

# 一种在线实时微服务调用链异常检测方法

张攀<sup>1</sup>,高丰<sup>2</sup>,周逸<sup>1</sup>,饶涵宇<sup>3</sup>,毛冬<sup>3</sup>,李静<sup>2</sup>

(1.国家电网有限公司信息通信分公司,北京 100031; 2.南京航空航天大学 计算机科学与技术学院,南京 211106;

3.国网浙江省电力有限公司信息通信分公司,杭州 310016)

**摘要:**微服务架构逐渐成为大规模云应用的主流设计架构,微服务可靠性是云服务提供商亟须处理的关键问题。精确检测并定位微服务应用故障可有效保障应用的可靠性与稳定性,基于微服务调用链的异常检测可在系统发生故障时及时发现系统异常行为并触发告警。针对当前主流检测方法无法保证异常告警的实时性和准确性问题,提出一种基于自然语言处理与双向长短期记忆(BiLSTM)网络的微服务调用链异常检测方法 MicroTrace。对调用链中记录的事件进行解析,将事件表示为语义序列与响应时间序列,利用词汇嵌入式表示算法提取事件的向量化表示,通过基于注意力机制的 BiLSTM 同时检测微服务实例的调用路径与性能异常。在真实微服务调用链数据集上的实验结果表明,该方法的查准率和查全率均可达 96% 以上, F1 度量值相比于多模态-LSTM 方法至少提升了 6.8%。

**关键词:**微服务;调用链;深度学习;异常检测;数据挖掘

开放科学(资源服务)标志码(OSID):



中文引用格式: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.

英文引用格式:ZHANG P, GAO F, ZHOU Y, et al. An online real-time anomaly detection method for microservice call chains[J]. Computer Engineering, 2022, 48(11): 161-169.

## An Online Real-Time Anomaly Detection Method for Microservice Call Chains

ZHANG Pan<sup>1</sup>, GAO Feng<sup>2</sup>, ZHOU Yi<sup>1</sup>, RAO Hanyu<sup>3</sup>, MAO Dong<sup>3</sup>, LI Jing<sup>2</sup>

(1.State Grid Information & Telecommunication Branch, Beijing 100031, China;

2.College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China;

3.Information & Telecommunication Branch, State Grid Zhejiang Electric Power Co., Ltd., Hangzhou 310016, China)

**[Abstract]** The microservice architecture is gradually becoming the mainstream design architecture for large-scale cloud applications, and the reliability of systems based on microservices is a key issue that must be addressed by cloud service providers. Accurately and effectively detecting and locating the faults in microservice applications is crucial for ensuring the reliability and stability of applications. Anomalies in microservice call chains are detected to identify abnormal behaviors in the system in a timely manner and trigger an alarm when the system fails. However, the real-time performance of alarms indicating abnormal behaviors cannot be guaranteed by using current mainstream methods, which require the establishment of a knowledge base by augmenting data from the microservice call chain. Therefore, an anomaly detection method for microservice call chains based on natural language processing and a Bi-directional Long Short-Term Memory (BiLSTM) network is proposed herein. First, events are extracted into semantic sequences and response time sequences. Second, Word2vec is used to extract the vectorized representation of the event, detect the call path anomalies in call chains, and identify the performance anomalies of microservice instances caused by BiLSTM based on the attention mechanism. Finally, the proposed method is verified on an actual microservice call chain dataset. The experimental results show that the precision and recall of the proposed method can exceed 96% and that the detection accuracy improves by at least 6.8% compared with that of the Multimodal-LSTM method.

**[Key words]** microservice; call chain; deep learning; anomaly detection; data mining

DOI: 10.19678/j.issn.1000-3428.0063817

## 0 概述

云计算技术的发展加速了大规模应用迁移上云的

进程,极大提升了大规模应用的开发、扩展和运维效率<sup>[1]</sup>。当前,微服务架构由于在传统面向服务的开发模式上进一步去中心化,因此逐渐成为大规模云应用

基金项目:国家电网有限公司科技项目“业务应用改造上云与全链路运行分析技术研究”(5700-202152169A-0-0-00)。

作者简介:张攀(1989—),男,高级工程师、博士,主研方向为电力信息通信;高丰,硕士研究生;周逸,硕士;饶涵宇、毛冬,工程师、硕士;李静(通信作者),副教授、博士。

收稿日期:2022-01-23 修回日期:2022-04-24 E-mail: gao\_feng@nuaa.edu.cn

的主流设计架构<sup>[2-3]</sup>,然而微服务实例之间复杂的依赖关系在增加故障发生频率的同时也加大了故障诊断难度<sup>[4-5]</sup>,尤其当微服务调用拓扑中某一微服务发生故障时,故障会随着调用拓扑扩散,而且微服务间共享CPU、内存等资源也会引发故障传播,即使微服务之间没有调用关系,也可能在较短时间间隔内产生大量异常告警<sup>[6]</sup>。为避免故障在系统内大范围传播,需要尽快尽早地检测和定位故障发生的根因(Root Cause, RT)<sup>[7]</sup>。目前,微服务系统的监测数据主要包括关键性能指标(Key Performance Indicator, KPI)、日志和调用链<sup>[8]</sup>。KPI数据负责监控系统的资源利用率,例如容器或物理机的CPU、内存以及硬盘等硬件资源利用率<sup>[9]</sup>。日志主要由代码中的日志输出语句输出非结构化的日志条目,通过日志解析方法抽取为结构化的日志模板后检测系统异常<sup>[10]</sup>。调用链数据是以类似图的结构记录微服务级别的调用关系、执行路径以及性能参数<sup>[11]</sup>。目前的微服务性能异常检测方法主要是基于KPI和日志的异常检测,并且只有小部分研究利用了调用链数据。

