微服务故障检测研究综述

王 璐 姜宇轩 李青山 霍其恩 王 展 谢生龙 歹 杰
(西安电子科技大学计算机科学与技术学院 西安 710071)

摘 要 微服务架构的独立部署、去中心化等特点契合当前软件系统对适应变更、弹性部署、高扩展性等需求，使得该类架构得到了广泛应用. 但这些特点同时也增加了软件配置、监控的难度，导致软件故障具有不确定性和不可控性，增加了对微服务软件开展故障检测的复杂性. 然而，鉴于微服务故障检测对提升软件性能和可靠性的重要作用，目前也涌现出一系列挑战问题复杂性的研究工作，但目前缺乏对这些研究工作的系统性地梳理和总结，导致该领域的基础概念不清晰、研究方向界限易混淆和重叠，影响了该领域的研究进展. 因此，本文对 2009 年至今共 105 篇研究工作开展了文献综述，不仅分析了关于微服务故障检测的相关工作，并讨论了单体架构、分布式架构等其他软件架构采用的故障检测方法应用于微服务架构的可能性. 本文旨在帮助相关学者和从业者梳理该领域的最新研究进展，完善故障检测技术体系，以切实推进微服务架构的应用和发展. 本文发现软件监控、故障诊断、故障预测、故障复现和测试是微服务故障检测领域的四个主流研究方向，因此本文进一步剖析了每个方向的研究现状与难点问题. 并且，本文给出了一个建议的微服务故障检测基础框架，阐述了四个研究方向在框架内的业务范围和内在联系，并讨论了各方向的发展趋势，为该领域的后续发展提供未来方向.

关键词 微服务；故障检测；软件监控；故障诊断；故障预测；故障测试
中图法分类号 TP311 **DOI 号** 10.11897/SP.J.1016.2023.02342

A Review of Microservice Fault Detection

WANG Lu JIANG Yu-Xuan LI Qing-Shan HUO Qi-En WANG Zhan XIE Sheng-Long DAI Jie
(*School of Computer Science and Technology, XiDian University, Xi'an* 710071)

Abstract Because the characteristics of independent deployment and decentralization, the microservice architecture meets the requirements of current software systems for adaptation to changes, flexible deployment, and high scalability, which make this kind of architecture widely used. However, these characteristics also increase the difficulty of software configuration and monitoring, leading to uncertain and uncontrollable software faults. It increases the complexity of fault detection of microservice software systems. However, in view of the important role of microservice fault detection in improving software performance and reliability, a series of research works challenging the complexity of problems have emerged. However, there is a lack of a systematic summary of these research works. It leads to unclear basic concepts, and confusing and overlapping research directions, which affects the research progress in this field. Therefore, this paper provides a literature review of 105 research works from 2009 to the present. It not only analyzes the related work on fault detection of microservice, but also discusses the possibility of applying fault detection meth-

收稿日期：2022-02-18；在线发布日期：2023-01-17. 本课题得到国家自然科学基金青年科学基金项目（No. 61902288）、国家自然科学基金面上项目（No. 61972300）、陕西省自然科学基金青年项目（No. 2020JQ-300）资助. **王 璐**，博士，副教授，中国计算机学会(CCF)会员，主要研究领域为AIOps智能软件运维、微服务架构与容器. E-mail: wanglu@xidian.edu.cn. **姜宇轩**，硕士研究生，中国计算机学会（CCF）会员，主要研究领域为微服务与故障检测. **李青山**（通信作者），博士，教授，中国计算机学会（CCF）高级会员，主要研究领域为面向智能体的软件工程、特定领域软件体系结构. E-mail: qshli@mail.xidian.edu.cn. **霍其恩**，硕士研究生，中国计算机学会（CCF）会员，主要研究领域为微服务与容器. **王 展**，硕士研究生，中国计算机学会（CCF）会员，主要研究领域为微服务与负载均衡. **谢生龙**，博士研究生，中国计算机学会（CCF）会员，主要研究领域为软件自优化、群体智能和多代理技术. **歹 杰**，博士研究生，主要研究领域为推荐系统、图神经网络、机器学习等.

ods adopted by other software architectures such as single architecture and distributed architecture to microservice architecture. This paper aims to help relevant scholars and practitioners sort out the latest research progress in this field, improve the fault detection system, and effectively promote the application and development of microservice architecture. This paper finds that software monitoring, fault diagnosis, fault prediction, fault recurrence and testing are the four mainstream research directions of the fault detection of microservice. Therefore, this paper further analyzes the current research status and difficult issues in each direction. In addition, this paper gives a suggested basic framework for microservice fault detection. In addition, this paper expounds on the business scope and internal connections of the four research directions within the framework, and discusses the development trends of each direction to provide future directions for the follow-up development of this field.

Keywords　microservice; fault detection; software monitor; fault diagnosis; fault prediction; fault testing

1　引　言

近年来，软件系统对快速响应需求变更、弹性部署、高可扩展性等特性的要求不断增加，促使微服务架构（Microservice Architecture，MSA）得到了广泛应用. 作为一种新型软件架构，微服务架构建议将单体架构应用拆分成一组独立运行的微服务模块，每个模块负责单一的业务功能并可独立部署，而微服务间则采用轻量化通信协议进行交互. 因其独特的设计，基于微服务架构的软件系统具备服务组件化、智能端点与管道扁平化、去中心化等特点，高度契合了当代软件系统的新需求，吸引了越来越多的企业将原有的单体架构软件迁移成为微服务软件，以节省软件运行与维护成本. 在 2022 年 1 月 11 日 IDC（互联网数据中心）发布了 2022 年中国未来网络十大预测①，其中有 3 项预测涉及到了云网络，预测中提到云网络软件、云管理以及商业规模在未来 3 年内将会有较快的增长，微服务作为云网络的基础思想之一也必然会迎来一个高速发展的阶段. 然而，随着微服务架构逐步成为互联网应用所采用的主流架构之一，针对微服务软故障检测工作的复杂性和必要性日益凸显.

微服务架构的相关特性在给软件系统带来收益的同时，也不可避免地为故障检测增加了诸多复杂性. 通常软件工程领域将导致软件系统无法正常运行或严重影响用户业务功能实现的特殊事件定义为软件故障. 故障检测则是指可及时预测、发现并分析定位软件故障起因的一系列处理措施. 首先，微服务软件的每个服务模块被独立开发、部署，从而产生了大量的部署、配置和监控需求. 然而，目前软件基础设施中的传统监控方法容易忽略潜在的环境部署问题，无法完全适用于微服务软件. 其次，微服务间存在较复杂的依赖，容易造成级联故障，从而导致局部系统瘫痪，这就要求软件故障检测方法能够尽早识别并避免级联故障. 并且，每个微服务都将进行多实例部署，并且服务间将通过网络进行异步调用，这为需要在多个物理和逻辑组件间进行协作的系统带来了潜在的交互故障等多种问题. 基于断点和程序切片的传统故障定位方法并无法对高度并发和动态的微服务系统实现准确故障定位. 此外，由于微服务软件系统的去中心化特点，每个微服务模块的结构、框架甚至编程语言均不尽相同，同时一般微服务运行时状态具有高复杂性和不确定性. 这些特殊性质导致了微服务软件故障类型具有高度的不确定性和不可控性，很难对故障环境进行复现和测试.

正因存在上述复杂性，目前业界迫切需要研究出适合微服务软件的故障检测技术，以有效提高软件性能和可靠性，并节省时间、人力开销，创造更多的产品经济效益. 例如，在软件监控方面，针对微服务架构带来的高部署、复杂依赖和服务调用的问题，目前亟需一种对微服务软件实行全面监控的方法，以便于开发运维人员实时获取系统状态，并能够对扩展服务做出有效的修正，及时告警软件故障. 在故障诊断方面，针对微服务软件的规模庞大，结构复杂特点，目前亟需对微服务故障诊断方法实现有效改进，以避免开发人员根据项目经验人工排查故障的低效和误差问题，达到减少时间和人力成本损失、提高软件运维效率的目标. 在故障复现和

①IDC. https://www.idc.com/getdoc.jsp?containerId=prCHC48766822 2022,1,11

测试方面，则需明确现有的微服务软件故障类型和根源问题，并能够在测试环境下复现，有利于保障微服务软件的可持续性和可用性.

因此，如何有效实现面向微服务软件的故障检测，是保障微服务软件性能与可靠性、提高产品经济和社会效益亟待解决的问题之一. 近年来，越来越多的研究学者开始探讨如何改进和完善该领域的相关方法[1]. 本文采用系统性文献综述的方法，从IEEE Xplore, the ACM Digital Library, Springer-Link等数据库中对 2009 年至今与微服务架构故障检测相关的研究工作、以及应用于其他软件架构的传统故障检测方法共 105 篇论文进行数据抽取和质量评价，对整体进行统计分析，并从研究方向和发展趋势两个方面进行了较为全面的分析，旨在帮助研究者和从业者了解目前的研究进展，如图 1 所示.

就研究方向而言，目前大部分研究工作基本围绕软件监控、故障诊断、故障预测、故障复现和测试四个方向开展.

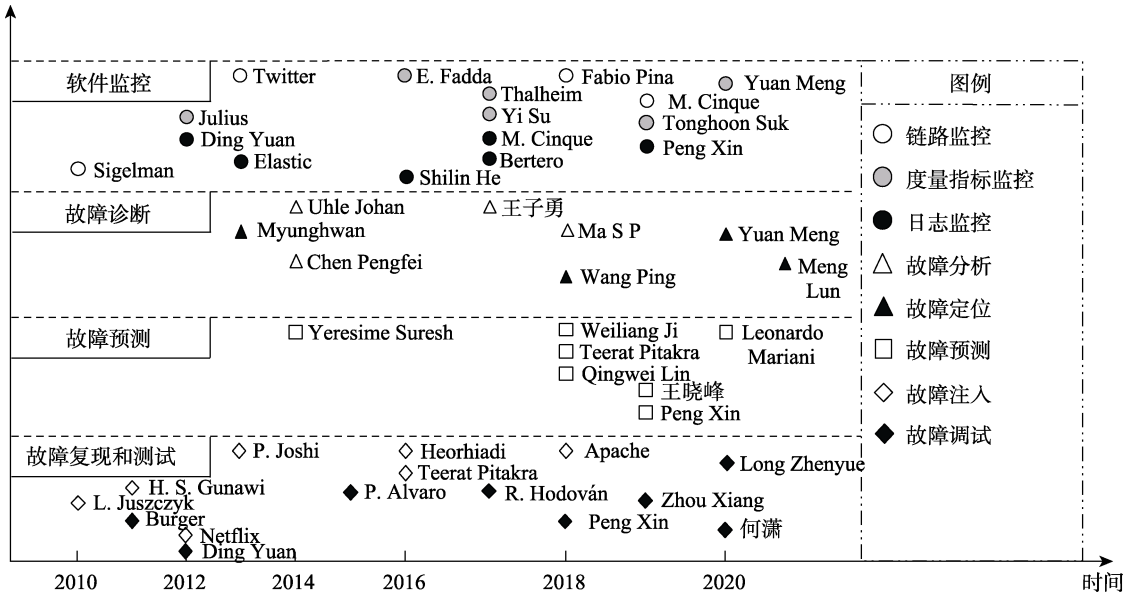

图 1 微服务故障检测算法时间轴图

（1）在软件监控方向，由于每个微服务允许独立动态部署，并且存在独立的数据库，导致微服务软件的监控数据具有一定的复杂性和资源异构性，需要对异构监控数据进行格式化和数据融合处理[3]. 并且，由于微服务软件的海量服务交互和复杂依赖，导致系统运行状态频繁变化，从而产生大量的监控数据信息. 因此，目前软件监控方法的相关研究主要关注如何保证对软件运行日志、服务调用信息和度量指标数据的完整采集.

（2）在故障诊断方向，由于每个微服务均可进行多实例动态部署，产生了大量不同状态的服务实例，从而很难利用物理节点准确诊断服务级故障位置. 并且，服务间基于轻量级通信机制进行异步交互，交互时序的不确定性给微服务故障的定位造成了极大的阻碍. 而传统故障诊断方法主要局限于业务层和功能层，利用人工经验进行微服务故障位置和起因诊断. 因此，针对微服务的故障诊断方法主要关注如何自动地诊断故障并细粒度定位故障位置和起因，从而减少成本开销，提高故障诊断的准确性和效率.

（3）在故障预测方向，由于微服务的复杂依赖，一个故障的发生通常伴随着一系列的级联故障，极大地影响系统运行. 因此，为了提高微服务软件的可靠性和可用性，目前研究主要考虑如何在故障发生前，基于对相关监控指标数据进行有效聚类，分析系统运行状态趋势，结合深度学习算法，判断潜在故障的发现概率，并考虑如何对故障事件有效预防，将对系统运行和功能正常执行的影响降至最低.

（4）在故障有效复现和测试方向，由于微服务软件故障发生的环境具有不确定性，会产生一系列与传统故障不同的软件故障类型. 因此，目前研究关注如何明确需要复现微服务的故障类型、故障表现形式和起因，在此基础上，结合微服务的实现机制，模拟故障发生环境，并分别在代码层、网络层和应用层进行故障注入，实现故障的有效复现和测试，提高目标系统的可靠性.

就研究趋势而言，在 2016 年之前，微服务架构还并未受到广泛关注，大部分的应用无法摆脱传统

单体软件的束缚，只能在原有传统软件的基础上对应用代码进行重构，无法将庞大、复杂的代码彻底修正为微服务架构. 因此，当时的研究工作依旧基于传统软件故障检测的思路，对方法进行改进和应用，存在一定的局限性，故障检测方法的效果也无法得到保障. 在 2016 年至 2018 年，随着微服务架构的不断发展，越来越多的企业开始应用微服务架构，实现从单体架构到微服务架构的迁移工作，微服务系统的特征更为鲜明，导致了传统的故障检测方法已无法适用. 因此，相关学者开始考虑如何设计符合微服务架构特点的故障检测算法，并结合机器学习、图论等领域相关算法更进一步提高微服务软件故障检测的有效性和准确性. 从 2018 年至今，随着微服务架构的不断完善和广泛应用，产业界对于微服务系统的故障检测提出了更高的要求，更加重视对于故障检测方法的透明性、低耦合以及效率问题. 因此，针对特定微服务故障类型以及实现无侵入式监控的软件故障检测方法受到广泛关注.

鉴于微服务故障检测问题的复杂性和必要性，目前已经存在大量研究工作. 本文从三个维度将本文综述内容与当前的总结性工作展开对比分析.

首先，在研究内容维度，现有的综述研究主要针对传统单体架构或分布式架构软件在软件监控[3]、故障诊断[4]、故障预测[5,6]、故障定位[7,8]、故障注入[9]等故障检测中部分研究方向进行了总结，缺乏对微服务故障检测相关工作的系统性梳理和较为全面的总结. 在以往的研究中，本团队提出了针对微服务运维治理的相关综述[10]，包括负载均衡、故障诊断和弹性伸缩等内容. 然而，目前还并未有专注于故障检测整体框架的总结性工作，这为后续微服务故障检测的研究造成了极大的阻碍. 因此，本文旨在深入剖析面向微服务系统的故障检测阶段，并分析当前的研究进展，为后续研究提供研究思路.

其次，在微服务架构维度，关于微服务软件系统的大部分总结性工作主要关注从单体架构迁移至微服务架构的方法[11]、微服务粒度划分的合理性[12-14]、微服务软件开发方法[15]、对微服务架构的优化[16,17]、以及对微服务架构相关概念和发展趋势的探讨[18-20]等内容，此外还包括对不同微服务系统的实证研究[21]. 上述总结性工作在微服务架构发展角度已相当完备，然而，当前依旧缺乏在剖析微服务架构，解决软件运行维护问题，以及对微服务架构内部的具体故障处理流程的综述. 因此，本文重点关注微服务系统的运行维护，提出了基础了故障检测框架，进一步挖掘微服务架构特点，旨在补充当前微服务生态，为后续研究提供基础.

最后，在软件可靠性维度，当前的研究主要关注于微服务安全问题[22]，以及对微服务风险识别能力的分析[23]等. 可以看出，当前针对微服务软件可靠性的总结性工作相对较少，已有的综述主要针对传统软件的可靠性模型研究[24,25]，缺乏针对如何保证微服务可靠性的探讨，其中包括软监控、故障复现和测试等相关内容的. 同时，Dragoni 等人强调了目前普遍缺少与微服务安全领域相关的研究，特别是微服务故障检测[26]，从而导致该领域的基础概念不清晰、研究方向界限易混淆和重叠，影响了领域的研究进展. 因此，本文结合了传统软件的可靠性保障方法，分析该类方法对于微服务的有效性探讨，同时，对当前较权威的用于软件测试的方法进行介绍，为后续微服务可靠性研究提供研究基础.

