Zipkin[13],Jaeger[14];第二类是分布式系统日志的收集及可视化工具,其以 ELK Stack[15]为代表;第三类是分布式系统执行的可视化及对比工具,如 ShiViz[16]。然而,单纯使用 Zipkin 进行调用链追踪只能知道调用链所涉及的服务,而一个服务可能有多个实例,这些实例随机分散在集群的节点上,用户需要人工确定具体调用的服务实例,进而查看与该条调用链相对应的日志;单纯使用 ELK Stack 进行日志的收集汇总,会导致日志信息的上下文缺失,即没有日志和服务的对应关系以及服务之间的调用关系;Shiviz 可以利用分布式系统的执行日志生成服务与服务之间的交互式通信图,同时提供不同系统执行之间的对比功能,方便开发者从服务交互和通信的维度方面进行问题排查。将日志和调用链关联,就是将分散的信息以调用链的维度聚合,从而方便开发者对问题进行定位。这些分散的信息包括:每一次请求所涉及的服务、服务之间的调用关系、调用的服务实例信息以及实例所打印的具体日志。然而,为了能够将日志和调用链关联展示,需要往日志信息中注入调用链信息,如果需要开发者手动修改已有打印日志的代码,以实现日志信息中调用链信息的注入,则对原系统代码具有极大的侵入性。同时,微服务系统在实际运行时会产生成千上万条调用链以及海量的日志信息,需要对这些信息进行一定的聚合分析,才能更好地帮助开发者根据这些信息对微服务故障根源进行定位。

为了解决上述问题,本文从微服务日志可视化分析的角度出发,首先定义了微服务的日志模型,在此基础上明确了可视化任务,并定义5种不同的策略来支持用户对微服务故障根源进行定位,5种策略包括:单条调用链的日志查看、调用链之间的对比、服务异步调用分析、服务多实例分析以及较长调用链的分段。同时,本文实现了原型工具 LogVisualization,该工具通过 AOP 的方式往日志信息中注入调用链信息,从而能够以较小的代码侵入性将调用链和相应的日志相关联。该工具可以收集存储集群的日志、调用链数据以及节点和服务实例信息,并将这些信息以调用链的维度进行聚合分析,实现本文所定义的5种策略,使用者可以通过组合不同的策略来对故障的根源进行定位。最后,本文将该原型工具在 TrainTicket[8]这一微服务系统上进行实验,通过将其与 Zipkin 以及 ELK Stack 进行对比,验证了该原型工具在微服务故障根源定位上的有用性和高效性。

本文的主要贡献点包括:1)首个研究微服务系统的日志可视化。针对微服务系统的特点,提出通过调用链信息关联一次业务请求在所有服务上产生的业务日志。2)定义微服务的日志模型。规范化微服务的日志组成,明确各组成部分之间的关系。3)定义微服务日志可视化的5种策略。微服务系统运行时会产生海量日志,通过所定义的5种日志可视化策略,统计、分析并可视化关键日志信息,能帮助用户从海量日志信息中分析出导致微服务故障的原因。4)设计与实现微服务异步调用分析以及服务分块的算法。5)设计并实现原型工具 LogVisualization,同时通过实际的对比实验验证原型工具的有效性。

2 相关工作

微服务是一个新兴的热门研究话题[17],目前已有的相关工作数量较少,而且现有研究工作主要关注微服务的设计[18]、测试[19-21]、部署[22-24]以及遗留系统改造[25]等方面,还没有相关文献研究基于调用链的微服务日志可视化及其对微服务故障根源定位的帮助。

可视化是利用计算机图形学和图像处理技术,将数据转换成图形或图像在屏幕上显示出来,再进行交互处理的理论、方法和技术[26]。一个经典的可视化实现流程,是先对数据进行加工过滤,再转变成视觉可表达的形式,然后将其渲染成用户可见的视图[27]。日志可视化分析,就是将日志作为原始输

入，过滤提取日志中的关键信息，并以恰当的方式可视化展示。然而，目前的日志可视化研究主要集中于特定领域，包括：网络安全日志分析[28-30]、域名日志分析[31-32]、Web 日志分析[33-34]，暂时没有相关文献提供日志可视化的方法指导，同时也没有相关工作研究微服务日志的可视化。

微服务系统本身是一个分布式系统，在分布式系统的日志收集分析可视化的工具支持方面，目前比较流行的有 Cloudera 公司提供的 Flume[35]、Facebook 公司的 Scribe[36] 以及 Elastic.co 公司的 ELK Stack[37]。Flume 是一种分布式、高可靠且高可用的服务，用于有效地收集、聚合以及传输海量的日志数据。Scribe 则是 Facebook 公司自己开发，同时在公司内部应用的开源日志收集系统，具有高可扩展性和鲁棒性，能够应对网络和节点故障。ELK Stack 是一套流行的一体化日志处理平台解决方案，提供日志收集、处理、存储、搜索、展示等全方位功能，主要由 ElasticSearch，LogStash，Kibana 和 Beats 构成[38]。然而这些分布式系统的日志收集和展示工具并没有特别考虑日志和调用链的关系，只是纯粹的日志收集可视化，没有提供日志和调用链关联展示的功能。ShiViz 是一个可视化引擎，可以根据微服务系统的执行日志生成服务与服务之间的交互式通信图，开发人员可以按需对图的某一部分执行展开、折叠或者隐藏操作。ShiViz 还支持两次系统执行之间的可视化对比，突出显示两次执行之间的差异。然而，为了使用该工具，需要将收集的调用链信息转化为特定格式，同时该工具只能较好地对服务交互方面的问题进行排查，无法直接将每次业务请求和产生的相应日志关联展示，而且在服务多实例问题排查方面的帮助十分有限。

分布式系统的故障定位方法主要包括调用链追踪、日志分析以及可视化[39]。在调用链追踪方面，根据 OpenTracing 的定义，一条调用链是 span 的有向非循环图，而所谓的 span，代表的是该调用链中服务的一次 RPC 调用[40]。目前绝大多数分布式系统的调用链追踪工具都是参考 Google 的 Dapper[41] 实现的。Dapper 是 Google 生产环境下的分布式系统调用链追踪工具，其设计目标有 3 个：低消耗、对应用程序透明以及可扩展性。基于 Google 的 Dapper，Twitter 公司开发了一款分布式追踪系统 Zipkin。Zipkin 收集服务调用的时序数据，将其用于服务监控和故障排查。Zipkin 搭配 ELK Stack 等日志收集工具，虽然可以为每一条调用链附加上日志，但是其具体实现方式，是跳转到 ELK 等日志收集的可视化页面，同时将调用链标识作为过滤条件，自动筛选和展示该调用链上的相应日志。这种实现方式需要在两个页面之间切换，而且每次只能查看单条调用链日志，面对可能导致微服务系统故障的诸多因素，并不能很好地辅助开发人员对故障根源进行定位。而受 Dapper 以及 Zipkin 的启发，Uber 公司开发了 Jaeger 这一分布式追踪系统。相比于 Zipkin，Jaeger 有着更好的客户端库（Go，Java，Node，Python，C++）支持，以便对应用程序进行插装。同时，Jaeger 的内存占用更低，设计上更为现代化，更具可扩展性[42-43]。

