计算机研究与发展
Journal of Computer Research and Development
DOI: 10.7544/issn1000-1239.202330543
61(3): 600−613, 2024

一种基于深度学习的微服务性能异常检测方法

方浩天　李春花　王　清　周　可
（武汉光电国家研究中心（华中科技大学）　武汉　430074）
（1479794847@qq.com）

A Method of Microservice Performance Anomaly Detection Based on Deep Learning

Fang Haotian, Li Chunhua, Wang Qing, and Zhou Ke
（*Wuhan National Laboratory for Optoelectronics* (*Huazhong University of Science and Technology*), *Wuhan* 430074）

Abstract　Microservice architecture is increasingly favored by cloud applications due to its good scalability and maintainability. Meanwhile, the complex interactions among microservices make it more difficult to detect performance anomalies in the system. Existing methods cannot adequately establish the complex relationship among microservices cross different call paths and their corresponding response time, resulting in low accuracy of anomaly detection and inaccurate root cause positioning. In this paper, we propose a Transformer based microservice performance anomaly detection and root cause positioning method TTEDA (Transformer trace explore data analysis), which constructs a call chain with microservice call sequence and its response time series, then captures the call relationship among microservices via self-attention mechanism, and the correlation between the response time of microservice and its call path is established through an encoder-decoder architecture, thus the normal response time distribution of microservice across different call chains is obtained. Based on the learned normal pattern, TTEDA can achieve accurate call chain anomaly detection and pinpoint the anomalies at the microservice level. Further, TTEDA uses the relationships among microservices and the propagation of anomalies to perform reverse topological sorting on abnormal microservices, achieving accurate and fast root cause localization. The effectiveness of TTEDA is evaluated on the dataset of the open source benchmark microservice system Train-Ticket and AIops Challenge dataset. Compared with similar methods AEVB, Multi-LSTM, and TraceAnomaly, TTEDA has an average precision improvement of 48.6%, 30.2%, and 3.5%, and an average recall improvement of 34.7%, 11.1%, and 4.1%. Compared with the root localization algorithms MonitorRank and TraceAnomaly, the accuracy of root localization is improved by 35.4% and 6.1%.

Key words　microservice; anomaly detection; root cause localization; call chain; Transformer

摘　要　微服务架构因具有良好的可扩展性和可维护性越来越受到云应用软件的青睐. 与此同时, 微服务之间复杂的交互使得系统的性能异常检测变得更加困难. 现有的微服务性能异常检测方法均不能很好地建立跨不同调用路径的微服务及其对应的响应时间之间的复杂关系, 导致异常检测准确率不高、根因定位不准确. 提出了一种基于 Transformer 的微服务性能异常检测与根因定位方法 TTEDA（Transformer trace explore data analysis）. 首先将调用链构建为微服务调用序列和对应的响应时间序列, 然后借助自注意力机制捕捉微服务之间的调用关系, 并通过编码器-解码器建立微服务的响应时间与其调用路径之间的关

收稿日期: 2023-06-21; **修回日期**: 2023-12-28
基金项目: 国家自然科学基金重点项目 (62232007); 国家自然科学基金创新群体项目 (61821003)
This work was supported by the Key Program of the National Natural Science Foundation of China (62232007) and the Innovation Group Project of the National Natural Science Foundation of China (61821003).
通信作者: 李春花 (li.chunhua@hust.edu.cn)

联关系，从而获得微服务在不同的调用链上的正常响应时间分布. 基于学习到的正常模式判断调用链的异常，并可将异常精确到微服务级别. 进一步地，利用微服务之间的调用关系以及异常的传播方式，对出现性能异常的微服务进行反向拓扑排序，实现了准确快速的根因定位. 在开源基准微服务系统 Train-Ticket 的数据集和 AIops 挑战赛数据集评估了 TTEDA 的有效性，相比于同类异常检测方法 AEVB, Multi-LSTM, TraceAnomaly, 精确率平均提高了 48.6%, 30.2%, 3.5%, 召回率平均提高了 34.7%, 1.1%, 4.1%. 相比于根因定位算法 MonitorRank 和 TraceAnomaly, 根因定位的准确率分别提高了 35.4 个百分点和 6.1 个百分点.

关键词 微服务；异常检测；根因定位；调用链；Transformer

中图法分类号 TP391

随着云计算的发展，越来越多的应用软件使用微服务架构来构建. 微服务架构[1]将一个大型应用程序拆分成许多小型的、松散耦合的微服务，可以更好地适应快速变化的业务需求. 然而，基于微服务架构的软件服务的故障也影响了用户的正常使用，并给企业带来造成了不小的损失. 例如，2018 年亚马逊在一次大型销售活动中因故障停机 1 h，造成了高达 1 亿美元的损失. 因此，保障线上微服务的高可靠性和高可用性十分重要[2]，一旦系统出现故障，需要及时发现并诊断问题以采取相应措施进行修复.

随着微服务系统的规模和复杂性的不断增加，同一个微服务可能会出现在不同的调用链中完成不同的业务请求，导致微服务的性能指标（例如响应时间）存在多个正常的数值区间，这使得常规的异常检测方法在微服务场景下变得不适用. 同时，微服务之间复杂的交互使得理解微服务是单独失败还是级联失败变得更加困难. 一旦某个微服务出现性能异常，异常可能会沿着调用链的调用方向进行反向传播，使得大量微服务表现出性能异常，然而这些被影响的微服务并非都是引起故障的根本原因. 因此，如何准确地发现微服务系统中的异常并定位异常的根本原因成了运维工作的难点. 另一方面，系统的运维数据（监控指标、日志以及调用链）规模变得越来越大. 据 Gartner 预测，企业 IT 基础设施平均每年生成的 IT 运营数据正在以 2~3 倍的速度不断增长，使得运维人员很难通过人工分析发现和诊断系统的各种问题.

由于微服务系统较高的复杂性，现代工业微服务系统通常配备了分布式跟踪系统，它可以跟踪微服务系统中跨服务实例的请求执行，采集系统的跟踪数据，用于分析和监视微服务应用程序的行为. 同时，面对系统海量的数据和更加复杂的软件架构，伴随着人工智能、机器学习等技术的兴起，智能运维[3]应运而生. 将运维知识、运维数据和人工智能结合，能够实现更加有效的异常检测与根因定位. Liu 等人[4]针对调用轨迹，提出了基于深度贝叶斯神经网络的异常检测方法 TraceAnomaly，通过比较调用轨迹的模式与学习到的正常调用轨迹的模式判断是否存在异常情况. Nedelkoski 等人[5]分别利用变分自编码器和卷积神经网络来检测跟踪异常并识别异常类型，他们还提出了一种多模态长短期记忆神经网络模型[6]，基于系统正常运行时收集的跟踪数据，通过组合 2 个单模态网络来学习跟踪数据中事件序列和响应时间序列的正常模式，基于预测发生的事件以及事件可能的响应时间来判断系统是否出现了异常.

这些方法虽然取得了一定的检测效果，但是都不能很好地建模微服务在跨调用路径下具有不同的响应时间的复杂模式，从而导致异常检测的效率不高，同时对检测的异常缺乏可解释性，也无法快速准确地定位问题的根因.

针对上述问题，本文针对微服务系统中的调用链，提出基于 Transformer 的性能异常检测与根因定位方法 TTEDA（Transformer trace explore data analysis）. TTEDA 首先将调用链构建为微服务调用序列和对应的响应时间序列. 然后，借助自注意力机制捕捉微服务之间的调用关系，并通过编码器-解码器建立微服务的响应时间与其调用路径之间的关联关系，使其能够有效学习到微服务在不同的调用链上的正常响应时间分布. 接着，基于学习到的正常模式检测出异常的调用链，并能够检测出异常调用链上的异常微服务. 最后，在异常检测的基础上，TTEDA 利用微服务之间的调用关系以及异常的传播方式，通过对出现性能异常的微服务进行反向拓扑排序来进行准确快速地根因定位. 同时利用所提出的方法设计了微服务性能异常检测与根因定位的框架.

本文的主要贡献包括 3 个方面：

1）提出了一种基于 Transformer 模型的性能异常检测与根因定位方法 TTEDA. 以调用链为研究对象，在考虑微服务调用路径的前提下，利用自注意力机

制学习微服务响应时间的正常模式,能够更加准确地检测到微服务的性能异常.

2)提出了一种调用链的特征构建方式,利用调用链的结构信息和特征信息将调用链构建为微服务调用序列和路径向量,解决了现有特征构建方式向量稀疏和表义不明的问题,能够实现有效的模型学习.

3)设计了针对微服务系统的异常检测和根因定位框架.首先利用历史跟踪数据离线学习调用链的正常模式,然后利用模型在线实时检测系统中的微服务性能异常推断根因,便于操作人员及时采取准确的措施,提高系统的可靠性.

4)在开源基准微服务系统 Train-Ticket 的数据集和 AIops 挑战赛数据集上验证了本文方法的有效性.实验结果表明,相比于同类异常检测方法 AEVB, Multimodal LSTM, TraceAnomaly, 精确率平均提高了 48.6%, 30.2%, 3.5%, 召回率平均提高了 34.7%, 11.1%, 4.1%;相比于根因定位方法 MonitorRank 和 TraceAnomaly, 根因定位的准确率分别提高了 35.4% 和 6.1%.

