计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2020.20190624
57(3): 525－541, 2020

面向微服务软件开发方法研究进展

吴化尧　　邓文俊
(计算机软件新技术国家重点实验室(南京大学)　南京　210023)
(hywu@nju.edu.cn)

Research Progress on the Development of Microservices

Wu Huayao and Deng Wenjun
(*State Key Laboratory for Novel Software Technology* (*Nanjing University*), *Nanjing* 210023)

Abstract　Microservices are the latest, and probably the most popular, technology to realize the well-known service-oriented architecture (SOA). They have been widely applied in many important industrial applications, and have also attracted increasing attentions in academia. In order to aid the effective development of high quality microservices, in this study, we present a systematic review of the microservices literature, focusing on the various software engineering activities in the development of microservices. Specifically, we collect and analyze the currently available methods, tools and practices for the requirements analysis, design and implementation, testing, and refactoring for Microservices. We also discuss the issues and opportunities in future researches of this field.

Key words　microservice; service-oriented architecture; software development; system design; refactor

摘　要　微服务是面向服务体系结构的最新发展趋势和研究热点,其不仅在工业实践中形成了广泛且重要的应用,在学术界也受到日益增长的关注.以软件工程生命周期中的各项活动为主线,系统全面地对当前的微服务软件开发方法进行梳理和总结,尤其分析了面向微服务软件开发在需求分析、设计与实现、测试以及重构上的已有方法、工具和实践,并讨论了该领域的未来研究方向,从而为更加科学有效地开发高质量微服务提供参考和借鉴.

关键词　微服务;面向服务架构;软件开发;系统设计;重构

中图法分类号　TP311

为了满足企业动态多变的业务需求、提高开发效率和业务扩展能力,软件工程的从业者在单体架构的基础上提出了面向服务的软件开发方法.这一新方法使用相互独立的服务作为构建应用程序的基本单元,可以在不影响系统运行的情况下对服务进行增删和修改,从而实现软件产品的快速构建和动态调整.此外,每个服务都可以选用最合适的技术体系进行独立开发,有助于提升软件开发和维护的效率.

微服务是面向服务软件开发的最新发展趋势,其通常采用去中心化的服务管理方式,在传统面向服务开发模式的基础上进一步降低了系统的耦合度.微服务还充分借鉴了云计算、容器技术以及 DevOps 等新的实践方式,提高了每个服务的可伸缩性,能

收稿日期:2019-08-30;修回日期:2019-12-13
基金项目:国家重点研发计划项目(2018YFB1003800);国家自然科学基金项目(61902174);江苏省自然科学基金项目(BK20190291)
This work was supported by the National Key Research and Development Program of China (2018YFB1003800), the National Natural Science Foundation of China (61902174), and the Natural Science Foundation of Jiangsu Province of China (BK20190291).

实现服务的快速部署和更改[1].在工业界,面向微服务的软件开发方法已得到了许多成功的应用,例如腾讯的微信①和优步②等都采用了基于微服务的架构.

目前,已有研究工作对面向服务软件开发方法进行了总结[2-4],但在遵循这些已有的理论和方法之外,微服务本身的一些新特性也要求人们设计更加契合的软件开发方法.本文因此尝试对面向微服务软件开发方法进行全面系统的总结,以帮助研究者和从业者了解该领域的最新研究进展.具体地,我们首先使用系统文献调研方法收集当前与面向微服务软件开发相关的研究论文;随后,分别在需求分析、设计与实现、测试以及重构这4个软件工程生命周期的关键活动上总结已有的方法、工具和实践.在此基础上,讨论面向微服务软件开发未来的研究方向.

1 背 景

软件架构定义了应用程序的组件结构及其之间的相互关系[5].随着客户需求的不断演化,应用程序通常会变得庞大且复杂,这就需要在软件开发时定义并选择最合适的架构方式来尽可能地以最小成本满足客户的各种功能和非功能性需求[6].

1.1 面向服务架构

单体架构将整个应用程序部署为一个统一的解决方案,其中各个组成模块无法独立运行[7].这种架构的优势在于其易于开发、测试和部署,但是随着应用程序规模的增长,单体应用程序会变得难以理解和维护,对其中任何组件的更新都必须重新测试和部署整个应用程序.此外,单体架构还会产生技术锁定,即所有组件都必须遵循同样的框架和技术体系[8].

为了克服单体架构的潜在缺陷,满足应用程序对于可扩展性、持续开发和技术选择自由等需求,面向服务架构(service oriented architecture, SOA)应运而生.SOA由Gartner公司在1996年首次提出[9],它将整个应用程序分解为若干个相互独立、自包含、可重用的服务,使得整个应用程序具有动态、松耦合和分布式的特性.其中,每个服务是一个实现特定业务能力的与平台无关实体,通过良好定义的标准化接口进行通信,开发者可以对服务进行描述、发布、发现和动态组合[10-11].

SOA的实现需要3种角色的参与,即服务提供者、服务消费者和服务注册中心[9].服务提供者提供服务并将服务注册到服务注册中心,服务消费者通过服务注册中心获取和使用服务,而服务注册中心作为中间平台联通服务提供者和消费者.与SOA相关的协议和标准包括描述3种参与者之间消息传输格式和方式的简单对象访问协议(simple object access protocol, SOAP[12]);描述服务功能、接口和位置等信息的Web服务描述语言(Web services description language, WSDL);以及对服务进行检索的统一描述、发现和集成协议(universal description, discovery, and integration, UDDI)等.

1.2 微服务架构

微服务架构可以视为面向服务软件架构的一个特定子类型.有研究者认为微服务一词由Lewis和Fowler在2014年首次提出[13-14],不过据上述2人所述,这一概念至少在2012年之前就已存在[15].Lewis和Fowler将微服务架构定义为通过一套小型服务(即微服务)的集合来构造单个应用程序,其中每个微服务都在自己的进程中运行,并以轻量级机制(例如HTTP)进行通信.

微服务架构在软件开发上主要有2种应用模式,其中一种是从需求分析出发,从无到有地开发一个新的微服务应用程序,本文第3~5节将分别从需求分析、设计与实现、测试这3个角度来梳理和总结相关研究的当前进展.另一种应用是将已有系统(通常是单体应用程序)重构到微服务架构,本文第6节将讨论这种应用的当前研究进展.

1.3 微服务架构与SOA的主要区别

微服务可以直观地理解为细粒度的SOA,但两者的差异并不局限在粒度上.Lewis和Fowler[15]在给出微服务定义的同时,也总结了微服务架构的九大特性,即服务组件化、按业务组织团队、做产品而不是项目、轻量级通信、去中心化治理、去中心化数据管理、基础设施自动化、容错设计以及演进式设计.上述特性通常被视为面向微服务开发的指导原则,从中我们可以发现微服务有别于SOA的一些特点:

1) 微服务的去中心化管理与SOA的集中式管理形成鲜明对比.每个微服务都是一个独立的应用

① https://cloud.tencent.com/developer/article/1346997

② https://cloud.tencent.com/developer/article/1346869

程序,具有独立的数据库和运行环境,这一自治性使得微服务能够独立部署和运行.而 SOA 通常采用统一的数据中心,并依赖于企业服务总线或其他同类重量级中间件等产品[16].

2) 相比于 SOA,微服务架构更加重视高可用性、可伸缩性、负载均衡、故障转移等特性.微服务运行在高可用的分布式环境当中,加上配套的监控和容错管理机制,系统更加稳定可靠.

3) 相比于 SOA,微服务的细粒度特性在提高开发效率、增强对需求变化的响应的同时也使得微服务数目成倍增长,微服务架构尤其需要相应的自动化基础设施来实现自动集成、测试和部署.

2 文献收集和汇总

我们采用系统文献调研(systematic literature review, SLR)方法[17]来进行文献收集,以确保系统全面地覆盖所有与微服务软件开发相关的研究论文.

