云原生环境下可观测性模型应用研究

Research on Application of Observability Model in Cloud Native Environment

杨 晨,童 博,赵纯熙(中讯邮电咨询设计院有限公司,北京 100048)
Yang Chen,Tong Bo,Zhao Chunxi(China Information Technology Designing & Consulting Institute Co.,Ltd.,Beijing 100048,China)

摘 要:
针对传统可观测性系统存在的数据孤岛、建设冗余、数据关联度不高等问题,提出一种云原生环境下可观测性模型。通过对多源可观测数据进行集成融合,采用无侵入式可观测性数据的采集和追踪方法,构建统一可观测数据标签体系,实现云原生下应用程序全链路可观测数据的追踪、可视和持续分析。实验结果表明,提出的可观测性模型对性能带来的影响最大不超过6%,在链路追踪效果上具有较强的优势。

关键词:
可观测性;eBPF技术;数据标签;云原生
doi:10.12045/j.issn.1007-3043.2025.01.013
文章编号:1007-3043(2025)01-0066-07
中图分类号:TN915
文献标识码:A
开放科学(资源服务)标识码(OSID):

Abstract:
In response to the problems of data silos, redundant construction, and low data correlation in traditional observability systems, it proposes an observability model in cloud native environment. By integrating and fusing multi-source observable data, and using non-invasive method for collecting and tracking observability data, a unified observable data label system is constructed to realize the tracking, visualization, and continuous profiling of observable data throughout the entire chain of cloud native applications. The experimental results show that the observability model proposed has a maximum impact on performance of no more than 6%, and has a strong advantage in link tracking performance.

Keywords:
Observability;eBPF technology;Data labeling;Cloud native

引用格式:杨晨,童博,赵纯熙. 云原生环境下可观测性模型应用研究[J]. 邮电设计技术,2025(1):66-72.

1 概述

1.1 云原生给传统监控带来挑战

随着企业数字化转型进程的加快,云原生架构凭借DevOps概念和容器化部署、持续集成交付、多微服务构建等特点,得到各行各业的广泛认可[1]。基于云原生架构的应用程序通常采用大规模容器化部署的形式,构建在由多个云资源支持的动态编排的微服务之上,虽然具备较好的灵活性和扩展性,但应用程序也呈现出多系统层级、多服务调用、多通信协议等典型特征,导致系统的复杂性指数级增长。为解决云原生应用程序的监控问题,工程师们受控制理论的启发,开始考虑一种新的范式——可观测性(Observability)[2]。

1.2 云原生下可观测系统的现状

在控制论的定义中,可观测性是指从黑盒系统外部输出的知识中推断出其内部状态的程度[3]。在IT领域中,可观测性被定义为通过观察系统和应用程序的外部输出,来理解其内部工作状态的能力,其目标是对复杂的云原生系统中发生的一切给出合理的解释[4]。可观测性的工具和方法主要依赖于指标(Metrics)、追踪(Tracing)、日志(Logging)这3个方面[5]。

目前,可观测性各个方面都有许多成熟的开源组件,基于开源组件可以很容易地为系统搭建一套开箱

收稿日期:2024-12-09

即用的可观测性设施。但是，对于系统数量多、运维难度大、云化程度高的大中型公司来说，云环境下系统层级的概念增多，服务调用链路也变得更长，基于标准开源组件的可观测性系统往往存在以下问题。

a）各类可观测性数据之间相互割裂，数据孤岛现象严重。当某次服务调用发生告警时，仅依靠指标数据、链路数据、日志数据中某单一类型的可观测性数据，无法实现告警根因的全面排查和准确定位。

b）基于开源组件的可观测性系统为获取云内服务调用全链路的可观测数据，通常采用SDK插桩的形式，在云内各组件中部署的侵入式的agent采集器，给应用程序的性能和安全性带来新的风险和挑战。

c）各类数据检索困难。不同来源数据的标签定义不同，缺少统一概念和定义的数据标签体系，无法使用某一数据标签作为唯一标识，对数据进行快速地筛选检索，难以下钻定位具体的主机、微服务、Pod、实例等内容。