微服务调用链异常检测可以在系统发生故障时及早检测出受故障影响的微服务实例,并启动故障根因定位,提高故障应对的响应速度,但微服务之间复杂的调用关系和调用链数据的特性给调用链异常检测带来了很大的挑战。第一,微服务之间的调用不仅包含单个服务内部的互相调用,而且包括服务与服务间的跨服务层调用。第二,同一微服务往往对应不同的微服务实例,为了保障业务的持续性,这些微服务实例通常部署在不同的容器或物理机上,跨容器或物理机的调用可能会引入通信时延。第三,微服务架构应用具有多线程、高并发的特性,一个微服务通常会在同一时刻内调用多个子微服务。第四,应用的不断迭代更新通常伴随着微服务调用拓扑的变化,例如新微服务的上线、旧微服务的下线以及微服务原有调用关系的变化。第五,微服务调用关系中会包含大量递归、循环调用。针对上述问题,一些异常检测方法通过构建调用链知识库<sup>[12-14]</sup>,在检测过程中根据匹配算法将新的调用链与知识库中的调用链作对比,进而检测异常调用链。但是,若应用于存在高并发、循环以及递归调用的大规模微服务系统时,该方法需要构建规模庞大的知识库,并且此类方法通常将微服务性能异常检测和调用路径异常检测分为两个独立的任务,而衡量微服务性能的响应时间不仅由微服务本身决定,还受其所有的子调用微服务的影响,因而此类方法检测效果不佳。此外,通过调用链匹配来检测异常的方法计算复杂度过高,检测过程耗时过长,因此更适用于离线异常检测,而不适用于对实时性要求更高的在线异常检测任务<sup>[15]</sup>。为了保证在线检测的实时性,一些基于深度学习的异常检测方法<sup>[16-17]</sup>被提出,但此类方法通常仅能检测一种类型的异常,容易产生异常的漏报,并且往往忽略了调用链中事件序列的上下文信息,导致难以学习到微服务调用路径与响应时间之间的关联关系,因此效果

不佳。

为解决上述问题,受自然语言处理技术在日志异常检测中的成功应用启发<sup>[18-20]</sup>,本文提出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(Bi-directional Long Short-Term Memory, BiLSTM)网络模型学习正常调用链的行为模式。在模型训练过程中,将调用链看作事件序列,对调用链中的事件进行解析,将事件解析为包含调用类型、调用发起网元以及服务网元的事件模板。利用词汇嵌入式表示算法提取调用链中事件序列的上下文信息,获得事件的语义向量表示,将调用链表示为语义向量序列。提取事件中包含的响应时间,获得与语义向量序列相对应的响应时间序列作为微服务性能异常检测的依据。基于语义向量序列和响应时间序列,采用基于注意力机制的BiLSTM实现同时检测调用路径异常和微服务性能异常。

## 1 相关工作

### 1.1 基于调用链知识库构建的异常检测方法

基于调用链知识库构建的异常检测方法<sup>[12]</sup>主要通过采集系统正常运行下产生的调用链,从而构建调用拓扑知识库。此类方法在检测过程中,利用匹配算法将输入调用链的调用拓扑与知识库作对比,进而判定是否存在异常。MENG等<sup>[12,15]</sup>通过构建调用树知识库,在检测时根据限制自上而下映射(Restricted Top-Down Mapping, RTDM)算法<sup>[21]</sup>计算输入调用树与知识库中所有正常调用树的树编辑距离,从而判定是否存在调用路径异常。该方法同时通过构建响应时间矩阵并基于主成分分析方法实现了微服务性能异常检测。JIN等<sup>[13]</sup>提出一种基于鲁棒主成分分析方法的离线调用链异常检测方法,通过构建标准调用树知识库和响应时间矩阵改善了检测精度。CHEN等<sup>[22]</sup>通过构建矩阵草图线性重建包含所有微服务正常响应时间的高维空间,实现了调用链中的微服务性能异常检测。LIU等<sup>[8]</sup>构建包含所有正常调用路径的知识库,然后人工设计调用链的向量化表示,利用深度贝叶斯神经网络实现了同时检测微服务调用路径异常和性能异常。但是,此类基于知识库构建的异常检测方法存在计算复杂度高、检测耗时长等问题,因此仅适用于离线调用链异常检测任务。

### 1.2 基于深度学习的异常检测方法

基于深度学习的异常检测方法<sup>[16-17]</sup>主要利用循环神经网络、深度生成模型等学习正常调用链数据的行为模式,在检测过程中将偏离此模式的输入数据判定为异常。NEDELKOSKI等<sup>[23]</sup>提出AEVB方法,通过对调用链中的事件分类,将相同类型事件的响应时间组成时间序列,转化为时序序列异常检测问题,但此方法不能发现存在调用路径异常的调用链。BOGATINOVSKI等<sup>[24]</sup>提出一种基于自监督学习的异常检测方法MSP,通过训练一个自编码器对

输入调用链中随机遮蔽的事件进行重构,在检测时根据重构结果预测调用链中每个位置上可能出现的事件,但此方法仅能检测微服务调用路径异常,忽略了微服务性能异常。BOGATINOVSKI 等<sup>[16-17]</sup>将调用链异常检测作为日志异常检测的辅助任务,但都仅利用了调用链中部分信息,因此效果不佳。为了实现同时检测调用路径异常和性能异常,减少异常的误报和漏报,NEDELKOSKI 等<sup>[25]</sup>提出一种基于多模态 LSTM(Multimodal-LSTM)方法,但此方法忽

略了调用链中事件序列的上下文关系,并且模型结构也无法有效学习到微服务调用路径与性能之间的关联关系,因此检测效果不能达到最优。

2 微服务调用链异常检测方法

为了实现在线实时的微服务调用链异常检测,本文提出 MicroTrace 方法,主要包含调用链数据解析、事件语义向量化以及调用链异常检测 3 个部分。MicroTrace 总体框架如图 1 所示。