同时，目前存在一些针对微服务系统开发的应用性能管理工具（Application Performance Management，APM），比如 Instana[44]。这些工具可以从宏观上对整个微服务系统建模，监控服务与服务之间的调用，帮助开发者对性能问题进行定位，但是也无法深入到服务内部，实现业务日志与调用链的关联展示。

3　可视化任务与调试方法

3.1　方法概述

3.1.1　概念定义

微服务系统在实际运行时会产生海量的调用链和日志信息，为了能够对这些信息进行聚合分析，本文给出以下两个定义。

定义 1（请求类型，Request Type）　一个微服务系统可以处理一种或多种业务请求，比如 TrainTicket 包括登录、查询车次、订票、退票、查询订单等业务请求，每一种业务请求都是一种请求类型。

定义 2（调用链类型，Trace Type）　是对属于同一请求类型的调用链的细分。在实际生产环境中，一种请求类型会有成千上万条调用链与之对应，因为用户可能多次发起该类型的请求。以调用链长度和调用链所经过的服务种类为依据，可以实现调用链的进一步分类。

调用链长度较长时，需要对调用链进行分段，以更好地帮助开发人员对问题进行定位。本文对调用链分段的方法是基于服务依赖图的，该概念的定义如下。

定义 3（服务依赖图，Service Dependency Graph）　表示的是微服务系统在实际运行时，服务与服务之间的依赖关系。服务依赖图是动态生成的，是对所有调用链中服务依赖关系的汇总。服务依赖图的节点是微服务系统运行期间所有被调用的服务，服务与服务之间如果有边相连，则边的方向代表服务与服务之间的依赖关系。

3.1.2　方法概览

基于日志可视化分析对微服务系统调试的方法概览如图 1 所示。

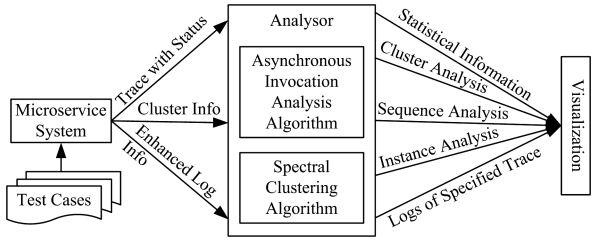


图 1　方法概览

Fig. 1　Approach overview

开发环境下，微服务的每一个业务逻辑都需要通过编写并运行测试用例的方式验证实现的正确性。一个测试用例可能包括多种请求类型，测试用例的每一次执行都会在对应的请求类型下生成一条调用链。同时，同一个测试用例会被执行多次，尤其是在对系统进行压力测试时，同一请求类型下会产生大量调用链，每条调用链经过的服务可能存在差异，形成不同的调用链类型。进一步地，通过测试用例成功与否，可以标识每条调用链的状态，即成功的测试用例产生正确的调用链，而失败的测试用例生成错误的调用链。

本文将调用链数据、日志信息以及集群信息（节点和服务实例信息）作为原始数据输入，提出的方法首先对数据进行统

计分析，主要分析每一种请求类型的错误率、每一种调用链类型的错误率、每一个服务的错误率。通过往日志信息中注入调用链信息，可以将调用链和相应的日志信息相关联。同时，通过算法分析同一调用链类型下的所有调用链和日志信息，这一方面可以分析该类型下存在的服务异步调用以及异步调用存在的调用序列和相应的错误率，另一方面可以分析该调用链类型所经过的所有服务以及服务实例的错误率。最后，通过谱聚类算法对服务依赖图进行分块，在分块的基础上可以实现对调用链的分段。

3.2 日志模型

为了规范化微服务日志可视化所需要的数据信息，本文定义了微服务的日志模型，具体如图 2 所示。

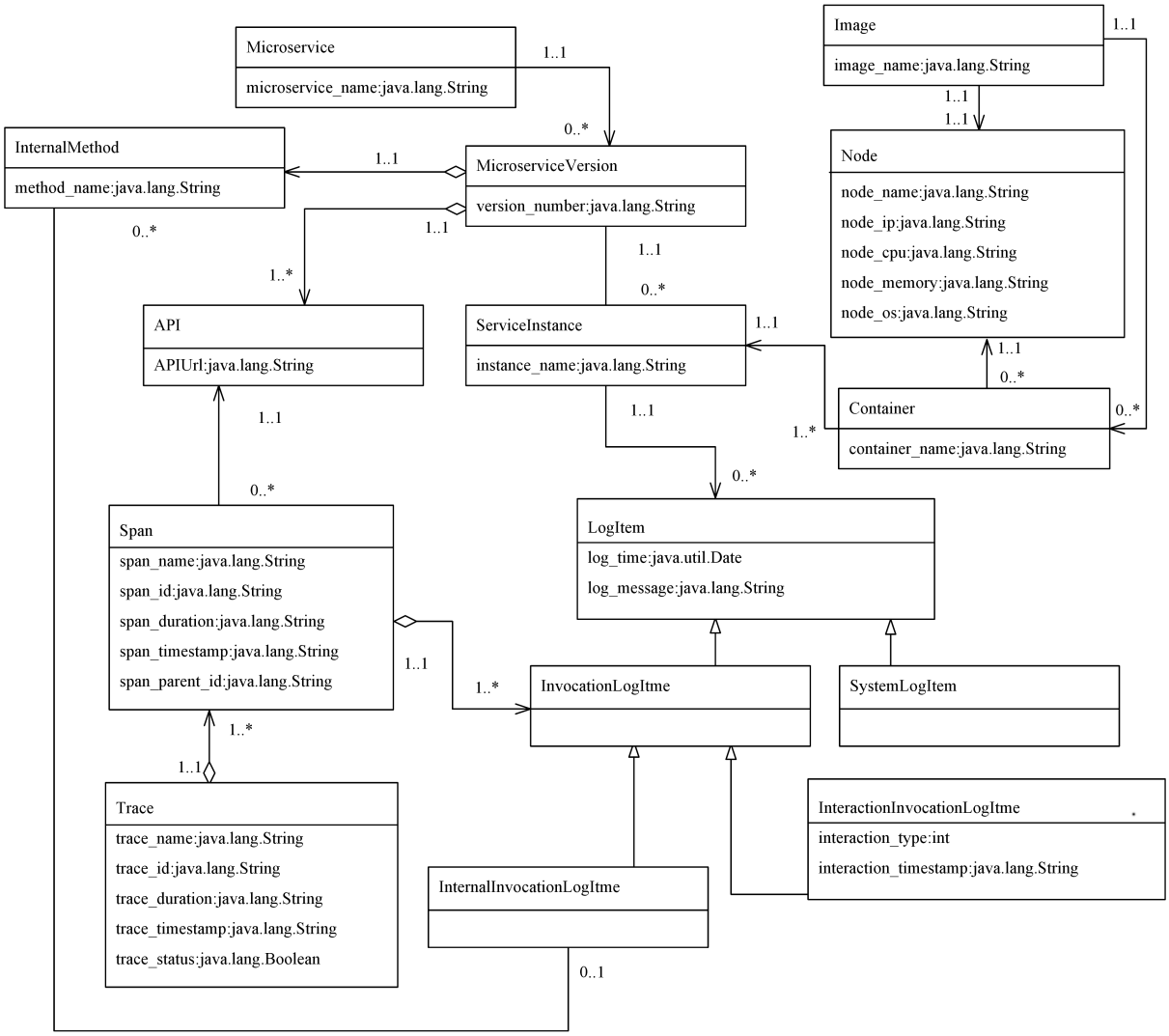


图 2 日志模型

Fig. 2 Log model

该模型从内容上可以分为 3 个部分：日志、调用链、节点与服务信息。首先，微服务系统运行时所产生的每一条日志都被称为日志项(LogItem)，每个日志项包含两个关键属性：日志时间戳以及日志内容。日志项分为两大类：一类是单个微服务的系统日志(SystemLogItem)，比如服务启动时所产生的日志项；另一类是服务与服务之间通信所产生的调用日志(InvocationLogItem)。调用日志又可以细分为交互调用日志(InteractionInvocationLogItem)和内部调用日志(InternalInvocationLogItem)。交互调用日志是指服务接收请求以及返回请求结果所产生的日志项；内部调用日志则是指服务在处理请求时，服务内部方法所产生的日志项。