因此，本文依据微服务故障检测所面临的复杂性和必要性，在参阅 2009 年至今 105 篇研究工作的基础上，尝试对微服务故障检测的研究情况开展全面性的综述总结，并深入剖析了该领域中重要研究方向所面临的问题和挑战. 并且，本文通过统计分析目前的研究成果和现实场景的需求，详细讨论了如何形成一个完善的微服务故障检测框架，以及未来可能的研究方向和发展趋势，为研究者和从业者提供参考，切实推动该领域的发展.

论文在第 2 章概述了故障检测的相关研究方向；第 3 章设计了综述的研究方案，并严格依照该方案对目前的研究进展进行总结分析；第 4 章提出了微服务故障检测的基本框架，并对未来的研究趋势开展了探讨和展望；第 5 章总结了全文的工作.

2　概　述

目前，随着微服务架构的广泛应用，也产生了一系列软件故障问题，越来越多的专家学者开始重视对微服务故障检测相关内容的研究. 本节根据相关研究工作，结合上述生产环境的需要，对微服务和故障检测的相关概念进行阐述，并在此基础上，结合已有的故障检测技术，描述目前微服务故障检测的实际应用场景.

2.1　基础概念

2.1.1　微服务概念

本文根据 2014 年美国软件开发者“教父”Martin Fowler 对微服务架构的定义①以及在此之后的相关

① Fowler M, Lewis J. Microservices. http://martinfowler.com/ articles/microservices.html 2014,3,25

研究，综合分析了微服务架构的基本概念. 微服务架构围绕业务领域将单体应用依据特定粒度标准拆分为多组可独立部署和运行的微服务. 结合去中心化治理思想，每个微服务运行在单一的进程中，可以考虑使用最佳工具实现单一的业务功能[27,28]. 不同微服务通过网络调用，利用轻量级机制相互通信，实现智能端点与管道扁平化，从而确保微服务模块的松耦合和服务的组件化.

为了更直观地展现微服务架构特点，如表 1 所示，本节对比分析了单体架构、分布式架构和微服务架构在部署、数据存储、服务调用等方面的差异[29,30]. 相较于传统架构，微服务架构具备持续快速交付、高可扩展性和高自治性[31]三个明显优势.

（1）持续交付指基于微服务架构的软件系统通常使用轻量级容器技术，独立部署微服务，并遵循 DevOps 方法，支持自动化的软件集成和持续交付. 其中，Docker 作为典型的容器技术，是一种基于 Linux 内核的虚拟化技术，在操作系统层面实现对进程的封装隔离，同时可以使用更少的资源来扩展服务实例，为微服务提供了理想的载体.

（2）高可扩展性指系统允许在运行态根据外部环境变化和内部服务执行情况进行动态伸缩扩展，以响应对网络、处理数据、数据库访问或日益增长的文件系统资源需求. 微服务软件系统中的可扩展性分为水平扩展性和垂直扩展性. 其中，水平扩展性指通过添加与现有节点功能相同的新节点，并在所有节点中重新分配负载，以此实现横向向外扩展；垂直扩展性是通过向节点添加处理、主内存、存储设备或网络接口来扩展以满足每个系统的更多请求时，从而实现垂直向上扩展.

（3）高自治性则指微服务架构将职责单一原则应用于服务设计，根据业务划分服务的上下界限，每个微服务独立治理单一的业务功能，拥有独立的数据存储方式，甚至可以选择最合适的工具进行开发管理，包括编程语言、部署方法、集成技术、架构模型等，实现了对技术替代性的优化，在技术选型和运维治理上具有高度自治性.

基于上述三个优势，微服务架构可解决应用部署和服务扩展困难的问题，并降低开发成本. 然而，微服务架构在带来软件开发、运行层面优势的同时，其具备的很多典型软件特征也会很大程度上影响软件运维的有效性，以下列举了影响软件运维中故障检测的三项基本特征：

（1）服务组件化：微服务强调对整个系统实现解耦，将单体应用按照一定的业务或技术维度拆分成独立的组件化服务，每个服务都独立开发、部署和管理. 该模式可有效避免一个服务的修改导致整个系统的重新部署，但同时也给服务监控和数据存储造成了很大的压力.

（2）智能端点与管道扁平化：微服务主张将与服务组件间通讯相关的业务逻辑放在组件端口而非通讯组件中，强调通讯机制应尽量简单且松耦合. 例如，基于 HTTP 协议的 RESTful 架构是微服务软件系统最常用的轻量级异步通讯机制. 然而，该通讯机制也会造成复杂的交互故障，从而影响故障检测的有效性.

（3）去中心化思想：微服务架构依据去中心化设计理念，对每个服务组件均可选择最佳工具（例如不同的编程语言和开发框架）进行独立开发；且每个微服务都存在一个独立的数据库，实现数据管理的去中心化，让数据存储和性能达到最优. 但是不同的开发方式及数据库实例，导致数据一致性也成为微服务架构中亟待解决的问题之一.

表 1 微服务架构与传统软件架构对比表

类别	单体架构	分布式架构	微服务架构
部署	应用整体部署	系统部署 重视机器隔离	独立部署 重视进程隔离
数据存储	单独数据库	每个机器 一个数据库	每个微服务 一个数据库
耦合程度	高耦合	耦合程度较高	耦合程度低 去中心化
服务调用	本地内部调用	不重视轻量级通信	重视基于 http 的轻量级通信
可扩展性	扩展性差	扩展性较强	扩展性强
测试	简单 不确定性低	较简单 不确定性高	复杂困难 不确定性高

2.1.2 微服务软件故障

软件故障是指软件不能按照预定义的条件执行特定功能的事件[32]. 引发软件故障的原因有很多，比如软件设计缺陷、代码问题、配置错误等. 软件故障的发生必然会导致系统无法正常运行或严重影响用户业务功能的实现，具体可表现为请求失败、响应延时等[33]. 传统软件故障主要集中在代码逻辑、基础设施部署和数据存储问题上. 然而，微服务架构作为一种新型的软件架构，独立部署和轻量级服务交互模式在提供快速部署、持续交付等优势的同时，也带来了新的故障问题，这些故障问题将会极大地影响软件服务的运行. 因此，为了保障对微服务软件的运维效率，提高软件运行可靠性和测试的准确性，需要对微服务软件可能出现的一些基

本故障类型有所了解.

本文根据目前相关研究工作及产业界现状，依据故障表现形式和故障根源划分了微服务故障. 如表 2 所示，将当前的微服务故障按照故障表现形式分为功能失效故障、性能遗漏故障[34]. 其中，功能失效故障指的是严重影响软件程序运行，导致服务无法执行的故障，例如机器宕机、网络信号中断等；性能遗漏故障是指会影响服务质量和性能，进而阻碍业务功能实现的一类故障，例如响应时间过长、连接超时等. 本文将微服务故障按照故障根源分为内部逻辑故障、服务交互故障[35,36]、部署环境故障. 其中，内部逻辑故障是由于不合理的软件设计和代码缺陷引发的故障，例如，变量冲突、版本不匹配等；服务交互故障是指服务间交互失效或相互影响导致的故障，例如，异步交互故障、资源冲突等；部署环境故障包括物理节点状态和基础设施部署配置故障，例如，节点宕机、网络中断[37]等. 从中可以看出，微服务系统和单体系统、分布式系统都存在内部故障和部署环境故障，而微服务系统由于其典型的服务组件化和去中心化的特点，导致其内部结构依赖关系复杂，基于轻量级的服务通信机制也容易受到网络影响，因此，服务交互故障成为了典型的微服务故障.

表 2　微服务故障划分表

类别	功能失效故障	性能遗漏故障
内部逻辑	版本冲突、变量冲突返回类型错误	代码逻辑繁琐复杂 时间复杂度过高
服务交互	资源被占用 第三方调用失败	响应超时 连接超时
部署环境	节点宕机 网络信号中断	网络信号差 CSS/SSL/Docker 配置错误

此外，相关研究还提出了关联依赖故障，作为故障根因定位的理论基础. 该故障类型是微服务故障的典型特征，其故障根因主要是由于有关联依赖关系的服务发生了上述三类软件故障，从而产生的级联反应.

2.1.3　微服务故障检测

由于微服务运行交互环境的复杂性和动态性，导致软件系统故障复杂多变，具备不确定性[38]，从而对微服务软件的故障检测有了更高的要求. 因此，完善微服务故障检测体系和技术，对提高微服务软件的可用性和可靠性具有极大的促进作用. 目前针对微服务领域的故障检测方法的相关研究主要集中在软件监控、故障诊断、故障预测、故障复现和测试四个方向.

软件监控需从基础设施、系统、应用、业务和用户端五个层面采集软件系统的运行状态信息. 然而，随着微服务的广泛应用，微服务去中心化多实例部署的特点驱动着软件监控从分层的视角转换成了以服务为中心的视角，包括指标度量监控、链路监控和日志监控. 微服务软件监控以采集、存储微服务基础设施平台运行过程中单个服务实例和服务节点状态为基础，将多服务监控信息图表聚合展示和分析，提供了微服务基础设施平台运行过程中的服务实例及运行节点状态采集功能，从而可视化微服务系统的整体状态，进一步实现对软件故障的及时告警[39].

故障诊断可基于监控数据，通过分析故障发生的层级关系，识别故障类型，计算故障影响因素与故障之间的相关性，实现对故障发生位置的精准定位和对故障发生根源起因的准确诊断[40-42]. 由于微服务软件的复杂依赖，微服务故障诊断需要挖掘不同影响因素与服务实例间的关联关系，建立关联关系模型，当故障发生时，判断当前系统状态与故障位置、起因之间的联系，从而实现对故障的细粒度诊断，进一步提高软件系统的可靠性和可维护性，保障业务功能的正常执行.

故障预测作为故障诊断过程的补充，在故障发生之前，其分析软件系统运行状态的演变趋势，预测是否对软件业务的正常运行存在故障威胁，并以此作为后续故障处理，降低故障影响范围和程度的主要依据. 微服务故障预测则需要考虑更多因素，例如实例状态、执行轨迹异同、节点状态等，通常结合数据挖掘、机器学习算法对故障发生的概率进行计算[43]，从而提高微服务故障预测的准确性，提高软件的可靠性和可用性.

故障复现和测试指通过故障注入等手段复现软件系统发生故障时的环境，监控此时软件系统的运行状态和故障表征，从而分析软件系统的故障处理能力，该过程也用于评估软件系统运行质量和安全性. 而在微服务系统中，微服务环境的多样化和复杂性，导致了故障发生的条件很难确定，完全依赖人工经验的故障注入和代码调试无法适用于微服务软件系统[36,38]. 因此，面向微服务架构的故障复现和测试需要将不同类型的故障自动化注入到服务的不同部分，并记录产生的一系列错误版本. 其中，故障注入的方法需要尽可能贴合实际的生产环境，从而提高微服务故障复现和测试方法的可靠性和可用性.

2.2 实际应用

微服务架构作为一种广泛应用于产业界的新型软件架构，其与实际应用场景紧密结合，本节通过分析产业界动态，更加深刻地综述微服务软件系统的故障检测特征. 目前，在实际生产环境中相关微服务故障检测工具已得到了广泛的应用. 而由于产业界大规模资源部署环境和对产品经济效益的追求，细粒度地故障诊断和预测并不适用于产业界大规模软件的部署运行和基于软件价值快速创造经济效益的要求. 因此，产业界目前的故障检测主要集中在两个方面：数据监控和不确定故障注入测试.

其中，海量数据监控主要用于可视化软件运行状态、做数据分析支撑和测试记录等. 微服务架构一般适用于规模庞大、结构复杂的应用软件开发，由于微服务本身独立动态部署、去中心化的设计思想，每秒都有海量异构监控数据的生成，给数据的存储和分析带来了巨大的压力；同时，微服务软件基于弹性伸缩方法，从服务扩容或迁移结束到流量接入的耗时降低和实例的频繁变化，大规模复杂软件系统的监控面临可用性的风险. 因此，为保证监控效率，在实际的生产环境下，势必需要对不同的监控内容进行资源倾斜. 例如，阿里巴巴的鹰眼系统重点关注于服务层面链路调用的监控数据采集；已得到广泛应用的 Prometheus 系统专注于对系统运行状态及资源使用情况的监控，主要包括对系统延迟、流量、服务错误和资源饱和度四个黄金指标的数据采集，其中，延迟是发送请求和接收响应所需的时间，流量是衡量流经网络的请求数量，服务错误用于衡量当前系统发生错误请求的速率，资源饱和度则定义了网络和服务器资源的负载，包括 CPU 利用率、磁盘容量等；以及目前主流的日志监控系统 ELK，是基于 Elasticsearch，Logstash，Kibana 三个开源软件，构建的一个完整的支持日志搜集、传输、存储、分析和告警功能的集中式日志监控系统.

由于独立开发导致技术上的分离，轻量级通信加上消息队列机制增加了故障问题的复杂性，服务间的复杂依赖也带来了很多的不确定性. 因此，在软件运维层面，针对微服务软件的不确定故障注入测试提出了更高的要求. 在当前生产环境下，有一些符合微服务特点的故障注入工具. 例如，ChaosBlade 是阿里巴巴开源的一款提供丰富故障场景实现，辅助软件系统提升容错性和可恢复性的混沌工程工具，可实现底层故障注入；ChaosMesh 是 PingCap 团队研发的一款自动化测试 kubernetes 环境的工具，已被应用于网易伏羲测试平台；Chaos-toolkit 是一个可集成在 IaaS 或 PaaS 平台的实验测试框架，可使用多个故障注入工具定制实验场景，与多个监控平台合作监测和记录指标信息. 上述工具已被广泛应用于微服务系统测试.

3 研究现状

目前，针对如何提高微服务软件的可靠性和可用性，保障软件的可持续运行的问题受到广泛关注. 而针对微服务故障检测的相关研究也层出不穷. 然而，微服务架构作为一种新型的软件架构，其故障检测技术的研究仍处于发展阶段，存在一些问题和挑战亟需解决. 因此，本文首先在系统的文献综述方法的基础上[44]，设计了合理的研究方案，根据文章的质量[45]、相关度等对文章进行筛选和评估；然后对微服务故障检测各个方向的研究现状和突破难点进行详细的综述.

3.1 综述研究方案

根据权威合理的研究方案设计方法，本节将具体的研究方案分为三个阶段. 第一阶段确定本文的综述方向，即微服务的故障检测；第二阶段确定本文的研究问题和搜索策略，根据定义的问题来确定搜索关键字，并在此基础上对搜集到的文献按照文献选择评估方法进行筛选；第三节点是抽取文献的关键信息进行分析总结，最后进行内容撰写.

研究问题的定义对任何系统性综述而言均最为关键，其主要职责在于将主要问题分解为具体问题. 定义出的问题会影响综述研究的偏向、关注点，需要结合目前研究进展、局限性和未来发展方向进行确定. 本文主要考虑三类问题. 第一类是对目前的研究工作的具体内容和研究的难点问题进行探索，分析目前的解决途径和关键技术，主要围绕架构特点、解决效率和局限性；第二类是如何通过对现有的研究工作进行分类，并考虑不同类别的主要职责以及类别间的关联关系，构建基础的理论体系框架，为进一步研究提供理论指导；第三类则是根据目前的研究现状和基本理论框架，对未来可能存在的挑战和发展趋势进行讨论，为后续研究指明了有价值的方向.

在确定研究问题的基础上，本文关注了从 2009 年至今与微服务架构故障检测直接相关的研究工作、以及应用于其他软件架构的传统故障检测方法，主要来源于中国知网、IEEE Xplore，ACM Digital Library，Springer Link，ELSEVIER，Web of Sciences 等学术资源数据库，囊括了大部分的中英文论文，

并在文献搜集过程中对选择的关键字进行更新迭代，最后筛选出核心关键字如“微服务”、“微服务故障检测”、“软件运维”、“软件故障诊断”等关键字对文献进行过滤，共搜集到 300 余篇论文. 然后，为了提高所选文献的相关性和严谨性[46]，本文建立了合理的文献选择方法和文章质量评估方法对相关文献做进一步筛选.

其中，文献选择方法主要考虑文献与研究问题的相关性，并从以下 5 个角度对文献进行文献选择：

（1）文献是否能为本文研究问题提供清晰且有价值的贡献；

（2）文献之间的主题是否有较大的相似性，若有，只选择其中之一；

（3）文献是否全面地描述了总体方法？

（4）文献的整体结构是否清晰；

（5）文献是否提供了对未来方向的阐述，以及技术的进一步发展的探讨.

首先为了避免筛选后的论文内容上出现较多重复，也为了能丰富地展示微服务故障检测领域的研究内容，本文通过评估论文主题之间的相似性，减少非必要论文数量，同时为了确保筛选后论文的质量，本文对首先按照论文发表平台等级进行了排序，去重操作主要针对序列后半部分论文. 此外，针对其余 4 个角度，本文制定了相应的量化规则，用于筛选搜集的论文. 如表 3 所示.