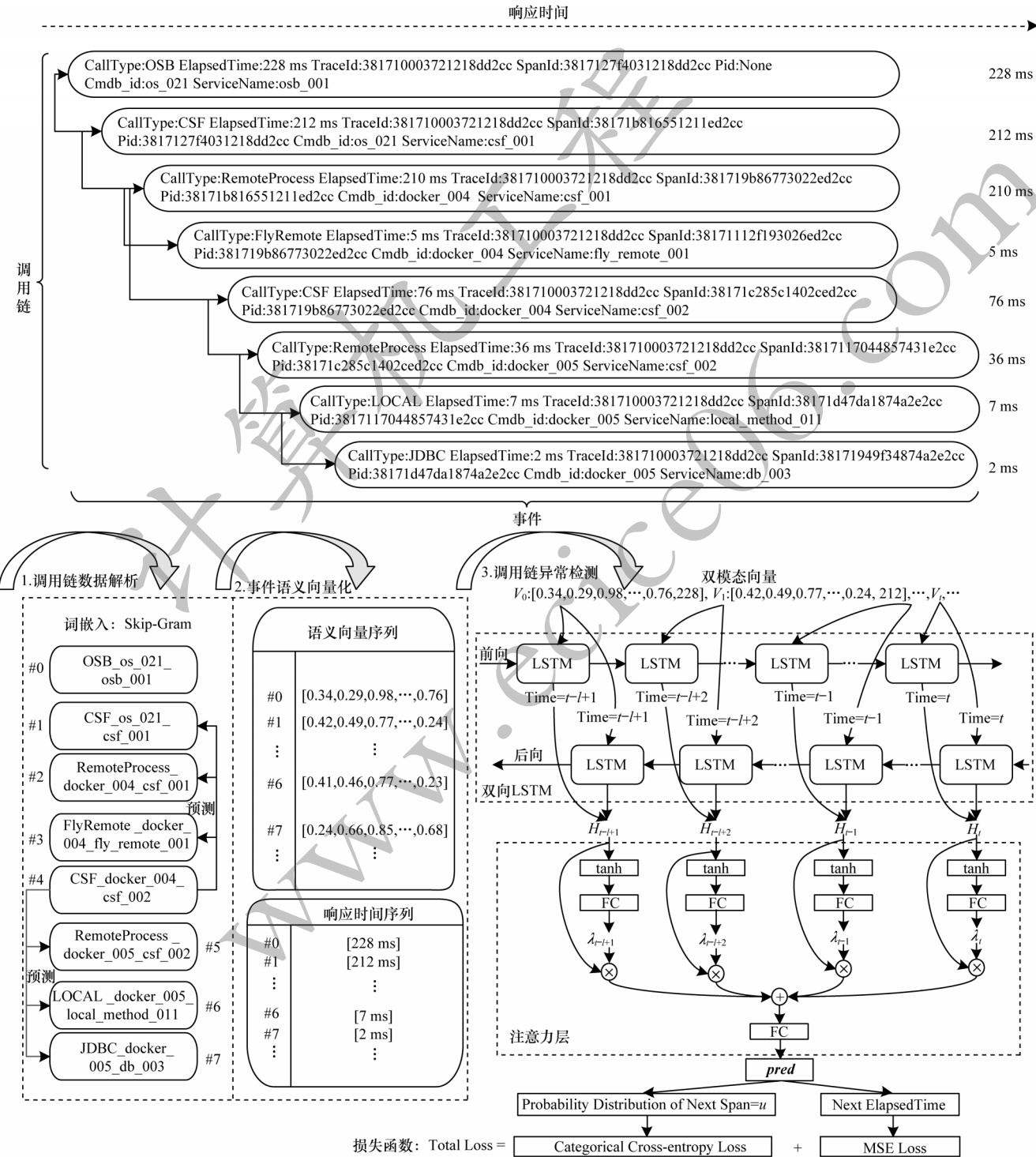


图1 微服务调用链异常检测方法总体框架

Fig.1 Overall framework of anomaly detection method for microservice call chains



## 2.1 调用链数据解析

调用链记录了业务应用内部微服务之间的调用关系以及微服务本身的响应时间。调用链由一系列带有时间戳的 event(称为事件)构成<sup>[11]</sup>,表1为微服务调用链数据示例,给出了一条调用链中的部分数据。本文将调用链  $T$  表示为按时间戳 StartTime 排序的事件序列  $[e_0, e_1, \dots, e_{N-1}]$ , 即  $T = [e_0, e_1, \dots, e_{N-1}]$ , 其中  $N$  代表调用链  $T$  的长度。在表1中, CallType 代表调用类型, StartTime 代表调用发起时间, 即时间戳, ElapsedTime 代表被调用的微服务的响应时间, TraceId 代表此条调用链的唯一标识, SpanId 代表该事件在调用链中的唯一标识, Pid 代表该事件在调用

链中的父事件的 SpanId, Cmdb\_id 代表发起此次调用的网元唯一标识, ServiceName 代表响应此次调用的服务网元(包含微服务和数据库等)。为保证在语义向量化步骤中能数值向量的形式准确地描述事件的行为, 本文采用 CallType、Cmdb\_id 以及 ServiceName 3 个字段对事件进行分类, 并将上述 3 个字段组成正则表达式 CallType\_Cmdb\_id\_ServiceName 作为事件类别的标记, 称为事件模板。例如, 表1中序号为 #0 的事件模板可以表示为 OSB\_os\_021\_osb\_001。本文还提取出事件中的微服务响应时间并组成响应时间序列  $[\eta_0, \eta_1, \dots, \eta_{N-1}]$ 。

表1 微服务调用链示例

Table 1 Example of microservice call chain

序号	CallType	StartTime	ElapsedTime/ms	TraceId	SpanId	Pid	Cmdb_id	ServiceName
#0	OSB	1590164528600	228	3721218dd2cc	127f4031218dd	None	os_021	osb_001
#1	CSF	1590164528609	212	3721218dd2cc	1b816551211ed	127f4031218dd	os_021	csf_001
#2	RemoteProcess	1590164528610	210	3721218dd2cc	19b86773022ed	1b816551211ed	docker_004	csf_001

在完成调用链数据解析后, 一个长度为  $N$  的调用链  $T$  就可以被表示为按时间戳排序的两条序列, 分别是事件模板序列  $T_p = [\rho_0, \rho_1, \dots, \rho_{N-1}]$  以及响应时间序列  $T_\eta = [\eta_0, \eta_1, \dots, \eta_{N-1}]$ 。

## 2.2 语义向量化

为了利用事件序列中的上下文信息, 本文基于自然语言处理思想, 采用词汇嵌入式表示方法 Word2vec<sup>[26]</sup> 提取事件序列中的上下文语义信息并

将事件模板转换为固定维度的数值型向量(称为语义向量化)。由于事件模板的种类较少, 本文省略了原 Word2vec 算法中的负采样过程。Word2vec 主要包含跳字(Skip-Gram)模型和连续词袋(Continuous Bag-of-Words, CBOW)模型两类, 本文采用跳字模型来实现事件模板的语义向量化, 如图2所示, 主要包含事件模板预处理以及语义信息提取两个步骤。