1 相关工作

近年来,随着云计算技术的不断发展,许多软件应用程序开始部署在云计算平台上[1],单体架构逐渐被面向服务的架构所取代.针对微服务系统的异常检测[7]与根因定位受到越来越多的关注,一些研究以分布式跟踪(调用链)为研究对象,提出了异常检测与根因定位的方法.

Liu 等人[4]提出了 TraceAnomaly 对调用轨迹进行异常检测.该方法将调用链编码为微服务跟踪向量,将其作为输入训练了一个具有后验流的深度贝叶斯神经网络,用于学习调用链的正常模式.通过比较调用轨迹的模式与学习到的正常调用轨迹的模式,来判断是否存在异常情况.在检测出异常跟踪的基础上, TraceAnomaly 通过比较与异常跟踪具有相同结构的跟踪向量,在相同维度上采用 3-sigma 原则确定异常微服务,并将处于最下游的微服务作为可能的异常根因. Nedelkoski 等人[5]使用变分自编码器对应用程序正常状态下的跟踪进行建模,并利用卷积神经网络对注入特定故障的应用程序运行时收集的跟踪信息进行训练,使其可以识别导致微服务性能异常的故障类型,并利用重构误差来检测异常跟踪.此外,他们还提出了一种多模态长短期记忆神经网络模型[6],通过组合 2 个单模态网络来学习跟踪数据中事件序列和响应时间序列的正常模式.其中一个网络用于学习事件之后可能发生的事件的概率分布,另一个网络则用于学习微服务响应时间的概率分布.在线检测时,模型通过预测一个事件之后可能发生的事件以及事件可能的响应时间来判断系统是否出现了异常.如果预测结果中未出现某个事件,或者微服务响应时间不在预测的范围内,则意味着应用程序出现了功能异常或性能异常.张攀等人[8]提出了一种在线实时微服务调用链异常检测方法 MicroTrace,利用基于注意力机制的双向长短期记忆网络模型学习正常调用链的模式.吴佳洁等人[9]提出了一种基于 TCN 和注意力机制的异常检测算法,用于检测从系统采集的时序数据中的点异常和窗口异常. Bogatinovski 等人[10]使用了一种自监督编码器-解码器神经网络对跟踪建模,它能够根据邻近事件的上下文预测跟踪中给定“掩码”位置记录的事件情况.训练好的网络提供了在输入跟踪的任何掩码位置可能出现的事件的概率分布.在线检测时能获得跟踪的每个位置应该记录的事件的排序列表.如果实际记录的事件不在相应位置预测出现的事件之中,视为异常事件.然后计算跟踪的异常得分(即异常事件数除以跟踪长度)来判断系统的功能异常. Jin 等人[11]首先对跟踪数据进行矩阵表示,然后通过主成分分析确定异常跟踪,接着使用无监督算法(如孤立森林[12]、单类支持向量机[13]、局部离群因子和 3-sigma 原则[14])处理微服务收集到的性能指标来分析微服务是否异常.文献 [4−14] 所述的方法虽然取得了一定的检测效果,但是没有很好地考虑到微服务的响应时间和调用路径之间的依赖关系,检测效果有待提高.

针对上述讨论的异常检测方法存在的问题,本文利用分布式跟踪数据,提出基于 Transformer 的跨调用路径的微服务异常检测方法,能够在考虑微服务的响应时间与调用路径的关联关系的前提下学习到微服务响应时间的正常分布模式,更加准确地检测到微服务的性能异常.同时以可解释的方式识别出调用链中的异常微服务,在异常检测的基础上能够直观快速地定位异常的根因.

2 TTEDA 方案描述

2.1 设计思想

由于微服务系统较高的复杂性,现代工业微服务系统通常配备了分布式跟踪系统,例如谷歌公司的 Dapper、Cloudera 的 HTrace、Twitter 的 Zipkin 等.分布式跟踪系统可以跟踪微服务系统中跨服务实例

的请求执行，能够用于分析和监视微服务应用程序的行为. 该系统将微服务操作的每次调用记录为一个跨度(Span)并将每个外部请求的执行过程，即实现同一用户请求的所有调用，记录为跟踪(Trace)，也称调用链. 本文对 AIops 挑战赛提供的某开源微服务系统中采集到的跟踪数据进行了分析，通过观察到的现象设计了本文的方案. 表 1 展示了跟踪数据的相关数据属性、格式以及示例，主要包括时间戳、对象名、span_id、trace_id、处理时间、操作名、parent_span 等信息.

由于微服务系统高度的动态性和复杂性，一个微服务同时会在不同的业务逻辑中发挥作用，即同一个微服务会出现在多条调用链中. 图 1(a)中展示了 4 个微服务在正常情况下部分响应时间的分布情况. 可以观察到，每个微服务的响应时间会呈现多个不同的区间，例如微服务 *A* 的响应时间有一部分小于 50 s，一部分集中在 50 s 附近，还有一部分大于 50 s，即出现了多个正常的指标区间. 进一步地，图 1(b)展示了不同调用链中同一微服务的部分响应时间的分布情况，在调用链 1 中微服务的响应时间主要集中在区间 2～4 s. 而在调用链 2 中微服务的响应时间主要集中在区间 50～100 s. 这说明了微服务的响应时间与其所处的调用链之间存在着一定的关联性. 微服务的响应时间受多个因素影响，其中包括微服务本身执行的操作以及从调用入口微服务到该使用服务的调用路径. 由于不同调用链涉及的调用链操作和调用的路径不同，导致微服务在不同的调用链中会表现出不同的响应时间特征. 这种微服务间调用链关联性的认识对于分析和优化微服务系统的性能至关重要.

以往的研究中，一些工作[5,11]会针对每个微服务的响应时间进行建模，学习微服务的正常模式. 然而由于微服务的响应时间在正常情况下也会出现显著不同，所以可能导致建模失效，并因此带来不准确的检测结果. 一些工作[4,6,8-9]考虑了微服务的响应时间和微服务的调用路径之间的关联关系，但是缺乏一定的可解释性，在检测到异常时往往无法确定调用链中的异常微服务，因此也难以定位异常的根因. 为此，本文基于 Transformer[15]提出一种适用于微服务系统的性能异常检测与根因定位方法，在建模微服务响应时间的正常分布时，考虑了微服务所处的调用路径. 本文针对调用链数据设计了具有明确意义的特征构建方式，并利用 Transformer 模型的自注意机制捕捉微服务之间的调用关系，通过编码器-解码器架构建立微服务的响应时间与其调用路径之间的关联关系，从而学习微服务在跨调用路径下响应时间的正常分布情况. 基于此检测异常调用链并确定其中的异常微服务，根据异常的传播模式有效地定位导致该异常的根因微服务.

2.2 异常检测

2.2.1 异常检测框架

本文针对微服务系统设计了基于 Transformer 的异常检测框架 TraceCatcher，如图 2 所示，该框架分为离线训练和在线检测 2 个部分.

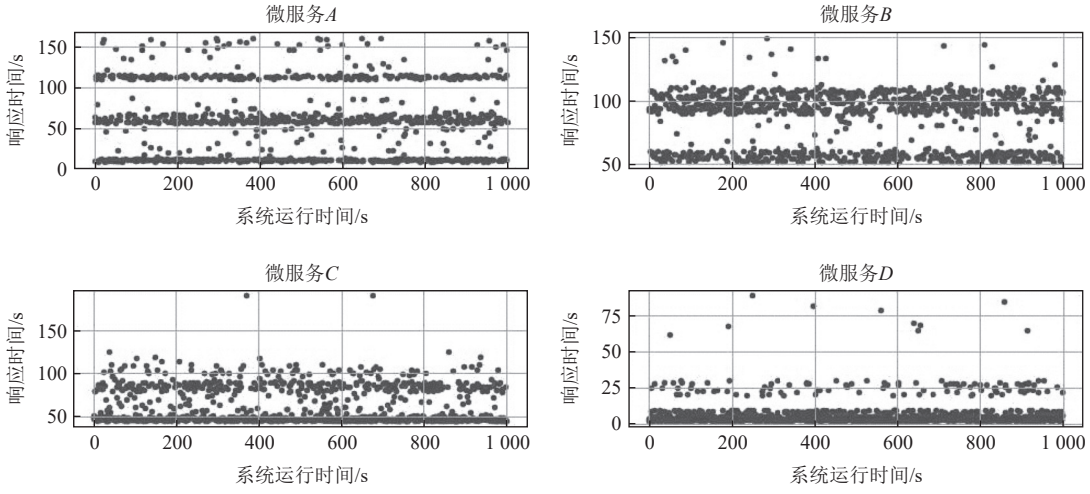
在离线训练阶段，TraceCatcher 利用从系统采集的跨度日志，将其组装成跟踪，并进一步处理得到训练样本. 在组装跟踪的过程中，通过跨度中传递的 Trace_ID 聚合属于同一跟踪的所有跨度，并使用深度优先搜索算法搜索跟踪中从根节点到叶子节点的所有路径，得到不同的微服务调用序列和对应的响应时间序列. 为了提高模型的鲁棒性和降低模型的运算复杂度，本文使用 CART[16]回归树对同一微服务的多个响应时间进行分箱，将分箱后的微服务响应时间作为特征值构建路径向量，最终得到多条成对的微服务调用序列和路径向量组成的训练样本，分别作为 Transformer 模型编码器和解码器的输入. 通过模型学习跟踪中不同微服务的响应时间的正常分布情况.