表 3　文献筛选量化表

文章问题		得分
文献是否能为本文研究问题提供清晰且有价值的贡献	论文研究领域	是 1，部分 0.4，否 0
	关键字是否契合本文	是 1，部分 0.6，否 0
	论文发表平台	Q1 或 A* (+2) / Q2 或 A (+1.5) / Q3 或 B(+1) / Q4 or C (+0.5) / Unranked (+0)
文献是否全面地描述了总体方法		是 1，部分 0.5，否 0
文献的整体结构是否清晰		是 1，部分 0.5，否 0.2
文献是否提供了对未来方向的阐述，以及技术的进一步发展的探讨		是 1，否 0.7

而文章质量评估方法是在 Roehrs 等人提出的文章质量标准[45]的基础上，结合 Kitchenham 与 Brereton 提出的通用检查表的质量工具[47]，制定了针对计算机软件技术领域论文的质量评估方法，如表 4 所示基于六个问题进行文献质量评估.

在文献质量评估过程中，最高评级为 7 分，最低评级为 0 分. 本文通过三到四位作者对文献质量进行评估，并将六个问题的单项质量分数累加，得到每篇论文的最终质量分数. 当评估结果不同时，可以通过讨论达成共识，或取平均质量分数. 最后，本文保留了大于等于最高得分一半分数的文献.

表 4　文献质量评估表

文章问题		得分
是否清楚地表明了研究的目的？		是 1，部分 0.5，否 0
充分分析了现状问题和研究背景？		是 1，部分 0.5，否 0
提出的方法是否为微服务领域特有？		是 1，部分 0.5，否 0
这篇文章的研究成果与目前其他研究成果相比是否拥有显著的贡献？	是否结合其他领域方法	是 0.5，否 1
	论文发表平台	同问题 6
	是否为原创而非改良已有方法	是 1，否 0.5
实验案例和实验数据是否权威？实验证明是否合理？		是 1，部分 0.5，否 0
是否在知名期刊或会议上发表过？（参考期刊采用 JCR 指数①，会议采用核心排名②）		Q1 或 A* (+2) / Q2 或 A (+1.5) / Q3 或 B(+1) / Q4 or C (+0.5) / Unranked (+0)[48]

本文基于上述文献选择方法和文章质量评估方法，通过对会议或期刊质量、突破难点问题和主要贡献对文章质量进行选择和评估，最后遴选出 105 篇参考文献. 此外，为了进一步从相关文献中提取关键信息，从而辅助分析目前的研究进展. 本文首先按照文献主要研究内容对文献进行分类处理，主要研究内容的提取主要根据文献的关键字及文献的研究动机进行甄别，然后对研究内容进行排序及相似内容合并操作，确定研究主题的提取. 然后针对分类后的文章，设计了文献信息提取方法. 本文通过设置数据提取表格来提取研究文章信息，主要包括两部分. 一部分要求所提取的元素能够准确表现文章的基本信息，包括论文名称、第一作者、论文年份以及论文页数；另一部分是要求所提取的元素能够代表文章的研究内容，包括研究目标、研究背景、相关方法和技术和应用领域.

最后，根据上述提取的关键信息，本文重点围绕微服务故障检测的四大研究方向，即软件监控、故障诊断、故障预测、故障的复现和测试，对筛选的文献进行精读，并对相关方向的研究现状进行详细分析、讨论和阐述.

3.2　软件监控

目前在大规模软件系统的构造中，例如，Google 的数据中心服务[49]、阿里巴巴的云中心服务，都将软件监控作为系统重要的组成部分[50]. 传统的软件

① JCR, Journal citations report, https://bit.ly/2uNFfBb 2018.

② CORE, Computing research & education, https://bit.ly/2wke TXZ. 2018.

监控技术主要针对软件系统状态相关的度量指标和日志信息采集，将监控信息图表聚合展示和分析，可视化系统运行状态的演变趋势. 相关专家学者更多是对日志信息的挖掘、度量指标选择、处理以及系统状态可视化设计的合理性进行研究.

然而，在微服务系统中，由于服务间通过轻量级通信机制实现异步调用，易受网络状态等因素影响，从而产生了微服务软件特有的服务交互故障. 因此，为了更全面、更有效地实现软件故障检测，面向微服务的软件监控应将服务交互监控作为重要的研究工作之一. 同时，由于微服务弹性环境的动态特性，每个微服务实例随时可能通过增加或减少来适应工作负载变化，还可以通过重启、迁移到不同节点等方式提高系统资源使用效率①. 这种动态性使得微服务交互监控变得困难，传统的埋点、调试手段也不再适用. 此外，由于微服务故障可能不会在服务层出现，例如某些性能方面的故障，为了给后续故障检测提供数据诊断依据，需要对微服务软件的整体状态信息和所有微服务组件的状态信息进行全面监控[51]，给软件监控造成了巨大的压力. 因此，如何针对微服务软件特点，设计出有效、全面、便捷的微服务软件监控工具[52]，是学术界和产业界亟需解决的问题.

表 5 微服务软件监控划分表

研究内容	基本概念	相关工具	引用
链路监控	通过对调用链路信息的跟踪采集分析快速发现故障	Dapper Zipkin Skywalking	[53-58]
度量指标监控	从事件发生时间和当前值的角度记录监控信息	Zabbix Prometheus	[37,40,55, 59-63]
日志监控	对软件运行状态下的日志的实时采集和持久化存储	ELK	[50,64-70]

本节根据当前微服务软件监控的相关研究，结合产业界的监控工具，综合分析软件监控的研究情况，如表 5 所示，将微服务软件监控的主要工作内容分为三个部分：链路监控（Trace）、度量指标监控（Metric）和日志监控（Log）[53]. 其中，日志监控和度量指标监控是传统软件监控的主要研究内容，而链路监控是为了专门应对微服务软件服务交互问题而产生的监控需求，是微服务软件监控的重要组成部分[54].

3.2.1 链路监控

链路监控主要是为了监控微服务软件系统中复杂异步通信机制产生的服务交互故障，而提出的一种采集服务间请求、调用信息的监控手段. 链路监控通过对调用链路信息的跟踪采集分析，确定应用服务的关键路径并可视化，从而快速发现故障问题[18]. 随着微服务架构的广泛应用，针对微服务软件的链路跟踪也越来越受到重视. 当前针对软件链路监控的研究工作主要分为主动跟踪和被动跟踪两种[53]. 其中，（1）主动跟踪是指通过对目标系统进行一定程度上的干预，从而强化链路信息特征，保证链路监控的全面性和准确性，但该方法必然会对目标系统的运行造成一定影响，且对系统环境要求较高. （2）被动跟踪是指在对目标系统无侵入的前提下，通过网络数据包传输拦截、网关请求拦截等方式被动采集链路信息，不需要对目标软件进行模式化，能够适应系统环境的变化.

由谷歌公司提出的 Google Dapper[55]是典型通过代码植入方式实现主动链路跟踪的方法. Dapper 主要通过插桩的方式对服务调用信息和相关运行状态指标信息进行采集. 该方法尽可能地将代码植入限制在一个很小的通用组件库中，实现对开发人员的应用程序透明. 当前工业界所设计的链路跟踪工具，例如阿里巴巴开发的鹰眼系统、Twitter 公司开发的 Zipkin②（Distributed Tracking System）以及具有较高活跃度的 Skywalking 等，均是基于 Google Dapper 的设计思路进行扩展和完善的，且都满足了微服务链路监控普遍部署和持续监控的需求. 然而，上述工具所产生的跟踪链路信息生成和搜集开销、存储和分析数据所占用的资源量，都势必会在一定程度上影响目标系统的运行. 同时，在软件系统升级的场景中，也需要重新修改跟踪代码，维护费用极高. 相关学者研究如何尽可能降低主动跟踪方法对目标系统的影响，如 Donghun Cha 团队提出基于服务网格的服务跟踪方法[56]等，而实际产业界的大规模、高吞吐量在线服务的微服务软件链路监控中，引入了采样的概念，即通过一定时间范围内对服务调用信息采集的采样频率的调整，以降低对目标系统的影响程度，提高链路监控的灵活性.

被动跟踪最大的特征在于对目标系统代码的无侵入性，解决了传统监控方法通过在软件架构层中检测或通过代理方式对相互依赖的调用信息进行采

① CanaryRelease, https://martinfowler.com/bliki/CanaryRelease.html 2017, 11,17

② Twitter, Zipkin, http://twitter.github.com/zipkin, 2013.

集，而导致的代码植入问题. F′abio Pina 提出通过在网关添加监控搜集功能实现被动链路跟踪方法[57]. 该方法可以基于网关搜集例如响应时间、IP、目的端口等基本软件信息，将监控工具与目标系统解耦，不影响目标系统的正常运行和自主负载伸缩.

除了通过网关进行数据采集的手段之外，还有针对微服务特有的轻量级通信机制，所设计的嗅探技术，其可以通过网络数据包拦截的方式进行被动链路跟踪. 例如，Marcello Cinque 提出将微服务日志和黑箱跟踪结合，基于具象状态传输（REST）通信模型，设计出一种通过拦截网络数据包的手段进行被动跟踪方法[58]. 该方法通过获取网络数据包，读取数据包的表头，根据 REST 通讯规范对数据请求进行判断分析，从而实现该服务链路的监控，同时保证了该方法的应用程序透明性. 该方法减少了收集的监视数据量，与传统监控方法相比，性能开销可以忽略不计.

如表 6 所示，当前微服务链路监控研究已逐步完成了从主动跟踪到被动跟踪的转变，从而适应大规模复杂软件. 其中主动跟踪方法注重对目标系统的全面部署和持续监控，监控的链路信息更加完整、准确. 而相比于主动跟踪方法，被动跟踪方法在无代码侵入、应用程序透明性和环境适应性等方面更有优势，对目标系统的影响较小. 因此，针对不同微服务软件的链路监控需求，需要设计不同侧重点的链路监控方法. 对大规模、高吞吐量的在线服务系统而言，在资源充裕以及对目标系统影响不大的条件下，更适合采用主动跟踪方法；而在监控对性能要求较高，且资源相对较少的软件系统时，则更加适合采用被动跟踪方法. 因此，在未来的研究工作中，如何合理地选择链路监控方法，并针对微服务特点对相关算法进行相应的优化，是在微服务链路监控方向需要长期研究并解决的难点问题.

3.2.2　度量指标监控

为了实时地获取软件系统运行状态信息，更加直观地监控软件系统运行状态，并考虑合理性和美观性对相关数据进行可视化展示，需要实现度量指标监控，从而对基础设施部署、网络状态、服务调用等度量指标进行量化监控. 度量指标监控是指从事件发生时间和当前值的角度记录的监控信息，可以聚合起来查看某些指标数据和指标趋势，并作为后续故障检测算法的前置基础和数据支撑[18]. 基础的指标度量类型分为度量、计数器、直方图、TPS 计算器和计时器五种，主要监控了包括系统运行瞬时状态的数据信息、事件发生的概率信息、数据统计分布情况等与软件系统运行时变化的聚合状态信息. 而在微服务系统环境下，面向多实例部署和独特的服务部署、通信机制，度量指标监控需要增加额外的监控数据类型，例如容器、VM 等状态，并进行更复杂的信息聚合处理.

表 6　链路跟踪方法对比分析表

类别	定义	举例说明	方法	场景	评价指标	特点
主动跟踪	通过对目标系统进行一定程度上的干预，从而强化链路信息特征	Dapper[55]	通过 trace 和 span 构建跟踪链路	大规模分布式系统 故障发现 动态展示	性能开销 可扩展性	有效获取链路信息； 无法应对多个 追踪请求合并场景
		Zipkin	基于 Dapper 的扩展	大规模分布式系统 故障发现 动态展示	性能开销 扩展性 应用级透明性	出色的扩展性 友好的数据展示界面 提供多种客户端 SDK
被动跟踪	通过网络数据包传输拦截、网关请求拦截等方式被动采集信息	F′abio Pina[57]	通过网关搜集链路数据	Docker 容器	性能开销 数据完整性 应用透明性	监控系统与应用程序 解耦可扩展性强
		Marcello[58]	基于嗅探方法 网络包拦截	Kubernetes 集群	性能开销 数据完整性 应用透明性	应用程序透明 非侵入式 性能开销小

产业界现有的软件监控工具也对度量监控指标有所涉及. 例如，传统的 Dapper[55]有留存 Annotation 模块用于自定义度量指标来丰富 Dapper 的链路跟踪，从而能够监控更高级别的系统运行状态，并帮助软件故障调试；Prometheus①作为一个独立开源的监控、告警工具，主要实现对软件系统的延迟、流量、错误和饱和度四种维度的全面监控. 上述四种维度也被称为监控的四个黄金信号，代表了软件系统中最重要的衡量因素. 同时，根据 Brendan Gregg 所提出的 USE 性能分析方法，如表 7 所示，本文将以本文所遴选的 105 篇相关文献以及当前产业界广泛应用的度量指标作为数据来源对 CPU、存储、网

① Promethus, https://github.com/prometheus/prometheus, 2012

络、服务和通信五个层级从上述四种维度进行统计分析，并建议将此作为度量监控指标选择的基础集合.

表 7 度量指标划分表

监控方向	延迟	流量	错误	饱和度
CPU	CPU 调度延迟	CPU 利用率	特定的 CPU 故障事件	缓存运行队列长度
存储	读写等待时长	读写 I/O 级别	文件系统磁盘错误	I/O 队列长度
网络	网络驱动等待队列	每秒传入传出字节	网络设备错误、丢包	重新传输段
服务	完成请求时间	每秒请求数	处理请求访问资源故障	资源使用量
通信	响应请求时间	每秒处理的请求数	处理请求访问资源故障	资源使用量

通过基于四个黄金信号的监控框架能有效地发现和理解软件系统变化，并以此作为评估在不同层级中需要监控的度量指标的标准. 然而，在特定领域下，由于系统环境的复杂性和针对领域的需求，四个黄金指标并不能完全满足复杂软件度量指标监控的需要. 因此，一个完善的度量指标监控体系，除了将四个黄金指标作为监控主体外，还需要针对特定的监控需求增加额外的数据指标，从而更有效地理解软件变化，发现故障问题. 在微服务领域内，Fadda[59]和 Tonghoon Suk[37]都关注到了对微服务 VM 相关指标的监控，通过集群协调器从不同监控维度获取 VM 的状态以及资源使用情况，并在此基础上计算 VM 的风险状态；Meng Yuan[41]针对微服务故障类型划分故障组件，监控 KPI（用户感知指标）、QPS（WEB 用户每秒查询）、中间件状态和部署环境相关指标（容器 CPU 利用率等）等度量指标；易俗[60]考虑了网络负载、机器性能和工作情况，提出一种综合地动态调整心跳检测频率的策略，通过心跳检测判断节点运行状态；Thalheim[61]重点阐述了 Sieve 监控工具中通过提取分析度量指标之间依赖从而实现监控指标缩放精简和根本原因分析功能，考虑了微服务度量监控指标之间的依赖关系，进一步完善了监控体系. 此外，为了实现实时的故障告警，Wang 提出了 OpernStack 监控系统[62]，用于对微服务系统的指标监控. 然而，该方法无法解决告警监控数据过大导致的性能和准确性问题. 因此，Bădică A 提出通过采样的方法计算软件运行指标在故障检测中的最小可接受阈值[63]，从而确保软件系统的可靠运行，同时解决了海量数据监控对目标系统的影响.

上述研究方法主要是针对微服务软件的特点对 VM、容器、依赖关系等度量指标的监控，结合四大黄金信号，构建了一个完整的软件度量指标监控体系. 然而，随着微服务架构的不断发展，需要监控的度量指标数量也会日渐增加，而臃肿的度量指标监控，也会对软件系统运行造成一定的影响. 因此，如何通过挖掘分析度量指标之间的关系，对度量指标的选择做合理、有效地筛选、精简和侧重，是未来在度量指标监控方向需要面临的挑战.

3.2.3 日志监控

日志监控也是软件监控的重要组成部分. 在大规模软件系统的运行过程中，框架代码、系统环境和业务逻辑等都会产生日志信息. 传统日志监控方法需建立集中式日志监控系统，将所有节点的日志统一搜集、传输、存储、分析以及告警，从而提高故障检测效率和故障定位准确度. 由于微服务系统的每个服务都可选择最合适的工具进行开发，加剧了日志数据格式和语义的异构性，导致日志的关键字、正则表达式和相关规则目录很难得到维护[64]，大部分的运行日志需要人工检查和处理，造成了极大的人力和时间开销. 因此，面向微服务软件的日志监控主要研究如何对微服务多源复杂日志的实时采集和持久化存储，并对日志信息进行格式化处理[39]，从而实现多源日志数据的融合[50]和可视化.