我们利用 ACM Digital Library,IEEE Xplore,ScienceDirect,Springer Link 和 DBLP 这 5 个英文数据库以及中国知网中文数据库对 2019 年 6 月之前发表的微服务开发相关文章进行检索,表 1 给出了文献检索使用的中、英文关键词:

Table 1 Keywords of Literature Search

表 1 论文检索关键词

Type	Keywords
English Keywords	("microservice" OR "micro service") AND "software engineering" ("microservice" OR "micro service") AND "requirement" ("microservice" OR "micro service") AND "design" ("microservice" OR "micro service") AND "development" ("microservice" OR "micro service") AND "test" ("microservice" OR "micro service") AND ("refactor" OR "migrate" OR "migrating" OR "migration")
Chinese Keywords	微服务软件工程 微服务需求 微服务测试 微服务开发 微服务设计 微服务重构 微服务迁移

对于检索到的文献,我们首先阅读标题、摘要和结论,并应用下述包含和排除标准来进行第 1 轮筛选.具体地,我们将符合全部 3 个条件的文献考虑在内(包含标准):

1) 关注微服务软件开发周期(需求分析、设计与实现、测试、重构)的一个或多个阶段.

2) 发表形式为会议、期刊或书籍.

3) 发表日期为 2014 年至 2019 年 6 月.

并将符合 2 个条件中的任一条件的文献排除在外(排除标准):

1) 仅将微服务软件开发作为例子来讨论.

2) 以中文和英文之外的语言发表.

第 1 轮筛选完成之后我们共得到 103 篇文献.然后,应用滚雪球方法对上述 103 篇文献进行进一步扩充,即对文献的所有参考文献都应用上述标准进行筛选,补充新的相关文献.重复这一步骤直到不再加入新的文献为止,这一步骤结束后我们获得的文献总数为 134 篇.最后,我们对上述 134 篇文献进行第 2 轮筛选,最终获得 91 篇文献作为本文调研的对象.图 1 给出了这些文献在软件工程生命周期各项活动上的分布,其中设计与实现是当前研究最多的子领域.

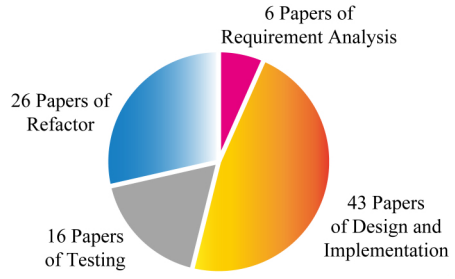

Fig. 1 Distribution of microservice research papers on software engineering activities

图 1 研究论文在软件工程问题上的分布

3 面向微服务的软件需求分析

软件需求包括功能需求和非功能需求,其中功能需求是软件产品要实现的功能,用户利用这些功

能来完成特定的任务;非功能需求则刻画了软件在应用时的相关约束和特性[18],是软件系统的重要质量属性[19].

表2给出了与微服务需求分析相关的6篇论文中所涉及的需求分析对象以及考虑的功能需求和非功能需求.我们发现在功能需求的获取上,半数研究直接引用已有需求[20-21]或者分析类似应用的需求[22],而未涉及需求分析相关方法和流程的使用.我们同时也发现,由于微服务架构的引入对应用程序的非功能属性带来新的挑战和要求,使得现有研究愈发重视应用程序的非功能需求.在已有的6篇涉及需求分析的研究中,半数未考虑非功能需求[22-23]或者考虑了但没有给出具体说明[21],其余的研究均考虑且直接给出对应非功能需求分析结果.

Table 2 Current Researches on Requirement Analysis of Microservices

表2 微服务需求分析的已有研究

Reference	Object of Requirement Analysis	Sources of Functional Requirements	Non-Functional Requirements Considered
Ref[20]	Migrate the Electronic Seat Reservation System to Microservices	Requirements of Existing Systems	Consistency, Fault Tolerance, Scalability, Efficiency, Load Balance, Portability, etc.
Ref[21]	Online Movie Streaming System	Requirements Database	From the Requirements Database
Ref[22]	Public Complaint Systems	Design Documents and Literature for Similar Applications	
Ref[23]	Supervisory Control and Data Acquisition System		
Ref[24]	Remote Care Applications	Participatory Design Methodology	Scalability, Security, Usability
Ref[25]	Solution for Connected Car Domain	Requirements of Existing Systems	Scalability, Reusability, Continuous Deployment, Code Quality, etc.

具体地,Richter等人[20]曾将电子座位预订系统迁移到微服务,他们在一致性上考虑了避免为不同顾客提供相同的座位和多次预订同一座位,在容错上考虑了容忍多个服务实例、虚拟机或基础设施组件的故障并自动从故障中恢复,在可扩展性上考虑了服务、虚拟机和服务器的水平可伸缩性以及应用程序的较高负荷上限等非功能性需求.Wizenty等人[24]开发了一个远程护理应用程序,他们在可扩展性方面要求能够灵活地集成和提供新的功能以及能够同时处理上千个应用实例,在安全性方面考虑了对用户个人数据的保护,同时在高可用性的基础上增加弹性、健壮性和功能独立性.Schneider等人[25]则研究了互联汽车解决方案,在可扩展性上要求能够以有效的方式对使用频繁或者资源紧张的组件进行复制,在重用上要求划分的服务将能在多个项目中使用,在持续部署上要求添加到服务中的特性必须立即可见以便能够自动测试与其他服务的兼容性.

4 面向微服务的软件设计与实现

面向微服务的软件设计与实现是微服务开发研究中的重点.目前,在面向微服务的设计上已形成了一些架构设计模式,如使用API网关来管理外部请求和进行服务组合、每个服务配备一个数据库来进行数据存储等[26],但相关研究尚不成熟.与之相比,学术界关于SOA架构设计模式已有很多研究,为了验证这些已有的设计模式能否用于面向微服务软件的设计,Bogner等人[27]对3本SOA著作[28-30]中提及的118种设计模式进行了定性分析.他们发现分别有63%,25%和12%的模式完全适用、部分适用和不适用于微服务,可见SOA与微服务在设计上具有许多共性.

但是,与SOA相比,微服务本身的特点也为软件设计与实现带来了诸多新的挑战.例如微服务的最优划分粒度难以把握,过细的粒度会导致微服务个数成倍增加,这大大增加了微服务之间相互调用和通信的复杂度.此外,每个微服务使用一个独立的数据库或者数据表,保证共享数据的一致性需要更多的操作和成本,而微服务所处的分布式环境也会对网络延迟、分布式事务以及安全性等提出新的要求.

在本节中,我们参考Haselböck等人的一次访谈研究[31],将微服务系统的设计空间划分为2类主要的设计领域:系统设计以及组织结构和流程.表3给出了这2类设计领域的详细分类,以及每个子设计领域需要考虑的研究问题以及对应的文献统计.值得一提的是,Haselböck等人[32-33]还对微服务接口、服务发现、容错、负载均衡4个子设计领域进行了研究,提出了相应的设计决策模型.

Table 3 Design Domain of Microservice Software

表 3 微服务软件设计领域

Design Areas	Sub-design Areas	Research Questions	Reference
System Design	User Interface	How to design interfaces for service communication	Ref[34-37]
	Data Management	How to limit Microservices' access to other services and ensure data consistency	Ref[38-39]
	Modularization	How to divide Microservices	Ref[33,40-45]
	Service Discovery	How to register an existing service in the service registry	Ref[33,46-49]
	Service Composition	How to combine existing services into a more complete application	Ref[50-53]
	Fault Tolerance	How to detect and recover from failures while the service is running	Ref[33,49]
	Security	How to ensure the security of external access to the system and inter-invocation between services within the system	Ref[24,35-37,54-56]
	Load Balance	How to achieve load balance between instances of the same service	Ref[33,57-58]
	Monitoring & Logging	How to design data and log monitoring indicators, and their collection and analysis strategies	Ref[59-62]
Organizational Structure and Processes	Organization	How to organize the Microservices development team	Ref[63]
	Deployment	How to deploy Microservices	Ref[24,64-72]
	DevOps	How to use the DevOps practice in Microservices development	Ref[73-74]

4.1 系统设计

系统设计关注微服务架构的设计,包括接口、数据管理、模块化、服务发现、服务组合、容错、安全、负载均衡、监控和日志等的设计.