针对上述问题，需要一种能融合多源可观测数据、支持无侵入式采集、具备统一数据标签的技术架构来提高云原生架构下应用程序的可观测性。

2 研究背景

2.1 可观测性技术相关工作

近年来，可观测技术在研究学者和业内工程师的推动下得到很大的进展[6]。Cyril等人[7]构建一种基于eBPF（extended Berkeley Packet Filter）技术的可观测系统，对分布式部署的系统进行全链路监控和可观测数据的分析。Marcelo等人[8]展示了基于eBPF技术所开发出的新一代云内可观测监测工具的技术架构，并通过实验对比，验证了基于eBPF的可观测工具在性能、消耗等方面均优于传统工具。Liu等人[9]基于阿里云Kubernetes（简称K8s）集群，设计一种内核级别的可观测模型，使用eBPF技术对用户应用程序L4/L7层的组件性能进行非侵入式收集，能够在不修改内核和应用程序代码的情况下，对用户基于云环境的系统性能数据进行采集。可以看出，eBPF技术作为可观测能力建设的核心能力，目前已经得到学术界和业内工程师的广泛关注和认可。

2.2 eBPF技术

eBPF是一项革命性的技术，它不需要修改编译内核源代码或者加载内核模块，就可以高效地在内核中运行用户编写的沙箱程序，使内核释放出巨大的可能性。与传统的内核编程方式相比，eBPF技术更加方便快捷，能基于现有的抽象层来打造更加智能、功能更加丰富的基础设施软件，而不会增加系统的复杂度，也不会降低执行效率和安全性[10]。

虽然eBPF技术并不是新出现的技术，但如今云、容器以及分布式应用的发展使基于eBPF技术的程序有了更多的契合场景。当内核函数被执行时，相应的eBPF程序也被执行，程序的目的可以是多样的，如用来实现内核事件、性能指标、应用追踪等数据的自动获取[11]，因此，eBPF技术在云安全、容器网络、分布式应用追踪以及可观测性等方面得到了广泛应用与创新。

3 云原生下可观测性模型

3.1 可观测性模型总体架构

基于对云原生下可观测性技术的分析和研究，本文提出一种云原生环境下的可观测性模型，模型总体架构如图1所示。模型由多源可观测性数据集成、可观测数据无侵入式采集、统一可观测数据标签体系3个模块的内容组成。

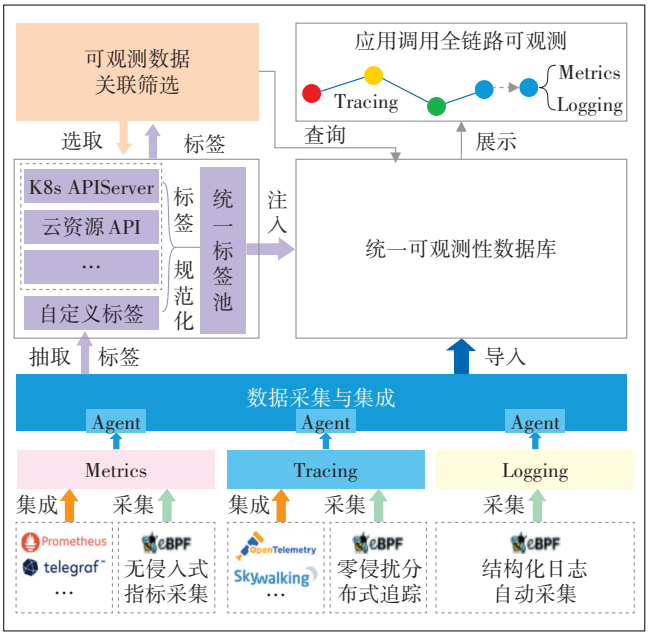

图1　可观测性模型总体架构

首先，通过与现有可观测性系统进行对接，对多源可观测性数据进行集成和统一存储，在消除各类可观测数据之间数据壁垒的同时，为可观测数据消费者提供通用化的数据支撑和数据关联分析能力。

其次，提出一种基于eBPF技术的可观测数据无侵

入式采集方法。针对内核各层级事件,编写具有针对性的eBPF程序,并将编译后的程序运行在Linux内核自带探针的接口处,实现各类可观测性数据的零侵扰式获取和追踪。

最后,通过对云原生环境下K8s、云资源等标准数据标签进行对接同步,对采集到的可观测数据进行标签抽取,构建标准化的可观测数据标签体系,注入可观测性数据库中,提供可观测数据的快速查询和检索能力。