在目前产业界，ELK 是一种主流的日志监控结构，它提供了日志监控一系列的解决方案. ELK 由 Elasticsearch、Logstash①、Kibana 三个开源软件组成. 其中，Elasticsearch 是个开源分布式搜索引擎，提供对日志信息的分析和存储功能；Logstash 主要用于日志信息的采集、分析和过滤，支持大量数据采集；Kibana 是在前两者采集的日志数据的基础上，帮助分析、可视化日志信息，并实现告警. 目前产业界大部分的日志监控工具也是基于 ELK 结构进行合理的优化设计. 同时，在学术界也存在大量基于 ELK 的研究，例如 Walidatush Sholihah 团队[65]就基于 ELK 监控框架研究了服务器日志分析工作. 然而，ELK 在提供海量日志信息采集、存储功能的同时，在微服务潜在的关联关系分析和处理、特定故障的告警等方面还缺乏完善的解决方案，产生了一定的时间和人力开销.

因此，为了解决开销成本问题，Yuan Ding 通过对四个大型开源软件中的当前日志消息进行定量特征研究，构建了一个自动化的日志代码检查器[66]；Christophe Bertero 利用自然语言处理技术，实现最小人工干预的自动化日志处理方法[67]. 上述两种方

① Logstash. https://www.elastic.co/products/logstash.2021

法都是通过对日志信息自动化处理的方法减少人力成本，但并没有从根本上解决由于微服务关联关系复杂导致的日志信息异构问题、故障类型复杂多样导致的告警问题等. 因此，Cinque 提出一种基于熵的方法处理非结构化运行日志[68]，解决微服务日志信息异构的问题；He Shilin 对比分析了六种基于日志的异常检测方法[69]，并发布了开源数据包，为后续实现针对微服务的故障检测提供方法参考；李文海设计了一个 LogVisualization 日志监控工具[70]，该工具定义了微服务的日志模型，以较小代码侵入性采集日志信息，并针对四种典型的微服务故障实现可视化调试，从而解决了微服务故障告警问题. 上述方法有效地解决了日志数据异构和特定故障告警问题，实现对微服务系统的全面监控，从而提高了基于日志监控的故障检测效率.

根据上述研究结果分析，如表 8 所示，针对微服务日志监控的研究工作完成了从日志采集可视化，海量数据存储，多源异构数据处理，再到实时高效监控和较少开销的演变过程. 综上可以看出，在完善微服务监控体系整体功能的基础上，大部分研究更关注如何摆脱传统监控体系的局限性，设计符合微服务特点、用于解决实际问题的微服务监控方法，并提高监控方法的可用性、全面性、无侵入性以及多源数据处理能力等，从而为后续故障检测过程提供数据和理论支撑，提高故障检测的准确性和效率. 在未来研究工作中，应该进一步设计出对软件系统无侵入式、多源数据融合的全新监控模式，优化微服务监控方法，降低对目标系统的影响.

表 8　日志监控方法对比分析表

典型方法	具体实现	场景	评价指标	特点
Christophe Bertero[67]	自然语言技术 最小人工干预 自动化日志处理方法	开源 VNF：Clearwater	准确性 性能开销	无法定位故障位置 通用性差 人力开销成本低
M. Cinque[68]	基于熵的方法处理非结构化运行日志	Apache Storm Cassandra	相似性 信噪比 详细程度	能够处理异构数据
李文海 彭鑫[70]	定义微服务的日志模型 实现故障告警	TrainTicket 系统 Kubernetes 集群	代码入侵性 数据完整性 定位准确性	较小代码侵入性 可视化界面完善 准确，高效

3.3　故障诊断

故障诊断作为软件运维的重要组成部分，对提高软件系统的运行质量和可靠性有重要影响. 然而，目前的传统故障诊断方法还停留在针对单体软件故障类型进行诊断，大部分研究主要通过检测度量指标，或利用人工经验构建知识图谱，或预先设置故障告警规则[29]，与故障库进行规则匹配[71]，或基于依赖图和故障树进行故障分析，因此很难自动地诊断故障并细粒度定位故障起因[33].

同时，微服务系统相比于传统的单体架构和分布式架构，其软件架构、通讯机制和服务依赖更加复杂. 独立开发导致技术上的分离. HTTP 通信与 Queue 等消息队列组合的机制增加了问题诊断的难度. 服务间的复杂依赖关系带来了很多的不确定性，每一个服务实例可以动态的创建、部署和销毁，当前的故障诊断方法不一定能完全适用于微服务软件，给面向微服务软件的故障诊断带来了更大的问题和挑战. 因此，随着微服务架构的广泛应用，越来越多的专家学者开始考虑如何针对微服务架构特点，在原有故障诊断算法的基础上，结合图论、深度学习等算法，设计出在故障诊断的各个流程对微服务的不同层面进行故障诊断的有效方法.

当前大部分关于故障诊断的研究工作都涵盖了故障分析和故障定位两部分内容. 其中，故障分析是故障诊断的前置输入，通过分析故障与微服务系统中复杂的依赖、性能指标等软件架构和软件运行特征间的关系，构建关联关系模型，从而为后续故障处理提供数据支持；故障定位则是在故障分析的基础上，通过相关算法定位故障位置和起因，决定了故障诊断结果的可靠性和有效性. 因此，为了深入分析故障诊断的工作内容和内部关联关系，如表 9 所示，本节从微服务故障检测各个阶段职能角度出发，将故障分析和故障定位的概念、相关方法和引文进行研究分析.

3.3.1　故障分析

故障分析主要针对目标系统计算故障与故障影响因素的相关性，在此基础上构建多种形式的关联关系模型，从而可视化故障发生的因果关系. 其中，如何对关联关系进行建模是故障分析的关键问题. 根据 Uhle[72]等团队的总结性工作，本文将从方法

表 9 微服务故障诊断划分表

研究内容	基本概念	相关方法	引用
故障分析	分析故障与影响故障因素间的层级关系，构建关系模型	依赖图建模 影响图建模	[51,72-80]
故障定位	基于关系模型定位故障位置 识别故障起因	随机游走算法 执行轨迹比较	[40,51,81-83]

和输出模型层面将面向微服务的故障分析方法分为依赖图、调用图和影响图建模三种类型. 其中，依赖图建模是通过分析挖掘微服务软件每个服务之间的依赖关系构建关系模型的方法；调用图建模是通过跟踪软件运行时服务间相关调用的执行轨迹构建关系模型的方法；影响图建模是分析了故障表征和故障起因之间的层级关系，同时考虑传播延迟、节点状态等不同软件指标对故障的影响，依据相关领域的知识图谱的构建关系模型的方法.

在上述三种故障模型构建方法中，基于调用图的故障分析方法应用最为广泛. 其中，王子勇团队提出一种基于执行轨迹的调用图建模方法[73]. 该方法在测试和运行阶段，对请求调用的执行轨迹进行信息采集，从而尽可能全面地获取请求调用的执行轨迹，构建调用树模型. 然而，该方法通过埋点方式获取轨迹信息，难以检测与特定请求相对应的跨节点执行轨迹，同时，微服务业务逻辑复杂，难以全面获取不确定的执行轨迹调用链，在一定程度上影响了调用树的全面性和可靠性.

为解决上述方法的问题，Yen 等人提出一种结合请求调用和服务依赖的故障依赖图构建方法[74]. 该方法通过挖掘服务间依赖关系，从而分析和可视化微服务之间的服务调用链. 该方法能够可视化微服务交互情况，帮助开发和运维人员进行服务管理. 相对于独立的调用图模型，覆盖范围更加广泛，关系模型的可靠性和可用性在一定程度有所提高. 此外，为解决当前服务调用分析面临的结构复杂，数据庞大且不平衡等问题，Yu Qing-Yang 提出基于控制流分析构建服务调用模型的高级抽象方法[75]，能够有效检测微服务故障和性能异常.

相比于调用图模型由于异常传播途径复杂，难以全面分析服务间关系，基于影响图的故障分析方法在这方面有着巨大的优势. Wang Ping 团队提出了一种数据驱动的影响图[51]代替精确的需要调用图捕捉故障的传播模式，在 PC 算法的基础上设计了影响图构造算法，从原始指标中建立影响图，帮助描述和分析异常传播. 该算法能够自动构建影响图，不受后端动态和性能基线的影响. Tang 提出通过挖掘微服务与异常事件的相关性构建分层相关图，能够细粒度分析故障关系[76]. 然而，上述方法仅通过人工经验和相关领域知识对故障相关监控度量指标进行选择，难以满足软件故障诊断可靠性和全面性的需要.

为提高故障分析方法的准确性和有效性，Chen Pengfei 引入了一种基于两层因果结构的故障影响图[77]. 该方法通过 BCP 变化点检测，挖掘软件运行过程中产出的长期数据序列中变化点，从而自动化构建双层因果图模型. 然而，该方法只考虑在性能问题中与非正常消耗物理资源（如 CPU）或逻辑资源（如锁）相关的故障，无法针对微服务特点，对服务调用过程中的故障进行分析.

因此，基于影响图的故障分析方法的关键在于被选取影响因素的合理性和全面性. Aggarwal Pooja 团队提出针对运行时日志解析错误率与微服务间的因果关系[78]. Meng Ma 提出了异常行为图，定义了两个二值运算和一个相似函数，实现针对不同类型度量与服务间的动态关联[79]. Horovitz，Shay 提出通过日志分析故障与 KPI 间的关联关系[80]. Meng Yuan 在考虑 KPI 和指标度量之间的联系的基础上，结合传播延迟导致不同时间节点之间存在因果关联关系，设计了一种新型的路径条件时间序列 PCTS 算法[40]，可准确捕获时间序列数据的顺序关系. 该算法首先通过改进的 PC 算法学习时间序列各点的因果图；然后结合不同时间序列的关系，生成故障因果图. 该算法考虑到传播延迟在不同节点的影响，极大地提高了关系模型的可靠性.

如表 10 所示，在微服务环境下，全面且精准捕获调用图的成本很高，且再精准的调用图也无法包含所有的错误传播路径. 例如，由于云的资源共享特性，多个容器在同一个物理节点上运行. 一个容器的缺陷耗尽了网络带宽，这将导致其他容器出现问题. 然而，在调用图中，这些容器之间可能没有网络连接. 因此，基于调用图的故障分析方法存在一定的局限性. 而依赖图作为调用图的扩展模型，相比较基于调用图的故障分析方法，其建模成本更加高昂，但故障分析的结果也更为全面. 此外，影响图建模方法作为目前较为热门的用于故障分析建模方法，通过直接计算整体故障表征和故障起因间的关联关系，忽略了内部复杂的逻辑流程，实现故障分析. 基于上述分析，可以看出当前针对故障分析的研究工作已完成了从服务调用、服务依赖等单

表 10　故障分析方法对比分析表

类别	定义	举例说明	方法	场景	评价指标	特点
调用图	跟踪服务间相关调用的执行轨迹构建关系模型	王子勇[73]	动态插桩检测 服务请求	Bench4Q	性能开销 查准率 查全率	难以检测与特定请求对应的跨节点轨迹 难以全面获取不确定的执行轨迹调用链
依赖图	分析挖掘微服务软件服务间 依赖关系构建关系模型	Yen[74]	基于 MAT 构建 服务依赖关系	基于 MSA 系统	查全率 时间开销	建模成本更加高昂 依赖图更全面
影响图	分析故障和起因间层级关系 考虑不同软件指标构建关系模型	Wang Ping[51]	基于数据驱动的动态影响图构建方法	Bluemix 模拟环境	相关性 准确性 精度	忽略系统内部 逻辑流程 建模成本较低
		Chen Pengfei[77]	基于 BCP 变化检测 基于 PC 的因果图构建	TPC-W 在线图书购物系统	可扩展性 稳定性 精度	只考虑性能问题

维度分析，逐步迁移至多维度分析，该研究结合了调用图、依赖图以及基于度量指标的故障影响图的特征，从而更加全面的针对微服务系统故障进行分析，进一步提高了故障诊断的准确性.

3.3.2　故障定位

故障定位是在关联关系模型的基础上，利用图论、概率统计和博弈论等相关算法准确定位故障发生的位置，识别故障起因. 而故障定位算法选择的优劣一定程度上决定了该故障诊断方法的有效性和准确性. 目前，针对微服务系统运行特征，相关研究学者从服务调用、服务依赖等维度提出了不同的故障定位方法，主要包括基于图模型游走的故障定位算法和基于树编辑距离的轨迹比较方法两类.

在基于图模型游走的故障定位算法中，随机游走策略的应用最为常见. 例如，Sun 提出的基于随机游走的故障定位方法，通过故障排除路径生成算法实现[81]. Myunghwan Kim 提出一种面向调用图模型的基于随机游走策略的无监督启发式方法[82]，用于对故障的根本原因进行排序，辅助运维人员缩小搜索空间规模，节省了开销. 同时，研究还引入了对历史数据的伪异常聚类算法，通过离线运行计算密集型聚类算法和在线运行随机漫步算法，且由于算法是无监督的，省去了学习该系统所需要的时间开销. 然而，该方法存在固有的基于调用图的缺陷，没有考虑同一节点不同服务相关性，一些异常调用轨迹无法分析.

因此，有效的故障定位算法应考虑服务相关性的影响. Wang Ping 提出了一种面向影响图的基于二阶随机游走的启发式调查算法来识别云事件的根本故障服务[51]. 该方法通过给定的影响图，从故障发生位置开始遍历，计算向前、向后过渡的概率，随机沿着影响图游走. 通过在相邻服务中随机选择下一个服务，按顺序访问不同的服务，记录每个服务被访问的次数，并以降序输出异常服务等级列表作为根本原因识别结果. 该方法假设与故障的相关性越高的异常度量指标越有可能是故障起因. 该方法能有效计算出不同服务之间的相关性，从而精准识别故障起因.

然而，上述研究直接将最相关的影响因素作为故障起因，缺乏合理性说明. 因此，Meng Yuan 提出面向影响图，考虑时间原因的随机游走方法[40]. 该方法认为相关性与故障起因并不等同，因此不能通过相关性进行随机游走. 该方法通过部分相关法消除混杂因素的影响，同时考虑了指标的异常程度和传播延迟的问题，将不同的度量指标划分成不同优先级别，然后结合影响关系、时间顺序和监视度量数据的优先级，实现对故障起因的精准识别.

相比于基于图游走的故障定位方法关注故障发生的根源问题，基于树编辑距离的轨迹比较方法更关注定位服务级别的故障位置. 例如，王子勇团队提出利用树编辑距离评估请求处理的故障，并通过分析执行轨迹差异定位故障发生的方法调用[73]. Meng Lun 将微服务多个组件的请求描述为组件调用树[83]，通过分析执行轨迹的差异和响应时间波动定位微服务软件应用程序故障. 上述方法都利用树编辑距离来计算以调用树为特征的历史序列痕迹的相似度，从而减少分析轨迹的数量，以提高检测异常的效率. 并且，该方法还通过与基线对比，计算出轨迹的异常程度，然后利用 PCA 对异常分量进行定位. 该方法能快速定位故障位置.

此外，还有一部分研究学者结合了影响图与调用图的特点，开展相关研究. 例如，Alvaro Bran-

don[84]团队通过图相似度比较的方式实现故障定位，提出了面向微服务架构的根本原因分析架构，将软件实时异常与异常数据库进行相似度比较从而实现异常排查. 该方法能够有效降低时间和人力开销，提高故障诊断的准确度. Li Min 团队通过半监督学习方法获得故障阈值[85]，并基于调用跟踪的排序算法定位故障根本原因. 然而，该方法强烈依赖于异常数据库中对于异常影响因素间关联关系的数量和准确性，存在一定的局限性.

上述方法都是基于关系模型的故障定位方法，目的是在实现粗粒度地查找服务级别问题的前提下，从性能层面细粒度定位故障根源问题. 其中，基于树编辑距离的轨迹对比分析方法能有效地定位服务级别故障发生位置，而基于图游走的故障识别算法则能从性能层面识别故障起因.

分析上述研究结果可知，目前针对微服务系统的故障诊断方法的研究工作主要集中于如何对关联关系模型进行建模，考虑如何加入更多的影响因素以提高故障诊断的全面性和准确性. 然而，故障影响因素种类的增加，关联关系模型的复杂化，会极大地影响故障诊断效率. 同时，如表 11 所示，上述方法并没有针对特定的微服务故障类型进行细粒度的故障诊断，传统软件与微服务软件的差异性并没有完全体现，传统的故障诊断方法不适用于微服务软件的问题也未得到根本性解决. 因此，在未来的研究工作中，应该更多的关注于微服务软件本身的特性，针对典型的微服务故障类型进行故障诊断的算法设计，细粒度定位故障起因和位置，从而保障微服务故障检测方法的可用性和准确性，进一步提高微服务软件的可靠性.