1) 接口

微服务接口设计需要考虑如何设计微服务之间进行通信的接口,包括采用何种通信机制、提供什么样的接口形式等.

就通信机制而言,微服务内部通信具有交互频繁、传输数据量不确定、接口数量多等特点,在实现上较为复杂.而微服务之间的通信主要有同步请求/响应模式和异步消息通信模式 2 种方式:

① 同步请求/响应模式无需中间件,具有较好的通用性,但是该模式会在一定程度上降低了系统可用性[75].目前主要实现有基于 HTTP 协议的 Restful API,基于 RPC 和序列化支持多种消息格式的 Thrift,以及二进制格式的 Protocol Buffer 和 Avro 等,其中 Restful API 是当前研究的主流[34-37].

② 异步消息通信模式无需等待请求响应,并且支持发布/订阅、发布/异步响应和请求/异步响应等多种通信机制,能够提高系统的可用性.目前主要实现有 AMQP 的 RabbitMQ 以及 Apache 的 Kafka.

此外,微服务接口设计还需要接口管理规范以表明接口如何被调用以及对关系复杂的接口进行监控和维护.Swagger,API Blueprint 和 RAML 是目前 3 种主流的 Restful API 接口管理规范.

2) 数据管理

数据管理旨在对服务完成特定功能所需的数据进行管理,其关注如何设计微服务的数据库、限制微服务对其他服务数据库的访问能力、保证共享数据在全局上的一致性等.

由于微服务之间采用的数据库类型可能不同,要保证共享数据在全局上的一致性必须实现跨异构数据库的数据复制.针对这一问题,Viennot 等人[38]提出了一个用于集成大规模、数据驱动服务的 Synapse 框架,它支持多种 SQL 和 NoSQL 数据库之间的数据复制,使得依赖于它的服务能以相互隔离、可伸缩的方式共享数据,确保不同服务中数据的一致性.

此外,微服务架构在跨不同流程的服务组合时会面临数据丢失的风险.为此,Messina 等人[39]提出了数据库即服务的架构模式,该模式将数据库功能作为一个微服务,使得服务与数据之间严格耦合.每当数据库需要扩展其功能时,可以直接在数据库中嵌入实现该扩展功能的业务逻辑,从而在降低架构复杂性和相关风险的同时提高微服务应用的可伸缩性.

3) 模块化

模块化的核心是如何划分微服务,其中的一个重要问题是微服务粒度的确定.SOA 中服务的规模可以处于小型应用程序到大型企业级系统之间[76],而微服务则通常是小型、细粒度的服务.

领域驱动的设计(domain-driven design, DDD)[77]是目前进行服务划分的常用方法.DDD是一系列软件设计实践、技术和原则的集合,能够将领域模型解耦成包含部分业务流程且相互区别的有界上下文(bounded context)并表达它们之间的关系.DDD中的每个有界上下文能自然地成为一个微服务的候选,但在使用DDD进行微服务划分时也需要有针对性地解决一些新的问题[40]:

① DDD中的有界上下文和领域模型缺乏对语义表达的支持,因此在微服务划分时会产生语法、结构和语义上的理解或转换问题.针对这一问题,Diepenbrock等人[41]提出了基于DDD建模微服务的元模型,该模型能够有效表达领域模型和共享模型的语义以及它们在有界上下文间的关系.

② DDD不是一个形式化的建模语言,这在一定程度上阻碍了模型的验证和转换.针对这一问题,Rademacher等人[42]使用UML配置文件来辅助领域驱动的微服务建模,其不仅为DDD提供原型和约束,还支持将结构化领域模型表示为UML类图,解决了模型的验证和转换问题.

③ DDD缺乏对软件开发过程的描述,其中的概念和任务不能直接扩展到微服务开发.针对这一问题,Steinegger等人[43-44]提出了一个DDD软件开发过程来构建微服务.他们首先在软件工程领域中对DDD活动进行分类,然后根据DDD的分类和要求研究构建微服务应用程序所需的各类软件开发活动,并将这些活动应用到案例研究中.

尽管目前已有针对微服务的划分方法,但上述方法所产生的结果是否满足高内聚、低耦合等划分的核心原则仍需进一步的验证.为此,钟陈星等人[45]提出了一个基于领域驱动设计中有界上下文理论的微服务粒度评估模型,该模型能够评估划分后服务的内聚性、耦合性、用例收敛性和实例收敛性,并给出对应微服务划分方案的综合评分.

4) 服务发现

服务发现将服务注册到服务注册中心,并提供服务实例的具体网络位置(IP地址和端口)供其他服务调用.SOA相关研究中已有很多服务发现方法(如基于服务质量[78]、语义[79]、Petri网[80]的服务发现),这些方法理论上也可用于微服务架构.然而,由于微服务中服务数量庞大、且在开发技术体系与SOA存在很大差异,因此已有的服务发现方法未必都适用于微服务.

针对容器中微服务的服务发现和通信问题,Stubbs等人[46]提出了一个可扩展的开源解决方案Serfnode,其中新加入的容器能够在Serfnode镜像中广播自己以供其他服务发现.针对云平台环境下微服务数目众多、且在给定需求和优先级条件下难以从中找到最合适服务的问题,Franca等人[47]提出了一个从技术、社会(如使用信息和评论)和语义3个角度来挑选微服务的框架DIRECTOR,能够在数百个候选对象中推荐出最符合需求和优先级条件的微服务.此外,针对传统的单节点部署在容错能力上的缺陷,Tang等人[48]提出了一个分布式的服务发现机制,通过服务发现数据的特点改进了用于解决分布式不一致性问题的Raft算法,从而确保集群中节点服务发现数据的一致性.

目前,在服务发现上已有很多开源软件产品,例如Eureka,Etcd,Consul,Zookeeper和Redis等,其中Eureka是目前研究中最常用的服务发现系统[24,34,37,49].

5) 服务组合

服务组合关注如何将已有的多个功能独立的服务组合成功能更为复杂和完善的整体应用,以满足一定的应用需求.服务组合有编排(orchestration)和协同(choreography)两种方式:服务编排通过一个中央协调器来协调对多个服务的调用,而服务协同是指在没有中央协调器的情况下,所有服务以对等的方式相互协作[76].SOA通常使用WS-BPEL和WS-CDL语言来实现服务编排和协同;与之相比,微服务的去中心化和独立性使其更倾向于服务协同的方式.

目前,在服务组合上已有很多系统和工具可供使用.其中,Spring Cloud通过轻量级的事件驱动机制来实现服务组合,并利用RabbitMQ或Kafka等事件处理中介来协调组合服务的调用;而Netflix的服务组合开源框架Conductor将系统所需的微服务和系统服务结合在一起,形成一个完整的服务组合蓝图来实现某项功能[50].

此外,Yahia等人[51]开发了一个用于微服务组合的事件驱动轻量级平台Medley,可以在主流服务器和嵌入式设备上进行扩展,且仅消耗较少的资源.Camilli等人[52]提出了一种基于工作流的微服务组合建模形式化方法,并指定了一种基于Petri网的微服务形式化语义,以工作流组合的形式来验证微服务组合方案的可行性.Zhang等人[53]针对视频云计算平台的微服务提出了一种性能感知的服务路径选择方法,该方法首先建立一个性能评估模型来衡

量视频处理任务的特点以及微服务实例的处理能力和信息传输条件,然后基于该模型使用最短路径算法搜索和更新最佳微服务组合路径.