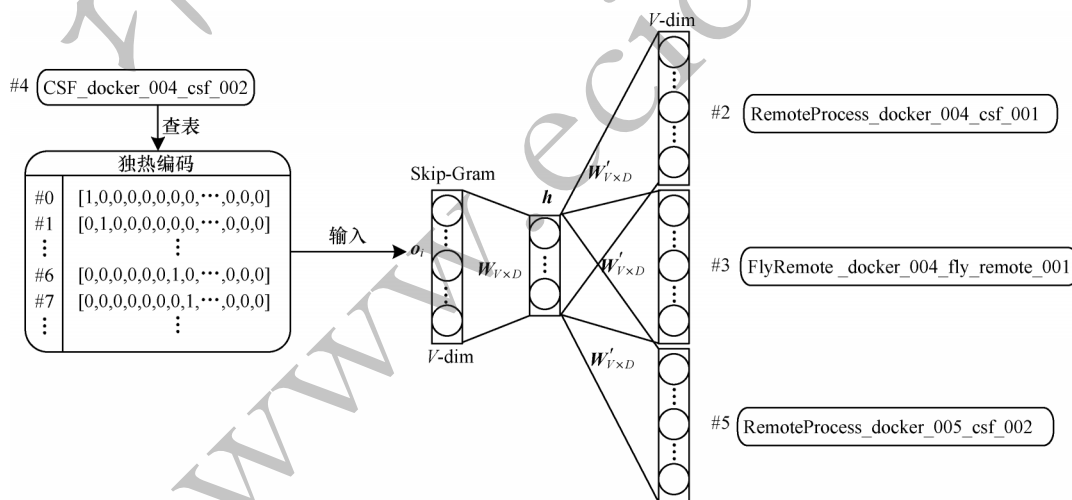


图2 语义向量化框架

Fig.2 Framework of semantic vectorization

### 2.2.1 事件模板预处理

在完成调用链数据解析后会得到一个事件模板集合  $S = \{\rho_0, \rho_1, \dots, \rho_{V-1}\}$ , 其中  $V$  代表事件模板种类。本文利用独热编码(one-hot encoding)将集合  $S$  中的事件模板  $e$  映射为维度为  $V$  的向量  $\mathbf{o}$ , 得到集合  $S$  对应的独热编码向量集合  $O = \{\mathbf{o}_0, \mathbf{o}_1, \dots, \mathbf{o}_{V-1}\}$ , 其中

$$\mathbf{o}_i \in \mathbb{R}^V, i \in [0, V-1].$$

### 2.2.2 语义信息提取

描述事件模板的语义向量需要满足可区分性和可对比性两个需求。可区分性指语义向量可以高度区分不同的事件模板。例如事件模板 CSF\_os\_021\_csf\_001 和 JDBC\_docker\_005\_db\_003 有不同的

行为模式,因此两者的语义向量应该具有较小的余弦相似度。可比性指相似事件应该有相似的语义向量。例如,事件模板 CSF\_os\_021\_csf\_001 和 CSF\_os\_022\_csf\_001 不仅在事件的行为模式上相似,在调用链中的位置也经常有交集(两者都经常作为 OSB\_os\_0xx\_osb\_0xx 的下一个调用),因此两者的语义向量应该具有较高的余弦相似度。为满足以上两点需求,给定一条调用链的事件模板序列  $T_\rho = [\rho_0, \rho_1, \dots, \rho_{N-1}]$ ,通过查询集合  $O$  将  $T_\rho$  映射为  $T_o = [o_0, o_1, \dots, o_{N-1}]$ ,其中,  $o_i \in \mathbb{R}^V, i \in [0, N-1]$ 。在跳字模型中,采用一个事件模板  $\rho_i$  预测它的上下文  $\rho_b$  来完成事件模板的语义向量化,具体过程如下:

首先,将  $\rho_i$  对应的  $o_i$  映射到低维空间中:

$$h = W_{V \times D}^T o_i \quad (1)$$

其中:  $h \in \mathbb{R}^D$  代表  $o_i$  的低维表示,且  $D < V$ 。

然后,根据式(2)和式(3)由事件模板的低维表示  $h$  预测  $\rho_b$ :

$$y = \text{Softmax}(W'_{V \times D} h) \quad (2)$$

$$P(\rho_b = \rho_k | \rho_i) = y_k, k \in [0, V-1] \quad (3)$$

其中:  $y$  为跳字模型的输出;  $P(\rho_b = \rho_k | \rho_i)$  为跳字模型在给定  $\rho_i$  的条件下,预测上下文  $\rho_b = \rho_k$  的概率。模型的优化目标为最大化给定事件模板  $\rho_i$  生成上下文事件模板的概率,等同于最小化式(4)表示的损失函数:

$$E = -\log_a \sum_{\rho_b} P(\rho_b = \rho_i | \rho_i) \quad (4)$$

其中:  $C$  为  $\rho_i$  对应的上下文事件模板的个数。

最后,在模型收敛后,可由式(5)得到事件模板  $\rho_i$  对应的语义向量  $v_i$ :

$$v_i = W_{V \times D}^T o_i \quad (5)$$

### 2.3 异常检测

除了微服务之间复杂的调用关系会对调用链异常检测带来困难以外,如何同时检测调用路径异常和微服务性能异常也是调用链异常检测要面临的挑战之一<sup>[8]</sup>,因此本文采用基于注意力机制的 BiLSTM 作为调用链异常检测的最后一环。在完成数据解析和语义向量化之后,长度为  $N$  的调用链就被表示为语义向量序列  $T_v = [v_0, v_1, \dots, v_{N-1}]$  和响应时间序列  $T_\eta = [\eta_0, \eta_1, \dots, \eta_{N-1}]$ ,为了同时检测调用路径异常和微服务性能异常,本文将上述序列按元素拼接为双模态向量序列  $T_\nu = [V_0, V_1, \dots, V_{N-1}]$  作为 BiLSTM 的输入,其中  $V_i = [v_i, \eta_i]$ 。

本文将调用链异常检测建模为下一个事件模板预测以及时序数据预测任务。BiLSTM 接收长度为  $l$  的子序列  $T_l = [V_{t-l+1}, V_{t-l+2}, \dots, V_t]$  作为输入,从前向和后向两个方向挖掘子序列中包含的信息,并且每个 LSTM 块都会输出一个隐向量  $H$ 。由于不同的事

件对于多分类的结果有不同的影响,并且在预测调用链中靠近根节点的事件时,通常要在子序列中填充占位符,因此在得到 BiLSTM 的输出后,采用注意力机制自动地为子序列中不同的事件施加不同的权重。

首先,由每个 LSTM 块的输出  $H$  得到每个时刻事件的权重:

$$\lambda_i = w_i^a \tanh(H_i), i \in [t-l+1, t] \quad (6)$$

其中:  $w_i^a$  为注意力层中第一个全连接层的参数。

然后,由每个时刻对应的 LSTM 块的输出  $H$  以及对应的权重  $\lambda$  得到注意力层的输出:

$$\text{pred} = \text{Softmax}\left(w' \left( \sum_{i=t-l+1}^t \lambda_i H_i \right)\right) \quad (7)$$