表 11 故障定位方法对比分析表

类别	定义	举例说明	方法	场景	评价指标	特点
随机游走策略	通过给定的影响图，从故障发生位置开始遍历	Myunghwan[82]	基于随机游走的无监督启发式方法	LinkedIn	精度 准确性	没有考虑不同服务相关性；异常调用轨迹无法分析
		Meng Yuan[40]	结合时间因素的随机游走方法	基于微服务的在线购物平台	准确性 性能开销	故障定位准确性高
轨迹比较方法	通过分析执行轨迹的差异和响应时间波动定位软件故障	Meng Lun[83]	基于树编辑距离分析轨迹差异	Bench4Q； Social Network	准确性 性能开销 召回率	依赖离线轨迹数据 构建复杂度过大
其他	图相似度比较方法	Brandon[84]	将实时异常和历史异常相似度对比	Grid'5000 测试平台	准确性 精度	准确性有较大提高 没有考虑时间维度

3.4 故障预测

故障预测作为软件故障检测的重要组成部分，是通过利用软件监控获取的系统状态数据和基于软件故障分析建立的关系模型，在此基础上建立故障预测算法，对目标潜在的软件故障发生的概率进行计算. 因此，故障预测的有效性和准确性强烈依赖软件监控和故障诊断的质量. 同时，软件体系结构的差异也会对软件预测方法的适用性造成一定影响. 目前，传统软件系统的故障预测方法主要包括基于异常数据的时间曲线拟合方法和基于签名的故障模式匹配方法[86,87]，能够有效对目标系统的潜在软件故障进行预测. 然而，微服务软件系统复杂的部署结构使其面临更多能够导致系统发生故障的潜在威胁，故障类型也更加多样化[88]. 因此，如何针对微服务软件实现有效、全面的故障预测的问题也得到了广泛关注.

在各类方法中，基于异常数据的时间曲线拟合方法应用最为广泛. 该方法主要根据时间变化对节点数据进行状态拟合，从而判断指标数据随时间的变化趋势，实现对故障的预测. 目前该类方法主要针对软件系统的节点故障、网络故障和服务故障三部分进行故障预测. 其中，Lin Qingwei[89]采用 LSTM 算法拟合时间数据，采用随机森林模型合并空间数据，将时间、空间两种时间对节点故障的倾向进行排序，从而判断节点故障概率. 然而，由于微服务软件本身的动态性，实例可随时的创建和删除，导致该类面向分布式节点故障的故障预测方法无法完全满足需要. Ji Weiliang[90]基于日志数据，采用了 CNN 算法挖掘关键信息，实现对网络故障的预测；王晓峰[91]则考虑了服务故障问题，采用了 XGBoost 算法，针对某个服务，采集某段时间内的主机性能数据作为一条特征数据，然后设定标签数据预测当前主机服务调用耗时. 根据上述研究工作，可以看出基于异常的时间曲线拟合方法过于依赖软件监控数据，受限于获取到数据的类型，同时，算法计算量的多少也会影响故障预测方法的效率和

质量.

基于签名的故障模式匹配方法是将已知的故障行为状态作为模式匹配，通过故障注入方法以及历史故障事件训练模型，或者根据可观测度量指标进行分析，从而实现对故障类型的预测. 该方法主要应用于代码层面的缺陷分析预测当中，大多数采用线性回归、逻辑回归和人工神经网络方法实现故障预测. 例如，Yeresime Suresh[92]通过统计和机器学习方法，以 CK 度量元为基础，预测代码层会出现的软件故障与缺陷，然后对故障进行分类，通过模型实现代码故障预测. 然而，该类方法没有涉及时间趋势，因此，不能很好地处理性能指标类故障，只适合处理离散的故障事件. 因此，Leonardo Mariani[86]提出将基于异常数据的时间曲线拟合方法和基于签名的故障模式匹配方法结合的分层方法，以弥补基于签名的故障模式匹配方法在处理性能指标故障这种连续性变量问题时的不足. 但该方法仍未解决基于签名的故障模式匹配方法只能预测故障可能发生的类型，而无法对故障发生的时间、位置进行识别的问题，并受限于固有的故障模式库，无法预测不确定的软件故障.

同时，上述两种方法只能预测整体系统存在的特定故障事件，并未考虑软件架构和依赖关系的影响，不能判断故障对系统的影响程度. 因此，Teerat Pitakrat 等人[93]考虑了故障传播影响，将相关的故障预测方法和架构知识相结合，采用贝叶斯网络建立故障传播模型，分析了内存泄露、系统过载和节点崩溃三类故障问题，从而更加全面地针对目标系统实现故障预测，降低故障的影响程度. 此外，Zhou Xiang[38]等提出一种名为 MEPFL 的潜在故障预测方法. 该方法是典型的针对微服务软件架构而设计的故障预测方法，特别针对微服务软件系统的三种典型故障类型，通过学习系统运行日志，在跟踪级和微服务级训练故障预测模式，以预测潜在故障.

根据上述研究成果可以看出，传统的针对单体软件和分布式软件的故障预测方法都存在一定的局限性，如表 12 所示，现有的大多数方法都是在此基础上，对传统故障预测方法进行改进和完善. 然而，只有少部分研究考虑了具体的软件体系架构，例如微服务架构，针对特定的软件故障类型和影响因素设计故障预测方法. 因此，在未来的研究工作中，应更加着重研究软件体系架构对故障类型和故障预测质量的影响，从而针对性地设计故障预测方法，提高软件可靠性，保证系统可持续运行.

表 12　故障预测方法对比分析表

类别	定义	举例说明	方法	场景	评价指标	特点
基于异常数据的时间曲线拟合方法	根据时间变化对节点数据进行状态拟合，判断指标数据变化趋势	Lin Qingwei[89]	采用随机森林模型合并空间数据对故障倾向排序	生产云服务系统	准确性 召回率	无法满足微服务实例动态创建的需求
		Ji Weiliang[90]	基于日志数据和 CNN 算法挖掘关键信息，进行预测	/	准确性 精度	适用于大规模系统 可扩展性强
基于签名的故障模式匹配方法	将已知故障行为状态作为模式匹配，通过故障注入方法进行分析	Suresh[92]	基于 CK 度量元 基于统计和机器学习算法	MATLAB 环境	准确性 精度 完整性	不涉及时间趋势 不能很好地处理性能指标类故障，
		Leonardo[86]	将异常数据和签名的方法相结合	Cloud-based IP Multimedia Subsystem	精度 召回率 准确性	只能预测故障可能发生的类型 受限于模式库
其他	/	Zhou Xiang[38]	通过学习系统运行日志获取故障信息	TrainTicket 案例系统	时间开销 准确性	在跟踪级和微服务级能够有效预测
		Teerat Pitakrat[93]	采用贝叶斯网络建立故障传播模型	a distributed RSS feed reader application	准确性 精度 召回率	全面故障预测， 降低故障影响程度

3.5　故障复现和测试

故障复现和测试过程可用于评估软件系统运行质量和安全性，以及软件故障检测方法的有效性，同时，为了构建合理、有效的故障分析模型，不仅需要对软件系统正常运行情况下的状态信息进行监控，还需要对软件故障情况下的日志、执行轨迹等状态信息进行监控，从而获取特定故障对软件系统的影响形式和状态表征[36]，是软件故障检测环节不

可或缺的一部分. 故障复现与测试的主要工作内容包括故障注入测试、软件增量调试等一系列可能引发软件故障的因素的复现测试. 当前的国内外微服务故障检测的研究中，故障注入和故障调试的相关方法已经很好的应用于微服务故障检测过程中，作为一种有效的故障分析建模中数据获取手段，同时能够对于故障检测方法进行评估. 与传统软件相比，微服务软件架构的复杂性、独特的故障类型和不确定的运行环境，给软件故障复现与测试方法提出了更高的要求.

根据在微服务环境下故障复现和测试的不同目的，本文将故障复现和测试的主要研究工作分为故障注入、故障调试和方法评估三个部分. 其一关于如何针对微服务软件的不同故障类型，研究如何贴合实际运行场景，对目标微服务系统进行故障注入，从而获取该故障对软件系统的影响表征，为后续故障分析建模提供数据支撑. 同时，作为评估微服务故障检测方法的有效性以及目标系统对该类故障的处理能力；其二关于如何通过一系列故障调试手段增强软件故障的表现形式，挖掘不同影响因素和故障间的关系，从而辅助开发人员对目标系统进行故障调试，进一步完善故障检测体系，同时降低时间和人力开销. 其三关于如何针对微服务故障检测方法进行有效性验证，包括了当前广泛应用的评价指标和案例系统的描述，旨在为后续研究人员在方法评估方面的实验设计提供参考.

3.5.1 故障注入

故障注入将不同的故障类型引入软件系统的不同阶段，生成目标系统的一系列错误版本. 在微服务环境下，迅速发展的代码需要进行快速测试，并专注于故障恢复逻辑，而不是业务逻辑. 因此，故障注入的主要目的是评估目标系统的故障处理能力，并掌握目标系统的故障处理规则，从而确保微服务软件的可用性、可靠性和性能. 目前针对故障注入方法的研究主要分为低级故障注入、代码注入和面向服务体系结构注入三种形式[96]. 其中，低级故障注入是对 CPU、内存和网络等方面故障的注入方法，该方法是目前应用最广泛的软件压力测试方法；代码注入是针对目标系统内部代码，注入指定的高级故障策略语言的方法，通常作为代码埋点调试的主要手段；面向服务体系结构的注入是随机向目标系统注入无法通过系统测试捕获的不可预见的故障的注入方法.

目前低级故障注入方法，主要是对目标系统整体的性能和负载进行测试，从而模拟出节点宕机，网络延迟，访问请求量大导致的软件故障问题. Rabiya Abbas 团队[94]对 JMeter、LoadRunner、Loadstorm 等目前较为常用的故障注入测试工具进行了分析比较实验. Apache JMeter 是典型的针对企业级系统的低级故障注入测试工具①. 它作为一种开源压力测试工具，可以模拟在 HTTP 网络、FTP 服务器、数据库和对象等应用程序各个阶段的负载，采用多线程的方法从不同压力类别测试目标系统的强度和性能. LoadRunner、Loadstorm 等工具在负载测试上具有较为广泛的应用. 然而，大部分的测试工具都通过脚本对低级故障进行注入，仍需要人工对目标系统的状态进行验证，无法做到自动化测试，往往会造成人力和时间的浪费；LoadRunner 甚至还需要复杂的场景设置，强烈依赖编程实现，不符合于微服务软件性能和负载自动化测试的要求. 此外，上述针对系统性能和负载的测试，在一定程度上很难针对性地复现微服务故障，从而无法有效地评估微服务故障检测方法的优劣，存在一定的局限性.

在代码注入方面，现有的注入方法都存在一定的局限性. 例如 Joshi[95]和 Gunawi[97]提出代码注入方法都提供了用于指定高级故障的策略语言. 然而，该类方法依靠特定语言功能，例如仅面向于 JAVA 开发的软件系统，通过 JAVA 特有的 AspectJ 注入错误代码，同时，代码的注入需要掌握目标系统内部实现逻辑，对源代码进行修改. 因此，代码注入方法无法适应于微服务软件多语言的特点，只能作为目标微服务系统局部功能模块内部的代码调试，从而只能对于代码层面的微服务故障进行测试验证.

由于微服务架构与传统软件体系架构的巨大差异，传统故障注入方法无法适用于微服务软件. 而面向服务体系结构的故障注入方法能够根据微服务架构特点随机对不可预知的故障类型进行注入测试，是一种当前被广泛应用于微服务故障检测方法评估的有效手段. 例如，Genesis2[98]设计了用于故障注入实验的基于模型的测试平台. 该平台由代表应用程序服务和基础架构组件的存根组成. 相比之下，Victor Heorhiadi[96]提出的用于系统测试微服务的故障处理能力框架 Gremlin，在实时服务上运行，使开发人员能够测试其行为可能与存根任意偏离的真实服务. 该框架依赖于网络上标准消息交换模

① Papers, Apache JMeterTM. http://jmeter.apache.org/. 2018,5.

式，在特定请求流上注入故障. 由于该方法不需要对代码进行侵入，因此适用于微服务多语言的应用场景. 同时，Meiklejohn 提出一种 Filibuster 原型[99]，用于对微服务系统系统地识别弹性问题，是一种有效的服务级故障注入方法. Chaos Monkey 是 Netflix 的随机故障注入工具①，能够处理无法通过系统测试捕获的不可预见的故障，并具有限制局部应用程序区域的能力. 但是，该工具无法自动分析应用程序行为，依旧需要人工干预. 因此，Alvaro 实现了一个谱系驱动故障注入(LDFI)方法[100]，基于 Netflix 实现自动化故障测试.

根据表 13 可以看出，目前针对微服务软件故障注入测试的研究主要关注于解决多语言性和复杂配置的问题，及实现自动化、无侵入式的故障注入，使其更适用于现实微服务软件的运行场景. 但大多微服务故障注入方法还是基于传统软件测试方法的基础上进行改进，缺乏完善的故障测试体系框架. 因此，在未来的研究工作中，还需要分析微服务典型的故障类型，并在此基础上，结合工业界微服务软件特点，设计出全新的故障注入框架，监控评估目标系统的故障处理能力，提高微服务故障检测方法评估的有效性和合理性.

表 13　故障注入方法对比分析表

类别	定义	举例说明	特点	缺陷
低级故障注入	对目标系统的性能和负载引起的故障进行注入	节点宕机 网络延迟超时 Rabiya Abbas[94]	注入与语言和运行无关 依赖人工，自动化能力不足 依赖编程实现和场景模拟	无法针对性地进行故障复现 无法有效评估故障检测方法效果
代码注入	根据代码内部逻辑注入错误代码	Joshi[95]	依靠特定语言的高级故障策略； 需掌握系统内部代码逻辑	只针对系统代码层面的故障类型 存在对源代码的侵入性
面向体系结构注入	根据体系结构特点随机对不可预知的故障类型进行注入测试	Chaos Monkey Gremlin[96]	不需代码入侵，适用微服务 注入无法通过系统测试捕获的故障	缺乏针对业务服务的故障注入

3.5.2　故障调试

故障调试是指通过一系列手段增强软件故障的表现形式，从而辅助开发人员对目标系统进行故障调试. 目前，针对传统分布式软件进行故障调试的一种基本方法是跟踪和可视化系统的可执行轨迹[101]，如基于程序行为切片方法实现测试用例的全执行路径覆盖[102]. 然而，微服务软件具有的高复杂性和动态性，微服务实例可以动态地创建和销毁，对微服务软件故障调试提出了更高的要求[30].

目前的研究主要集中于如何利用增量调试[103]方法解决微服务故障调试问题，如表 14 所示. 传统的故障增量调试方法无法适应微服务软件复杂的结构和动态部署的特点，例如 Burger[104]提出的一种有效故障定位调试方法 JINSI，通过结合 delta 调试和动态切片的手段获取故障复现的最小方法数，以及被扩展到的粗粒度增量调试等[105]. 因此，越来越多的专家学者开始重视研究如何针对微服务典型特征实现故障增量调试. 其中，最具代表性的是 Zhou Xiang 等人提出的一种基于增量调试算法的微服务系统调试方法[106]，包括在增量调试期间定义、部署/操作和执行自动化增量测试. 还有 Yuan Ding[107]和何潇[108]通过向目标系统注入适当的辅助信号，从而自动增强现有的日志记录代码和故障表现形式，从而辅助后续故障调试工作.

表 14　日志监控方法对比分析表

类别	典型方法	具体实现	场景	评价指标	特点
增量调试	Zhou Xiang[106]	基于增量调试 自动化测试	Train Ticket	可扩展性 有效性	具有良好的可扩展性 可靠性保障
	Yuan Ding[107]	注入辅助信号 增强故障表现形式	八个应用程序	性能开销 覆盖率 有效性	适用微服务 性能良好 粗粒度
其他	IntelliFT[109]	针对微服务的引导 弹性测试技术	Train Ticket	有效性 成功率	有效发掘 微服务 处理逻辑缺陷 粗粒度指标

① Netflix-Chaos Monkey Released Into The Wild, http://techblog. netflix.com/2012/07/chaos-monkey-released-intowild.html, 2012, 7

此外，还有一种针对微服务应用程序的引导弹性测试技术 IntelliFT[109]，用于发掘微服务软件故障处理逻辑缺陷；一种沿袭驱动的错误注入方法[110]，用于测试目标系统的故障容器机制是否有效. 上述方法都有利于提高微服务软件的故障调试质量和效率，降低时间和人力开销. 然而，目前的微服务故障调试方法受限于粗粒度的指标参数对比，考虑的方法维度也不够全面. 因此，在未来的研究工作中，应该进一步细化故障调试方法，扩展调试维度，加入请求调用链、轻量级通信等符合微服务特性的影响因素，从而完善故障检测体系框架，辅助软件系统运维.