6) 容错

容错使得软件系统在运行时能够检测故障并从故障中恢复,防止故障对系统运行造成影响.微服务涉及的故障通常包括由于服务崩溃或者网络延迟等原因造成的无法访问服务,以及服务过载等.

目前,Hystrix[49]是常用的微服务容错开源框架,其综合考虑了服务超时、负载、错误及服务隔离等因素.Hystrix能够封装服务调用逻辑,使每个服务在单独线程中执行,并提供了断路器、调用超时设置和资源隔离3方面功能来保障微服务的可靠性.

7) 安全

目前与微服务安全相关的设计主要关注外部对于系统访问的安全性以及系统内部微服务之间相互调用的安全性.

在外部对于系统访问的安全性上,目前研究主要通过API网关[24,35-37,54]来进行身份验证和授权.API网关不仅能够封装微服务系统内部架构,为客户端提供统一接口,还能根据请求调用单个微服务或者通过服务编排调用多个微服务并返回结果(即起到服务发现和组合的作用).此外,通过API网关还能实现负载均衡、缓存、路由、访问控制、服务代理、监控和日志等功能.Zuul,Kong和gRPC等是目前微服务开发常用的开源网关,同时OAuth2标准①常被用于保障外部对于系统的访问安全.此外,Fu等人[55]还分析了传统访问控制技术在微服务环境中的局限性,并提出了一种基于角色的访问控制模型以提高访问策略的表达能力.

在系统内部微服务之间相互调用的安全性上,网络中错综复杂的情况使得微服务之间不能完全彼此信任.针对这一问题,Yarygina等人[56]提出了一个结合MTLS(mutual transport layer security)、自托管PKI(public key infrastructure)和安全令牌机制的安全框架,以确保微服务之间通信的安全和可信.

8) 负载均衡

一个微服务通常具有多个实例,负载均衡将对同一微服务的多个请求分配到该微服务的特定实例进行处理,从而保证同一微服务的每个实例所处理的请求在数目上大致保持一致.

负载均衡可分为客户端负载均衡和服务端负载均衡,其中服务端负载均衡需要使用一个独立的负载均衡服务[75].目前,Ribbon是实现客户端负载均衡的常用工具,API网关的请求转发和微服务间的调用等实际上都是通过Ribbon来协调实现的.与之相比,服务端负载均衡可以通过硬件或者软件来实现.硬件指的是在服务器节点间安装专门用于负载均衡的设备如Array,F5等;而软件指的是在服务器上使用一些具有均衡负载功能的软件如LVS,Nginx等来完成请求分发工作.

此外,Niu等人[57]提出了基于消息队列的面向微服务调用链的负载均衡算法,该算法通过减少网络开销和减少操作复杂度来降低微服务响应时间;Rusek等人[58]则为运行在OpenVZ容器中的微服务提供了一个非集中式的负载均衡系统,该系统相比于集中式的容器编排系统能够提升一定的性能.

9) 监控和日志

微服务监控的主要目的是在系统运行时对基础设施、微服务等的性能进行观察,而日志则用于在微服务等发生故障后快速排除错误.目前这一设计领域关注微服务监控的整体设计,监控指标的设计以及对应监控数据或日志的收集和分析.

在微服务监控的整体设计上,Haselböck等人[59]提出一个指导监控数据生成、管理、处理和展示的决策指导模型.刘一田等人[60]提出了一个对微服务状态和负载进行监控,并基于聚类分析算法对实时和历史数据进行统计分析的微服务监控框架.

在监控指标的设计以及对应监控数据或日志的收集和分析上,王子勇等人[61]以服务组件的请求处理流为监控指标,利用调用树来刻画请求处理的执行轨迹并监测使执行轨迹发生偏离的系统故障,最后通过分析执行轨迹差异来识别引起故障的关键方法调用.

此外,Sun等人[62]还提出了安全即服务模式,该模式通过为网络虚拟机管理程序添加新的API原语以支持细粒度、灵活的虚拟网络流量监控,确保微服务应用程序的安全.

4.2 组织架构和流程

微服务组织架构和流程关注微服务软件开发团队的组织模式,以及开发和部署微服务的流程.

1) 开发团队组织

"康威定律"指出:开发团队组织的结构在很大

① https://oauth.net/2/

程度上会影响其在系统设计上的结果[81].微服务系统的结构与传统软件的结构并不相同,因此必须以不同于传统软件开发团队组织的方式来组织微服务开发团队.Carrasco 等人[63]认为应根据不同微服务组建跨职能团队,一个团队完全对自己的微服务负责,而不是多个团队共同开发一个微服务,即每个团队具有完全开发微服务所需的所有技能以确保团队和微服务的独立性.

2) 部署

与传统 SOA 使用整体式部署方式不同[82],微服务采用独立部署且主要有 2 种部署方式:1)将多个微服务实例部署在单个物理机或虚拟机上,优点是部署速度快,资源的利用率高,缺点是服务实例之间隔离性较差,可能会出现某个实例占用过多内存或 CPU 资源的情况;2)将单个微服务实例部署在单个物理机或虚拟机上(通常是轻量级的虚拟机),该方法在部署上更加轻量灵活,但可能面临资源利用率不高、服务实例的维护效率低等问题[75].

其中,通过以 Docker① 为代表的容器技术来部署微服务已成为微服务软件部署的必然趋势[24],其还可与 Kubemetes, Mesos 和 Docker Swarm 等容器编排工具相结合实现容器分配和管理的自动化和可视化[83].当使用容器来部署微服务时,也存在一个容器部署一个微服务和一个容器部署多个微服务 2 种方式.Shadija 等人[64]对这 2 种方式进行了性能上的比较,发现在服务延迟上两者并没有明显区别.

微服务在部署时还需要解决容器调度和资源分配等问题.例如,针对容器调度问题,Zhou 等人[65]提出了一个云计算环境下对容器中微服务进行离线和在线调度的框架,旨在最大限度地降低部署微服务的成本.针对容器资源的分配问题,Guerrero 等人[66]使用多目标遗传算法对容器配置进行优化,该方法优于 Kubernetes 中提供的容器管理策略.郝庭毅等人[67]为容器内微服务提供了一种弹性资源供给方法,该方法基于应用负载、资源消耗和响应时间的关联关系,利用卡尔曼滤波来预测不同资源配置下的服务响应时间.Wan 等人[68]则提出了一个优化容器放置位置和任务分配的算法,能在保证应用程序服务延迟要求的同时最小化应用程序部署和运营的成本.

此外,在与微服务部署相关的平台和工具的支持上,李超等人[69]提出了微服务应用部署引擎 OpsFlow,通过组合多种自动化部署技术以应对不同的微服务部署环境.Zheng 等人[70]提出了一个以微服务为中心且支持服务质量标准的 SmartVM 服务部署框架,该框架能够简化构建和部署微服务的流程,在部署成本、资源利用和服务质量上均优于传统的微服务部署方法.Gabbrielli 等人[71]提出了一个以面向服务编程语言 Jolie 编写的服务部署工具 JRO,用于实现微服务的自动和优化部署.Guo 等人[72]提出了一个基于微服务和容器技术的部署平台.

3) DevOps

DevOps 是一组集成自动化软件开发、部署和维护的技术,旨在提高开发团队持续交付的能力以适应客户需求的变化,同时保证软件产品的质量[73].

Chen[73]在 4 年的实践研究中发现,DevOps 对于软件可部署性和可修改性的要求很高,单体架构难以满足这一要求,而微服务架构在这点上十分契合.同时,对于微服务软件而言,每个微服务都是一个可部署单元,需求的快速变化要求微服务能够被快速自动地部署.因此,DevOps 和微服务的结合势在必行.