其中:  $\text{pred} = [u, \eta_{\text{pred}}]$ ;  $w'$  为注意力层中第 2 个全连接层的参数。

最后,将  $u$  作为模型输出的事件模板预测结果,  $\eta_{\text{pred}}$  作为模型预测  $t+1$  时刻对应的事件响应时间,即:

$$P(e_{t+1} = e_{i'} | e_{t-l+1}, e_{t-l+2}, \dots, e_t) = u_{i'}, i' \in [0, V-1] \quad (8)$$

获取在系统正常运行一段时间内产生的调用链数据并将其按照一定比例分为训练集和验证集。在模型训练阶段,同时采用多分类交叉熵损失(针对事件模板预测)以及均方误差损失(针对响应时间预测)之和作为模型的总体损失函数,在此损失函数的引导下采用 RMSprop(Root Mean Square Prop)<sup>[27]</sup> 更新模型参数,并对模型参数采用 L2 正则化。待模型收敛后,将验证集输入模型中,首先利用模型的响应时间预测值  $\eta_{\text{pred}}$  与  $t+1$  时刻的观测值  $\eta_{t+1}$  计算均方误差  $r$ :

$$r = (\eta_{\text{pred}} - \eta_{t+1})^2 \quad (9)$$

然后计算  $r$  的均值  $\mu$  以及标准差  $\sigma$ ,根据坎特立不等式(Cantelli inequality),对于  $\forall \varepsilon \geq 0$ :

$$P(r \geq \mu + \varepsilon) \leq \frac{\sigma^2}{\sigma^2 + \varepsilon^2} \quad (10)$$

将  $\varepsilon = \beta\sigma$  代入式(10)中得到:

$$P(r \geq \mu + \beta\sigma) \leq \frac{1}{1 + \beta^2} \quad (11)$$

由于验证集中包含的都为正常的调用链数据,因此多数预测值和观测值之间的均方误差  $r$  会很接近均值  $\mu$ ,在微服务性能异常检测过程中采用  $r_{\text{threshold}} = \mu + \beta\sigma$  作为判断  $r$  是否异常的阈值,如果模型的预测值  $\eta_{\text{pred}}$  与实际观测值  $\eta_{t+1}$  的均方误差  $r > r_{\text{threshold}}$ ,则可判断此事件中响应的微服务存在性能异常。

在调用路径异常检测过程中,由于收敛后的模型学习了正常调用链的行为模式,模型的预测值  $u$

实际上为在给定历史事件子序列后下一个可能出现的事件的概率分布,因此本文根据模型预测的各事件出现概率大小,选出最有可能出现的 $m$ 个候选事件,如果实际观测值不在这 $m$ 个候选事件中,则判断为异常。

MicroTrace模型训练算法如算法1所示。

#### 算法1 MicroTrace模型训练算法

输入 训练集 $\Sigma$ ,验证集 $\Omega$

输出  $pred = [u, \eta_{pred}], r_{threshold}$

```

1. for  $T_v$  in  $\Sigma$  do //开始训练,  $T_v$ 代表调用链
2. for  $T_i$  in  $T_v$  do //  $T_i$ 为  $T_v$ 的子序列
3. 将  $T_i$ 输入到双向LSTM层中
4. 取每个LSTM块的输出  $H_i$ 
5.  $\lambda_i \leftarrow w_i^a \tanh(H_i)$  //计算注意力层权重
6.  $pred \leftarrow \text{Softmax}\left(w' \left(\sum_{i=t-1+1}^t \lambda_i H_i\right)\right)$ 
7. 对  $pred$ 切片得到  $u, \eta_{pred}$ 
8.  $\text{Total Loss} \leftarrow \text{Cross-entropy Loss} + \text{MSE Loss}$ 
9. 利用梯度下降算法更新模型参数
10. end for
11. end for //模型训练完毕
12. for  $T_v$  in  $\Omega$  do
13. for  $T_i$  in  $T_v$  do //  $T_i$ 为  $T_v$ 的子序列
14.  $T_i$ 输入到模型中得到输出  $pred$ 
15.  $r \leftarrow (\eta_{pred} - \eta_{t+1})^2$  //计算模型预测值与观测值的均方根
//误差
16. end for
17. end for
18.  $\mu \leftarrow E(r), \sigma \leftarrow SD(r)$  //  $r$ 的均值和标准差

```

19.  $r_{threshold} \leftarrow \mu + \beta\sigma$  //计算响应时间阈值

20. Return  $pred = [u, \eta_{pred}], r_{threshold}$

### 3 实验结果与分析

#### 3.1 数据集

采用AIOps2020挑战赛公开的预赛数据集,该数据集是由某大型运营商提供的真实数据,共分为黄金业务指标、调用链数据和KPI数据三部分,其中,黄金业务指标反映了业务系统的整体状态,记录了业务系统的平均响应时间、每分钟接收请求数量、每分钟成功响应请求数量以及响应请求成功率。调用链数据记录了该微服务架构的业务系统中各微服务之间的调用关系和执行路径。KPI数据记录了物理机、虚拟机和容器的关键性能指标数据,例如内存利用率、CPU利用率等。本文采用标记为2020\_04\_20中的调用链数据作为训练集,标记为2020\_05\_22、2020\_05\_23以及2020\_05\_24中的调用链数据作为测试集。训练集和测试集的黄金业务指标如图3所示,其中实线为业务系统的平均响应时间(average\_time),虚线为响应请求成功率(success\_rate),圆点为故障注入时间节点。从图3可以看出,在2020\_04\_20训练集中系统运行平稳,而在2020\_05\_22、2020\_05\_23以及2020\_05\_24测试集中由于故障注入导致系统运行状态产生波动。数据提供方注入故障的类型包括容器CPU利用率故障、容器内存利用率故障、数据库类型故障以及主机或容器网络类型故障等。测试集的部分故障注入类型以及时间如表2所示。