微服务系统每次处理请求时的完整服务调用序列，构成了一条调用链(Trace)，每条调用链有唯一标识、时间戳等属性，其中状态属性用于标识调用链的对错。调用链的基本组成单位是 span(Span)，span 表示单个服务与服务之间的调用。每个 span 也有自己的唯一标识，该标识会记录服务调用的时间戳以及调用时长，同时 span 通过 span_parent_id 属性明确了彼此之间的顺序关系，没有 span_parent_id 属性的 span 被称为根 span。服务之间的调用是通过服务提供的 API 实现的，因此 span、调用日志以及 API(API)这三者可以相互关联。同时，在微服务系统中，每个服务(Microservice)都可能有多个版本(MicroserviceVersion)，不同版本的微服务所提供的 API 以及实际内部方法(InternalMethod)的实现可能存在差异。每个服务版本都可能有多个实例(ServiceInstance)，每个实例可能包含一个或多个容器(Container)，这些容器运行在集群的工作节点(Node)上，负责加载和运行指定的镜像(Image)。同时，服务实例在进行业务处理时会产生日志，因此每个日志项都可以和服务实例相关联。

3.3 可视化任务与调试策略

可能导致微服务故障的因素很多，而本文的可视化任务是在将微服务运行时产生的海量日志与调用链关联的基础上，可视化展示关键信息，从而支持用户对以下 4 种故障类型进行根源定位：

1）有异常抛出的普通故障；

2）没有异常抛出的逻辑故障；

3）存在服务异步调用时，特定的服务序列导致的故障；

4）服务多实例运行，未实现状态同步或者某个版本的实例导致的故障。

其中，需要可视化展示的关键信息包括：1）请求类型及其错误率；2）调用链类型及其错误率；3）指定调用链类型下存在的服务异步调用序列及对应的错误率；4）服务本身的错误率、服务的所有实例及对应的错误率；5）指定调用链下的所有日志信息及日志分类。

为了完成上述可视化任务，基于定义的日志模型，本文从日志和调用链关联展示的角度出发，将分散的信息以调用链维度进行聚合，并定义了 5 种策略。用户在实际使用时，可以通过组合各种不同的策略对可能导致微服务故障的根源进行定位。

策略 1　单条调用链日志查看。当微服务故障的根源只涉及到单个服务时，用户往往只需查看某条失败的调用链对应的日志即可定位到故障的根源。此种策略下，用户选定一条调用链，即可看到该条调用链所经过的所有服务以及该调用链对应的所有日志。同时，日志提供了两种排序方式：API 和时间。1）按照服务调用的 API 排序，同一个 API 调用产生的日志进一步按照时间排序，对应于请求接收、请求处理以及请求返回的顺序。按照这种方式组织日志可以方便用户以服务的 API 为单位进行故障根源查找。2）用户也可以选择按照时间对日志链中的所有日志项进行排序，方便查看此次请求处理的整个过程。用户针对单条调用链的所有日志项可以应用一些过滤器进行过滤，比如关键字筛选。通过过滤，某些情况下用户可以对微服务系统中的数据溯源，从而更好地定位故障根源。此外，针对调用链所经过的所有服务，用户可以直观地看到具体被调用的服务实例信息、服务实例所在的节点信息以及产生的日志信息。

策略 2　不同调用链的对比。同一个请求类型下的不同调用链具有可对比性，因为同一请求类型下的所有调用链具有相同的业务逻辑，正常情况下，同一请求类型下的所有调用链之间不会存在太大差异。如果一条调用链的长度明显短于其他调用链，而且该条调用链所经过的服务包含于其他调用链所经过的服务，那么该调用链的最后一个服务可能就是问题所在。此种策略下，用户可以选择两条调用链，而可视化界面将会区别展示两条调用链共同经过的服务以及各自经过的服务。同时，对于共同经过的服务，用户可以查看该服务在不同调用链上产生的日志。日志所包含的信息，比如输入输出数据，可以指引用户思考调用链差异产生的原因，从而辅助用户对问题根源进行定位。

策略 3　服务异步调用分析。服务与服务之间的通信方式可能很复杂，比如服务与服务之间的通信方式是异步调用，如果没有处理好异步调用的返回顺序，可能会导致微服务系统故障。利用此种策略，用户可以对某一调用链类型中的异步调用进行分析（见 3.4 节算法 1）。如果该调用链类型中存在异步调用，可视化界面会以特定方式显示异步调用所涉及的服务。同时，在所有调用链中实际存在的服务调用序列以及不同序列对应的错误率，会以图的形式显示。用户可以进一步查看每一个服务序列对应的所有调用链以及每条调用链的所有日志信息。

策略 4　服务多实例分析。微服务系统的每一个服务一般都是多实例运行的，以应对高负载、高并发的场景。然而，对于携带状态的微服务，如果无法保证不同服务实例的状态一致性，或者不同实例对应于服务的不同版本，都可能会导致问题出现。可视化界面支持用户对某一请求类型或者调用链类型下的服务进行多实例分析。可视化界面会以图的方式展示该类型下所有调用链涉及的微服务以及微服务的错误率。用户可以继续查看每一个服务被调用的所有实例以及相应的错误率。如果存在某个服务实例的错误率远高于或者远低于其他实例的情况，那么导致系统故障的根源很可能就是服务多实例没有做好状态同步或者某一版本的服务存在问题。

策略 5　调用链分段。实际生产环境中，微服务系统在处理单个请求时，可能会涉及到成百上千个微服务，产生的调用链因为长度较长，不适合进行完整的分析。需要找到一种方式来对调用链进行分段，从而允许用户以段为单位进行调用链分析。此策略下，用户首先输入一个整数值，该值用于根据服务之间的调用关系对服务依赖图分块（见 3.4 节算法 2），调用链中处于相同分块的微服务即形成一段。由于处于相同分块的微服务调用是比较频繁的，因此块内部服务调用出错的概率理论上低于块与块之间服务调用出错的概率。通过这种方式，可以指引用户重点查看不同块之间服务的调用情况，辅助用户查找故障根源。

3.4 相关算法

当属于同一调用链类型的调用链彼此经过的服务序列存在差异时，该调用链类型中存在服务异步调用。为了识别异步调用所涉及的服务范围，本文设计了服务异步调用分析算法。同时，针对策略 5，本文利用谱聚类方法实现了微服务依赖图分块算法，用于对服务依赖图进行分块，进而实现调用链的分段。