3.2 多源可观测性数据集成

多源可观测性数据集成能力作为可观测技术框架的基础,通过与现有可观测性系统进行对接,获取并集成已有的可观测数据,消除可观测数据孤岛。在集成Metrics指标数据时,通过与当前主流指标采集工具进行对接,采用配置远程写入的方式,将Prometheus或telegraf采集到的指标数据统一发送至采集器。

在集成Tracing链路数据时,对标准的OpenTelemetry数据来说,可以直接将K8s环境和主机环境中的链路数据以OTel的格式发送给链路数据采集器;而Skywalking采集到的链路数据则需要将数据通过otel-collector进行OpenTelemetry标准协议处理,之后再发送给采集器,最后统一导入可观测性数据库(见图2)。

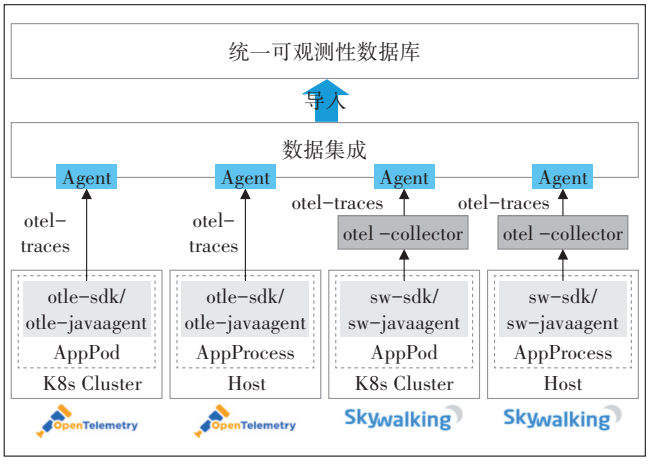

图2 链路数据集成方式

通过将集成的标准OTel链路数据与基于eBPF采集到的链路数据进行融合,使链路信息更加完整,有效消除分布式调用链中的盲点;并且通过提取指标数据中已有的pod_name(Telegraf)、pod(Prometheus)、instance(Prometheus)等原始标签,与标签体系中的相关指标自动进行计算和匹配,从而将集成的指标数据与其他观测数据打通,实现多源可观测数据的统一存储,有效消除数据壁垒,提升集成数据的关联分析和数据下钻能力。

3.3 无侵入的可观测数据采集

3.3.1 无侵入式采集框架

为实现多源可观测数据的无侵入式采集,本研究采用了Linux内核中提供的能够注入eBPF程序的扩展点,即探针(Probe)。基于这些探针的类型,编写用来解析应用协议的eBPF代码,在内核运行到这些探针的入口时,运行完成编译的、通过安全验证的、已注入内核中的eBPF代码,并以此来实现对内核数据流的采集。本研究主要使用内核中的4种不同类型的探针。

a)内核探针(Kernel probes,Kprobes)。Kprobes能够动态跟踪内核中的内部函数及组件等。

b)跟踪点(Tracepoints)。Tracepoints能静态跟踪内核中的函数组件。

c)用户空间探针(User-space probes,Uprobes)。Uprobes能动态跟踪在用户空间中运行的程序。

d)用户静态定义的跟踪点(User Statically Defined Tracepoints,USDT)。USDT能够静态跟踪在用户空间中运行的程序。

在内核加载的eBPF程序可在发生指定的系统调用时被执行,它将系统调用时的线程级数据向上聚合到容器、Pod甚至服务级,并将聚合后的关键信息传递回用户状态。因此,本研究采用如图3所示的eBPF采集框架,通过将完成编译的eBPF程序注入到如Kprobe、Uprobe、Tracepoint等探针的入口处,实现无侵入式的可观测数据采集。

整个采集框架包括数据采集、数据聚合以及数据导出3个部分。

a)数据采集部分。数据采集模块以Daemonset的形式部署在K8s集群的每个worker节点上,通过eBPF Controller设置配置信息和过滤规则(图3中①),并将eBPF程序存储在配置信息中,动态更新到内核,以将配置程序动态部署到数据采集探针点(图3中②);

b)数据聚合部分。当内核事件发生后(图3中③),在探针处部署的eBPF程序随即被触发,并对采集到的内核数据进行匹配聚合和字段过滤(图3中④),检索过滤后的主要数据通过初步解析器进行整合,并排队到缓冲区中(图3中⑤)。