在线检测阶段，将新产生的跨度日志经过相同

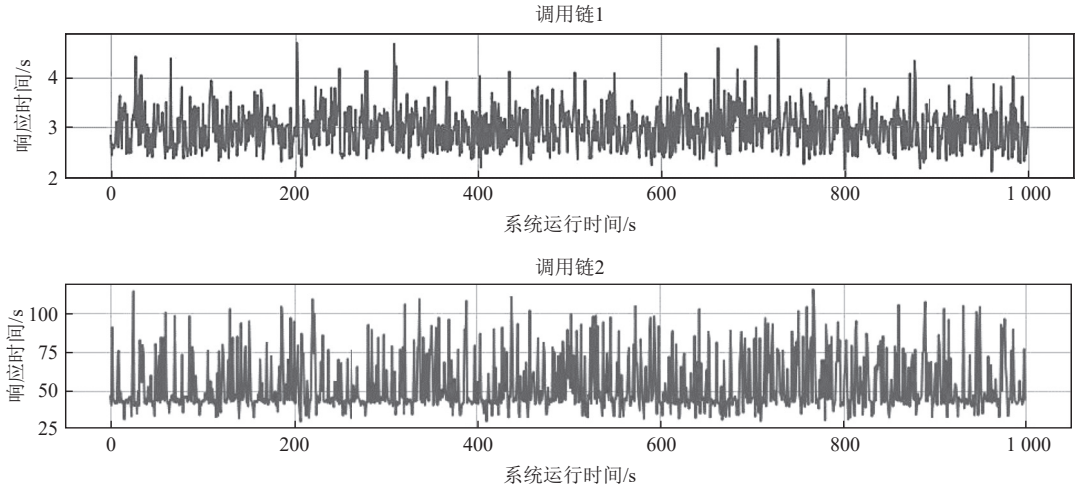
Table 1 Trace Data Attributes and Content Instances

表 1 跟踪数据属性与内容实例

id	属性名	中文含义	数据格式	示例
1	timestamp	时间戳	正整数（INT）	1647705600361
2	cmdb_id	对象名	字符串	frontend-0
3	span_id	请求链路中节点的唯一 id	进行加密过的 16 位 16 进制字符串	a652d4d10e9478fc
4	trace_id	全局唯一请求 id	进行加密过的 16 位 16 进制字符串	9451fd8fdf746a80687451dae4c4e984
5	duration	处理时间	正整数（INT）	49877
6	operation_name	执行的操作名	字符串	CheckoutService/PlaceOrder
7	parent_span	请求链路中上一级节点的 span_id	进行加密过的 16 位 16 进制字符串	952754a738a11675



(a) 同一微服务的多个响应时间



(b) 同一微服务在不同调用链中响应时间的分布

Fig. 1 Relationship between the microservices' call trains and its response time distribution

图 1 微服务的调用链及其响应时间分布的关系

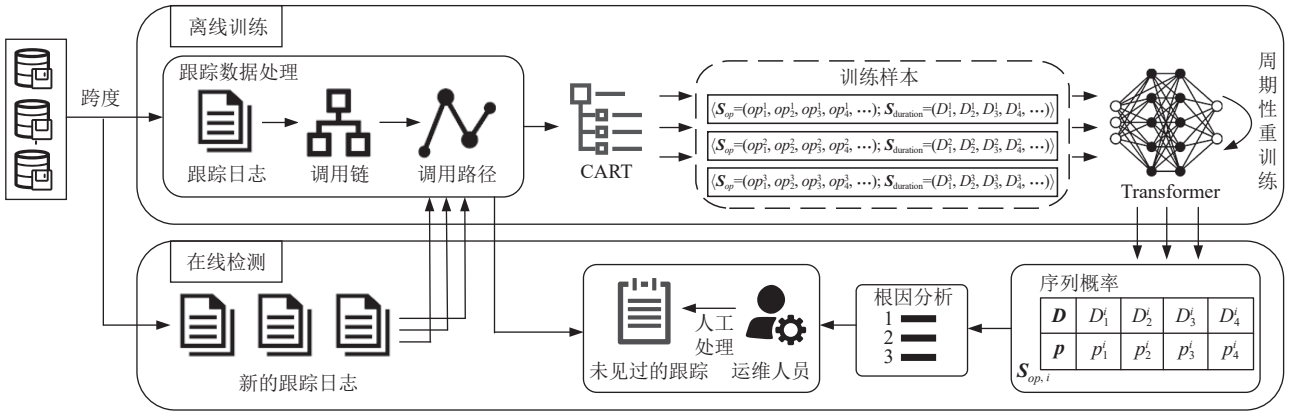


Fig. 2 Transformer-based anomaly detection framework TraceCatcher

图 2 基于 Transformer 的异常检测框架 TraceCatcher

的数据处理得到微服务调用序列和路径向量. 将微服务调用序列作为训练好的 Transformer 模型编码器的输入，经过模型的编码器-解码器层生成一个输出向量，并经过模型的线性层和 Softmax 层输出每个微服务的特征值的概率分布，利用概率分布获取到路径向量中微服务实际特征值的概率，作为微服务的

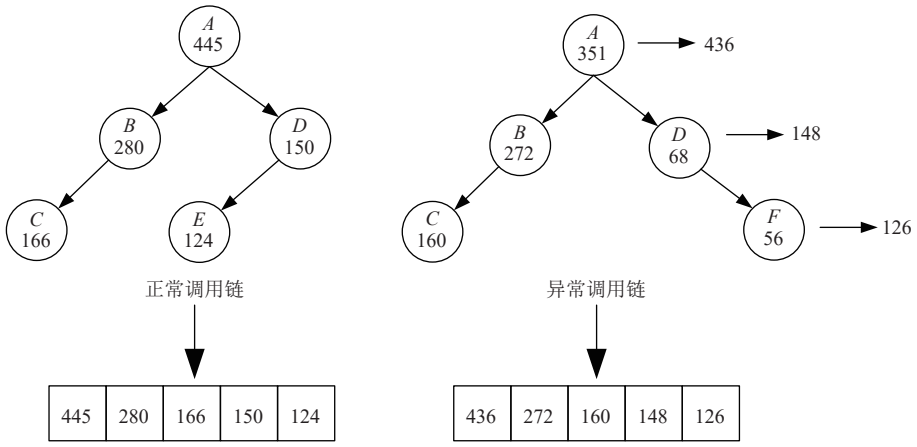
异常得分来判断微服务是否出现了性能异常.

2.2.2 特征构建

在检测微服务性能异常的过程中，需要先将跟踪数据构建为特征向量的形式用于建模. 现有的方法通常是将微服务系统中的调用链以某种编排方式，例如先序、后序或者广度优先遍历等，表示为 1 个 1 维向量. 下面介绍 2 种常见的方式.

1)1 维特征向量构建. 这种方式通过对调用树进行先序遍历来对微服务进行排序，然后利用微服务的特征值(如微服务 *A* 的响应时间 445 s)来构建表示调用链的特征向量，能够反映微服务之间的部分先后调用关系，但并不能完全地体现微服务之间的调用关系. 在一些情况下，无法做到有效区分正常和异常的调用链. 如图 3 所示，经过特征构建之后，正常调用链 1 和异常调用链 2 的 1 维特征向量非常相似，并因此会导致模型错误地将异常调用链 2 识别为正常调用链 1.