3.4.1 服务异步调用分析算法

为了对微服务系统中的服务异步调用进行分析，首先给出如下定义：微服务集合 $S=\{s_1,\cdots,s_n\}$，调用链 $L=(s_i,\cdots,s_m\mid s_i\in S,1\leqslant i,m\leqslant n)$，调用链集合 $C=\{l_i,\cdots,l_p\mid 1\leqslant i,p\leqslant A_{m-1}^{m-1},\forall c_k\in C,\forall l_i,l_j\in c_k:\mid l_i\mid=\mid l_k\mid$，对于 l_i 和 l_j 中 $\forall s_q$，其数量相等$\}$。对于上述定义，有如下性质：

1）任意服务都可以出现在调用链的任意位置；

2）调用链有序且服务可重复；

3）同一调用链集合中，任意两个调用链中任意一个服务的调用次数相同；

4）任意调用链都是以同步调用开始的。

算法分析的服务调用链基于日志产生，按该服务被调用的时间进行排序。异步调用的起始服务后的调用都会被归结

为异步调用，比如以下微服务调用：

$$s_1 \rightarrow s_2 \rightarrow \begin{Bmatrix} s_3 \rightarrow s_4 \\ s_5 \rightarrow s_6 \rightarrow s_7 \end{Bmatrix} \rightarrow s_8$$

s_1 和 s_2 为起始的同步调用，s_3 和 s_5 为异步调用的起始服务，s_4，s_6，s_7 为异步调用中调用的服务，最后调用服务 s_8。最终形成的调用链为 $s_1 \rightarrow s_2 \rightarrow$（异步调用）$\rightarrow s_8$。对于异步调用的部分，保持每个异步链的相对序列不变，然后组合。服务异步调用分析算法如算法 1 所示。上述调用可得的调用链如下：

$s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_5 \rightarrow s_4 \rightarrow s_6 \rightarrow s_7 \rightarrow s_8$

$s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_5 \rightarrow s_6 \rightarrow s_4 \rightarrow s_7 \rightarrow s_8$

…

算法 1 服务异步调用分析算法

输入：C[m][n]//调用链集合二位数组，每一行为一个调用链

输出：AsynServices //异步调用服务集合

```
Begin
/* 以第一个调用链 C[1][n]为基础，找出二维数组中元素相同的
   列，即找出同步调用的服务 */
1. for i←1 to m-1
2.    for j←0 to n-1
3.       do if C[i][j] ≠ C[1][j]
4.          then Count[j]++
5.    end for
6. end for
/* 对于输入中不相同的列，将其列号记录到异步调用下标集合
   asynIndics 中，对于相同的列，将其对应的服务记录到同步调用服
   务集合 syncServices 中 */
7. for i←0 to n-1
8.    do if Count[i]=0
9.       then syncServices.add(C[1][i])
10.   else
11.      asynIndics.add(i)
12. end for
/* 判断同步调用的服务是否在异步调用范围中出现，若出现，则为异
   步调用，否则为同步调用 */
13. for i← 0 to m-1
14.   for j in asynIndics
15.      do if syncServices.contains(C[i][j])
16.         then syncServices.remove(C[i][j])
17.   end for
18. end for
19. Services←Set(C[1][n])
20. AsynServices←Services.remove(syncService)
End
```

最终在异步调用部分出现的服务，都会被归类到异步调用范围，上述调用链中异步调用范围为：

$\{s_3, s_4, s_5, s_6, s_7\}$

3.4.2 微服务依赖图分块算法

谱聚类算法[45]从图论演化而来，之后在聚类中得到了广泛应用。它的主要思想是把所有的数据看作空间中的点，这些点之间使用边连接。距离较远的两个点之间的边权重值较低，而距离较近的两个点之间的边权重值较高，通过对所有的数据点组成的图进行切图，让切图后不同的子图间的边权重和尽可能低，而子图内的边权重和尽可能高，从而达到聚类的目的。

在微服务系统中，服务与服务之间的调用形成了调用链，不同的调用链之间形成了服务调用网络。虽然服务与服务之间的调用是有向的，但是当我们在分析服务与服务之间的调用关系时，可以忽略其调用方向，因为我们只关注服务与服务之间的紧密程度。我们将调用链形成的调用网络映射为无向图 $G(V,E)$。V 表示无向图中点的集合$(v_1, v_2, \cdots v_n)$，即调用网络中的微服务的集合；E 表示无向图中边的集合$(e_1, e_2, \cdots, e_n)$，即调用网络中微服务之间的调用的集合。我们定义权重 w_{ij} 为点 v_i 和点 v_j 之间的权重，即服务 i 与服务 j 之间的调用次数，因此 $w_{ij}=w_{ji}$。基于以上定义，利用谱聚类算法，本文实现了微服务依赖图分块算法。

算法 2 微服务依赖图分块算法

输入：微服务调用网络映射的邻接矩阵 $\mathbf{W}$，聚类数量 k

输出：分类点集 $A_i, \cdots, A_k$，$A_i=\{j \mid y_i \in C_i\}$

```
Begin
1. 计算归一化拉普拉斯矩阵 L_sym
2. 计算前 k 个 L_sym 的特征向量 u_1, ···, u_k
3. 令 U∈R^{n×k}，U 包含列向量 u_1, ···, u_k
4. 矩阵 T∈R^{n×k} 将行归一化为范数 1，即赋值 t_ij = u_ij / (Σ_k u_ik^2)^{1/2}
5. for i=1, ···, n，令向量 y_i∈R^k 为矩阵 T 的第 i 行
6. 将点集 (y_i)_{i=1,...,n} 使用 k-means 算法聚类到分类 C_1, ···, C_k
End
```

输入中的 k 值由用户提供，允许输入的值为 2～9，因为开发者本身对于所调试的微服务系统比较熟悉，所以让开发者自行输入、调整分类的数量会更为合理。

4 系统设计及实现

4.1 系统设计

基于定义的微服务日志模型及 5 种策略，本文实现了一个可以将日志与调用链关联展示的原型工具 LogVisualization，其体系结构如图 3 所示。

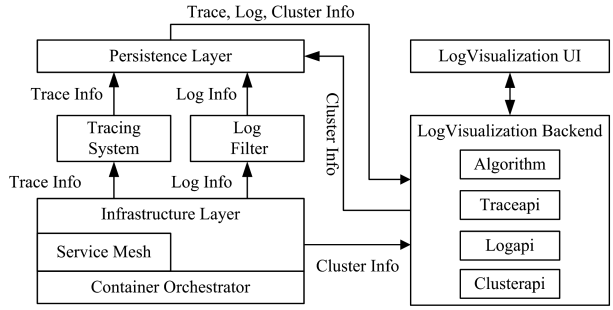


图 3 原型工具的体系结构

Fig. 3 Architecture of prototype tool

容器编排工具和服务网格共同构成基础设施层，由基础设施层管理集群以及微服务系统在集群上的部署运行。目前，运行微服务系统的最好形式就是容器[46]，因为容器可以封装应用程序的业务代码以及所有必需的运行时环境和配置，同时容器可以实现不同应用程序之间的隔离。借助于容