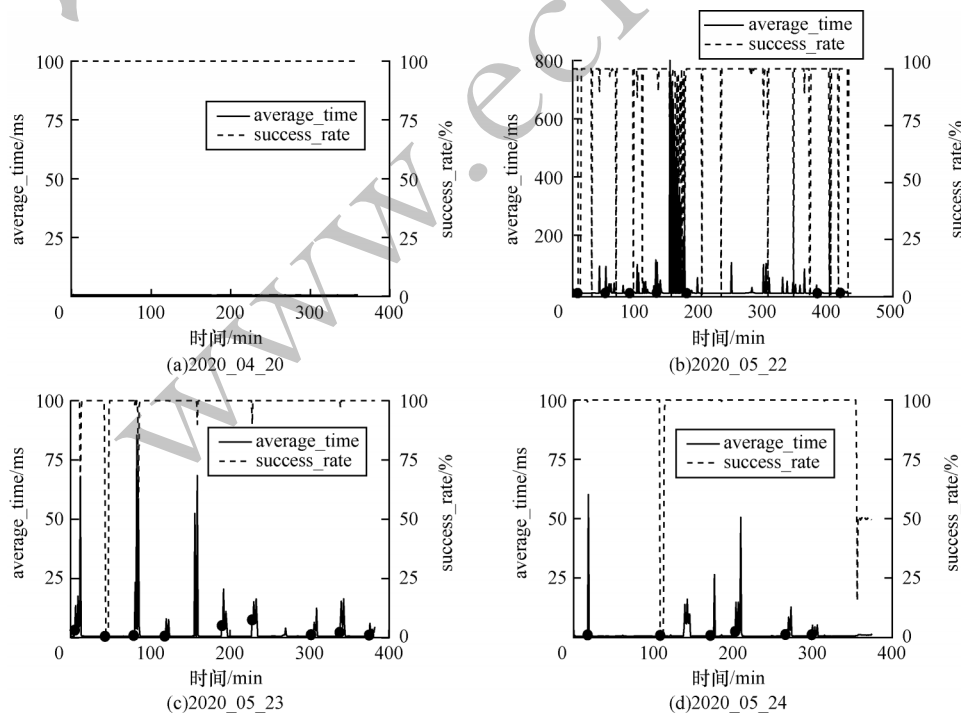


图3 训练集和测试集的黄金业务指标时序图

Fig.3 Timing diagrams of golden business indicators in training set and test set



表2 故障注入类型及时间  
Table 2 Fault injection type and time

序号	故障注入类型	故障注入网元	故障注入开始时间	故障注入持续时间/min
0	db close	db_003	2020-05-22 00:07:00	5
1	CPU fault	docker_001	2020-05-22 00:48:00	5
2	network delay	docker_005	2020-05-22 01:18:00	5
3	network delay	os_018	2020-05-22 01:48:00	5
4	CPU fault	docker_005	2020-05-22 02:18:00	5
5	network delay	docker_007	2020-05-22 05:18:00	5
6	network delay	docker_006	2020-05-22 05:48:00	5

3.2 对比方法与评价指标

选取DeepLog\_A、MSP<sup>[24]</sup>、AEVB<sup>[23]</sup>以及Multimodal-LSTM<sup>[25]</sup>4种当前最优方法作为对比方法,其中DeepLog\_A为DeepLog的变体,为减少计算量,将DeepLog<sup>[10]</sup>中执行路径异常检测模块命名为DeepLog\_A作为对比方法之一。

采用查准率(P)、查全率(R)以及F1度量值(F)作为衡量检测效果的指标。查准率表示在检测出的异常中真异常的比率。查全率表示在所有真异常中被模型标记为异常的比率。F1度量值为综合考虑查准率和查全率的性能衡量指标,计算公式如式(12)所示:

$$F = \frac{2 \times P \times R}{P + R} \tag{12}$$

3.3 检测效果对比

图4给出了本文MicroTrace与4种对比方法的实验结果。从图4可以看出, MicroTrace查准率、查全率以及F1度量值均达到96%以上,相对于检测效果次优的Multimodal-LSTM的F1度量值约提升了6.8%,这主要归因于MicroTrace利用语义向量化有效提取了调用链数据中事件之间的因果关系,通过异常检测模型充分考虑了微服务的响应时间与调用路径之间的关联关系,同时检测出了调用路径异常和微服务性能异常,然而DeepLog\_A、MSP仅能检测调用路径异常,AEVB仅能检测微服务性能异常,因此在面对同时存在两种异常的实验场景下检测效果欠佳。

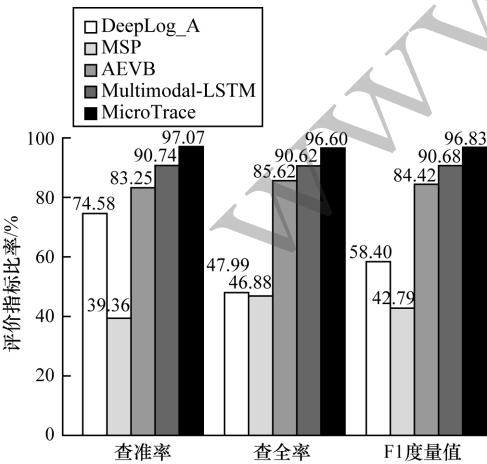


图4 5种异常检测方法的检测效果对比

Fig.4 Comparison of detection effect of five anomaly detection methods

结合图3和表2可以看出,在注入db close(关闭数据库)、db connection limit(数据库访问限制)等故障时,系统响应请求成功率会在故障期间降为0。此类型故障可类比于将系统中某一个微服务的所有实例全部下线,从而导致所有需要调用此微服务的调用链被截断,进而产生异常的调用路径。表3和表4给出了在2020-05-22 00:07:00向db\_003(数据库)注入db close故障时产生的两种类型的异常调用链的部分信息,分别将其称为AT\_1和AT\_2。AT\_1和AT\_2都为系统发生故障时产生的一条完整的调用链,可以从表3和表4看出两者都在调用db\_003时发生了截断,导致一条调用链仅包含极少数事件(正常调用链通常包含几十个甚至几百个事件),即产生了调用路径异常。然而将表3和表4中的异常调用链和表1中正常调用链数据相比可以发现,AT\_1中的事件响应时间与正常调用链中的事件响应时间明显不同,而AT\_2中的事件响应时间却和正常调用链中的差距很小,因此如果仅检测响应时间异常,AT\_2类型的异常调用链极有可能被漏报从而导致查全率下降,这也是AEVB检测效果不佳的原因之一。在注入CPU fault(CPU类型故障)、network delay(网络延迟故障)等故障时,系统的平均响应时间通常会有较大幅度的增加,并有可能导致系统响应请求成功率下降,此类故障可能会使系统中产生调用路径正常但响应时间异常的调用链数据,然而仅考虑调用路径异常的检测方法(比如DeepLog\_A和MSP)在面临此类型异常时会产生漏报或误报。MicroTrace可以同时检测出调用路径异常和响应时间异常,因此在所有方法中检测效果达到了最优。