在微服务开发中应用 DevOps 的一个关键要素是 DevOps 工具链.Ebert 等人[74]研究了一系列适用于微服务开发的自动化工具,包括有助于实现快速迭代的 Apache Ant, Maven, Rake 和 Gradle 自动化构建工具;尽早将开发者工作集成起来的 Jenkins, TeamCity 和 Bamboo 持续集成工具;实现基础设施管理自动化的 Puppet, Chef 和 Ansible 配置管理工具;以及保持基础设施稳定性和性能的 Graylog2 日志工具和 Nagios, New Relic 监控工具.

除了针对上述单个设计领域的具体研究外,微服务框架和服务网格技术还能为多个设计领域的实现提供统一的解决方案,方便开发者快速构建微服务.其中,微服务框架是微服务体系结构的具体实现及解决方案,是企业、研究者在构建微服务时将众多关键技术如服务部署、服务通信和服务发现等集成为一个整体,从而形成的一种框架或方案.目前研究主要使用 Spring Cloud② 这一微服务框架[37,49],其为微服务架构中涉及的配置管理、服务治理、断路器、智能路由、控制总线、分布式会话和集群状态管理等操作提供了一种简单的开发方式.

① https://www.docker.com/

② https://spring.io/projects/spring-cloud/

另一方面,服务网格被称为"下一代微服务",其通过统一的控制平面来定义服务发现、熔断、限流、降级、分布式跟踪等策略,并为每一个微服务实例部署一个称为随行(sidecar)的代理.微服务之间通过该代理进行通信,该代理负责在服务通信时应用和执行预先定义的治理策略,从而为所有微服务提供完全集成的服务治理环境.目前较为成熟的服务网格有 Istio①,Linkerd②和 Consul③等.

5 面向微服务的软件测试

与传统软件测试方法类似,对微服务的测试通常包括单元测试、集成测试、组件测试和端到端测试这 4 个层次[84].其中,单元测试关注对微服务内部的模块进行测试;集成测试验证微服务内部模块之间、以及内部模块与外部组件之间的交互;组件测试用于验证每个微服务本身能否满足相应需求;而端到端测试则是测试微服务应用程序的整体行为,以最终软件产品的视角来验证应用程序是否满足业务目标.然而,微服务系统通常由众多服务组成,这些服务同时运行、互相调用、并且容易发生变化,这些特性会为面向微服务的软件测试带来了一些新的挑战.

我们共收集到 16 篇与微服务测试相关的文献,按研究内容可分为组件测试、端到端测试、回归测试、验收测试、变异测试和微服务调试.

在组件测试上,Wu 等人[85]提出了一个基于故障注入的微服务应用容错性能测试框架,以验证目标服务的容错能力.Heorhiadi 等人[86]提出了一个对微服务故障处理能力进行测试的框架,该框架允许操作员通过操纵服务间通信来执行自动化测试.Camargo 等人[87]提出了一个对微服务进行性能测试的方法,该方法将包含测试参数的规范附加到每个服务,从而对这些参数进行自动化测试.Rahman 等人[88]研究了微服务的并行测试,他们利用 Docker 容器的操作系统级虚拟化特点来为微服务的并行执行提供沙箱机制.此外,还有研究者利用组件测试来帮助确定微服务部署方案,例如 Avritzer 等人[89]通过分析部署完成后的微服务负载来定量评估不同内存、CPU 分配条件下的微服务部署方案,从而帮助确定最优部署配置.

在端到端测试上,Ma 等人[90]提出了一个能够自动生成微服务系统对应的服务依赖图的方法.该方法能够在开发的早期阶段对存在风险的调用链进行分析和测试,并在开发新版本系统时追踪不同服务间的联系.Meinke 等人[91]则使用基于学习的测试来评估微服务系统功能的正确性和鲁棒性.

在回归测试上,Kargar 等人[92]提出了一个将回归测试结合到微服务系统开发持续交付过程的自动化测试方法.在新版本交付时,该方法能根据资源使用率、系统故障率、系统性能等指标将现版本系统与之前版本进行比较,并在需要时阻止发布最新版本.

在验收测试上,Rahman 等人[93]提出了一个可重用的自动测试框架,该框架能够检验微服务仓库在可重用性、可审计性和可维护性上是否满足预期和验收标准,同时还能减少开发人员和测试人员之间的冲突,允许他们在实现相同应用程序目标的同时能够独立地对微服务仓库进行迭代检验.

在变异测试上,Winzinger[94]提出了一个针对微服务测试创建变异算子的初步设想,使用该算子能够有效评估微服务测试用例的质量.

在微服务调试上,Zhou 等人[95]对工业界中微服务系统的典型故障、调试实践以及开发人员所面临的挑战进行了调研,并开发了一个基准微服务系统来验证这些实践的有效性,其发现最新的跟踪和可视化技术能够提升这些现有实践.此外,他们还针对微服务的高度复杂性和动态性,提出了一种增量式的调试方法[96],以及一种基于系统跟踪日志学习的故障预测和定位方法[97].李文海等人[98]提出了一个基于日志可视化分析的微服务软件调试方法,该方法通过日志、调用链、节点与服务信息构建日志模型,并针对典型微服务故障提出具体的可视化调试策略.Rajagopalan 等人[99]则开发了一个对微服务性能故障进行定位和修复的自动化工具,该工具能够将不同版本的微服务进行统一测试和部署,并能通过部分回滚来修复当前版本的性能问题,直到找到一个不存在性能故障的组合.

6 面向微服务的重构

面向微服务的软件重构旨在将传统上难于维护和扩展的单体应用程序重构为微服务应用程序(即

① https://istio.io/

② https://linkerd.io/

③ https://www.consul.io/

将单体应用程序迁移到微服务).由于微服务自身的特点,面向微服务的重构主要具有3项挑战:

1) 现有应用的模块数目通常十分庞大,在确定接口和通信方式、构建支持微服务运行的稳定分布式环境、以及构建自动化基础设施等方面都需要巨大的工作量;

2) 将数据库有效拆分和分配给对应的微服务,同时保持数据一致性的复杂性;

3) 微服务是无状态的,但是单一的遗留代码通常是有状态的[100],需要处理这种从有状态到无状态的转换.

目前,与面向微服务重构相关的研究主要集中在重构的流程、方法和决策,微服务的提取,以及相关的实证和经验研究.

在重构流程上,Francesco 等人[101]认为微服务的重构过程与将已有系统迁移到面向服务架构的过程一致,包括逆向工程(分析已有系统并且识别出能够作为服务的候选元素)、架构转换(将已有系统体系结构重构为面向微服务的体系结构)和前向工程(最终确定、实施和部署新系统的设计)这3个步骤.Taibi 等人[102]给出了一个由3个流程组成的迁移过程框架,其中前2个流程用于从头开始重新开发整个系统,另一个用于在现有系统之上以微服务架构创建新功能.Fan 等人[103]则从软件生命周期的角度考虑迁移流程,并总结了每个阶段的方法和工具.

在重构方法上,Fritzsch 等人[104]将已有的重构方法分为静态代码分析辅助方法、元数据辅助方法、工作负载-数据辅助方法以及动态微服务组合方法.静态代码分析辅助方法从应用程序的源代码中派生出分解方案;元数据辅助方法[105]使用更抽象的输入数据(例如 UML 图)描述用例和接口;工作负载-数据辅助[106]方法通过度量应用程序上模块或功能级别的操作数据(例如通信、性能等)来确定合适的服务分解粒度;动态微服务组合方法[107-109]通过描述微服务运行时环境来解决服务分解问题.

在重构决策上,Christoforou 等人[110]给出了与微服务重构相关的关键概念和驱动因素,并提出一个决策支持系统以支持微服务重构过程中的决策制定.