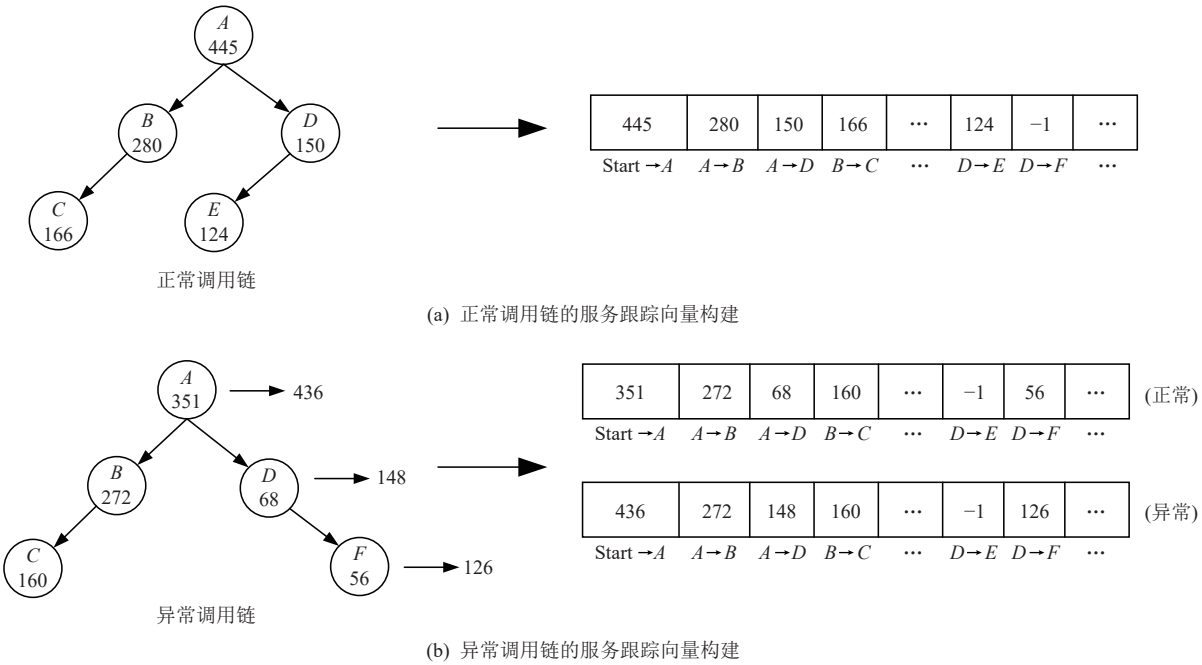
2)服务跟踪向量构建. 除了 1 维特征构建方式，在 TraceAnomaly 中提出了一种不同的针对调用链的 1 维特征向量构建方式，并将其称之为服务跟踪向量(service trace vector, STV). 如图 4 所示，该方法将所有调用链中出现的不同的调用对(调用者，被调用者)对应到向量的不同维度上，在向量中每个维度的值代表相应调用对中被调用者的响应时间. 对于未出



(a) 正常调用链的1维特征向量构建　(b) 异常调用链的1维特征向量构建

Fig. 3 Construction of similar one-dimensional feature vectors with normal call chain and abnormal call chain

图 3 正常调用链与异常调用链相似的 1 维特征向量构建



(a) 正常调用链的服务跟踪向量构建

(b) 异常调用链的服务跟踪向量构建

Fig. 4 TraceAnomaly service trace vector construction

图 4 TraceAnomaly 服务跟踪向量构建

现在调用链中的调用对，该方法用“−1”来设置向量中对应维度的值，以此表示该调用对在向量中缺失. 当调用关系过多，而一条调用链中包含的调用关系较少时，向量中会出现大量的无效维度，会导致向量中有效维度过于稀疏，在一定程度上会影响模型训练的收敛速度和检测的效果.

为了解决上述2种特征构建存在的问题，同时更好地支持模型训练和检测异常并定位异常根因，本文基于调用链的领域知识提出了一种新的方法，该方法将跟踪数据表示为成对的微服务调用序列和路径向量用于建模. 如图5所示，在同一条调用链中，一个微服务可能会被调用多次，每次调用可能执行不同的业务逻辑，在执行不同的业务逻辑时微服务会执行不同的操作，可能会有多个特征值. 本文通过构建基于事件表示的调用链树形结构，将每个节点表示为一个调用事件，如节点 $A\rightarrow B$ 表示微服务 A 调用了微服务 B 的事件. 然后使用深度优先搜索算法从调用链的树形结构中提取出不同的调用序列，并从节点信息中获取特征(响应时间)，构建相应的路径向量.

采用上述方式，可以有效地保留调用链的结构和特征信息，同时避免了特征向量过于稀疏的问题，使模型能够更加有效地建模服务的正常模式. 此外，对于不同的调用链类型，服务调用序列和路径向量的维度会有所不同，由于Transformer能够处理不同长度的序列，因此不会对模型建模造成影响.

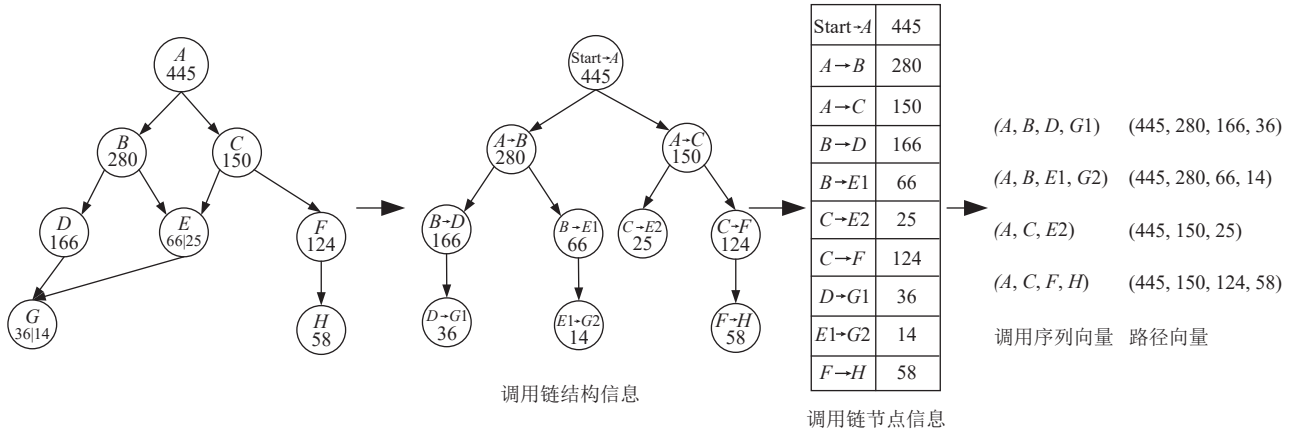


Fig. 5 Constructing call sequence vectors and path vectors

图5 构建调用序列向量和路径向量

2.2.3 模型输入

通过特征构建，调用链被表示为不同的微服务调用序列向量($\boldsymbol{op}_0$, $\boldsymbol{op}_1$, ⋯, $\boldsymbol{op}_{n-1}$)和路径向量($\boldsymbol{D}_0$, $\boldsymbol{D}_1$, …, $\boldsymbol{D}_{n-1}$). 为了使模型能更好地理解系统中的行为，$\boldsymbol{op}_i$ 由表1中对象名和微服务操作名组成.

对于一个微服务调用序列，Transformer首先使用一个 $V\times d_{\text{model}}$ 维的嵌入矩阵将输入序列中的每个微服务映射成词向量用于表示微服务的语义信息，其中 V 是微服务的数量，d_{model} 是词向量的维度. 在训练过程中，模型会通过更新嵌入矩阵的参数来学习更好的语义表示，而在推理时则会使用固定的嵌入矩阵将微服务映射为词向量. 然后需要对序列中的每个元素进行位置编码，引入位置编码有助于模型理解微服务之间的依赖关系，建模事件模板的上下文信息.Transformer采用正弦和余弦位置编码为序列中的不同位置赋予不同的编码向量，编码向量的维度与词向量的维度相同. 对于给定的位置 pos 和给定的维度 i，位置编码的计算方式如式(1)(2)所示：

$$PE_{(pos,i)}=\sin\left(\frac{pos}{10\,000^{\frac{2i}{d_{\text{model}}}}}\right), \tag{1}$$

$$PE_{(pos,i)}=\cos\left(\frac{pos}{10\,000^{\frac{2(i-1)}{d_{\text{model}}}}}\right), \tag{2}$$

其中 $PE_{(pos,i)}$ 表示位置 pos 的第 i 维的位置编码，d_{model} 表示向量的维度. 位置编码的参数 $10\,000^{2i/d_{\text{model}}}$ 是一个基于位置和词向量维度计算的值，当 i 是偶数时这个值是 $10\,000^{2i/d_{\text{model}}}$，当 i 是奇数时这个值是 $10\,000^{2(i-1)/d_{\text{model}}}$. 从位置编码函数的定义中可以看出，对于给定位置 pos 的每个维度 i，其对应的位置编码值都是由不同周期的正弦曲线和余弦曲线组成. 这种位置编码方式可以确保不同位置在所有 d_{model} 维度上被编码成不完全相同的值，使得序列中不同位置上的事件模板都能获得独一无二的位置编码. 其次，由于三角函数的特性，如正弦函数 $\sin(a+b)=\sin(a)\cos(b)+\cos(a)\sin(b)$，其对于 $pos+k$ 处的信息，

可以由*pos*位置编码值计算得到，因此，这种位置编码方式能够表示序列不同距离的元素的相对关系. 位置编码会在模型训练过程中被动态更新和调整，以更好地捕捉序列中的依赖关系.

通过将词向量和位置编码相加，得到模型的输入向量. 词向量用来表征事件模板的语义信息，位置编码用来表征微服务在序列中的位置关系.

2.2.4 模型训练

Tranformer模型主要由编码器和解码器2部分组成，每个部分主要包含了2个子层：1)多头注意力机制层，可以捕捉输入序列中的多层次信息和长距依赖关系；2)前馈神经网络，用于对注意力机制计算的输出作进一步处理.