器编排工具，可以实现微服务系统的自动部署/扩容和管理。服务网格是一个比较新的概念，针对每一个服务实例，服务网格都会并行部署一个边车进程，从而控制服务与服务之间的网络通信，实现微服务系统中请求的可靠传递[47]。借助于基础设施层，可以实现日志和调用链数据的收集和发送。调用链数据会被发送到调用链追踪工具中，而日志则会被发送到过滤层，其一方面去除无用的日志，另一方面对日志进行处理，提取日志中的关键属性。同时，原型工具后台会定期调用基础设施层的 API 来获取集群信息，主要包括集群节点的信息以及运行在集群中的服务实例信息，这些信息都会被发送到持久化层进行存储。

后台提供了 3 类 API 用于支持前台界面的可视化分析，包括：1）调用链 API，负责调用链分类，统计和分析调用链所涉及的服务的信息等；2）日志 API，用于获取某条调用链或者某个服务实例的所有日志；3）算法 API，用于分析调用链当中的异步调用以及对调用链分段提供算法支持。

4.2　系统实现

原型工具的具体实现如图 4 所示。在基础设施层的具体实现中，本文使用了 Kubernetes[48] 进行容器的编排管理。作为一个开源容器集群管理项目，Kubernetes 提供了容器化应用部署、规划、更新、维护的一系列机制，保证了云平台中的容器始终能够按照用户的期望状态运行。Kubernetes 中所有的容器均在 Pod 中运行，一个 Pod 可以承载一个或者多个相关的容器，Pod 一旦被创建，Kubernetes 就会持续监控 Pod 以及 Pod 所在节点（Node）的健康状况。

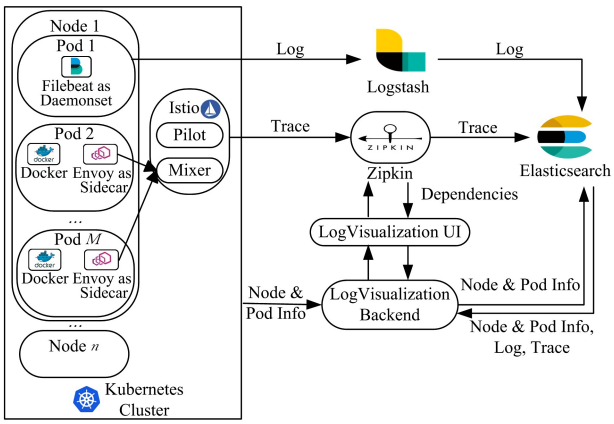


图 4　LogVisualization 的实现

Fig. 4　Implementation of LogVisualization

在此基础之上，本文利用 Istio[49] 建立已部署微服务系统的服务网络，对服务间的通信调用进行监控拦截。Istio 会在每一个 Pod 中新启容器，运行单独的进程 Envoy，Envoy 负责从 Istio 的 Pilot 模块拉取对应的指令和规则，修改容器的 iptables 来代理应用程序的所有出入流量，并定期向 Mixer 模块发送监控数据。这种对原应用程序代码无侵入、通过额外的容器来扩展和增强主容器的方法被叫做边车模式。通过边车，Istio 可以实现调用链信息的自动生成和传递，结合 Zipkin，可以将调用链信息持久化到 Elasticsearch。

为了收集系统日志，集群的每个节点都会以 DaemonSet 的方式运行 Filebeat。Filebeat 属于 Beats，能够从指定目录读取应用程序的日志，对日志进行简单的过滤和处理（例如异常日志的多行拼接）后再将其发送给 Logstash。日志在 Logstash 中经过定制的 grok 插件，被分为请求、响应、内部方法和异常信息 4 种类型，同时提取出每个类别对应的关键字段。最后，Logstash 会将日志存储到 Elasticsearch。

核心部分 LogVisualization 以前后台分离的形式实现。后台接收前台发送的请求，从 Elasticsearch 查询相应信息，组装数据并返回给前台；另外，后台与 Kubernetes API Server 交互，定期获取集群中所有节点与 Pod 的相关信息并存储到 Elasticsearch 中。前台在 Zipkin 界面的基础上进行修改，与后台进行交互，实现调用链与日志的关联可视化。

4.3　数据图形映射与示例分析

LogVisualization 的可视化界面如图 5 所示。整个可视化界面包含 4 个视图，视图与视图之间有一定的联动性，这样既可以区别展示不同的信息，又可以实现信息之间的关联，从而使其有更好的用户体验。在界面最上方，用户能够根据需要，以天为单位，指定分析日志和调用链数据的时间段。选定分析时间段以后，可视化界面左侧会以 hierachical graph[50] 的形式显示该时间段内的所有请求类型，以及每个请求类型下存在的调用链类型。同时，每个调用链类型下会显示属于该调用链类型的所有调用链标识，每条调用链标识下会显示该条调用链经过的所有服务。每个请求类型和调用链类型都包含一些统计信息，如正确以及错误的调用链数量。每条调用链也具有统计信息，如异常、错误以及正常的日志数量。之所以采用 hierachical graph，是因为 hierachical graph 的层次感强，可以很好地表达请求类型、调用链类型与调用链标识之间的递进关系，方便用户根据自己的判断逐层深入，探索导致微服务故障的原因。

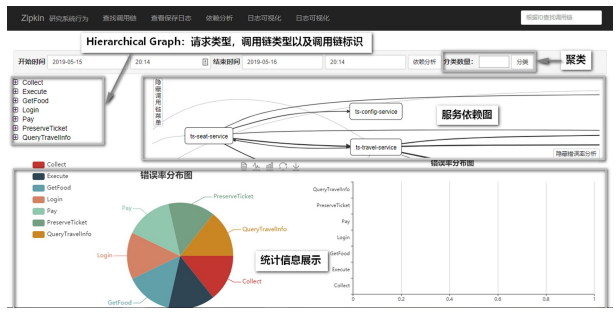


图 5　可视化界面（电子版为彩色）

Fig. 5　Visualization Page

为了更加直观地展示请求类型和调用链类型的统计信息，可视化界面下方会通过饼状图的方式显示所有请求类型的错误率，点击某一请求类型后会自动展示该请求类型下所有调用链类型的错误率。同时，该饼状图和 hierachical graph 是联动的。采用饼状图的形式使得用户可以很直观地看到同一请求类型下所有的调用链类型的错误率，从而使得用户可以选择错误率高的调用链类型继续探索。除了饼状图以外，界面下方还有柱状图，用以展示服务的不同实例错误率，其主要目的是让用户可以从服务多实例的角度进行分析。

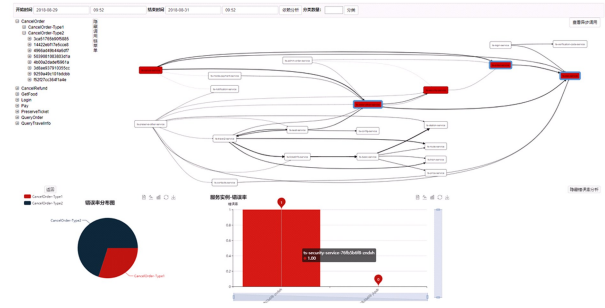
可视化界面右侧的中间部分还会展示指定时间段内的服务依赖图，同时会根据服务的错误率对服务的颜色进行渲染，