c)数据导出部分。为避免频繁地从内核空间复制数据,采集框架通过Perf缓冲区,将聚合的数据流信息从eBPF映射输出到用户状态流日志中(图3中⑥),

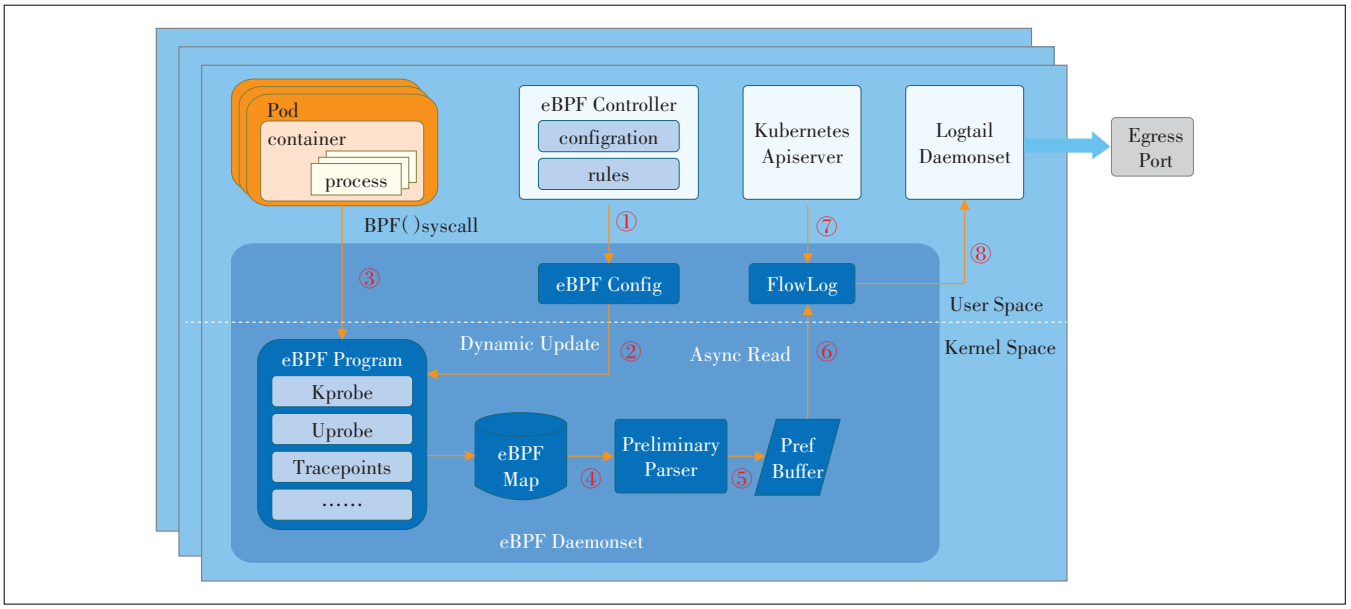

图3　eBPF采集框架

并通过调用K8s API接口，进一步匹配聚合相关的拓扑信息（图3中⑦），最后由Logtail Daemonse将采集到的数据进行导出操作（图3中⑧）。

通过eBPF采集框架，可自动获取每一个内核事件在应用函数、系统函数、网络通信等方面的可观测数据，通过对数据进行匹配分析，能够零侵扰地获取应用的RED（Request、Error、Delay）、吞吐、时延、异常等指标，实现可观测指标数据的采集。此外，通过一次服务调用的指标数据可按照各层级数据标签进行匹配和融合，构建并生成结构化的可观测日志数据。

3.3.2 全层级链路追踪

云原生环境下，各服务间的通信复杂，往往包含应用服务、数据库、云内网络等多个层次，通常需要许多不同层级的协议来进行通信，并且对于每一层级的协议，内核会选择一个特定的探针入口函数作为协议对应事件的hook点。因此，为实现可观测链路数据的追踪和采集，本研究采用不同的探针来采集追踪不同层级的可观测数据。同时，由于不同协议包含不同的网络事件，因此在使用探针时，需要使用具体探针的入口函数来处理相应协议header报头中的数据，表1给出了TCP、UDP、HTTP这3种代表性的网络协议及其对应的探针类型和入口函数。

因此，对于某一层级的协议发生的调用，利用eBPF技术提供的跟踪点，内核会选择具体的跟踪点函数，来实现对该协议调用事件的追踪。例如，当系统实例启动TCP连接时，入口函数tcp_v4_connect()就会