本文的核心思路是通过关联微服务的响应时间和调用路径来学习微服务在不同调用链中的响应时间的正常分布模式. 为此，首先利用自注意力机制对微服务的上下文信息进行建模，以捕捉微服务之间的调用关系；然后利用编码器-解码器的架构关联微服务的响应时间和调用路径.

1)捕捉微服务之间的依赖关系. 编码器的第1层为多头注意力层. 利用自注意机制，模型可以计算微服务之间的相关性分数，捕捉微服务之间的依赖关系. 给定一组输入矩阵$(\boldsymbol{X}_1,\boldsymbol{X}_2,\cdots,\boldsymbol{X}_n)$，其中$\boldsymbol{X}_i$表示通过词嵌入和位置编码构造的序列中第$i$个微服务的向量表示. 在自注意力机制中，模型首先通过权重矩阵$\boldsymbol{W}^q$, $\boldsymbol{W}^k$, $\boldsymbol{W}^v$对$\boldsymbol{X}_i$进行线性变换，分别得到query，key，value向量$\boldsymbol{q}$, $\boldsymbol{k}$, $\boldsymbol{v}$，在训练过程中，权重矩阵$\boldsymbol{W}^q$, $\boldsymbol{W}^k$会不断更新. 然后，利用向量$\boldsymbol{q}$和$\boldsymbol{k}$计算微服务之间的相关性分数(注意力分数)，计算过程如式(3)所示. 由$\propto_{i,j}$组成的注意力得分(attention score)矩阵代表了微服务调用序列中不同微服务之间的依赖程度.

$$\boldsymbol{Q}\boldsymbol{K}^{\mathrm{T}}=\begin{pmatrix}\boldsymbol{q}_1\\ \boldsymbol{q}_2\\ \vdots\\ \boldsymbol{q}_n\end{pmatrix}(\boldsymbol{k}_1\boldsymbol{k}_2\cdots\boldsymbol{k}_n). \tag{3}$$

接着，利用Softmax函数对$\boldsymbol{\alpha}_{i,j}$归一化处理得到$\boldsymbol{\alpha}'_{i,j}$:

$$\boldsymbol{\alpha}'_{i,j}=Softmax\left(\boldsymbol{\alpha}_{i,1},\boldsymbol{\alpha}_{i,2},\ldots,\boldsymbol{\alpha}_{i,n}\right)=\exp\left(\boldsymbol{\alpha}_{i,j}\right)/\sum_t\exp\left(\boldsymbol{\alpha}_{i,t}\right). \tag{4}$$

随后，通过将向量$\boldsymbol{v}$与归一化后的相关性分数进行加权和计算得到表征微服务上下文的向量$\boldsymbol{Y}_i$. 如式(5)所示：

$$\boldsymbol{Y}_i=\boldsymbol{\alpha}'_{i,1}\times\boldsymbol{v}_1+\boldsymbol{\alpha}'_{i,2}\times\boldsymbol{v}_2+\cdots+\boldsymbol{\alpha}'_{i,n}\times\boldsymbol{v}_n=\sum_{j=1}^{n}\boldsymbol{\alpha}'_{i,j}\times\boldsymbol{v}_j. \tag{5}$$

总的来说，通过自注意力机制，对于一组输入矩阵$\boldsymbol{X}=(\boldsymbol{X}_1,\boldsymbol{X}_2,\cdots,\boldsymbol{X}_n)$，可以将其转换为包含上下文信息的另一组矩阵$\boldsymbol{Y}=(\boldsymbol{Y}_1,\boldsymbol{Y}_2,\cdots,\boldsymbol{Y}_n)$，整个过程如图6所示. 因此，模型能够有效建模微服务的上下文信息，并捕捉到微服务之间的依赖关系.

为了增强自注意力层的学习能力，模型使用了多头注意力机制，即将多个自注意力机制集成在一起，每个注意力模块称为一个注意力头. 使用多头注意力，模型可以更全面地捕捉输入序列的特征. 具体地，在训练过程中，模型会将数据$\boldsymbol{X}$分别输入到多个注意力头中，每个注意力头会分别进行$\boldsymbol{q}$, $\boldsymbol{k}$, $\boldsymbol{v}$向量的计算，并得到不同的特征矩阵$\boldsymbol{Y}^i$，将多个$\boldsymbol{Y}^i$拼接得到最终的特征矩阵$\boldsymbol{Y}$. 编码器的第2层是前馈神经网络(feedforward neural network, FFN)，用于对多头自注意力层的输出进行非线性变换和特征提取，为模型提供更多的非线性变换和学习能力.

2)建立微服务调用序列和响应时间序列的关联关系. 在模型训练过程中，编码器中的多头注意力机

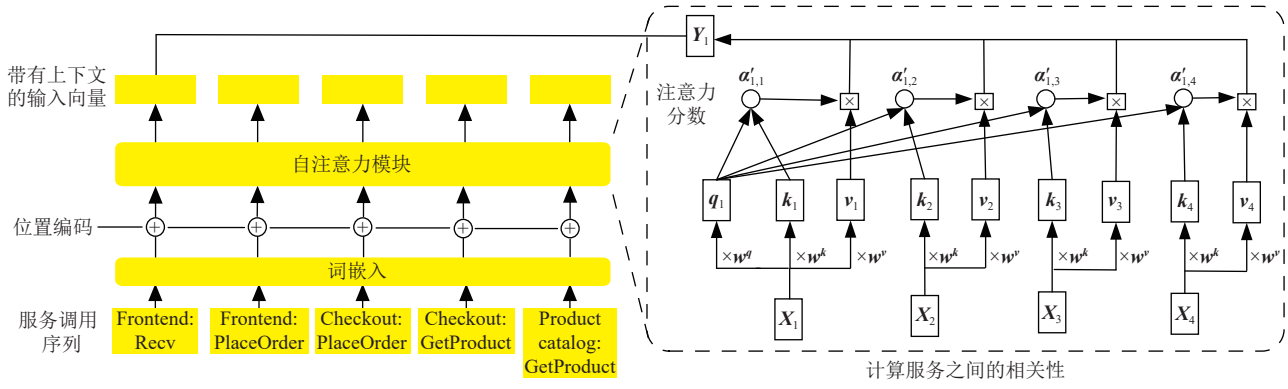


Fig. 6 Computing the vectors with the context of the microservice through the self-attention mechanism

图6 通过自注意力机制计算微服务带有上下文的向量

制和前馈神经网络能够生成带有微服务上下文信息的特征矩阵,从而能够捕捉微服务之间的依赖关系.利用编码器-解码器注意力机制则能够建模微服务调用序列和响应时间序列之间的关联关系,并最终结合微服务的调用关系学习微服务响应时间的正常分布模式.

编码器-解码器注意力模块的部分输入来自于带掩码的多头注意力(masked multi-head attention)模块,该模块使用掩码技术处理路径向量.模型在推理时,因为在生成目标输出时无法使用到未来时间步信息,因此在训练时采用了掩码的方式,将目标输入序列中当前时间步之后的内容进行掩蔽.也就是,将需要掩蔽的向量设置为负无穷,在经过点积和 Softmax 计算注意力得分时,这些位置的权重会趋于零,不会对后续的计算结果产生影响.

在编码器的注意力模块中,输入向量被用于构建向量 $\boldsymbol{q}$, $\boldsymbol{k}$, $\boldsymbol{v}$,而在编码器-解码器注意力模块中,向量 $\boldsymbol{q}$ 来自于带掩码的多头注意力层对路径向量处理后的输出,向量 $\boldsymbol{k}$ 和向量 $\boldsymbol{v}$ 则来自编码器对微服务调用序列处理后得到的特征矩阵 $\boldsymbol{Y}$.通过 Attention 的方式,即注意力计算过程,从而使得微服务调用序列和响应时间序列在训练过程中被关联起来.

3)学习微服务响应时间的正常分布模式.解码器输出的特征向量在经过一层线性层和 Softmax 层之后,会为目标序列的每个位置输出一个概率分布.通过模型预测的概率分布与真实目标序列之间的差异,计算模型的损失函数.

具体地,模型的损失函数基于目标序列与模型预测的响应时间序列之间的交叉熵损失函数计算得到.解码器的输出通过线性层和 Softmax 函数,可以得到每个微服务的特征值的概率分布情况.将模型生成的目标序列中下一个响应时间特征值对应的概率作为该时间步的预测值.该预测值与真实值之间的交叉熵损失函数可通过式(6)来计算.训练时,模型旨在为每个时间步预测下一个响应时间,通过计算预测值与真实值之间的差异,最小化损失函数的值来不断更新模型参数,以此达到建模微服务正常响应时间分布的目的.

$$L=-\sum_{t=1}^{T_y}\sum_{i=1}^{n}y_{t,i}\lg\hat{y}_{t,i}, \tag{6}$$

其中 T_y 表示目标序列的长度,$y_{t,i}$ 表示目标序列在时间步 t 时第 i 个特征值的 one-hot 表示,$\hat{y}_{t,i}$ 表示模型在时间步 t 时预测的第 i 个特征值的概率值.