微服务的提取主要关注如何从单体应用中提取出候选的微服务.目前提出的方法大致可划分为基于有向图的方法、基于无向图的方法以及基于机器学习的方法:

1) 在基于有向图方法提取微服务上,Levcovitz 等人[111]根据单体应用中业务功能入口、业务功能和数据库表构建依赖关系图,并根据图中的依赖关系识别出候选微服务.Chen 等人[112]根据业务逻辑构建并优化数据流图,并根据"操作+输出数据"的描述风格提取出候选微服务.Ren 等人[113]结合静态和动态分析来构建单体应用程序的功能调用图,并利用功能之间的耦合程度以及功能聚类识别出候选微服务.

2) 在基于无向图方法提取微服务上,Gysel 等人[114]提出了 Service Cutter 方法,旨在从域模型、用例等软件工程构件中提取出系统的耦合信息并表示为无向加权图,并通过图形聚类算法识别出微服务.Mazlami 等人[115]同样使用了图聚类算法来提取微服务,他们首先将3种形式化耦合策略嵌入到图聚类算法中,随后利用微服务提取模型和上述图聚类算法对从系统代码库中构造出的无向加权图进行处理,从而推荐候选微服务.

3) 在基于机器学习方法提微服务上,Abdullah 等人[116]提出了一种基于应用程序访问日志和非监督机器学习的微服务分解方法,他们还提出了一种动态选择合适资源类型的方法,以提高重构后微服务应用程序的可扩展性和整体性能.

此外,Escobar 等人[117]评估了 Enterprise Java Bean(EJB)之间的耦合性,从而对不同 EJB 进行分离和分组,并将一个或若干个 EJB 映射为一个微服务.Baresi 等人[118]提出了一种基于 OpenAPI 接口规范的微服务划分方案,能通过对单体应用的接口进行分析从而自动识别出候选微服务.Kecskemeti 等人[119]还基于 ENTICE 项目的成果提出将单体应用解耦成微服务的方法,并讨论了相关技术的优势.

在与微服务重构相关的实证和经验研究上,已有研究致力于从工业界实践中进行学习、或者分享自身重构经验.对于前者,Carrasco 等人[63]通过对34篇学术论文以及24篇传统学术出版发行渠道之外的灰色文献进行调研,总结了在微服务重构过程中常见的9个陷阱.Francesco 等人[101]进行了一项微服务迁移实践的实证研究,他们通过调查问卷总结了来自16家 IT 公司的18名从业者在逆向工程、架构转换和前向工程阶段进行的各项活动(如对已有系统信息的研究、设计新架构执行的任务和如何开始迁移等)和遇到的各种挑战(如原系统的高度耦合、服务界限的识别和基础设施的设置).Taibi 等

人[102]调查了21名从业者并分析了他们迁移到微服务的动机和遇到的困难,他们发现微服务的可扩展性和可维护性是主要的动机,而主要困难则在于将单体系统解耦成微服务、将遗留数据库中的数据进行迁移以及分割给各个服务等.Kalske等人[120]则对微服务软件重构相关的出版物和案例研究进行了调研,以获取公司进行重构的原因和这一过程中可能面临的各类挑战.

而在分享自身重构经验上,Furda等人[100]研究并报告了将单体应用程序迁移到微服务时在多租户、有状态和一致性这3个方面面临的挑战.Balalaie等人[121]报告了将单体架构迁移到微服务架构的经验,他们发现微服务架构在提高可伸缩性和可用性的同时也引入了新的复杂性和困难.他们同时还研究了一些工业规模的软件迁移项目,从中识别和收集了一组微服务重构的设计模式[122],并在一个现实案例中探讨了如何使用DevOps来实现平稳迁移[123].

此外,Zdun等人[124]还提出了一种对微服务重构质量进行检验的方法.该方法首先对重构进行建模,然后根据若干约束和度量来检验微服务之间的依赖性,以及重构后的微服务是否可以独立部署、或者可以独立部署到什么程度.

7 未来研究方向

面向微服务软件开发作为一个新的技术发展趋势,早已在注重实践的工业界得到了广泛的运用,并派生出一系列成熟的开源工具和平台.但通过本文调研,我们发现目前学术界对于微服务开发的研究尚处于早期阶段,尤其缺少在一流会议和期刊上发表的相关研究(例如CCF-A类).此外,我们也发现在面向微服务软件开发方法的整体研究趋势上,国内相关研究的水平与国际前沿相比仍存在一定的差距.

在微服务软件需求分析方面,目前相关研究通常是作为开发或者迁移特定应用程序研究的一部分,并主要沿用传统的软件工程需求分析方法[20-22,24];而不同微服务软件在非功能需求上往往会存在很多共性特征[20,24-25].因此,在微服务软件需求分析上,一方面我们需要针对微服务的特点来研究更加快速、灵活和精确的需求分析方法,并开发相应的需求分析工具,提高需求分析的效率、可伸缩性和正确性;同时还需要综合考虑微服务设计、测试等若干后续开发阶段,使得需求的变化能够快速反馈到其他阶段,而其他阶段的结果也能够在需求上及时验证.另一方面,在非功能需求分析上也需要对不同领域、特征的微服务软件进行更加系统全面的总结,制定相应的实践标准,为研究者提供规范的参考.

在微服务软件设计与实现方面,在工业界人们已开发了很多微服务框架(如Spring Cloud等)和开源解决方案(如Netflix开源组件、服务网格Istio、Docker等),并得到了广泛的应用.虽然学术界也提出了很多面向微服务设计与实现的工具,但这些工具在功能和性能上能否满足实践中大规模系统开发的需求仍需要进一步的验证.此外,微服务软件设计过程中存在的众多影响因素也为整体设计质量的控制带来了困难[32].因此,在微服务软件设计与实现上,一方面我们需要全面了解已有的工业界实践方式,在此基础上分析当前仍存在的问题和挑战;同时,进一步推广学术界的最新研究成果,使这些研究成果的有效性能得到充分的验证.另一方面,我们还需要研究微服务软件设计的质量评估方法,从而对相应设计结果的整体质量进行评估,更好地协助开发人员解决设计与实现上的决策问题.

在微服务测试方面,目前研究在组件测试、端到端测试、回归测试、验收测试、变异测试和软件调试等方面进行了一定程度的探索,但仍缺乏专门针对微服务内部逻辑进行测试的方法.此外,现有研究主要关注自动化测试框架的构建,缺乏对于面向微服务的测试建模、测试用例生成、测试预期输出判定以及故障定位等各个阶段的深入研究.因此,在微服务测试上,一方面我们需要持续研究如何将传统测试技术引入到微服务测试中,通过对已有测试技术的改进和优化来进一步提高微服务测试的有效性,并研究新的测试覆盖标准,确保测试的准确、快速和全面.另一方面我们也需要在微服务测试的测试建模、测试用例生成、测试预期输出判定以及故障定位等各个阶段开发相应的自动化测试方法和工具,形成系统的微服务测试方法体系.

在微服务重构方面,目前研究数量相对较多,其中研究者不仅在重构流程和方法上进行了探索,还提出了很多将单体应用迁移为微服务的方法,并开展了一批实证研究.但是,已有的重构方法是否适用于现实中的大规模系统也仍需进一步的验证.同时,在不同重构方法的比较和评估上也需要开展更多的经验研究,并在此基础上提取出适用于不同重构场景的标准化、规范化方法.

8 总 结

面向微服务软件开发是近年来软件工程领域研究的前沿和热点话题.为了了解该领域的当前研究进展,本文首先收集了目前与微服务软件开发相关的91篇研究论文,随后从需求分析、设计与实现、测试以及重构的角度对已有的方法、工具和实践进行了分析和总结,并对微服务软件开发可能的一些研究方向进行了讨论.

经过不足10年的发展,微服务在工业实践中已经取得了广泛且丰富的应用,但学术界对于微服务开发的研究尚处于早期阶段,与实践应用之间仍存在着很大鸿沟.作为面向服务体系结构的最新研究和发展趋势,微服务在未来具有良好的研究和应用前景.本文希望能为相关研究者和开发团队提供参考和借鉴,以进一步提高微服务软件开发的效率和质量.