错误率越高，则渲染的颜色越深。其中，服务依赖图的绘制是在 dagre-d3[51]的基础上进行定制的；服务的错误率是指在指定时间段内，经过该服务的所有调用链中，错误的调用链所占的比例。这里选用圆角矩形表示服务，是因为人眼处理圆角更容易，圆角更适合头部和眼睛的自然运动[52]。而服务与服务之间的调用以有向边表示，边的方向表示服务调用方向，其粗细表示服务调用次数的多少，即服务之间依赖程度的高低。此种方式简单直观，符合人的视觉观感。同时，错误率的颜色渲染选择了红色，这是因为红色属于三原色之一，在可见光光谱上位于长波末端的位置，是视觉效果最强烈的色彩，容易吸引人眼的注意[53]。从而可以引导用户重点关注错误率高的服务。

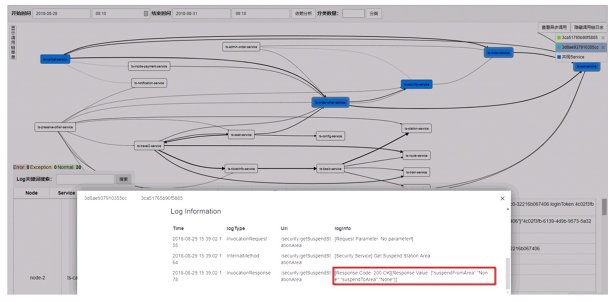
接下来以 TrainTicket 中的多实例错误为例，介绍用户在对该种类型的错误进行根源定位时可能会使用的不同策略。该错误的具体表现为：某些情况下，管理员需要锁定两个站点，对两个站点之间的票进行管理。此时用户的订票、退票操作如果涉及到被锁站点，则都不应该成功，而应提示用户暂时无法进行该项操作。但是系统实际运行时，涉及到被锁站点的部分退票操作却被成功执行了。

为了对问题根源进行定位，用户从可视化界面的饼状图中可以首先看到在所有请求类型中，CancelOrder 的错误率最高，因此用户查看 CancelOrder 下存在的调用链类型及对应的错误率。此时用户看到 CancelOrder-Type2 的错误率远高于 CancelOrder-Type1。接下来用户对 CancelOrder-Type2 进行分析，包括服务异步调用分析以及服务多实例分析，结果如图 6(a)所示。从图 6 中可以看到，该调用链类型下存在异步调用(被加上蓝色边框的服务，就是算法 1 分析得到的异步调用所涉及的服务)。同时，柱状图显示，所经过的服务 ts-security-service 的不同实例中，一个实例的错误率为 1(100%)，另一个实例的错误率为 0，错误率存在较大差异。因此用户在 CancelOrder-Type2 下选择一条正确以及一条错误的调用链进行对比分析，结果如图 6(b)所示。其中，两条调用链所经过的所有服务都用相应的色彩高亮显示(深蓝色表示两条调用链共同经过的服务，青绿色和浅蓝色用于表示两条调用链各自经过的服务)。

用户重点关注 ts-security-service，因此会查看该服务在不同调用链上所产生的日志信息。此时，用户点击服务依赖图上的 ts-security-service 服务，该服务在两条调用链上被调用的实例及其对应的日志信息以两个选项卡的形式展示。此种形式方便用户切换调用链，进而进行日志内容的对比。从图 6 中可以看到，在错误的调用链上，用户可以看到打印出的日志信息中显示该条调用链所经过的 ts-security-service 服务实例上被锁定的站点显示为“None”(空值)，而成功的调用链上显示的是管理员锁定的两个站点。因此，此时用户定位到了错误根源，即由于 ts-security-service 的两个实例的状态不一致，未同步被锁站点，导致系统出现故障。



(a)CancelOrder-Type2 实例和异步调用分析



(b)调用链对比和日志展示

图 6 原型工具使用示例(电子版为彩色)

Fig. 6 Usage example of prototype tool

5 实验

5.1 实验准备

本文实验使用的 TrainTicket 是一个开源的仿 12306 火车票业务系统，共有 41 个业务微服务，包括注册登录、订票、退票、行车路线查询等主要流程。为了验证本文所提出的几种策略，本文往 TrainTicket 系统中注入了 4 种不同类型的故障，这些故障位于 Git 仓库不同的分支上[54]。故障的类型和重现过程如表 1 所列。

表 1 实验故障列表

Table 1 Table of experiment fault

故障序号	故障类型	验证策略	故障重现过程
故障一	有异常抛出的简单错误	单条调用链查看	这是用户退票过程中注入的一个故障，当退票用户为 VIP 用户时，某个验证身份的参数传错导致退票失败
故障二	无异常抛出的逻辑错误	不同调用链对比	计算车票价格的模块由于逻辑错误导致用户会在页面上看到某条火车线路上二等座价格比一等座还高出很多，某条火车线路票价又是正常，由于是单纯的业务逻辑错误，整个过程不会存在打印异常日志
故障三	多实例状态不一致	服务多实例分析	当管理员锁定了两个车站后，所有关于这两个车站的车票都将无法退票。但由于锁定状态仅存储在某个实例内部，多个实例之间没有做状态同步，导致锁定车站后出现仍然可以退票的情况
故障四	异步调用顺序不一致	服务异步调用分析	系统退票过程中退票服务会异步发送两个请求给退款服务和订单服务，而这两个服务都会修改数据库中订单的状态，只有退款服务首先完成对订单状态的修改才能够成功退票，否则退票失败

5.2 实验过程与结果

实验参与者为 4 个对 TrainTicket 业务流程有一定了解的微服务开发人员。针对每一个故障，参与者首先访问部署好的 TrainTicket 系统，并按照步骤在系统界面上重现故障场景，在此基础上使用指定的工具对故障根源进行分析和定位。实验中使用的工具为 LogVisualization，此外，还将 Zipkin＋Kibana 作为对比的工具组合，其中 Zipkin 可以收集可视化调用链数据，而 Kibana 可以将存储在 Elasticsearch 中的日志数据可视化展示并提供搜索过滤功能。故障和参与者使用工具的对照如表 2 所列。

表 2　实验工具-故障对应表

Table 2　Tool-fault table of experiment

参与者	故障一	故障二	故障三	故障四
参与者 A	Zipkin＋Kibana	LogVisualization	Zipkin＋Kibana	LogVisualization
参与者 B	LogVisualization	Zipkin＋Kibana	LogVisualization	Zipkin＋Kibana
参与者 C	Zipkin＋Kibana	Zipkin＋Kibana	LogVisualization	LogVisualization
参与者 D	LogVisualization	LogVisualization	Zipkin＋Kibana	Zipkin＋Kibana

在进行故障排查时，要求参与者记录定位到故障服务所需的时间，使用本文构建的工具时需要记录运用到哪几种类型的故障排查策略以及策略使用的先后顺序，最终得到的实验结果如表 3 所列。

表 3　实验结果

Table 3　Experiment results

故障	工具	是否定位到故障服务？（是/否）	定位的总时间/min	使用策略及先后顺序（1－＞2－＞3）	能够判断/猜测出故障的根本原因（是/否）
故障一	LogVisualization	是	15	1－＞2－＞3	是
		是	10	1	是
	Zipkin＋Kibana	是	23	－	是
		是	24	－	是
故障二	LogVisualization	是	20	1－＞2	是
		是	18	1－＞2	是
	Zipkin＋Kibana	是	18	－	否
		是	16	－	否
故障三	LogVisualization	是	10	1－＞2－＞3	是
		是	15	1－＞2－＞3	是
	Zipkin＋Kibana	是	20	－	是
		是	16	－	是
故障四	LogVisualization	是	4	1－＞2－＞4	是
		是	8	1－＞2－＞4	是
	Zipkin＋Kibana	是	15	－	是
		是	12	－	是