本文使用大量的从跟踪数据中提取的微服务调用序列和路径向量进行无监督训练.考虑到在训练样本中可能会存在一些异常的噪声数据,但由于模型采用随机梯度下降进行训练,每次按照输入数据中的最小批次的数据去更新模型参数,这使得模型只学习到输入序列中最显著的模式.因此,模型的最终性能不会受到训练集中的少量异常数据的影响.

2.3 根因定位

根因定位是现代基于微服务的软件系统运维中的一个重要环节,通过快速准确地定位系统异常的根本原因,可以有效地提高系统的可靠性和稳定性.本文提出的方法在异常检测基础上,利用微服务的调用关系,同时能够快速准确定位异常的根因.

针对一条异常调用链,本文的异常检测方法能够细粒度检测出调用链中的异常微服务,在此基础上,对异常微服务进行反向拓扑排序,以形成排序列表.具体而言,通过遍历调用链中的异常微服务,迭代输出异常出度(指向异常微服务的边的个数)为 0 的微服务,每输出一个异常微服务,就将调用它的微服务的异常出度减 1,直到输出所有的异常微服务.然后按照列表中的顺序输出不存在调用关系的微服务,将其确定为异常的根本原因.如图 7 所示,在线检测时,调用路径中的根因微服务 G 和微服务 E 以及被影响上游微服务会获得较低的异常得分从而被判定为异常微服务,然后在这些异常微服务中不断寻找异常出度(即是否有指向异常服务的边)为 0 的节点,将其选出并更新指向节点的异常出度,最终得到排序列表 $[G,E,D,C,B,A]$,将其作为根因列表输出,其中微服务 G 和 E 排序靠前且不存在依赖关系,会被判定为最终根因.在一些情况下,调用链中可能会出现多个根因,如图 7 所示,本文的异常检测与根因定位方法同样适用于多根因的情况.最后将根因列表反馈给运维人员,运维人员可以更全面地了解系统的运行状态,以便分析系统故障原因并采取相应措施.

3 实验评估

3.1 数据描述

本文的数据集来自于 AIops 挑战赛的数据集以及一个开源的基于微服务架构的火车订票系统 Train-Ticket.

1)AIops 挑战赛数据集.该数据来源于基于微服务架构的模拟电商系统.系统基于谷歌公司开源的 hips-

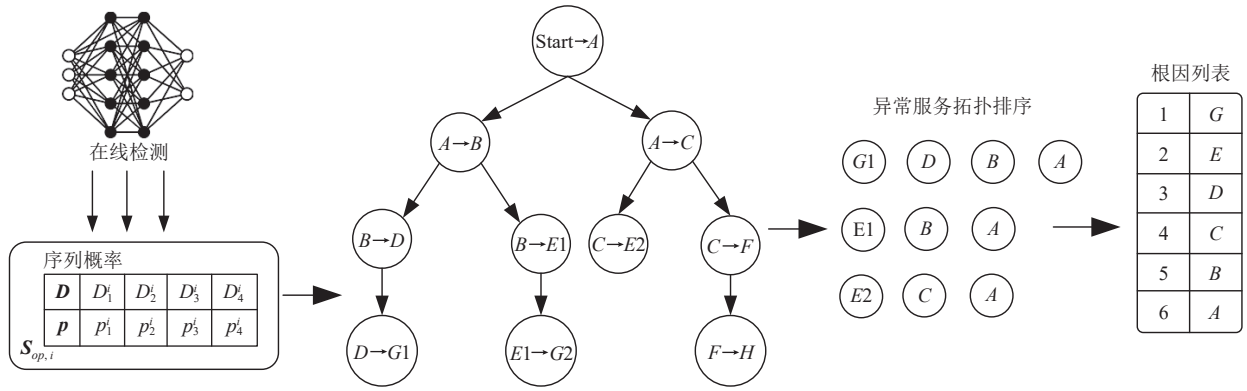


Fig. 7 Locating the root cause by detecting anomalous microservices

图 7 通过检测的异常微服务定位根因

tershop 进行改造，采用动态部署架构，主要包含 10 个核心服务和 6 个虚拟机. 每个服务部署了 4 个 Pod，共有 40 个 Pod，40 个 Pod 动态部署在 6 个虚拟机上，分别在服务(Service)级别、Pod 级别和虚拟机节点(Node)级别注入故障. Service 级别故障是指对该服务下的所有 Pod 都注入故障，而 Pod 级别的故障只针对某一个 Pod 进行故障注入. 其中 Service 级别和 Pod 级别的故障主要是针对 k8s 容器进行故障注入，包括网络延迟、CPU 压力突增、内存压力突增、磁盘读写压力等. 针对 Node 级别的故障注入包括内存压力突增、磁盘空间满、磁盘读、磁盘写、CPU 压力等. 这些故障部分会导致服务的响应时间异常. 表 2 展示了部分故障注入信息.

Table 2 Partial Fault Injection Information

表 2 部分故障注入信息

时间戳/s	级别	Cmdb_id	故障类型
1 647 737 295	Node	Node-6	Node 节点 CPU 故障
1 647 739 015	Pod	Cartservice2-0	k8s 容器内存过载
1 647 743 634	Pod	Currencyservice-0	k8s 容器读 I/O 过载
1 647 749 225	Service	Currencyservice	k8s 容器读 I/O 过载开源
1 647 769 665	Pod	Adservice2-0	k8s 容器网络延迟
1 647 770 896	Service	Shippingservice	k8s 容器网络延迟

2) Train-Ticket 数据集. 火车票订票系统是目前最大的开源微服务系统之一，并且已经广泛应用于现有的一些研究中[17-19]. 该系统一共包含 41 个微服务，实现了完整的订票流程. Train-Ticket 通过 Kubernetes 部署在 7 台物理机器上运行，其中每个服务都部署了多个实例. 通过系统在正常运行状态下收集正常数据集用于训练，由于偶发的系统问题，可能会存在一小部分的异常数据. 异常数据采用故障注入的方法构造，通过网络模拟工具模拟网络延迟来实现服务的响应时间异常. 此外，考虑了在 3 个不同级别组件上注入故障，包括微服务、容器和 API 级别. 最后，训练集中包含了系统在无故障注入条件下收集到的 214 782 条调用链数据，测试集中一共包含 35 977 条调用链数据，其由 31 964 条正常调用链数据和 4 013 条异常调用链数据组成.

3.2 实验设置

实验环境如表 3 所示. 为了验证模型的性能，本文选用准确率、精确率、召回率、*F*1-Score 作为评价指标.

Table 3 Experimental Environment Configuration

表 3 实验环境配置

组成	基本配置
CPU	AMD Ryzen 73700X 8-Core Processor
内存	16 GB DDR4
显卡	NVIDIA GeForce RTX 3060
硬盘	2 TB HDD
操作系统	Windows 10
CUDA	Cuda 11.4
环境库	Python v3.8，Numpy，Panda，Pytorch

3.3 模型性能分析

3.3.1 异常检测效果评估

本文分别在 AIops 数据集和 Train-Ticket 数据集上对本文方法进行了评估.AIops 数据集中包含从系统中采集到的连续 4 天的各种类型的数据(指标、日志和调用链). 本文使用其中的调用链数据对本文方法 TTEDA 进行评估，调用链的基本信息如表 1 所示. 每天的调用链数据规模如表 4 所示.

表 4 中 Path 数量表示所有的 Trace 中包含的微服务调用序列的数量. 第 1 天的数据 (Day-1) 是系统在正常状态下运行时采集的数据，其他 3 天分别注

Table 4 AIops Dataset Information

表 4 AIops 数据集信息

数据	Span 行数	Trace 数量	Path 数量
Day-1	9 211 341	421 984	3 996 879
Day-2	9 168 470	420 102	3 977 743
Day-3	9 027 465	413 955	3 918 783
Day-4	9 004 532	412 958	3 911 309

入了不同类型的故障. 每次采用 1 天的调用链数据进行模型训练. 测试集来自于注入故障的后 3 天数据, 其中在不同层级注入的不同类型的故障并不是都会导致微服务的响应时间异常, 少部分经过被注入故障的微服务的请求可以被正常处理.

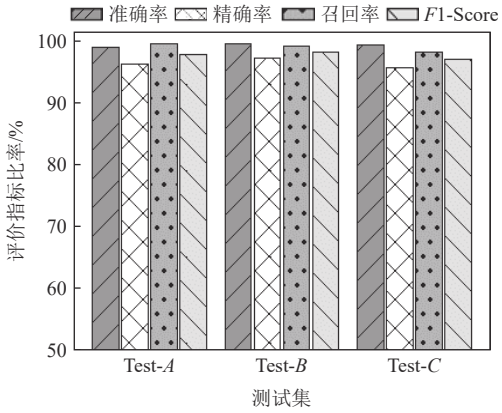
为了筛选出合适的测试集, 首先根据故障注入时间以及注入单元筛选出故障注入后 10 min 内经过被注入故障的微服务的所有调用链数据, 然后根据相同结构的调用链的响应时间模式, 将显著偏离调用链响应时间模式的调用链和微服务标记为异常. 具体而言, 本文基于分布密度和3σ原则划分数据, 密度阈值设为 0.1. 将数据集中各个调用链中的各个位置的微服务响应时间按降序排列, 在故障注入的一段时间内的调用链数据中, 如果调用链中的某个位置上的微服务响应时间超过对应位置微服务响应时间的密度阈值和$\mu+3\sigma$阈值, 则将该调用链和微服务标记为异常. 测试集的数据规模如表 5 所示.