参 考 文 献

[1] Xiao Zhongxiang, Wijegunaratne I, Qiang Xinjian. Reflections on SOA and Microservices [C] //Proc of the 4th Int Conf on Enterprise Systems. Piscataway, NJ: IEEE, 2016: 60-67

[2] Stojanovic Z, Dahanayake A. Service-Oriented Software System Engineering: Challenges and Practices [M]. Hershey, Pennsylvania: IGI Global, 2005

[3] Papazoglou M P, Van Den Heuvel W J. Service-oriented design and development methodology [J]. International Journal of Web Engineering and Technology, 2006, 2(4): 412-442

[4] Lemos A L, Daniel F, Benatallah B. Web service composition: A survey of techniques and tools [J]. ACM Computing Surveys, 2016, 48(3): 33.1-33.41

[5] Tvedt R T, Costa P, Lindvall M. Evaluating software architectures [J]. Advances in Computers, 2004, 61(5): 1-43

[6] Sharma A, Kumar M, Agarwal S. A complete survey on software architectural styles and patterns [J]. Procedia Computer Science, 2015, 70: 16-28

[7] Dragoni N, Giallorenzo S, Lafuente A L, et al. Microservices: Yesterday, today, and tomorrow [G] // Present and Ulterior Software Engineering. Berlin: Springer, 2017: 195-216

[8] Namiot D, Sneps-Sneppe M. On Micro-services architecture [J]. International Journal of Open Information Technologies, 2014, 2(9): 24-27

[9] Ling Xiaodong. A review of SOA [J]. Computer Applications and Software, 2007, 24(10): 122-124 (in Chinese)
(凌晓东. SOA综述[J]. 计算机应用与软件, 2007, 24(10): 122-124)

[10] Yang Zhiyi, Yang Gang, Zhang Haihui. An approach for implementing service-oriented and event-driven information integration platform [J]. Journal of Computer Research and Development, 2008, 45(10): 1799-1806 (in Chinese)
(杨志义, 杨刚, 张海辉. 一种面向服务的事件驱动架构信息集成平台构造方法[J]. 计算机研究与发展, 2008, 45(10): 1799-1806)

[11] Papazoglou M P, Traverso P, Dustdar S, et al. Service-oriented computing: State of the art and research challenges [J]. Computer, 2007, 40(11): 38-45

[12] Li Lei, Niu Chunlei, Chen Ningjiang, et al. A high-performance strategy for optimizing Web services [J]. Journal of Computer Research and Development, 2007, 44(7): 1191-1198 (in Chinese)
(李磊, 牛春雷, 陈宁江, 等. 一种高效的Web服务性能优化策略[J]. 计算机研究与发展, 2007, 44(7): 1191-1198)

[13] Butzin B, Golatowski F, Timmermann D. Microservices approach for the Internet of things [C] //Proc of the 21st IEEE Int Conf on Emerging Technologies and Factory Automation. Piscataway, NJ: IEEE, 2016: 209.1-209.6

[14] 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

[15] Lewis J, Fowler M. Microservices: A definition of this new architectural term [OL]. [2019-06-27]. https://www.martinfowler.com/articles/microservices.html

[16] Jamshidi P, Pahl C, Nabor C. et al. Microservices: The journey so far and challenges ahead [J]. IEEE Software, 2018, 35(3): 24-35

[17] Budgen D, Brereton P. Performing systematic literature reviews in software engineering [C] //Proc of the 28th Int Conf on Software Engineering. New York: ACM, 2006: 1051-1052

[18] Sotelo K I G, Baron C, Esteban P, et al. How to find non-functional requirements in system developments [J]. IFAC-PapersOnLine, 2018, 51(11): 1573-1578

[19] SEBoK Editorial Board. Guide to the systems engineering body of knowledge (SEBoK)[EB/OL]. (2019-10-30) [2019-12-12]. http://sebokwiki.org

[20] Richter D, Konrad M, Utecht K, et al. Highly-available applications on unreliable infrastructure: Microservice architectures in practice [C] //Proc of the 39th IEEE Int Conf on Software Quality, Reliability and Security Companion. Piscataway, NJ: IEEE, 2017: 130-137

[21] Hassan S, Bahsoon R. Microservices and their design trade-offs: A self-adaptive roadmap [C] //Proc of the 13th IEEE Int Conf on Services Computing. Piscataway, NJ: IEEE, 2016: 813-818

[22] Suryotrisongko H, Jayanto D P, Tjahyanto A. Design and development of backend application for public complaint systems using microservice spring boot [J]. Procedia Computer Science, 2017, 124: 736—743

[23] Porrmann T, Essmann R, Colombo A W. Development of an event-oriented, cloud-based SCADA system using a Microservice architecture under the RAMI4.0 specification: Lessons learned [C] //Proc of the 43rd Annual Conf of the IEEE Industrial Electronics Society. Piscataway, NJ: IEEE, 2017: 3441—3448

[24] Wizenty P N, Rademacher F, Sorgalla J, et al. Design and implementation of a remote care application based on Microservice architecture [G] //LNCS 11176: Proc of the 2018 Int Conf on Software Technologies: Applications and Foundations. Berlin: Springer, 2018: 549—557

[25] Schneider T, Wolfsmantel A. Achieving cloud scalability with Microservices and DevOps in the connected car domain [OL]. [2019-12-11]. http://ceur-ws.org/Vol-1559/paper16.pdf

[26] Taibi D, Lenarduzzi V, Pahl C. Architectural patterns for Microservices: A systematic mapping study [C] //Proc of the 8th Int Conf on Cloud Computing and Services Science. Setúbal, Portugal: SciTePress, 2018: 221—232

[27] Bogner J, Zimmermann A, Wagner S. Analyzing the relevance of SOA patterns for Microservice-based systems [C/OL] //Proc of the 10th ZEUS Workshop. 2018: 9—16 [2019-08-01]. http://ceur-ws.org/Vol-2072/paper2.pdf

[28] Erl T. SOA Design Patterns (paperback) [M]. New York: Pearson Education, 2008

[29] Erl T, Carlyle B, Pautasso C, et al. SOA with Rest: Principles, Patterns & Constraints for Building Enterprise Solutions with Rest [M]. Upper Saddle River, NJ: Prentice Hall Press, 2012

[30] Rotem-Gal-Oz A, Bruno E, Dahan U. SOA Patterns [M]. Shelter Island, NY: Manning Publications, 2012

[31] Haselböck S, Weinreich R, Buchgeher G. An expert interview study on areas of Microservice design [C] //Proc of the 11th Conf on Service-Oriented Computing and Applications. Piscataway, NJ: IEEE, 2018: 137—144

[32] Haselböck S, Weinreich R, Buchgeher G, et al. Microservice design space analysis and decision documentation: A case study on API management [C] //Proc of the 11th Conf on Service-Oriented Computing and Applications. Piscataway, NJ: IEEE, 2018: 1—8

[33] Haselböck S, Weinreich R, Buchgeher G. Decision guidance models for Microservices: Service discovery and fault tolerance [C] //Proc of the 5th European Conf on the Engineering of Computer-Based Systems. New York: ACM, 2017: 4.1—4.10

[34] Le V D, Neff M M, Stewart R V, et al. Microservice-based architecture for the NRDC [C] //Proc of the 13th Int Conf on Industrial Informatics. Piscataway, NJ: IEEE, 2015: 1659—1664

[35] Luo Qinkai, Ni Chengzhang. Application of workflow technology based on Micro-service in cloud management platform [J]. Computer Technology and Development, 2019, 29(9): 122—127 (in Chinese)
(罗钦凯, 倪成章. 基于微服务的工作流技术在云管平台的应用[J]. 计算机技术与发展, 2019, 29(9): 122—127)