表3 AT\_1异常调用链

Table 3 Anomalous call chain AT\_1

CallType	ElapsedTime/ms	Cmdb_id	ServiceName
OSB	62	os_021	osb_001
CSF	50	os_021	csf_001
RemoteProcess	48	docker_003	csf_001
FlyRemote	4	docker_003	fly_remote_001
CSF	35	docker_003	csf_002
RemoteProcess	31	docker_008	csf_002
LOCAL	5	docker_008	db_003

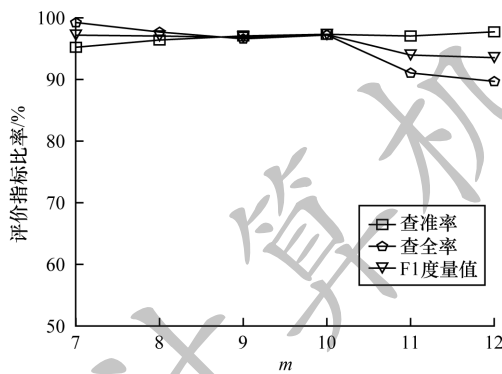
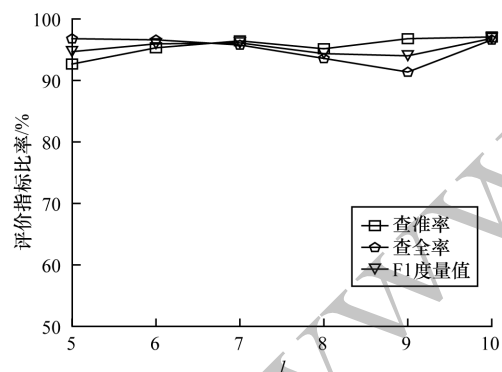
表4 AT\_2异常调用链

Table 4 Anomalous call chain AT\_2

CallType	ElapsedTime/ms	Cmdb_id	ServiceName
OSB	281	os_022	osb_001
CSF	270	os_022	csf_001
RemoteProcess	267	docker_002	csf_001
FlyRemote	4	docker_002	fly_remote_001
CSF	99	docker_002	csf_002
RemoteProcess	34	docker_005	csf_002
LOCAL	5	docker_005	db_003

### 3.4 模型超参数分析

为验证 MicroTrace 对于超参数的鲁棒性,设计模型对于预测候选事件参数  $m$  和时间窗口大小  $l$  的鲁棒性实验,其中,第1个实验固定  $l=10$ ,  $m \in \{7, 8, 9, 10, 11, 12\}$ ,第2个实验固定  $m=9$ ,  $l \in \{5, 6, 7, 8, 9, 10\}$ ,实验结果如图5、图6所示。

图5 超参数  $m$  的实验结果Fig.5 Experimental results of hyper parameter  $m$ 图6 超参数  $l$  的实验结果Fig.6 Experimental results of hyper parameter  $l$ 

从图5可以看出,随着  $m$  的增加,模型查准率会逐渐上升,查全率会逐渐下降,但F1度量值波动不大,这是因为随着  $m$  的增加模型对调用链中的事件异常行为的容忍度会越来越高,致使检测结果中的假阳性数量减少,假阴性数量增加,导致查准率增大,查全率减小。从图6可以看出,随着  $l$  的增大,模型查准率、查全率和F1度量值波动不大,证明了本

文方法对于超参数的变化并不敏感,具有较强的鲁棒性,便于在真实系统环境中部署。

### 3.5 消融实验分析

为验证 MicroTrace 各组成部分对检测效果的影响,设计了该方法的3种变体,分别命名为 MicroTrace\_Alpha、MicroTrace\_Beta 和 MicroTrace\_Gama,其中, MicroTrace\_Alpha 相对于原方法仅利用了调用链数据中事件之间的依赖关系来检测调用路径异常, MicroTrace\_Beta 相对于原方法仅利用了调用链数据中的微服务的响应时间来检测微服务性能异常, MicroTrace\_Gama 相对于原方法删去了事件模板的语义向量化过程,仅采用独热编码对事件模板进行向量化。所有上述方法均采用默认参数,即  $m=9$ ,  $l=10$ ,实验结果如表5所示,其中最优指标值用加粗字体标示。

表5 消融实验结果

Table 5 Ablation experiment results

方法	查准率	查全率	F1度量值
MicroTrace	<b>97.070</b>	<b>96.603</b>	<b>96.836</b>
MicroTrace_Alpha	90.996	49.896	64.451
MicroTrace_Beta	87.481	93.724	90.495
MicroTrace_Gama	95.708	93.360	94.519

从表5可以看出, MicroTrace 在查准率、查全率以及F1度量值上均优于其他变体方法,验证了该方法的各模块对于检测效果的贡献。例如, MicroTrace 的F1度量值相对于其他3种变体方法分别提升了50.2%、7.0%以及2.5%,这体现出同时检测调用路径异常和性能异常的策略可以减少模型的误报和漏报。 MicroTrace\_Alpha 的检测效果远劣于 MicroTrace\_Beta,这是因为大部分故障会表现为服务延迟增加或者请求超时<sup>[14]</sup>,即导致调用路径异常的故障通常也会同时表现为性能异常。 MicroTrace 相对于 MicroTrace\_Gama 的检测效果有较小幅度的提升,这证明了在事件模板向量化的过程中考虑事件之间的依赖关系有助于异常检测模型更好地学习到调用路径与响应时间之间的关联关系。

## 4 结束语

本文提出一种在线实时微服务调用链异常检测方法 MicroTrace,将调用链建模为自然语言序列,采用词汇嵌入式表示算法提取调用链中事件序列中的上下文信息,并利用基于注意力机制的 BiLSTM 模型学习正常调用链的行为模式,实现了同时检测调用路径异常和微服务性能异常。通过在真实调用链数据集上的实验结果证明了该方法的有效性,相比于现有方法减少了异常的漏报和误报数量。下一步将针对异常的微服务集合研究根因定位算法,并在此基础上设计微服务系统故障诊断闭环解决方案,实现系统故障的自动定位与诊断。