从表 3 的数据可以总结出：除了故障二，使用本文构建的工具 LogVisualization 定位故障的时间比结合使用 Zipkin 和 Kibana 要短很多。定位故障二的时间之所以更长，是因为在故障二中，使用 Zipkin＋ Kibana 的参与者并没有找到故障的根源。除此之外，我们可以看到参与者在实际的故障排查过程中的策略选择和顺序与前文的总结是相符的，LogVisualization 工具提供的几种策略对于特定故障的排查是有用的，而且与 Zipkin＋Kibana 组合相比，LogVisualization 更为高效。

本文在参与者完成全部 4 个故障的实验后，围绕工具的使用感受、两种工具的对比等方面对他们进行了简短的访谈。4 位参与者均表示：1) LogVisualization 学习成本低、上手快，图标和颜色的高亮能够帮助他们快速定位出错的服务，精确到异常的日志；2) 与 Zipkin＋Kibana 相结合的方法比较，LogVisualization 提供了对于整个系统运行状态的总体概览，可以直观地看到各个业务流程、服务乃至实例的错误率和比重，从而引导他们逐步深入直至找到故障根源；3)将调用链和日志相结合并显示在同一界面避免了反复切换 Zipkin 和 Kibana 以及查找对应错误日志的麻烦，更加准确和高效；4)工具提供的异步调用检测、不同调用链的对比分析等功能为故障排查提供了新的思路，同时能够更好地监控服务之间的调用情况。

最后，参与者也认为本文构建的工具目前还有一些可以改进之处：1)应当在右上角添加异步调用的标识，方便工具使用者明确其具体意义；2)借鉴 Kibana 中日志的查找和过滤功能，也对日志内容中的字段进行了提取和分类，可以更好地实现数据的查找和溯源。

6　讨论

相比于传统的巨石应用，单个微服务的开发相对简单，但是微服务系统运行过程中一旦发生故障，对故障根源的定位会更加困难。微服务系统也是一种分布式系统，虽然通过调用链追踪工具可以记录微服务处理请求时所经过的服务，但是要从微服务运行时产生的海量日志信息中查找调用链对应的日志信息，目前还没有工具支持。文中的工具能够将日志和调用链关联展示，同时提供了不同的策略，辅助用户对微服务系统故障根源进行定位。但所提工具仍存在以下不足：

1)通过 AOP 往日志中注入调用链信息，仍然需要修改原系统代码，对原微服务系统具有一定的侵入性。为避免这种对原系统实现的修改，一种解决方法是改造 Istio，监控微服务之间的网络通信，自动往服务调用产生的日志中注入调用链信息。

2)实验部分相对薄弱。由于参与实验的人员需要对测试的微服务系统有一定了解且没有参与本文的相关工作，同时需要具有一定的微服务开发经验，限于这些客观因素，参与实验的人员数量较少，不能充分代表微服务开发者。另外，用于测试的 TrainTicket 微服务系统无法产生较长的调用链，因此调用链分段的策略无法在实验中被验证。

结束语　本文首次研究微服务日志的可视化，定义了微服务日志模型，同时从微服务故障调试的角度明确了可视化任务，定义了 5 种可视化调试策略：单条调用链日志查看、不同调用链对比、服务异步调用分析、服务多实例分析以及调用链分段。为了实现服务异步调用分析以及调用链分段，本文设计并实现了相应的算法。在此基础上，本文实现了一个原型工具 LogVisualization，通过该工具可以将微服务系统的日志和调用链关联展示。工具的可视化界面实现了定义的 5 种故障调试策略，用户可以通过组合不同的策略来对导致微服务系统故障的根源进行定位。同时，通过在实际微服务系统当中的应用以及与 Zipkin 和 ELK Stack 的实验对比，证明了

原型工具的有用性和高效性。

由于目前的原型工具无法自动地往日志信息中注入相应的调用链信息，对业务微服务系统仍有一定的侵入性，因此未来的一个工作方向是改造 Istio，通过拦截服务与服务之间的网络通信，自动往日志信息中注入调用链信息。同时，考虑将调用链中的延迟信息、集群的资源使用情况等以合适的方式展示在可视化界面中，以更好地支持用户对导致微服务系统故障的根源定位。

参 考 文 献

[1] JAMES L, MARTIN F. "Microservices" [EB/OL]. [2018-7-15]. https://martinfowler.com/articles/microservices.html.

[2] ZIMMERMANN O. Microservices tenets[J]. Computer Science-Research and Development, 2017, 32(3/4): 301-310.

[3] FRANCESCO P D, MALAVOLTA I, LAGO P. Research on Architecting Microservices: Trends, Focus, and Potential for Industrial Adoption[C]// IEEE International Conference on Software Architecture. 2017: 21-30.

[4] HEORHIADI V, RAJAGOPALAN S, JAMJOOM H, et al. Gremlin: Systematic Resilience Testing of Microservices[C]// IEEE International Conference on Distributed Computing Systems. 2016: 57-66.

[5] ZHENG H, LI D, LIANG B, et al. Automated Test Input Generation for Android: Towards Getting There in an Industrial Case [C]// IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice Track. 2017: 253-262.

[6] RUSLAN M. Microservices at Netflix Scale[EB/OL]. [2018-7-15]. https://gotocon.com/dl/goto-amsterdam-2016/slides/RuslanMeshenberg_MicroservicesAtNetflixScaleFirstPrinciplesTradeoffsLessonsLearned.pdf.

[7] JAMSHIDI P, PAHL C, MENDONCA N C, et al. Microservices: The Journey So Far and Challenges Ahead[J]. IEEE Software, 2018, 35(3): 24-35.

[8] ZHOU X, PENG X, XIE T, et al. Benchmarking Microservice Systems for Software Engineering Research[C]// Proceedings of International Conference on Software Engineering: Companion Proceeedings. 2018: 323-324.

[9] DO N H, DO T V, XUAN T T, et al. A scalable routing mechanism for stateful microservices[C]// Innovations in Clouds, Internet & Networks. 2017: 72-78.

[10] SHASHA M. Application Delivery Service Challenges in Microservices-based Applications[EB/OL]. [2018-7-21]. http://www.thefabricnet.com/application-delivery-service-challenges-in-microservices-based-applications/.

[11] ZHOU X, PENG X, XIE T, et al. Delta debugging microservice systems[C]// Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. 2018: 802-807.

[12] ZELLER A, HILDEBR R. Simplifying and Isolating Failure-Inducing Input[J]. IEEE Transactions on Software Engineering, 2002, 28(2): 183-200.

[13] ZIPKIN. Zipkin[EB/OL]. [2018-7-20]. https://zipkin.io/.

[14] JAEGER. Jaeger[EB/OL]. [2018-7-20]. https://www.jaegertracing.io/.

[15] ELK. ELK Stack[EB/OL]. [2018-7-20]. https://www.elastic.co/elk-stack.

[16] SHIVIZ. Shiviz[EB/OL]. [2018-7-20]. https://bestchai.bitbucket.io/shiviz/.

[17] PAHL C, JAMSHIDI P. Microservices: A Systematic Mapping Study[C]// International Conference on Cloud Computing & Services Science. 2016: 137-146.