表1　典型协议与内核探针的入口函数示例

网络协议	探针类型	入口函数
TCP	tracepoint	inet_sock_set_state
	tracepoint	tcp_retransmit_skb
	kprobe	tcp_v4_connect
	kprobe	tcp_drop
	kprobe	tcp_send_loss_probe
UDP	kprobe	ip4_datagram_connect
	kprobe	ip6_datagram_connect
	kprobe	udp_sendmsg
	kprobe	udp_recvmsg
	kprobe	udp_destruct_sock
HTTP	uprobe	net/http.(*http2serverConn).processHeaders
	uprobe	golang.org/x/net/http2.(*serverConn).process-Headers
	uprobe	net/http.(*http2clientConnReadLoop).handleRe-sponse
	uprobe	net/http.(*http2ClientConn).writeHeader
	uprobe	net/http.(*http2ClientConn).writeHeaders

被对应的探针唤醒，对TCP调用连接进行追踪；当TCP连接状态发生变化时，入口函数inet_sock_set_state()可以用来记录连接状态和请求时延的变化。并且，通过eBPF所提供的辅助函数bpf_ktime_get_ns()，能够为探针收集到的事件赋予纳秒级别的时间戳，利用时序来关联2个相邻的调用事件，并且通过将数据流进行聚合，将请求/响应到达时序、TCP发送时序等信息进行关联融合，实现对TCP协议一次请求整个链路轨迹的追踪。UDP协议相关事件以类似的方式进行追

踪和收集。

对于不同层级的调用追踪,通过eBPF程序和cBPF程序可从应用函数调用和网络流量的报头中自动解析出TracingID和SpanID,可以用来关联各层级的调用链路追踪数据,有效实现从应用程序到网络的全层级可观测性数据的无侵入式追踪。

3.4 全栈可观测数据标签

全栈可观测数据标签体系作为可观测模型的核心,是对可观测数据进行快速筛选,实现故障快速根因定位的重要抓手。由于数据标签体系需要包含服务调用全链路各层级的数据标签,因此本文通过对接云环境中的资源标签,以及从可观测数据中提取出标签,并将2种来源标签的概念和定义进行规范化处理,打造形成全层级的统一标签体系。

为对接同步云原生环境下各类资源的标签,设计了一种云环境标签对接同步模型。该模型由以下5个部分内容组成。

a) Agent。监视并上报所在云环境的资源信息,并获取所在环境中网络的接口信息。

b) Controller.genesis。合并采集器上报的接口信息,接收采集器上报的云环境的资源信息。

c) Controller.cloud。抽象和转换采集器上报的资源信息,根据采集器上报的接口,补充Pod和Node的IP和Mac信息。

d) Cloud.recorder。将抽象后的资源信息保存至MySQL数据库,并定时更新。

e) Cloud.tagrecorder。抽取资源的基础标签信息,以字典的方式保存至ClickHouse,并定时更新。

将获取的可观测数据标签进行清洗和规范化处理,得到如图4所示的统一可观测数据标签体系。基于标签体系,可以为集成和采集到的可观测数据进行统一标签注入,用户可以使用标签对可观测数据进行筛选和检索,实现从全栈、全链路的视角完整观测一次请求所调用的指标、链路、日志数据,发生告警后能够迅速定位问题所在,彻底打破数据孤岛,增强故障根因下钻分析的能力。

4 可观测模型性能评估

云原生架构下的应用程序通常构建在多个支持动态编排的微服务之上,因此,本章针对不同架构的微服务应用程序,对本文提出的云原生下可观测性模型的性能开销、观测效果等能力进行测试和评估。

服务信息标签 Service Info Tag	应用服务/app_service 、应用实例/app_instance、端点/endpoint、进程/IDprocess_id、内核线程名/process_kname……
应用层标签 Application Layer Tag	应用协议/17_protocol、协议版本/Version、请求类型/request_type、请求域名/request_domain、请求资源/request_resource……
传输层标签 Transport L ayer Tag	客户端口/client_port、服务端口/server_port、请求TCP Seq号/req tcp_seq、响应TCP Seq号/resp_tep_seq……
网络层标签 Network Layer Tag	IP地址/ip、IPv4标志/is_ipv4、Internet IP标志/is_internet、网络协议/Protocol……
链路信息标签 Tracing Info Tag	TraceID/trace_id、SpanID /span_id、ParentSpanID/parent_span_id、Span类型/span_kind、X-Request-ID/x_request_id……
通用标签 Universal Tag	路由器/Router、DHCP网关/Dhcpgw、负载均衡器/Lb、负载均衡监听器/lb_listener、NAT网关/Natgw……
流量信息标签 Flow Info Tag	流日志ID/flow_id、开始时间/start_time、结束时间/end_time……
……	……