3.5.3 基准系统

基准系统是指针对微服务故障检测方法的效果、性能等各个维度进行评估时所需要进行场景模拟的案例系统和数据集，用于对比和验证该故障检测方法的有效性. 由于微服务故障检测效果的评价是当前微服务领域研究的重点内容，而当前国内外公认的具有一定公信力的基准系统和数据集还相对较少，这给微服务研究者和实践者开展故障检测研究造成了极大的阻碍. 因此，本节重点阐述了当前学术界广泛认可的应用于微服务故障检测的基准系统和测试数据集，旨在为后续研究提供数据和实验环境支持.

在案例系统方法，复旦大学彭鑫教授提出的 TrainTicket 开源微服务基准系统[38]，包含了 40 个以上的业务微服务，并结合了消息中间件缓存、分布式缓存以及数据库服务等组件，并支持 Java、Go 和 Python 等多语言进行开发，同时还支持使用 Jaeger、Promethus 等可视化监控工具对系统进行监控，能够有效模拟真实场景下微服务软件系统的运行状态，为微服务故障检测方法提供相对完善的环境. 同时，Yu Gan 团队设计了一个 DeathstarBench 系统[111]，采用交互式微服务构建方法，包括了社交网络、媒体服务和电子商务商店等场景，旨在量化微服务系统在整个系统堆栈中的影响，具有一定的代表性和可扩展性. 此外，Acme Air 作为一个公开可用的案例系统，也得到了广泛的应用. 该系统基于微服务架构设计，用于模拟航空公司系统，在 NodeJS 中实现，用 Docker 平台部署，能够有效模拟真实场景. 而在数据集方面，清华大学举办的第一届 AIOps 竞赛数据集能够有效适用于微服务应用系统故障发现和根因定位，主要包括 KPI、业务异常信息等多维度指标，能够有效推动微服务故障检测方向的研究.

4 研究与展望

综上所述，随着微服务架构得到广泛应用，针对微服务的故障检测方法研究逐渐成为了工业界和学术界研究工作的重点. 本文结合现实生产环境需求，对传统软件的故障检测方法和微服务故障检测方法的相关研究进行了总结和分析. 总体来说，目前研究成果由于缺乏完善的故障检测体系的指导，导致一些理论、方法和技术存在一定的局限性，还有许多的问题和挑战亟需解决. 因此，本文首先通过分析汇总目前微服务故障检测的相关过程和方法，提出了针对微服务软件系统的基础故障检测框架，然后为了帮助进一步改进现有的故障检测技术，本文详细阐述了该领域未来的发展趋势.

4.1 微服务故障检测框架

合理的软件故障检测框架通过对工作内容的有效划分，能够规范故障检测过程，明确研究内容，在各个阶段展开深入研究，从而切实推动软件故障检测的研究进展. 由于微服务软件系统的复杂性和动态性，导致微服务故障检测方法的规模过于庞大，很难系统地、全面地实现对微服务软件的故障检测. 目前主流的针对微服务故障检测的研究工作主要围绕软件监控、故障检测、故障预测、故障复现和测试四个方向展开. 然后，在一定研究工作的基础上，通过不同故障检测阶段的协调配合，最终实现微服务故障的有效检测. 然而，大部分的总结性工作都没有对故障检测的各个阶段进行详细阐述，仅仅只是通过数据获取、算法输入、算法输出等过程的简要分析和描述，导致了微服务故障检测方法各个阶段概念和内容的混淆和模糊. 因此，建立针对微服务的故障检测框架对于提高微服务软件的可靠性和可用性至关重要.

本文讨论了微服务故障检测框架的缺失势必会带来三方面的影响. 从功能角度看，多样化的微服务故障检测方法很难得到统一，不同的故障类型和故障检测手段，对软件系统的影响程度和检测结果都不同，势必会影响故障检测方法的全面性和准确性；从发展角度看，目前针对微服务故障检测框架的总结性研究较少，且针对不同故障类型的偏重和手段不同，使得微服务故障检测框架存在很大的差异. 而基础框架的差异性会导致一个独立的微服务故障检测技术只能在原有的基础上进行扩展和完善，从而很难满足不断发展和完善的微服务架构的需要；从研究角度看，目前针对微服务故障检测的研究方向呈现多元化、复杂化和异构化的趋势. 该

趋势导致了研究学者对于微服务故障检测的概念理解缺乏统一，在一定程度上制约了微服务架构的发展. 因此，为了有效提高微服务故障检测的可用性和可靠性，推动微服务故障检测技术的研究和发展，必须构建合理、完善的故障检测框架，辅助梳理故障检测的基本内容，明确具体的研究方向.

本节在一些传统的软件故障检测过程的基础上，结合现有的面向微服务架构的故障检测技术的相关研究进行总结分析，同时考虑微服务软件故障检测的目的，提出了微服务故障检测的整体框架如图 2 所示，该框架主要从 4 个部分划分故障检测流程和工作范围.

（1）软件监控. 作为故障检测框架的基础模块，负责监控目标系统，包括监控数据采集和故障发现告警两部分. 首先，该模块基于目标微服务系统在运行过程中产生的状态数据作为输入，从日志、链路和度量指标三个维度进行数据采集，进行聚类分析处理，同时，考虑不同应用端和不同模块的编程语言和架构差别导致的多源数据异构问题，实现持久化存储；然后，根据监控数据的变化趋势和阈值进行判断故障的发生，及时产生告警信息. 最后，将持久化监控数据和告警信息作为软件监控的输出.

（2）故障诊断. 作为故障检测框架的核心模块，负责对目标系统的运行故障诊断，分为故障分析和故障定位两个部分. 首先，该模块基于软件监控建立的监控指标数据库作为输入，通过对故障、故障影响因素和故障起因进行关联关系挖掘和监控数据分析；然后，基于关联关系构建依赖图、调用图、影响图等关系模型作为故障分析阶段的输出结果；在此基础上，根据故障关系模型，通过故障定位方法诊断故障发生的位置和起因作为最终输出，从而及时进行故障处理，保证软件系统的正常运行.

（3）故障预测. 负责对目标系统的故障发生进行有效预测. 首先，在故障发生之前，将软件监控采集到的监控数据信息作为输入，进行发展趋势分析；然后，由于微服务软件不确定环境下发生故障的概率较高，需要根据故障诊断产生的关联关系模型作为输入，对目标潜在的软件故障发生的概率进行计算，并以此作为故障预测模块的输出结果. 从而及时进行故障预防处理，提高目标微服务系统的可靠性和可持续运行的能力.

（4）故障复现和测试. 负责对目标微服务系统进行故障注入和故障调试测试. 首先，针对微服务复杂依赖的特点，该模块基于结合人工经验，将微服务软件故障进行组合，以此作为测试输入，并测试结果作为输出结果反馈到故障诊断阶段. 同时，考虑到微服务软件故障发生时内部复杂的级联反应，该过程的主要信息包括故障先决条件、预期故障症状以及故障影响因素，从而从整体上明确了软件运行状态受故障影响的变化特征. 其中，故障注入过程是通过复现故障先决条件，模拟真实场景下软件故障的发生，获取故障症状和故障影响因素之

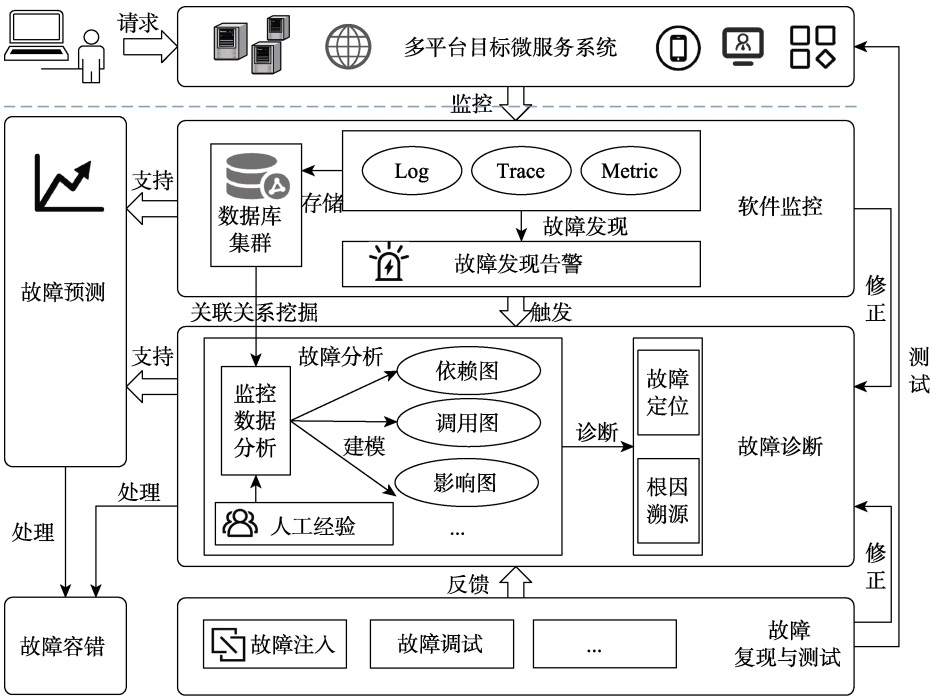

图 2　微服务故障检测框架图

间的关系，为故障关系建模提供正向反馈；故障调试过程通过增量调试、注入信号和切片等手段判断故障的起因，能够获取故障起因和故障症状之间的关系，为故障关系建模提供反向反馈.

在上述故障检测模块工作内容阐述的基础上，本文对不同模块间的关联关系进行分析. 首先，软件监控作为基础模块，其监控数据输出为故障诊断、故障预测和故障复现提供数据支撑，同时将告警信息作为触发故障诊断的前提条件；然后，故障诊断中分析产生的故障关系模型作为内部故障定位和外部故障预测的模型支撑；此外，故障预测基于监控数据和故障关系模型，对故障发生概率进行计算，提高目标系统的可用性；最后，故障复现与测试，是借助软件监控获取故障测试的运行状态，并基于故障注入的正向反馈、故障调试的反向反馈对故障诊断模块的故障关系模型进行有效性修正.

该微服务故障检测框架从内容完备性、关联科学性和可修正性三个维度满足了一个合理的、完善的故障检测框架的要求. 在内容完备性方面，该框架包含了软件监控、故障诊断、故障预测、故障复现与测试四个部分的内容，涵盖了目前主流的针对微服务故障检测的研究工作，能够基本覆盖微服务故障检测的全部流程. 在关联科学性方面，该框架的各个模块之间紧密结合、相辅相成，明确了输入与输出，同时实现了各模块之间的解耦，有利于后续开展针对性研究. 在可修正性方面，该框架会针对后续研究，不断完善基础模块功能，增加服务容错等工作，优化内部逻辑，调整框架结构，以满足微服务故障检测快速发展的要求.

4.2 未来发展趋势

微服务架构作为一种新型的软件架构，其故障检测的研究仍处于发展阶段，相关技术和方法还存在局限性，很难对微服务软件实现有效的运维治理. 因此，为了有效推动微服务故障检测体系的完善，本文在对现有的研究工作的整理和分析的基础上，通过对目前产业界和学术界研究现状和发展态势的分析，从研究方向和应用领域两个方向展开讨论，重点阐述当前微服务故障检测所面临的挑战和未来的发展趋势，旨在为后续研究提供一定有效思路.

首先，在研究方向方面，本节重点讨论了软件监控、故障诊断、故障预测、故障的有效复现和预测以及其他方向上的未来发展趋势和关键难点：

（1）软件监控. 采集监控数据的方式和数据的处理过程会直接影响软件监控方法的有效性和全面性. 在未来研究工作中，针对微服务的软件监控方法建议从以下三个方面考虑：首先聚焦微服务特点，设计出对软件系统影响较小的无侵入式监控方法，用于获取监控数据，尽可能地降低对目标系统代码和业务的干预，并保证该数据信息的全面性和准确性；然后通过统一数据格式，将多源异构的监控数据进行融合，从而优化数据聚类方法，提高软件状态可视化效果；此外，由于大规模复杂微服务系统存在的海量监控信息，还应该研究如何保证软件监控方法可持续性地获取系统状态信息，并进行持久化存储.

（2）故障诊断. 目前大部分研究工作都是通过加入影响因素对关联关系进行建模，从而提高故障诊断的全面性和有效性. 然而，故障影响因素种类的增加，导致的关联关系模型的复杂化，会极大地影响故障诊断效率. 因此，未来研究工作的重点可考虑放在以下两方面：一方面，应该重点关注微服务的典型故障类型，考虑业务层、功能层和代码实现层等多维度的故障状态表征和起因，针对典型故障设计微服务故障诊断算法，从而实现细粒度定位故障位置和起因；另一方面，由于微服务软件系统的复杂性，人工经验的模糊性和不确定性会导致故障诊断结果的不准确，并且人工参与会导致时间和人力开销过大. 因此，如何实现微服务故障诊断方法的自动化，降低人工经验和人工参与的干预，也是亟需解决的难题.

（3）故障预测. 未来针对微服务软件的故障预测方法研究，可考虑技术和架构两个层面的研究内容. 在技术层面，参考传统的数据挖掘和深度学习方法，实现对故障类型分类和状态拟合，从而对故障进行预测；在架构层面，需要结合具体的软件架构特点，针对微服务故障类型和影响因素，主要从两个角度进行研究：一种是服务依赖关系导致的故障传播问题；另一种是由于故障间也存在影响关系，导致的系统级联故障和局部瘫痪问题.

（4）故障的有效复现和测试. 与传统故障测试方法类似，针对微服务的故障复现和测试应该明确故障的类型、故障表现形式和起因，在此基础上模拟现实场景进行故障复现. 然而，现有的研究工作很少关注微服务特有的故障类型，大多是在原有的传统故障复现和测试方法基础上进行改进，导致无法完全适用于微服务软件. 因此，未来研究应重点关注如何针对典型微服务故障类型，细化故障测试方法，考虑多维度影响因素，设计全面的故障复现和测试框架，监控评估目标系统的故障处理能力，从而对故障诊断过程有准确、有效的反馈，进一步

完善故障检测体系.

（5）其他方向. 根据统计分析的从 2009 年至今的相关研究来看，在微服务故障检测研究方向，除了本文列举的软件监控、故障诊断、故障预测、故障复现和测试之外，还包括软件故障容错、服务发现等研究方向[31]. 例如，作为软件监控的扩展，如何全面、实时地发现微服务部署情况，以便服务间快速可靠交互？如何设计合理的微服务故障容错机制[112]，提高微服务系统处理故障的能力，从而保证系统的正常、可靠运行?上述研究方向均可能成为未来微服务故障检测体系中不可或缺的一部分.

其次，在应用领域方面，本文通过对相关研究工作分析可知，伴随着微服务架构应用于边缘计算、AI 和 IoT 等各个领域，由于不同领域下场景的差异，给微服务故障检测带来了一系列的问题.

（1）在 IoT 物联网领域，由于物理网和微服务存在许多相同的特性，包括了轻量级通讯、独立部署的服务和去中心化开发等，能够有效规避故障风险，降低服务运维开销. 然而，当前针对微服务与 IoT 领域结合的研究大多集中在通过异构服务构建增量应用程序[113]，却忽略了针对服务安全机制的研究[114]. 因此，未来的研究应对重点关注于如何保证物联网服务在应用微服务的安全性和可靠性.

（2）在边缘计算领域，传统的安全解决方案无法保护微服务化的边缘计算网络. 在当前的研究中，已经有通过微服务部署人工智能的方式增强边缘计算学习大规模数据源的过程，从而实现模块化安全技术. 然而，在当前阶段，仍存在不同方面的问题亟需解决[115]. 例如，如何解决系统节点增加和资源约束带来的性能问题？如何优化边缘节点和服务资源的负载平衡和调度问题[116]？如何避免由于 AI 算力引发的性能故障？

（3）在微服务迁移方向，当前的微服务迁移缺乏一个成熟、完善的通用指南，由于传统原有系统的高度耦合[117]，导致拆分原有功能，迭代版本代码的难度增大，在系统运维层面的技术不够成熟. 因此，在未来的研究中，应该重点关注如下方面：如何为微服务迁移提供可靠的基础设施，避免相关故障[118]？如何对于已迁移微服务系统提供有效的资源监控和管理，有效告警故障发生[119]？

此外，通过本文的文献搜集和研究，可以发现人工智能、知识工程、大数据等技术已应用于解决微服务故障检测问题，属于智能化运维（Artificial Intelligence for IT Operations，AIOps）领域的重要研究方向之一. 但现有方法仍存在数据样本较少、数据标签标注不均衡[120,121]等问题，未来可进一步攻克相关难点问题，提升算法能力. 最后，随着微服务架构软件在更多领域的应用，如何建立符合领域需求的故障检测体系也是未来重要的发展方向. 例如，在边缘计算、雾计算、物联网等领域中应用的微服务架构软件[122]就迫切需要一种符合资源受限、环境多变特点的轻量级微服务故障检测体系.