[18] HASSAN S, BAHSOON R. Microservices and Their Design Trade-Offs: A Self-Adaptive Roadmap[C]// IEEE International Conference on Services Computing. 2016: 813-818.

[19] CAMARGO A D, SALVADORI I, MELLO R D S, et al. An architecture to automate performance tests on microservices[C]// International Conference on Information Integration & Web-based Applications & Services. 2016: 422-429.

[20] HEINRICH R, HOORN A V, KNOCHE H, et al. Performance Engineering for Microservices: Research Challenges and Directions[C]// International Conference on Performance Engineering Companion. 2017: 223-226.

[21] SCHERMANN G, SCHÖNI D, LEITNER P, et al. Bifrost: Supporting Continuous Deployment with Automated Enactment of Multi-Phase Live Testing Strategies[C]// International Middleware Conference. 2016: 12.

[22] HASSELBRING W. Microservices for scalability: keynote talk abstract[C]// Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering. 2016: 133-134.

[23] KLOCK S, VAN DER WERF J M E M, GUELEN J P, et al. Workload-Based Clustering of Coherent Feature Sets in Microservice Architectures[C]// IEEE International Conference on Software Architecture. 2017: 11-20.

[24] LEITNERP, JURGEN C, EMANUEL S. Modelling and managing deployment costs of microservice-based cloud applications [C]// International Conference on Utility & Cloud Computing. 2016: 165-174.

[25] LIN J, LIN L C, HUANG S. Migrating web applications to clouds with microservice architectures[C]// International Conference on Applied System Innovation (ICASI). 2016: 1-4.

[26] 百度百科. 可视化[EB/OL]. [2018-7-25]. https://baike.baidu.com/item/%E5%8F%AF%E8%A7%86%E5%8C%96.

[27] AntV. 数据可视化概览[EB/OL]. [2018-7-25]. https://antv.alipay.com/zh-cn/vis/blog/vis-introduce.html.

[28] YANG H. The Research and Implementation of Visual Log Analysis System[D]. Xi'an: Xidian University, 2010. (in Chinese)
杨华. 可视化日志分析系统的研究与实现[D]. 西安：西安电子科技大学，2010.

[29] ZHANG S, ZHAO J. Research advances on network security logs visualization[J]. Journal of Frontiers of Computer Science and Technology, 2018(5): 681-696. (in Chinese)
张胜，赵珏. 网络安全日志可视化分析研究进展[J]. 计算机科学与探索，2018(5)：681-696.

[30] HU G. Research and Implementation of The Network Security Log Data Analysis System[D]. Beijing: Beijing University of Posts and Telecommunications, 2012. (in Chinese)
胡钢. 网络安全日志数据可视分析系统的研究与实现[D]. 北京：北京邮电大学，2012.

[31] DONG Z W. Design and Implementation of Visual Analysis and Monitoring System for National Domain Name Log[D]. Beijing: University of Chinese Academy of Sciences(School of Engineering Science),2014. (in Chinese)
董再旺. 国家域名日志可视化分析监控系统设计与实现[D]. 北京：中国科学院大学，2014.

[32] CHEN W W,WU K C. Research and application of massive DNS log data analysis and visualization[J]. Application Research of Computers,2016,33(2):335-338. (in Chinese)
陈文文，吴开超. 海量域名日志数据分析与可视化研究及应用[J]. 计算机应用研究，2016，33(2)：335-338.

[33] ZHAO G. Visualization Analysis Research of Website Usability and User Behavior Based on Web Server Log[D]. Taiyuan: Shanxi University,2007. (in Chinese)
赵刚. 基于 Web 日志的网站可用性及用户行为可视化分析方法研究[D]. 太原：山西大学，2007.

[34] LU D. Implementation and Design of OPAC Search Log Visualize System[J]. Modern Computer,2016(12):67-70. (in Chinese)
鲁丹. OPAC 搜索日志可视化系统的设计与实现[J]. 现代计算机，2016(12)：67-70.

[35] APACHE FLUME. Welcome to Apache Flume [EB/OL]. [2018-7-26]. https://flume.apache.org/.

[36] GITHUB. facebookarchive/scribe[EB/OL]. [2018-7-26]. https://github.com/facebookarchive/scribe/wiki.

[37] LIU K. Architecture analysis and application of massive data log system[J]. Journal of Changchun University of Technology,2016,37(6):581-586. (in Chinese)
刘锴. 海量数据日志系统架构分析与应用[J]. 长春工业大学学报(自然科学版)，2016，37(6)：581-586.

[38] CHEN J J,LIU H H. Distributed ELK log analysis system based on Kubernetes[J]. Electronic Technology & Software Engineering,2016(15):211-212. (in Chinese)
陈建娟，刘行行. 基于 Kubernetes 的分布式 ELK 日志分析系统[J]. 电子技术与软件工程，2016(15)：211-212.

[39] BESCHASTNIKH I,WANG P,BRUN Y,et al. Debugging Distributed Systems[J]. Queue,2016,14(2):91-110.

[40] OPENTRACING. The OpenTracing Semantic Specification [EB/OL]. [2018-7-26]. https://opentracing.io/specification/.

[41] SIGELMAN B H,BARROSO L A,BURROWS M. Dapper,a Large-Scale Distributed Systems Tracing Infrastructure[EB/OL]. [2018-7-26]. https://storage.googleapis.com/pub-tools-public-publication-data/pdf/36356.pdf.

[42] JAEGER. Introduction[EB/OL]. [2018-7-26]. https://www.jaegertracing.io/docs/.

[43] SEMATEXT. Jaeger vs Zipkin-OpenTracing Distributed Tracers[EB/OL]. [2018-7-28]. https://sematext.com/blog/jaeger-vs-zipkin-opentracing-distributed-tracers/.

[44] INSTANA[EB/OL]. [2018-7-28]. https://www.instana.com/.

[45] LUXBURG U V. A tutorial on spectral clustering[J]. Statistics & Computing,2007,17(4):395-416.

[46] TECHBEACON. 3 reasons why you should always run microservices apps in containers[EB/OL]. [2018-7-28]. https://techbeacon.com/3-reasons-why-you-should-always-run-microservices-apps-containers.

[47] BUOYANT. What's a service mesh? And why do I need one? [EB/OL]. [2018-7-28]. https://blog.buoyant.io/2017/04/25/whats-a-service-mesh-and-why-do-i-need-one/.

[48] KUBERNETES[EB/OL]. [2018-7-28]. https://kubernetes.io/.

[49] ISTIO[EB/OL]. [2018-7-28]. https://istio.io/.

[50] SUGIYAMA K,TAGAWA S,TODA M. Methods for Visual Understanding of Hierarchical System Structures[J]. IEEE Transactions on Systems,Man and Cybernetics,1981,11(2):109-125.

[51] 沪江网. 以红色为主的色彩搭配，值得推荐[EB/OL]. [2018-7-28]. https://www.hujiang.com/fyuid_s/p886132/.

[52] 新浪博客. 用户体验设计之圆角和直角[EB/OL]. [2018-7-28]. http://blog.sina.com.cn/s/blog_5d7170af0101dnpk.html.

[53] GITHUB. Dagrejs[EB/OL]. [2018-7-29]. https://github.com/dagrejs/dagre-d3.

[54] GITHUB. Rogen319/logvisualization_trainticket[EB/OL]. [2018-7-28]. https://github.com/Rogen319/logvisualization_trainticket.git.