图4 统一可观测数据标签体系

4.1 实验环境

基于可观测模型的设计思路,使用C语言实现云原生下可观测性模型。为对模型性能进行测试和评估,本研究在标准配置的三节点K8s集群测试平台(版本1.24)进行实验验证。该集群由3台相同的服务器组成,单台服务器的参数配置如表2所示。

表2 实验环境配置

操作系统	Ubuntu 20.04
内核版本	V5.4.0
处理器	Intel Corei7-4790 3.60 GHz
内存	32GB
RAM	128GB

4.2 对比实验设置

为全面评估本文提出的云原生下可观测性模型的性能,本研究将从以下3个方面进行对比实验。

a) 数据采集开销。为验证在使用eBPF程序进行数据采集时对系统调用时延产生的损耗,首先在内核探针处无eBPF代码时,重复5万个系统调用,分别计算Kprobes、Uprobes、Tracepoints、USDT 4类探针探测调用事件的平均时间开销,并作为数据基线。然后在内核探针处部署eBPF程序,重复同样数量的系统调用,计算出平均时间开销,对比得出eBPF程序对系统调用带来的影响。

b) 链路追踪效果。为验证研究提出的可观测性模型在对真实微服务进行追踪时的追踪效果,本文选

择基于Spring Boot[12]和Istio[13]2种不同架构的微服务，分别使用Jaeger[14]、Zipkin[15]和本文提出的基于eBPF的可观测性模型进行链路追踪，并使用Grafana进行追踪结果的绘制和展示。通过Grafana生成的火焰图，对比不同可观测追踪工具在实际链路追踪中的效果。

c）链路追踪开销。为验证研究提出的可观测性模型对真实微服务性能产生的影响，同样选择Spring Boot[12]和Istio[13]2种不同架构的微服务作为测试目标，首先在没有可观测性工具的情况下部署这些微服务，记录吞吐量和延迟之间的关系并以此作为基线，然后分别安装Jaeger[14]、Zipkin[15]和本文提出的可观测性模型并重新对微服务的性能进行评估，对比不同可观测性追踪工具对微服务性能带来的影响。

4.3 实验结果分析

4.3.1 数据采集开销

通过对Kprobes、Uprobes、Tracepoints、USDT 4类探针在对调用事件进行探测时的平均时间开销进行计算，得到如图5所示的调用时间开销对比图。

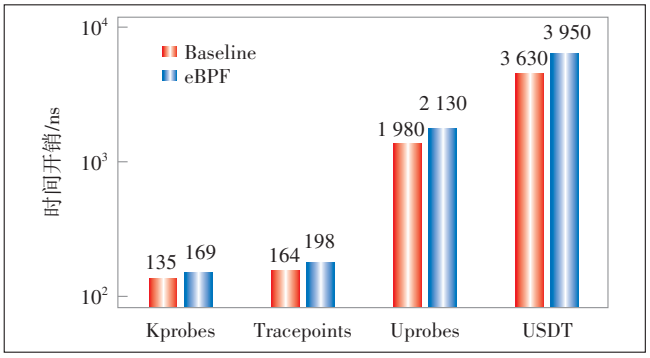

图5　内核探针时间开销对比

从图5可以看出，在Linux内核探针处部署完eBPF程序后，当使用eBPF程序进行可观测数据采集时，eBPF程序给每个系统调用带来的额外时间开销最大不超过320 ns，对于特别耗时的内核I/O操作来说，这个时间开销几乎可以忽略不计。因此，可以说明基于eBPF技术的可观测数据采集方式不会给系统调用带来明显的时间开销。

4.3.2 链路追踪效果

本研究选用Jaeger对Spring Boot架构下的微服务调用进行链路追踪，选用Zipkin对Istio架构下的微服务进行追踪，并使用本文提出的可观测性模型进行同步追踪，得到如图6所示的链路追踪效果火焰图。