4.3　有效性威胁

在本文研究过程中，共发现了三个有效性威胁. 我们将在本章对此进行讨论.

Internal validity（数据源和搜索词）. 本文考虑微服务故障检测相关的文献搜集的全面性，我们通过手动选择“微服务”、“微服务故障检测”“软件运维”、“软件故障诊断”等关键词在重要的数据库中进行文献搜索，对数据库以及关键词的选择上，涵盖了大部分相关文献，对关键词的手动选择可能会导致主观错误，许多包含了微服务故障检测概念的文章可能无法获得. 因此，在后续的研究中，我们会继续扩大数据库选择范围，采用定量技术选择有效的关键词进行搜索.

Conclusion validity（研究结果）. 本文的研究结果通过严格的工作流程设计产生，是合理和完备的. 结论具备一定的权威性. 我们考虑未来的工作在此研究基础上，进一步优化工作设计流程，提升研究结果的合理性和全面性.

External validity（外部影响）. 由于每个数据库的搜索机制和存储文献不同，我们不可能保证所有的相关研究都被检索到. 未知的技术不稳定可能发生在这些数据来源的内部，这可能会影响这些研究的正常检索. 这一事实被认为是一个威胁，我们不能保证这些数据库的搜索质量，但通过我们使用的研究方法，可以保证检索到的文献都会得到较好的处理.

5　总　结

随着微服务架构的广泛应用，在代码层、功能层和业务层等层面的故障类型也变得复杂化和多样化，针对传统软件的故障检测方法已不再适用.

目前，工业界所部署的微服务架构具有海量复杂的服务关系和资源，反观学术界由于缺乏市场数据与应用无法搭建出承载海量服务和基础资源的微服务架构，缺乏市场驱动，在研究内容上并不能切入领域痛点. 同时，针对故障检测方向的研究主要集中在如何改进原有的故障检测体系和方法，从而满足面向微服务系统高效故障检测的需求. 然而，

目前的研究仅仅局限于面向微服务架构的特点，并对传统的故障检测方法做出了针对性改进. 因此，这些方法并未形成完整体系，缺乏在产业界的实际应用，其可用性和价值还有待检验；同时，大部分的研究都是借助人工经验计算故障与故障因素的相关性进行匹配，很难自动诊断故障并细粒度地定位故障起因，缺乏一定的规范性和权威性.

因此，针对微服务软件的故障检测方法开展总结性工作，设计完整的故障检测框架，对提高微服务软件的可靠性和可用性具有重要意义. 本文首先对从2009年至今的105篇国内外与微服务故障检测直接相关或应用于其他软件架构的故障检测方法进行了研究综述，分析了在微服务软件中软件监控、故障诊断、故障预测、故障复现和测试四个研究方向的发展现状，并对相关关键难点问题进行详细阐述；然后，在此基础上构建了微服务故障检测的基础框架，总结分析了微服务故障检测的基本流程和内在联系；最后，对目前面临的挑战和未来发展趋势进行讨论，旨在完善微服务故障检测体系，帮助相关研究学者和从业者梳理微服务故障检测的发展脉络和最新的研究进展，以切实推动微服务架构的应用与发展.

参 考 文 献

[1] Zhang He, Wang Zhong-Jie, Chen Lian-Ping, Peng Xin. Preface to the topic of continuous software engineering-oriented microservice architecture technology. Journal of Software, 2021, 32(05): 1229-1230(in Chinese)
(张贺, 王忠杰, 陈连平, 彭鑫. 面向持续软件工程的微服务架构技术专题前言. 软件学报, 2020, 31(05): 1229-1230)

[2] Jin Wu-Xia, Zhong Ding, Zhang Yu-Yun, Yang Ming-Fan, Liu Ting. Maintainability evaluation of microservices based on multi-source feature space. Journal of Software, 2021, 32(05): 1322-1340(in Chinese)
(晋武侠, 钟定洪, 张宇云, 杨名帆, 刘烃. 基于多源特征空间的微服务可维护性评估. 软件学报, 2020, 31(05): 1322-1340)

[3] Wang Tao, Zhang Wen-Bo, Xu Ji-Wei, et al. Research on fault detection technology of distributed software system based on statistical monitoring in cloud environment. Chinese Journal of Computers. 2017, 40(2): 397-413(in Chinese)
(王焘, 张文博, 徐继伟, 等. 云环境下基于统计监测的分布式软件系统故障检测技术研究. 计算机学报, 2017, 40(2): 397-413)

[4] Jia Tong, Li Ying, Wu Zhong-Hai. A Survey of fault diagnosis of distributed software system based on log data. Journal of Software, 2020, 31(7): 1997-2018(in Chinese)
(贾统, 李影, 吴中海. 基于日志数据的分布式软件系统故障诊断综述. 软件学报, 2020, 31(7): 1997-2018)

[5] Hall T, Beecham S, Bowes D, Gray D, Counsell S. A systematic literature review on fault prediction performance in software engineering. IEEE Transactions on Software Engineering, 2012, 38(6): 1276-1304

[6] Meiliana, Karim S, Warnars H, Gaol F L, Soewito B. Software metrics for fault prediction using machine learning approaches: A literature review with PROMISE repository dataset//Proceedings of the 2017 IEEE International Conference on Cybernetics and Computational Intelligence (CyberneticsCom). Phuket, Thailand, 2017: 19-23

[7] Agarwal P, Agrawal A P. Fault-localization techniques for software systems: A literature review. Acm Sigsoft Software Engineering Notes, 2014, 39(5): 1-8

[8] Li Zheng, Wu Yong-Hao, Wang Hai-Feng, Chen Xiang, Liu Yong. Survey of software multi defect localization methods. Chinese Journal of Computers. 2022, 45(02): 256-288(in Chinese)
(李征, 吴永豪, 王海峰, 陈翔, 刘勇. 软件多缺陷定位方法研究综述. 计算机学报, 2022, 45(02): 256-288)

[9] Natella, Roberto, Domenico Cotroneo, and Henrique S. Madeira. Assessing dependability with software fault injection: A survey. ACM Computing Surveys, 2016, 48(3): 1-55

[10] Wang L, Jiang Y X, Wang Z, et al. The operation and maintenance governance of microservices architecture systems: A systematic literature review. Journal of Software: Evolution and Process. 2022. e2433

[11] Ponce F, G Márquez, Astudillo H. Migrating from monolithic architecture to microservices: A rapid review//Proceedings of the International Conference of the Chilean Computer Science Society, Concepcion, Chile, 2019: 1-7

[12] Zhong Chen-Xing, Li Shan-Shan, Zhang-He, Zhang Cheng. Granularity evaluation of microservices from the perspective of bounded context. Journal of Software, 2019, 30(10): 3227-3241 (in Chinese)
(钟陈星, 李杉杉, 张贺, 章程. 限界上下文视角下的微服务粒度评估. 软件学报, 2019, 30(10): 3227-3241)

[13] Hassan S, Bahsoon R, Kazman R. Microservice transition and its granularity problem: A systematic mapping study. Software: Practice and Experience, 2020, 50(9): 1651-1681

[14] Vera-Rivera F H, Gaona C, Astudillo H. Defining and measuring microservice granularity—a literature overview. PeerJ Computer Science, 2021, 7: e695

[15] Munaf, Raja M, Khakwani F, Rana T. Microservices architecture: Challenges and proposed conceptual design//Proceedings of the 2019 International Conference on Communication Technologies (ComTech). 2019: 82-87

[16] Neri D, Soldani J, Zimmermann O, et al. Design principles, architectural smells and refactorings for microservices: a multivocal review. SICS Software-Intensive Cyber-Physical Systems, 2020, 35(1): 3-15

[17] Joseph C T, Chandrasekaran K. Straddling the crevasse: A review of microservice software architecture foundations and recent advancements. Software: Practice and Experience, 2019, 49(10): 1448-1484

[18] Valdivia J A, Lora-González A, Limón X, et al. Patterns related to microservice architecture: A multivocal literature review. Programming and Computer Software, 2020, 46(8): 594-608

[19] Cerny T, Donahoo M J, Trnka M. Contextual understanding of microservice architecture: Current and future directions. ACM SIGAPP Applied Computing Review, 2018, 17(4): 29-45

[20] Chen F, Zhang L, Lian X. A systematic gray literature review: The technologies and concerns of microservice application programming interfaces. Software: Practice and Experience, 2021, 51(7): 1483-1508

[21] Waseem M, Liang P, Shahin M, et al. On the nature of issues in five open source microservices systems: An empirical study//Proceedings of the Evaluation and Assessment in Software Engineering. Trondheim, Norway, 2021: 201-210

[22] Berardi D, Giallorenzo S, Mauro J, et al. Microservice security: A systematic literature review. PeerJ Computer Science, 2022, 7: e779

[23] Yin Kang-Lin, Du Qing-Feng. Microservice resilience risk identification and analysis based on chaos Engineering. Journal of Software, 2021, 32(05): 1231-1255(in Chinese)
(殷康璘, 杜庆峰. 基于混沌工程的微服务韧性风险识别和分析. 软件学报, 2020, 31(05): 1231-1255)

[24] Zhang Ce, Liu Hong-Wei, Bai Rui, at el. Review on the research of fault detection rate in reliability model. Journal of Software, 2020, 31(09): 2802-2825(in Chinese)
(张策, 刘宏伟, 白睿, 王瞰宇, 王金勇, 吕为工, 孟凡超. 可靠性模型中故障检测率研究述评. 软件学报, 2020, 31(09): 2802-2825)

[25] Zhang Ce, Meng Fan-Chao, Kao Yong-Gui, at el. Summary of software reliability growth model. Journal of Software, 2017, 28(09): 2402-2430(in Chinese)
(张策, 孟凡超, 考永贵, 吕为工, 刘宏伟, 万锟, 蒋家楠, 崔刚, 刘子和. 软件可靠性增长模型研究综述. 软件学报, 2017, 28(09): 2402-2430)

[26] Dragoni N, Giallorenzo S, Lafuente A L, et a. Microservices: Yesterday, today, and tomorrow. Present and ulterior software engineering springer. cham, 2017: 195-216

[27] Yang Ou, Zhang Yi, Di Zhen-Wei. Application practice of microservice architecture in container cloud. Computers and Telecommunications. 2017, (7): 79-81(in Chinese)
(杨鸥, 张羿, 耿贞伟. 微服务架构在容器云中的应用实践. 电脑与电信, 2017(7) : 79-81)

[28] Li Zhen-Hao. Analysis of the development and impact of microservice architecture. Information System Engineering. 2017, (1): 154-155(in Chinese)
(李贞昊. 微服务架构的发展与影响分析. 信息系统工程, 2017(1) : 154-155)

[29] Zhao Jian-Tao, Huang Li-Song. Research on related technologies of microservice fault diagnosis. Network New Media Technology. 2020, 9(01): 57-64(in Chinese)
(赵建涛, 黄立松. 微服务故障诊断相关技术研究探讨. 网络新媒体技术, 2020, 9(01): 57-64)

[30] Liu Ying. evolution of microservice architecture based on distributed system. Communication World. 2018, (7): 97-98(in Chinese)
(刘颖. 基于分布式系统的微服务架构演进. 通讯世界, 2018, (7) : 97-98)

[31] Jamshidi Pooyan, et al. Microservices: The journey so far and challenges ahead. IEEE Software, 2018, 35(3): 24-35

[32] Gill, Sukhpal Singh, BUYYA, Rajkumar. Failure management for reliable cloud computing: A taxonomy, model, and future directions. Computing in Science & Engineering, 2018, 22(3): 52-63

[33] Chandola V, Banerjee A, Kumar V. Anomaly detection: A survey. ACM Computing Surveys, 2009, 41(3): 75-79

[34] Aguilera Marcos Kawazoe, Chen Wei, Toueg Sam. Failure detection and consensus in the crash-recovery model. Distributed Computing, 2000, 13(2): 99-125

[35] Zhou Xiang. Microservice fault location based on trajectory analysis [Ph. D. Dissertation]. Fudan University: School of Computer Science and Technology, Shanghai, 2019(in Chinese)
(周翔. 基于轨迹分析的微服务故障定位[博士学位论文]. 复旦大学: 计算机科学技术学院, 上海, 2019)

[36] Zhou Xiang, et al. Latent error prediction and fault localization for microservice applications by learning from system trace logs//Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. New York, USA, 2019: 683-694

[37] Suk Tonghoon, et al. Failure-aware application placement modeling and optimization in high turnover DevOps environment//Proceedings of the 2019 IEEE 12th International Conference on Cloud Computing (CLOUD). Milan, Italy, 2019: 115-123

[38] Zhou Xiang, et al. Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study. IEEE Transactions on Software Engineering, 2018, 47(2): 243-260

[39] Jiang Ying, Zhang Na, Ren Zheng. Research on intelligent monitoring scheme for microservice application systems//Proceedings of the 2020 International Conference on Intelligent Transportation, Big Data & Smart City (ICITBS). Vientiane, Laos, 2020: 791-794

[40] Meng Yuan, et al. Localizing failure root causes in a microservice through causality inference//Proceedings of the 2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS). Hangzhou, China, 2020: 1-10

[41] Mace J, Roelke R, Fonseca R. Pivot tracing: Dynamic causal monitoring for distributed systems. ACM Transactions on Computer Systems (TOCS), 2018, 35(4): 11

[42] Christina Heinze-deml, Marloes H Maathuis, Nicolai Meinshausen. Causal structure learning. Annual Review of Statistics and Its Application, 2018, 5: 371-391

[43] Qi Guangyang, Yao Lina, Uzunov Anton V. Fault detection and

localization in distributed systems using recurrent convolutional neural networks//Proceedings of the International Conference on Advanced Data Mining and Applications. Singapore, 2017: 33-48

[44] Petersen Kai, Vakkalanka Sairam, Kuzniarz Ludwik. Guidelines for conducting systematic mapping studies in software engineering: An update. Information and Software Technology, 2015, 64: 1-18

[45] Roehrs A, Da Costa C A, da Rosa Righi R, et al. Personal health records: A systematic literature review. Journal of medical Internet research, 2017, 19(1): e13

[46] I Kadi, A Idri, J L. Fernandez-Aleman. Knowledge discovery in cardiology: A systematic literature review. International Journal of Medical Informatics, 2017, 97: 12-32

[47] B Kitchenham, P Brereton. A systematic review of systematic review process research in software engineering. Information and software technology, 2013, 55. 12: 2049-2075

[48] S Y Chadli, A Idri. Identifying and mitigating risks of software project management in global software development//Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement. New York, USA, 2017: 12-22

[49] Barroso Luiz André, Hölzle Urs. The datacenter as a computer: An introduction to the design of warehouse-scale machines. Synthesis Lectures on Computer Architecture, 2009, 4(1): 1-108

[50] Liu Dong-Hong, Guo Chang-Guo, Wang Huai-Min. Monitoring-enabled distributed software system construction method. Journal of Software, 2011, 22(11): 2610-2624 (in Chinese)
(刘东红, 郭长国, 王怀民, 等. 监控使能的分布式软件系统构造方法. 软件学报, 2011, 22(11): 2610-2624)

[51] Wang Ping, et al. Cloudranger: Root cause identification for cloud native systems//Proceedings of the 2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID). Washington, USA, 2018: 492-502

[52] Marshall J, Kotonya G. A Runtime Visualizer For Microservices// Proceedings of the 2021 IEEE International Conference on Service-Oriented System Engineering (SOSE). UK, online, 2021: 72-80

[53] Cinque Marcello, et al. Characterizing direct monitoring techniques in software systems. IEEE Transactions on Reliability, 2016, 65(4): 1665-1681

[54] You Yong, Wang Hao, Gu Sheng-Hui, Sun Jia-Lin. A kind of link tracking log data storage design of monitoring system. Journal of Software, 2021, 32(05): 1302-1321 (in Chinese)
(尤勇, 汪浩, 任天, 顾胜晖, 孙佳林. 一种监控系统的链路跟踪型日志数据的存储设计. 软件学报, 2020, 31(05): 1302-1321)

[55] Sigelman Benjamin H, et al. Dapper, a large-scale distributed systems tracing infrastructure. 2010

[56] Cha D, Kim Y. Service mesh based distributed tracing system// Proceedings of the 2021 International Conference on Information and Communication Technology Convergence (ICTC). Jeju Island, Korea, 2021: 1464-1466

[57] Pina Fábio, et al. Nonintrusive monitoring of microservice- based systems//Proceedings of the 2018 IEEE 17th International Symposium on Network Computing and Applications (NCA). Cambridge, USA, 2018: 1-8

[58] Cinque Marcello, Della Corte Raffaele, Pecchia Antonio. Microservices monitoring with event logs and black box execution tracing. IEEE Transactions on Services Computing, 2019, 15(1): 294-307