## 参考文献

- [1] KRATZKE N, QUINT P C. Understanding cloud-native applications after 10 years of cloud computing—a systematic mapping study[J]. *Journal of Systems and Software*, 2017, 126: 1-16.
- [2] 施凌鹏,朱征,周俊松,等. 面向微服务架构的云系统负载均衡机制[J]. *计算机工程*, 2021, 47(9): 44-50, 58.  
SHI L P, ZHU Z, ZHOU J S, et al. Load balancing mechanism for microservice architecture in cloud-based systems[J]. *Computer Engineering*, 2021, 47(9): 44-50, 58. (in Chinese)
- [3] DMITRY N, MANFRED S S. On micro-services architecture[J]. *International Journal of Open Information Technologies*, 2014, 2(9): 24-27.
- [4] SOLDANI J, TAMBURRI D A, VAN DEN HEUVEL W J. The pains and gains of microservices; a systematic grey literature review[J]. *Journal of Systems and Software*, 2018, 146: 215-232.
- [5] SOLDANI J, BROGI A. Anomaly detection and failure root cause analysis in (micro) service-based cloud applications: a survey[J]. *ACM Computing Surveys*, 2023, 55(3): 59.
- [6] WENG J P, WANG J H, YANG J H, et al. Root cause analysis of anomalies of multitier services in public clouds[J]. *IEEE/ACM Transactions on Networking*, 2018, 26(4): 1646-1659.
- [7] KIM M, SUMBALY R, SHAH S. Root cause detection in a service-oriented architecture[J]. *ACM SIGMETRICS Performance Evaluation Review*, 2013, 41(1): 93-104.
- [8] LIU P, XU H W, OUYANG Q Y, et al. Unsupervised detection of microservice trace anomalies through service-level deep Bayesian networks[C]//*Proceedings of the 31st International Symposium on Software Reliability Engineering*. Washington D. C., USA: IEEE Press, 2020: 48-58.
- [9] XU H W, CHEN W X, ZHAO N W, et al. Unsupervised anomaly detection via variational auto-encoder for seasonal KPIs in Web applications[C]//*Proceedings of 2018 World Wide Web Conference*. Washington D. C., USA: IEEE Press, 2018: 187-196.
- [10] DU M, LI F F, ZHENG G N, et al. DeepLog: anomaly detection and diagnosis from system logs through deep learning[C]//*Proceedings of 2017 ACM SIGSAC Conference on Computer and Communications Security*. New York, USA: ACM Press, 2017: 1285-1298.
- [11] SIGELMAN B, BARROSO L, BURROWS M, et al. Dapper, a large-scale distributed systems tracing infrastructure[EB/OL]. [2021-12-05]. [https://www.researchgate.net/publication/239595848\\_Dapper\\_a\\_Large-Scale\\_Distributed\\_Systems\\_Tracing\\_Infrastructure](https://www.researchgate.net/publication/239595848_Dapper_a_Large-Scale_Distributed_Systems_Tracing_Infrastructure).
- [12] MENG L, JI F, SUN Y, et al. Detecting anomalies in microservices with execution trace comparison[J]. *Future Generation Computer Systems*, 2021, 116: 291-301.
- [13] JIN M X, LÜ A R, ZHU Y P, et al. An anomaly detection algorithm for microservice architecture based on robust principal component analysis[J]. *IEEE Access*, 2020, 8: 226397-226408.
- [14] ZHOU X, PENG X, XIE T, et al. Latent error prediction and fault localization for microservice applications by learning from system trace logs[C]//*Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. New York, USA: ACM Press, 2019: 683-694.
- [15] WANG T, ZHANG W B, XU J W, et al. Workflow-aware automatic fault diagnosis for microservice-based applications with statistics[J]. *IEEE Transactions on Network and Service Management*, 2020, 17(4): 2350-2363.
- [16] BOGATINOVSKI J, NEDELKOSKI S. Multi-source anomaly detection in distributed it systems[C]//*Proceedings of International Conference on Service-Oriented Computing*. Berlin, Germany: Springer, 2020: 201-213.
- [17] ZUO Y, WU Y L, MIN G Y, et al. An intelligent anomaly detection scheme for micro-services architectures with temporal and spatial data analysis[J]. *IEEE Transactions on Cognitive Communications and Networking*, 2020, 6(2): 548-561.
- [18] MENG W, LIU Y, ZHU Y, et al. LogAnomaly: unsupervised detection of sequential and quantitative anomalies in unstructured logs[C]//*Proceedings of the 28th International Joint Conference on Artificial Intelligence*. New York, USA: ACM Press, 2019: 4739-4745.
- [19] ZHANG X, XU Y, LIN Q W, et al. Robust log-based anomaly detection on unstable log data[C]//*Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. New York, USA: ACM Press, 2019: 807-817.
- [20] YANG L, CHEN J J, WANG Z, et al. PLELog: semi-supervised log-based anomaly detection via probabilistic label estimation[C]//*Proceedings of the 43rd International Conference on Software Engineering*. Washington D. C., USA: IEEE Press, 2021: 230-231.
- [21] REIS D C, GOLGHER P B, SILVA A S, et al. Automatic Web news extraction using tree edit distance[C]//*Proceedings of the 13th International Conference on World Wide Web*. Washington D. C., USA: IEEE Press, 2004: 502-511.
- [22] CHEN H Y, CHEN P F, YU G B. A framework of virtual war room and matrix sketch-based streaming anomaly detection for microservice systems[J]. *IEEE Access*, 2020, 8: 43413-43426.
- [23] NEDELKOSKI S, CARDOSO J, KAO O. Anomaly detection and classification using distributed tracing and deep learning[C]//*Proceedings of the 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. Washington D. C., USA: IEEE Press, 2019: 241-250.
- [24] BOGATINOVSKI J, NEDELKOSKI S, CARDOSO J, et al. Self-supervised anomaly detection from distributed traces[C]//*Proceedings of IEEE/ACM 13th International Conference on Utility and Cloud Computing*. Washington D. C., USA: IEEE Press, 2020: 342-347.
- [25] NEDELKOSKI S, CARDOSO J, KAO O. Anomaly detection from system tracing data using multimodal deep learning[C]//*Proceedings of the 12th International Conference on Cloud Computing*. Washington D. C., USA: IEEE Press, 2019: 179-186.
- [26] 喻靖民,向凌云,曾道建. 基于Word2Vec的自然语言隐写分析方法[J]. *计算机工程*, 2019, 45(3): 309-314.  
YU J M, XIANG L Y, ZENG D J. Natural language steganalysis method based on Word2Vec[J]. *Computer Engineering*, 2019, 45(3): 309-314. (in Chinese)
- [27] TIELEMAN T, HINTON G. Lecture 6. 5-rmsprop: divide the gradient by a running average of its recent magnitude[J]. *Neural Networks for Machine Learning*, 2012, 4(2): 26-31.