Table 5 Test Dataset Information

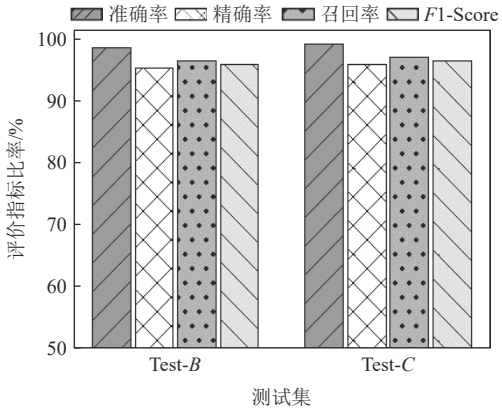
表 5 测试集信息

测试集	Trace 数量	异常 Trace 数量	异常比例
Test-*A*	14 287	2 902	0.203
Test-*B*	13 040	2 743	0.210
Test-*C*	20 999	3 213	0.153

本文分别利用第 1 天和第 2 天的数据训练模型 Model-1 和 Model-2, 然后利用训练好的模型检测测试集中的异常调用链, 最终的各项评价指标数据如图 8 所示. 图 8(a)是通过正常数据训练模型测试得到的各项指标结果, 图 8(b)的训练数据则包含了部分异常数据. 其中 Model-1 在 Test-*A* 上的准确率、精确率、召回率以及 *F*1-Score 分别达到了 98.57%, 95.80%, 99.20%, 97.47%; 在 Test-*B* 上的各项指标分别达到了 99.11%, 96.91%, 98.83%, 97.86%; 在 Test-*C* 上的各项指标分别达到了 98.93%, 95.30%, 97.85%, 96.56%. Medel-2 在 Test-*B* 上的准确率、精确率、召回率以及 *F*1-Score 分别达到了 98.14%, 94.88%, 96.09%,



(a) Model-1的异常检测效果



(b) Model-2的异常检测效果

Fig. 8 Detection effect of Model-1 and Model-2

图 8 Model-1 和 Model-2 的检测效果

95.49%; 在 Test-*C* 上的准确率、精确率、召回率以及 *F*1-Score 分别达到了 98.78%, 95.45%, 96.67%, 96.05%. 可以看出模型具有较高的检测性能, 并且性能不会受到训练集中少量异常数据的影响.

除了在 AIops 数据集上验证模型的性能和效果, 同样在 Train-Ticket 数据集上, 针对不同级别的故障注入, 也分别对模型的异常检测的效果进行了测试, 测试结果如图 9 所示. 在微服务、容器以及 API 这 3 个不同层级上, 模型的精确率分别达到了 89.26%, 85.29%, 83.53%; 召回率分别达到了 91.41%, 84.46%, 88.07%; *F*1-Score 分别达到了 90.32%, 84.87%, 85.74%; 准确率分别达到了 98.65%, 96.37%, 88.99%. 可以看出, 模型对异常具有较高的检测效率.

此外, 本文在 AIops 数据集和 Train-Ticket 数据集上对针对 3 种基线方法 TraceAnomaly[4], AEVB[5], Multi-LSTM[6] 的效果进行了评估, 并与本文方法 TTEDA 进行了对比. 如表 6 所示相比于 3 种异常检测的基准方法, TTEDA 在所有指标上都取得了最高的分数. 在 2 种数据集下, TTEDA 在精确率上平均提高了 48.6 个

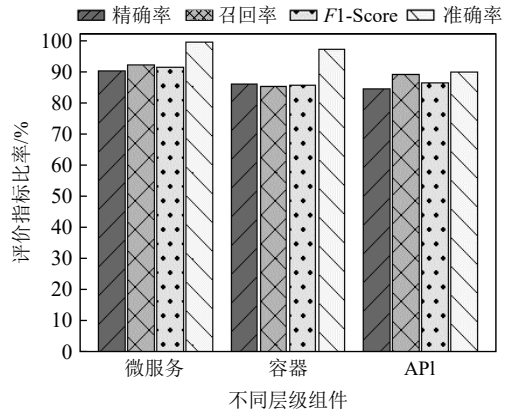


Fig. 9 Detection effect on the test set of different levels of Train-Ticket

图 9 在 Train-Ticket 不同层级测试集上的检测效果

Table 6 Anomaly Detection Effect of Different Methods

表 6 不同方法的异常检测效果 %

方法	AIops			Train-Ticket		
	精确率	召回率	*F*1-Score	精确率	召回率	*F*1-Score
AEVB	45.6	61.4	52.3	39.5	57.3	46.7
Multi-LSTM	61.7	91.2	73.6	60.2	74.7	66.6
TraceAnomaly	90.2	95.8	92.9	83.4	88.2	85.7
TTEDA（本文）	**95.9**	**98.5**	**97.2**	**86.5**	**89.6**	**88.0**

注：黑体值为最优值.

百分点、30.2 个百分点、3.5 个百分点，在召回率上平均提高了 34.7 个百分点、11.1 个百分点、4.1 个百分点.

上述实验结果可以看出，本文方法 TTEDA 的性能要优于其他的基线模型. 这说明本文将调用链表示为成对的微服务调用序列和路径向量是有效的，利用自注意力机制挖掘微服务的响应时间与其调用路径之间的关联关系可以有效地学习到微服务响应时间的正常分布模式，从而提高异常检测性能. 实验结果验证了本文所提出模型的有效性. 同时表 7 给出了不同方法的训练和测试开销，可见，本文方法 TTEDA 在带来良好性能的同时其开销是可以接受的.

Table 7 Training and Testing Overheads of Different Methods on AIops Dataset

表 7 不同方法在 AIops 数据集上的训练和测试开销

方法	训练时间/min	测试时间/s
AEVB	5 250	2 097
Multi-LSTM	36	1 972
TraceAnomaly	268	183
TTEDA(本文)	**112**	**948**

注：黑体值为最优值.

3.3.2 根因定位效果评估

本文选择了 3 种根因定位的基准算法与本文的方法 TTEDA 在 AIops 测试集上对比了根因定位方面的性能，实验结果如表 8 所示. 采用常见的评估排序结果的指标 Top-*k* 准确率（*A*@*k*）来评估算法根因定位的效果. *A*@*k* 计算过程如式（7）所示，其中 $k=1,2,3$.

$$A@k=\frac{1}{n}\sum_{i=1}^{n}pk(y_i,\hat{y}_i), \tag{7}$$

其中 n 表示样本中异常调用链的数量，$pk(y_i,\hat{y}_i)$ 表示通过算法确定的前 k 个根因微服务是否包含实际的根因微服务. *A*@*k* 表示根本原因包含在 Top-*k* 结果中的概率，即前 k 个结果中包含根因微服务的调用链数量占总的异常调用链数量的比例.

Table 8 Comparison of Root Cause Location Effects of Different Methods

表 8 不同方法根因定位效果对比 %

方法	*A*@1	*A*@2	*A*@3
SBFL	62.0	63.5	68.6
MonitorRank	65.7	67.6	72.8
TraceAnomaly	91.8	N/A	N/A
TTEDA(本文)	97.4	N/A	N/A

1）SBFL（spectrum-based fault localization）. 基于频谱的故障定位在程序调试中应用较多. SBFL 在运行测试用例时收集程序元素的覆盖率信息，然后使用预定义的评分函数来计算程序元素的可疑分数. 其基本思想是，由较少通过的测试和更多失败的测试覆盖的程序元素更有可能是根本原因. 这项技术在 Micro-Rank[20] 被用微服务的根因定位. 通过统计微服务出现在异常路径上的次数 O_{ef}、出现在正常路径上的次数 O_{ep}、不包含微服务的异常路径数量 O_{nf} 以及不包含微服务的正常路径数量 O_{np} 来计算微服务的异常评分，其中评分越高的微服务是根因的可能性越大，异常评分的计算为 $abnormal\_score=\dfrac{O_{ef}}{O_{ef}+O_{ep}+O_{nf}}$.

2）MonitorRank[20]. 该方法定期处理应用程序服务产生的调用数据，为每个时间段生成调用图. 基于调用图，利用微服务的邻接矩阵和指标之间的相关性，使用随机游走方法 PageRank 来定位根因.

3）TraceAnomaly[4]. 该方法在检测出异常跟踪的基础上，通过比较所有与异常跟踪具有相同结构的跟踪向量 HSTV 来定位根因，即在向量的每一维上使用 3-sigma 原则找出 1 个或多个异常服务，即显著偏离平均响应时间的服务. 然后将这些异常服务最下

游的微服务作为可能的根本原因.