通过实验发现，Jaeger只为单个调用追踪构造6个span，Zipkin生成了12个span。相比之下，本文提出的

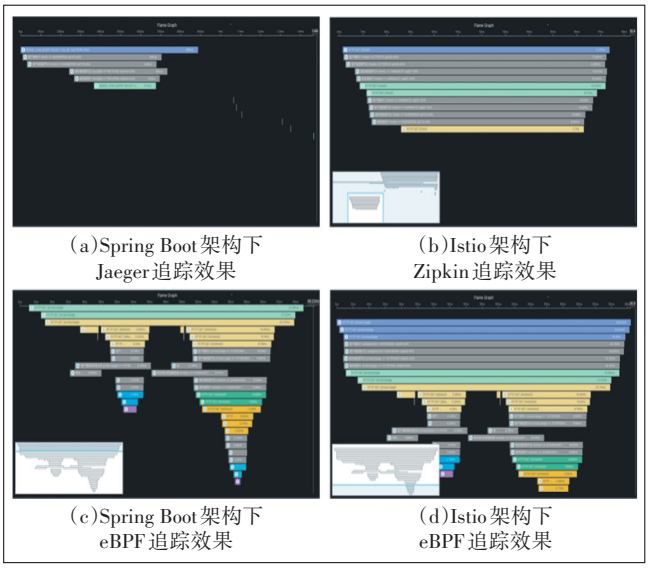

图6　不同架构下微服务全链路追踪效果对比

可观测性模型单次跟踪最少创建了38和39个span，能够将虚拟化所实现的逻辑通信逐步展开，实现对服务调用全链路的每个调用节点的逐跳追踪，与Jaeger和Zipkin可观测工具相比，追踪效果具有明显的提升。

4.3.3 链路追踪开销

本研究选用Jaeger和Zipkin作为基线方法，与本文提出的基于eBPF的可观测模型一起，在不同微服务架构下进行追踪，得到如图7和图8的性能评估折线图。

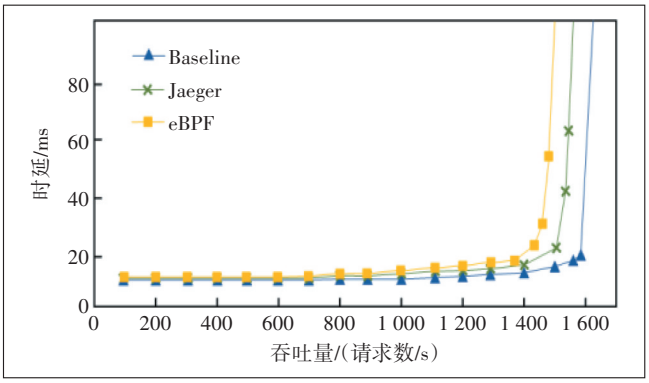

图7　Spring Boot架构下微服务全链路可观测追踪开销对比

通过观察图7和图8，可以看出在没有部署可观测追踪工具的情况下，Spring Boot架构的微服务的最大吞吐量大约是每秒1 580个请求，在部署Jaeger和本文提出的基于eBPF的可观测性模型后，最大吞吐量分别降至每秒1 520个请求和每秒1 480个请求，对性能产生的开销分别为4%和6%。对于Istio架构的微服务来说，在部署Zipkin和本文提出的基于eBPF的可观测

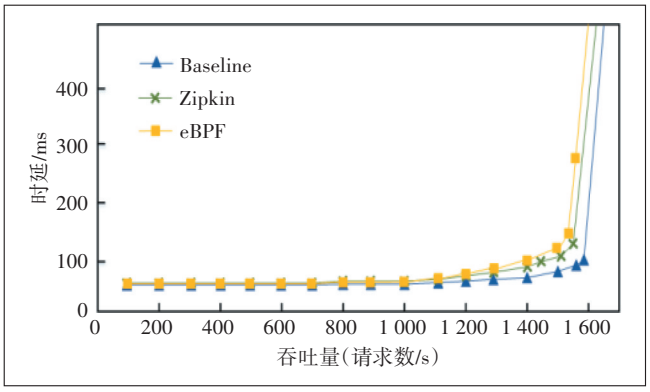

图8 Istio架构下微服务全链路可观测追踪开销对比

性模型后，最大吞吐量分别降至每秒780个请求和每秒760个请求，对性能产生的开销分别为2.5%和5%。

结合图7所示的链路追踪效果对比图，虽然本文提出的可观测性模型在开销性能上略高于其他跟踪工具，但在微服务调用追踪的效果上，本文提出的可观测模型的链路明显更全，能够清晰便捷地展示一次调用中各服务节点的网络状态、应用状态等内容，提供了对程序内部运行状态的持续分析能力。综上，实验结果表明，与Jaeger和Zipkin可观测追踪工具相比，本文提出的可观测性模型在链路追踪效果方面具有一定的优势。