[59] Fadda Edoardo, Plebani Pierluigi, Vitali Monica. Optimizing monitorability of multi-cloud applications//Proceedings of the International Conference on Advanced Information Systems Engineering. Ljubljana, Slovenia, 2016: 411-426

[60] Yi Su, Yin Hui-Wen, Wang Chuang, Zhang Yi-Chuan. Research on multi-factor adaptive heartbeat detection algorithm. Computer Engineering and Applications. 2017, 53(24): 86-93+128(in Chinese)
(易俗, 殷慧文, 王闯, 张一川. 多因素自适应心跳检测算法研究. 计算机工程与应用, 2017, 53(24): 86-93+128)

[61] Thalheim Jörg, et al. Sieve: Actionable insights from monitored metrics in microservices. arXiv preprint arXiv: 1709. 06686, 2017

[62] Wang H, Zhang X, Ma Z, et al. An microservices-based openstack monitoring system//Proceedings of the 2022 11th International Conference on Educational and Information Technology (ICEIT). Chengdu, China, 2022: 232-236

[63] Bădică A, Bădică C, Bolanowski M, et al. Cascaded anomaly detection with coarse sampling in distributed systems//Proceedings of the International Conference on Big Data Analytics. Springer, Cham, 2021: 181-200

[64] Cinque Marcello, Della Corte Raffaele. Pecchia Antonio. Advancing monitoring in microservices systems//Proceedings of the 2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW). Berlin, Germany, 2019: 122-123

[65] Sholihah W, Pripambudi S, Mardiyono A. Log event management server menggunakan elastic search logstash kibana (elk stack). JTIM: Jurnal Teknologi Informasi dan Multimedia, 2020, 2(1): 12-20

[66] Yuan Ding, Park Soyeon, Zhou Yuanyuan. Characterizing logging practices in open-source software//Proceedings of the 2012 34th International Conference on Software Engineering (ICSE). Zurich, Switzerland, 2012: 102-112

[67] Bertero Christophe, et al. Experience report: Log mining using natural language processing and application to anomaly detection//Proceedings of the 2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE). Toulouse, France, 2017: 351-360

[68] M Cinque, R Della Corte, A Pecchia. Entropy-based security analytics: Measurements from a critical information system//Proceedings of the 2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). Denver, USA, 2017: 379-390

[69] He Shilin, et al. Experience report: System log analysis for anomaly detection//Proceedings of the 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE). Ottawa, Canada, 2016: 207-218

[70] Li Wen-Hai, Peng Xin, Xiang Qi-Lin, Guo Xiao-Feng, Zhou Xiang, Zhao Wen-Yun. Micro-service system debugging method based on log visual analysis. Computers Science, 2019, 46(11): 145-155 (in Chinese)
(李文海, 彭鑫, 丁丹, 向麒麟, 郭晓峰, 周翔, 赵文耘. 基于日志可视化分析的微服务系统调试方法. 计算机科学, 2019, 46(11): 145-155)

[71] Pham Cuong, et al. Failure diagnosis for distributed systems using targeted fault injection. IEEE Transactions on Parallel and Distributed Systems, 2016, 28(2): 503-516

[72] Uhle Johan, Tröger Peter. On dependability modeling in a deployed microservice architecture. Operating Systems and Middleware Group[M. S. Thesis], Universität Potsdam, Germany, 2014

[73] Wang Zi-Yong, Wang Tao, Zhang Wen-Bo, Chen Ning-Jiang, Zuo Chun. A microservice fault diagnosis method based on execution track monitoring. Journal of Software, 2017, 28(06): 1435-1454 (in Chinese)
(王子勇, 王焘, 张文博, 陈宁江, 左春. 一种基于执行轨迹监测的微服务故障诊断方法. 软件学报, 2017, 28(06): 1435-1454)

[74] Ma Shang-Pin, et al. Using service dependency graph to analyze and test microservices//Proceedings of the 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC). Tokyo, Japan, 2018: 81-86

[75] Yu Qing-Yang, Bai Xiao-Ying, Li Ming-Jie, Li Qi-Yuan, Liu Tao, Liu Ze-Yin, Pei Dan. Performance modeling and anomaly location of large microservice systems based on trace control flow analysis. Journal of Software. 2022. 33(5): 1849-1864(in Chinese)
(于庆洋, 白晓颖, 李明杰, 李奇原, 刘涛, 刘泽胤, 裴丹. 基于调用链控制流分析的大型微服务系统性能建模与异常定位. 软件学报, 2021, 32(5): 1849-1864)

[76] Tang Y, Chen N, Zhang Y, et al. Fine-grained diagnosis method for microservice faults based on hierarchical correlation analysis//Proceedings of CCF Conference on Computer Supported Cooperative Work and Social Computing. Springer, Singapore, 2022: 14-28

[77] Chen Pengfei, et al. Causeinfer: Automatic and distributed performance diagnosis with hierarchical causality graph in large distributed systems//Proceedings of the IEEE INFOCOM 2014-IEEE Conference on Computer Communications. Toronto, Canada, 2014: 1887-1895

[78] Aggarwal Pooja, et al. Localization of operational faults in cloud applications by mining causal dependencies in logs using golden signals//Proceedings of International Conference on Service-Oriented Computing. Springer, Cham, 2020, 137-149

[79] Ma M, Xu J, Wang Y, et al. Automap: Diagnose your microservice-based web applications automatically//Proceedings of The Web Conference 2020. Taipei, China. 2020: 246-258

[80] Horovitz S, Boren B, Wizen B. Sequencis-distributed application fault characteristics discovery using service logs//Proceedings of the 2021 6th International Conference on Big Data and Computing. Shenzhen, China, 2021: 109-115

[81] Sun Y, Zhao L, Wang Z, et al. Fault root rank algorithm based on random walk mechanism in fault knowledge graph//Proceedings of the 2021 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB). Chengdu, China, 2021: 1-6

[82] Kim Myunghwan, Sumbaly Roshan, Shah Sam. Root cause detection in a service-oriented architecture. ACM SIGMETRICS Performance Evaluation Review, 2013, 41(1): 93-104

[83] Meng Lun, et al. Detecting anomalies in microservices with execution trace comparison. Future Generation Computer Systems, 2021, 116: 291-301

[84] Brandón Á, Solé M, Huélamo A, et al. Graph-based root cause analysis for service-oriented and microservice architectures. Journal of Systems and Software, 2020, 159: 110432

[85] Li M, Tang D, Wen Z, et al. Microservice anomaly detection based on tracing data using semi-supervised learning//Proceedings of the 2021 4th International Conference on Artificial Intelligence and Big Data (ICAIBD). Chengdu, China, 2021: 38-44

[86] Mariani Leonardo, et al. Predicting failures in multi-tier distributed systems. Journal of Systems and Software, 2020, 161: 110464

[87] Vachtsevanos G, Wang P. Fault prognosis using dynamic wavelet neural networks//Proceedings of the 2001 IEEE Autotestcon Proceedings. IEEE Systems Readiness Technology Conference. (Cat. no. 01ch37237). Valley Forge, USA, 2001: 857-870

[88] Yin Kang-Lin, Du Qing-Feng. Identification and analysis of microservice resilience risk based on chaos engineering. Journal of Software, 2021, 32(05): 1231-1255 (in Chinese)
(殷康璘, 杜庆峰. 基于混沌工程的微服务韧性风险识别和分析. 软件学报, 2020, 31(05): 1231-1255)

[89] Lin Qingwei, et al. Predicting node failure in cloud service systems//Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Lake Buena Vista FL, USA, 2018: 480-490

[90] Ji Weiliang, et al. A CNN-based network failure prediction method with logs//Proceedings of the 2018 Chinese Control And Decision Conference (CCDC). Shenyang, China, 2018: 4087-4090

[91] Wang Xiao-Feng, Wang Yi-Da, Sun Yu-Qiang, Song Xi-Tang, Wang Xin-Dong, Yu Bin. Research and application of distributed service failure prediction model based on XGBoost algorithm. Telecom Technology. 2019, (10): 13-16(in Chinese)
(王晓峰, 王一大, 孙玉强, 宋希堂, 王新东, 于滨. 基于XGBoost 算法的分布式服务故障预测模型研究与应用. 电信技术, 2019, (10): 13-16)

[92] Suresh Yeresime, Kumar Lov, Rath Santanu Ku. Statistical and machine learning methods for software fault prediction using CK metric suite: A comparative analysis. International Scholarly Research Notices, 2014: 1-15

[93] Pitakrat Teerat, et al. Hora: Architecture-aware online failure prediction. Journal of Systems and Software, 2018, 137: 669-685

[94] Abbas R, Sultan Z, Bhatti S N. Comparative analysis of automated load testing tools: Apache jmeter, microsoft visual studio (tfs), loadrunner, siege//Proceedings of the 2017 International Conference on Communication Technologies (Comtech). Chengdu, China, 2017: 39-44

[95] Joshi Pallavi, et al. SETSUDŌ: Perturbation-based testing framework for scalable distributed systems//Proceedings of the First ACM SIGOPS Conference on Timely Results in Operating Systems. PA, USA. 2013: 1-14

[96] Heorhiadi Victor, et al. Gremlin: Systematic resilience testing of microservices//Proceedings of the 2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS). Nara, Japan, 2016: 57-66

[97] Gunawi Haryadi S, et al. FATE and DESTINI: A framework for cloud recovery testing//Proceedings of the NSDI'11: 8th USENIX Symposium on Networked Systems Design and Implementation. San Jose, USA, 2011: 239

[98] L Juszczyk, S Dustdar. Programmable fault injection testbeds for complex SOA//Proceedings of the International Conference on Service-Oriented Computing. Berlin, Germany: Springer, 2010: 411-425

[99] Meiklejohn C S, Estrada A, Song Y, et al. Service-level fault injection testing//Proceedings of the ACM Symposium on Cloud Computing. Santa Cruz, USA, 2021: 388-402

[100] P Alvaro, K Andrus, C Sanden, C Rosenthal, A Basiri, L Hochstein. Programmable fault injection testbeds for complex SOA// Proceedings of the International Conference on Service- Oriented Computing. Berlin, Germany: Springer, 2010: 411-425

[101] Zhou Xiang, et al. Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study. IEEE Transactions on Software Engineering, 2018, 47(2): 243-260

[102] Wang Zhi-Wen, Huang Xiao-Long, Wang Hai-Jun, Liu Ting, Yu Le-Chen. Research and implementation of test case generation system based on program slicing. Computer Science. 2014, 41(09): 71-74 (in Chinese)
(王志文, 黄小龙, 王海军, 刘烃, 俞乐晨. 基于程序切片的测试用例生成系统研究与实现. 计算机科学, 2014, 41(09): 71-74)

[103] Zeller Andreas. Yesterday, my program worked. Today, it does not. Why?. ACM SIGSOFT Software Engineering Notes, 1999, 24(6): 253-267

[104] Burger Martin, Zeller Andreas. Minimizing reproduction of software failures//Proceedings of the 2011 International Symposium on Software Testing and Analysis. Toronto, Canada, 2011: 221-231

[105] Hodován Renáta, Kiss Ákos, Gyimóthy Tibor. Coarse hierarchical delta debugging//Proceedings of the 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME). Shanghai, China, 2017: 194-203

[106] Zhou Xiang, et al. Delta debugging microservice systems with parallel optimization. IEEE Transactions on Services Computing, 2019, (1): 1

[107] Yuan Ding, et al. Improving software diagnosability via log enhancement. ACM Transactions on Computer Systems (TOCS), 2012, 30(1): 1-28

[108] He Xiao, Guo Ya-Qi, Zhang Zhao, Jia Fan-Lin, Zhou Dong-Hua. Active fault diagnosis technology for dynamic systems. Chinese Journal of Automation. 2020, 46(08): 1557-1570(in Chinese)
(何潇, 郭亚琦, 张召, 贾繁林, 周东华. 动态系统的主动故障诊断技术. 自动化学报, 2020, 46(08): 1557-1570)

[109] Long Zhenyue, et al. Fitness-guided resilience testing of microservice-based applications//Proceedings of the 2020 IEEE International Conference on Web Services (ICWS). Beijing, China, 2020: 151-158

[110] Alvaro Peter, Rosen Joshua, Hellerstein Joseph M. Lineage-driven fault injection//Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. Victoria, Melbourne, 2015: 331-346

[111] Gan Y, Zhang Y, Cheng D, et al. Unveiling the hardware and software implications of microservices in cloud and edge systems. IEEE Micro, 2020, 40(3): 10-19

[112] Rasheedh J A, Saradha S. Reactive microservices architecture using a framework of fault tolerance mechanisms//Proceedings of the 2021 Second International Conference on Electronics and Sustainable Communication Systems (ICESC). 2021: 146-150

[113] B Butzin, F Golatowski, D Timmermann, Microservices approach for the internet of things//Proceedings of the 2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA). Berlin, Germany, 2016: 1-6

[114] Pahl Marc-Oliver, Lorenzo Donini. Securing IoT microservices with certificates//Proceedings of the NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium. Taipei, China, 2018: 1-5

[115] F Al-Doghman, N Moustafa, I Khalil, Z Tari, A Zomaya. AI-enabled Secure Microservices in Edge Computing: Opportunities and challenges. IEEE Transactions on Services Computing, to appear

[116] S Wang, Y Guo, N Zhang, P Yang, A Zhou, X Shen. Delay-aware microservice coordination in mobile edge computing: A reinforcement learning approach. IEEE Transactions on Mobile Computing, 2019, 20(3): 939-951

[117] P Di Francesco, P Lago, I Malavolta. Migrating towards microservice architectures: an industrial survey//Proceedings of 2018 IEEE International Conference on Software Architectur. Seattle, USA, 2018: 29-2909

[118] J Kazanavičius, D Mažeika, Migrating legacy software to mi-

croservices architecture//Proceedings of 2019 Open Conference of Electrical, Electronic and Information Sciences (eStream). 2019: 1-5

[119] P Jamshidi, C Pahl, N C Mendonça, J Lewis, S Tilkov. Microservices: The journey so far and challenges ahead. IEEE Software, 2018, 35(3): 24-35

[120] Gulenko A, Acker A, Kao O, et al. Ai-governance and levels of automation for aiops-supported system administration//Proceedings of the 2020 29th International Conference on Computer Communications and Networks (ICCCN). Honolulu, USA, 2020: 1-6

[121] Dang, Yingnong, Qingwei Lin, Peng Huang. AIOps: real-world challenges and research innovations//Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion). Montreal, Canada, 2019: 4-5

[122] Khoso F H, Lakhan A, Arain A A, Soomro M A, Nizamani S Z, Kanwar K. A microservice-based system for industrial internet of things in fog-cloud assisted network. Engineering, Technology & Applied Science Research 11. 2 (2021): 7029-7032

WANG Lu, Ph. D., associate professor. Her current research interests include microservice architecture and container, operation and maintenance of intelligent software in AIOps, etc.

JIANG Yu-Xuan, master. His current research interest is microservice and fault detection.

LI Qing-Shan, Ph. D., professor. His current research interests include agent-oriented software engineering, domain-specific software architecture, etc.

HUO Qi-En, master. His current research interest focus on microservice and docker.

WANG Zhan, master. His current research interest focus on microservice and load balancing.

XIE Sheng-Long, Ph.D. candidate. His current research interests include Software self-optimization, Swarm intelligence and Multi-agent technology, etc.

DAI Jie, Ph.D. His research interests include recommender system, graph neural network, machine learning.

Background

With the development of information technology, the traditional monolithic architecture and distributed architecture can no longer be adapted to the requirements of frequent changes in large-scale and complex software requirements, durable operation, and rapid deployment. Therefore, the microservice architecture has emerged. However, with the widespread application of microservice architecture in the industry, some software faults specific to microservice software cannot be detected and checked by traditional fault detection techniques. However, at present, domestic and foreign experts and scholars have little research on microservice fault detection, and most of them are improved on the basis of traditional software fault detection methods, which cannot fully meet the requirements of the complexity and heterogeneity of microservice software faults. At the same time, there are relatively few summaries for microservice failure detection, and there is a lack of a complete microservice failure detection framework.

Therefore, this paper summarizes and analyzes the existing fault detection methods of traditional software and microservice software. First, it explains the current problems and challenges from the four aspects of software monitoring, fault diagnosis, fault prediction, fault reproduction and testing. Then, a complete fault detection framework is established on this basis. Finally, the future development trends in various research directions are analyzed. It aims to help relevant researchers and practitioners sort out the development context and latest research progress of microservice fault detection, so as to effectively promote the application and development of microservice architecture.

This work was supported by the National Natural Science Foundation of China under grant No. 61902288, the National Natural Science Foundation of China under grant No. 61972300, and the Shaanxi Provincial Natural Science Foundation of China under grant No. 2020JQ-300.