由表 8 可知, 本文方法 TTEDA 根因定位最准确. 原因分析如下:

SBFL 基于服务的频谱数据来定位根因, 并没有考虑不同调用链的重要性, 不同调用链在系统中出现的次数不同, 出现次数较多的调用链中包含的服务计算的异常得分偏小, 且如果调用链出现异常, 其包含的所有服务的异常次数都会增加, 这其中也包含了正常的服务, 因此效果不佳. MonitorRank 使用随机游走算法定位根因, 如果某个正常的服务在不同的异常调用链中多次出现, 它可能会被方法多次遍历, 因此具有较高的异常分数, 这会导致方法出现误判, 使得算法性能变差. TraceAnomaly 在检测异常上, 通过比较与异常调用链具有相同结构的调用链来定位根因, 虽然算法取得了一定的效果, 但是这增加了一定的比较开销.

而本文方法将调用链表示为具有明确成对关系的微服务调用序列和路径向量, 通过模型将其关联, 从而能够细粒度地检测出异常调用链中的异常服务, 在正确检测到异常调用链的情况下准确定位到根因微服务. 相比于现有的根因定位方法 MonitorRank 和 TraceAnomaly, 根因定位的准确率分别提高了 35.4 个百分点和 6.1 个百分点. 由于本文是按照异常服务的拓扑顺序对微服务进行排序, 即在调用路径中越下游的异常服务排序越靠前, 在单根因的情况下, 如果根因列表中的第 1 位不是实际根因, 那么后续的服务可能均不是根因, 因此不适合计算 $A@2$ 和 $A@3$.

4 结 论

本文以微服务系统的调用链为研究对象, 通过分析发现微服务的响应时间与调用链存在一定的依赖关系, 进而提出一种基于 Transfomer 的微服务性能异常检测与根因定位方法 TTEDA. 通过构建微服务调用序列和路径向量的方式有效表示了调用链的结构信息和服务的特征信息, 借助自注意力机制捕捉微服务之间的调用关系, 并通过编码器-解码器注意力机制建立服务响应时间和调用路径之间的关联关系, 从而能够有效学习到微服务在不同调用链上的正常响应时间分布, 提高了异常检测的准确性, 且将异常检测粒度精确到了微服务级别. 此外, 本文方法主要是检测调用链中微服务的性能异常, 即服务的响应时间异常. 在实际的场景中, 往往还存在功能异常的情况, 即服务调用路径异常(例如调用中断), 这种功能异常不太常见, 但可能与攻击等严重问题有关, 因此依然值得关注. 同时还需要更加细粒度地定位根因, 如与服务有关的根因 KPI, 以便运维人员采取止损措施. 这些工作将作为本文的未来工作进行展开.

作者贡献声明: 方浩天提出了方法思路和实验方案, 并完成了实验和论文初稿的撰写; 李春花指导研究方案设计并修改论文; 王清参与论文修改和实验验证; 周可提出指导性意见并参与论文的修订.

参 考 文 献

[1] He Zhang, Li Shanshan, Jia Zijia, et al. Microservice architecture in reality: An industrial inquiry[C]//Proc of the 2019 IEEE Int Conf on Software Architecture (ICSA). Piscataway, NJ: IEEE, 2019: 51−60

[2] Luo Ruici, Ye Wei, Liu Xueyang, et al. A runtime resource management approach of microservices based on congestion game[J]. Acta Electronica Sinica, 2019, 47(7): 1497−1505(in Chinese)
(罗睿辞, 叶蔚, 刘学洋, 等. 基于拥塞博弈的微服务运行时资源管理方法 [J]. 电子学报, 2019, 47(7): 1497−1505)

[3] Pei Dan, Zhang Shenglin, Pei Changhua, et al. Intelligent operation and maintenance based on machine learning[J]. Communications of the CCF, 2017, 13(12): 68−72(in Chinese)
(裴丹, 张圣林, 裴昶华, 等. 基于机器学习的智能运维 [J]. 中国计算机学会通讯, 2017, 13(12): 68−72)

[4] Liu Ping, Xu Haowen, Ouyang Qianyu, et al. Unsupervised detection of microservice trace anomalies through service-level deep Bayesian networks[C]//Proc of the 31st IEEE Int Symp on Software Reliability Engineering (ISSRE). Piscataway, NJ: IEEE, 2020: 48−58

[5] Nedelkoski S, Cardoso J, Kao O. Anomaly detection and classification using distributed tracing and deep learning[C]//Proc of the 19th IEEE/ACM Int Symp on Cluster, Cloud and Grid Computing (CCGRID). Piscataway, NJ: IEEE, 2019: 241−250

[6] Nedelkoski S, Cardoso J, Kao O. Anomaly detection from system tracing data using multimodal deep learning[C]//Proc of the 12th IEEE Int Conf on Cloud Computing (CLOUD). Piscataway, NJ: IEEE, 2019: 179−186

[7] Chen Xingshu, Jin Yiling, Wang Yulong, et al. Anomaly detection of processes behavior in container based on LSTM neural network[J]. Acta Electronica Sinica, 2021, 49(1): 149−156(in Chinese)
(陈兴蜀, 金逸灵, 王玉龙, 等. 基于长短期记忆神经网络的容器内进程异常行为检测 [J]. 电子学报, 2021, 49(1): 149−156)

[8] Zhang Pan, Gao Feng, Zhou Yi, et al. An online anomaly detection method using on microservice call chain[J]. Computer Engineering, 2022, 48(11): 161−169(in Chinese)
(张攀, 高丰, 周逸, 等. 一种在线实时微服务调用链异常检测方法 [J]. 计算机工程, 2022, 48(11): 161−169)

[9] Wu Jiajie, Wu Shaoling, Wang Wei. Anomaly detection and position algorithm based on TCN and attention mechanism[J]. Netinfo Security, 2021, 32(11): 85−94(in Chinese)
(吴佳洁, 吴绍岭, 王伟. 基于 TCN 和注意力机制的异常检测和定位算法[J]. 信息网络安全, 2021, 32(11): 85−94)

[10] Bogatinovski J, Nedelkoski S, Cardoso J, et al. Self-supervised anomaly detection from distributed traces[C]//Proc of the 13th IEEE/ACM Int Conf on Utility and Cloud Computing (UCC). Piscataway, NJ: IEEE, 2020: 342−347

[11] Jin Mingxu, Lv A, Zhu Yuanpeng, et al. An anomaly detection algorithm for microservice architecture based on robust principal component analysis[J]. IEEE Access, 2020, 8: 226397−226408

[12] Li Kexin, Li Jing, Liu Shuji, et al. GA-iForest: An efficient isolated forest framework based on genetic algorithm for numerical data outlier detection[J]. Transactions of Nanjing University of Aeronautics & Astronautics, 2020, 36(6): 1026−1038

[13] Amer M, Goldstein M, Abdennadher S. Enhancing one-class support vector machines for unsupervised anomaly detection[C]//Proc of the ACM SIGKDD Workshop on Outlier Detection and Description. New York: ACM, 2013: 8−15

[14] Chandola V, Banerjee A, Kumar V. Anomaly detection: A survey[J]. ACM Computing Surveys, 2009, 41(3): 1−58

[15] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need[J/OL]//Advances in Neural Information Processing Systems. 2017[2023-12-24].https://proceedings.neurips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[16] Lawrence R L, Wright A. Rule-based classification systems using classification and regression tree analysis[J]. Photogrammetric Engineering and Remote Sensing, 2001, 67(10): 1137−1142

[17] Zhou Xiang, Peng Xin, Xie Tao, et al. Latent error prediction and fault localization for microservice applications by learning from system trace logs[C]//Proc of the 27th ACM Joint Meeting on European Software Engineering Conf and Symp on the Foundations of Software Engineering. New York: ACM, 2019: 683−694

[18] Zhou Xiang, Peng Xin, Xie Tao, et al. Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study[J]. IEEE Transactions on Software Engineering, 2018, 47(2): 243−260

[19] Hou Xiaofeng, Liu Jiacheng, Li Chao, et al. Unleashing the scalability potential of power-constrained data center in the microservice era[C/OL]//Proc of the 48th Int Conf on Parallel Processing. 2019[2023-12-23].https://dl.acm.org/doi/abs/10.1145/3337821.3337857

[20] Yu Guangba, Chen Pengfei, Chen Hongyang, et al. MicroRank: End-to-end latency issue localization with extended spectrum analysis in microservice environments[C]//Proc of the 2021 Web Conf . New York: ACM, 2021: 3087−3098

Fang Haotian, born in 1999. Master. His main research interests include AIops and AI for storage.
方浩天, 1999 年生. 硕士. 主要研究方向为数据库智能运维、智能存储.

Li Chunhua, born in 1971. PhD, associate professor. Member of CCF. Her main research interests include edge storage, KV storage system, intelligent data management, and AIops.
李春花, 1971 年生. 博士, 副教授. CCF 会员. 主要研究方向为边缘存储、键值存储系统、智能数据管理、数据库智能运维.

Wang Qing, born in 2001. Master. His main research interests include KV storage system and intelligent data management.
王　清, 2001 年生. 硕士. 主要研究方向为键值存储系统、智能数据管理.

Zhou Ke, born in 1974. PhD, professor. His main research interests include AI for storage, big data processing, and AIops.
周　可, 1974 年生. 博士, 教授. 主要研究方向为智能存储、大数据处理、数据库智能运维.