5 总结与展望

本文为解决传统可观测性系统中数据相互割裂、数据采集影响性能、数据检索困难等问题，提出一种云原生下的可观测性模型，并基于eBPF技术提出一种无侵入式的可观测数据采集和追踪方法，实现对云内服务的持续分析；结合规范化的数据标签处理方法，形成统一可观测数据标签体系；最后，通过实验验证了本文提出的可观测模型对服务性能带来的影响最大不超过6%，在全链路追踪效果上具有较大的优势。

在构建统一标签体系的研究过程中发现，对标签进行融合和规范化的过程中存在大量冗余标签，可能某一个指标数据会存在上百个标签，对标签使用者带来较大的挑战。因此，在下阶段的研究中，将致力于通过智能编码对标签进行压缩和筛选，进一步提高数据标签的准确性和实用性，实现可观测数据的快速检索以及告警故障的根因定位。

参考文献：

[1] PAHL C, BROGI A, SOLDANI J, et al. Cloud container technologies: a state-of-the-art review[J]. IEEE Transactions on Cloud Computing, 2019, 7(3): 677-692.

[2] POURMAJIDI W, ZHANG L, STEINBACHER J, et al. A reference architecture for observability and compliance of cloud native applications[EB/OL]. (2023-02-22)[2024-03-06]. https://arxiv.org/abs/2302.11617.

[3] Wikipedia. Observability[EB/OL]. [2024-03-06]. https://en.wikipedia.org/wiki/Observability.

[4] 刘畅. 基于eBPF的容器网络可观测性方法与实践[D]. 杭州：浙江大学，2020.

[5] NIEDERMAIER S, KOETTER F, FREYMANN A, et al. On observability and monitoring of distributed systems - an industry interview study[C]//Service-Oriented Computing: 17th International Conference, ICSOC 2019. Toulouse: Springer-Verlag, 2019: 36-52.

[6] KRATZKE N. Cloud-native observability: the many-faceted benefits of structured and unified logging--a multi-case study[J]. Future Internet, 2022, 14(10): 274.

[7] CASSAGNES C, TRESTIOREANU L, JOLY C, et al. The rise of eBPF for non-intrusive performance monitoring[C]//NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium. Budapest: IEEE, 2020: 1-7.

[8] ABRANCHES M, MICHEL O, KELLER E, et al. Efficient network monitoring applications in the kernel with eBPF and XDP[C]//2021 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN). Heraklion: IEEE, 2021: 28-34.

[9] LIU C, CAI Z G, WANG B S, et al. A protocol-independent container network observability analysis system based on eBPF[C]//2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS). Hong Kong: IEEE, 2020: 697-702.

[10] LEVIN J, BENSON T A. ViperProbe: rethinking microservice observability with eBPF[C]//2020 IEEE 9th International Conference on Cloud Networking (CloudNet). Piscataway: IEEE, 2020: 1-8.

[11] SOLDANI D, NAHI P, BOUR H, et al. eBPF: a new approach to cloud-native observability, networking and security for current (5G) and future mobile networks (6G and beyond)[J]. IEEE Access, 2023 (11): 57174-57202.

[12] Spring Boot. Spring-boot-istio-jaeger-demo[EB/OL]. [2024-07-06]. https://github.com/chanjarster/spring-boot-istio-jaeger-demo.

[13] Istio. Bookinfo application[EB/OL]. [2024-07-06]. https://istio.io/latest/docs/examples/bookinfo/.

[14] Jaeger. Jaeger: open source, distributed tracing platform[EB/OL]. [2024-07-06]. https://www.jaegertracing.io/.

[15] Zipkin. Zipkin[EB/OL]. [2024-07-06]. https://zipkin.io/.

作者简介：
杨晨，助理工程师，硕士，主要从事云内监控产品的研发设计工作；童博，高级工程师，硕士，主要从事新型IP网络技术、SDN及云网创新产品的研究、研发工作；赵纯熙，工程师，硕士，主要从事网络创新产品、云内监控产品的研发设计工作。