[36] Idoughi D, Abdelouhab K A, Kolski C. Towards a Micro-services development approach for the crisis management field in developing countries [C] //Proc of the 4th Int Conf on Information and Communication Technologies for Disaster Management. Piscataway, NJ: IEEE, 2017: 12.1—12.6

[37] Liu Gang. Dealer management system based on Microservice architecture [J]. Journal of Computer Applications, 2018, 38(增2): 243—249 (in Chinese)
(刘罡. 基于微服务架构的汽车经销商管理系统[J]. 计算机应用, 2018, 38(S2): 243—249)

[38] Viennot N, Lécuyer M, Bell J, et al. Synapse: A Microservices architecture for heterogeneous-database Web applications [C] //Proc of the 10th European Conf on Computer Systems. New York: ACM, 2015: 21.1—21.16

[39] Messina A, Rizzo R, Storniolo P, et al. A simplified database pattern for the Microservice architecture [C] //Proc of the 8th Int Conf on Advances in Databases, Knowledge, and Data Applications. Wilmington, DE: IARIA XPS Press, 2016

[40] Rademacher F, Sorgalla J, Sachweh S. Challenges of domain-driven Microservice design: A model-driven perspective [J]. IEEE Software, 2018, 35(3): 36—43

[41] Diepenbrock A, Rademacher F, Sachweh S. An ontology-based approach for domain-driven design of Microservice architectures [C] //Proc of the 2017 INFORMATIK. Bonn, Nordrhein-Westfalen, Germany: Gesellschaft für Informatik, 2017: 1777—1791

[42] Rademacher F, Sachweh S, Zündorf A. Towards a UML profile for domain-driven design of Microservice architectures [G] //LNCS 10729: Proc of the 2017 Int Conf on Software Engineering and Formal Methods. Berlin: Springer, 2017: 230—245

[43] Steinegger R H, Giessler P, Hippchen B, et al. Overview of a domain-driven design approach to build Microservice-based applications [C] //Proc of the 3rd Int Conf on Advances and Trends in Software Engineering. Wilmington, DE: IARIA XPS Press, 2017

[44] Hippchen B, Giessler P, Steinegger R, et al. Designing Microservice-based applications by using a domain-driven design approach [J]. International Journal on Advances in Software, 2017, 10(3/4): 432—445

[45] Zhong Chenxing, Li Shanshan, Zhang He, et al. Evaluating granularity of Microservices-oriented system based on bounded context [J]. Journal of Software, 2019, 30(10): 3227—3241 (in Chinese)
(钟陈星, 李杉杉, 张贺, 等. 限界上下文视角下的微服务粒度评估[J].软件学报, 2019, 30(10): 3227—3241)

[46] Stubbs J, Moreira W, Dooley R. Distributed systems of Microservices using docker and serfnode [C] //Proc of the 7th Int Workshop on Science Gateways. Piscataway, NJ: IEEE, 2015: 34—39

[47] França M, Werner C. Perspectives for selecting cloud Microservices [C] //Proc of the 3rd IEEE Int Conf on Software Architecture Companion. Piscataway, NJ: IEEE, 2018: 56—59

[48] Tang Weilun, Wang Li, Xue Guangtao. Design of high availability service discovery for Microservices architecture [C] //Proc of the 3rd Int Conf on Management Engineering, Software Engineering and Service Sciences. New York: ACM, 2019: 253—257

[49] Wu Lei, Zhan Jian, Song Lihua. Research of application of Micro-service architecture in smart home gateway system [J]. Computer Technology and Development, 2019, 29(11): 200—205 (in Chinese)
(吴磊, 湛健, 宋丽华. 微服务架构在智能家居网关系统中的应用研究[J]. 计算机技术与发展, 2019, 29(11): 200—205)

[50] Baraiya V, Singh V. Netflix conductor: A Microservices orchestrator [EB/OL]. [2019-06-27]. https://medium.com/netflix-techblog/netflix-conductor-a-microservices-orchestrator-2e8d4771bf40

[51] Yahia E B H, Réveillère L, Bromberg Y D, et al. Medley: An event-driven lightweight platform for service composition [G] //LNCS 9671: Proc of the 2016 Int Conf on Web Engineering. Berlin: Springer, 2016: 3—20

[52] Camilli M, Bellettini C, Capra L, et al. A formal framework for specifying and verifying Microservices based process flows [G] //LNCS 10729: Proc of the 2017 Int Conf on Software Engineering and Formal Methods. Berlin: Springer, 2017: 187—202

[53] Zhang Haitao, Yang Ning, Xu Zhengjun, et al. Microservice based video cloud platform with performance-aware service path selection [C] //Proc of the 15th IEEE Int Conf on Web Services. Piscataway, NJ: IEEE, 2018: 306—309

[54] Krylovskiy A, Jahn M, Patti E. Designing a smart city Internet of things platform with Microservice architecture [C] //Proc of the 3rd Int Conf on Future Internet of Things and Cloud. Piscataway, NJ: IEEE, 2015: 25—30

[55] Fu Guo, Sun Jin, Zhao Jiantao. An optimized control access mechanism based on Micro-service architecture [C] //Proc of the 2nd IEEE Conf on Energy Internet and Energy System Integration. Piscataway, NJ: IEEE, 2018

[56] Yarygina T, Bagge A H. Overcoming security challenges in Microservice architectures [C] //Proc of the 10th IEEE Symp on Service-Oriented System Engineering. Piscataway, NJ: IEEE, 2018: 11—20

[57] Niu Yipei, Liu Fangming, Li Zongpeng. Load balancing across Microservices [C] //Proc of the 31st IEEE INFOCOM IEEE Conf on Computer Communications. Piscataway, NJ: IEEE, 2018: 198—206

[58] Rusek M, Dwornicki G, Orłowski A. A decentralized system for load balancing of containerized Microservices in the cloud [C] //Proc of the 2016 Int Conf on Systems Science. Berlin: Springer, 2016: 142—152

[59] Haselböck S, Weinreich R. Decision guidance models for Microservice monitoring [C] //Proc of the 1st IEEE Int Conf on Software Architecture Workshops. Piscataway, NJ: IEEE, 2017: 54—61

[60] Liu Yitian, Liu Shijin, Guo Wei, et al. Flexible Microservice monitoring framework [J]. Computer Systems & Applications, 2017, 26(10): 139—143 (in Chinese)
(刘一田, 刘士进, 郭伟, 等. 柔性微服务监控框架[J]. 计算机系统应用, 2017, 26(10): 139—143)

[61] Wang Ziyong, Wang Tao, Zhang Wenbo, et al. Fault diagnosis for Microservices with execution trace monitoring [J]. Journal of Software, 2017, 28(6): 1435—1454 (in Chinese)
(王子勇, 王焘, 张文博, 等. 一种基于执行轨迹监测的微服务故障诊断方法[J]. 软件学报, 2017, 28(6): 1435—1454)

[62] Sun Yuqiong, Nanda S, Jaeger T. Security-as-a-service for Microservices-based cloud applications [C] //Proc of the 7th IEEE Int Conf on Cloud Computing Technology and Science. Piscataway, NJ: IEEE, 2015: 50—57

[63] Carrasco A, Bladel B, Demeyer S. Migrating towards Micro-services: Migration and architecture smells [C] //Proc of the 2nd Int Workshop on Refactoring. New York: ACM, 2018: 1—6

[64] Shadija D, Rezai M, Hill R. Microservices: Granularity vs. performance [C] //Proc of the 10th Int Conf on Utility and Cloud Computing. New York: ACM, 2017: 215—220

[65] Zhou Ruiting, Li Zongpeng, Wu Chuan. Scheduling frameworks for cloud container services [J]. IEEE/ACM Transactions on Networking, 2018, 26(1): 436—450

[66] Guerrero C, Lera I, Juiz C. Genetic algorithm for multi-objective optimization of container allocation in cloud architecture [J]. Journal of Grid Computing, 2018, 16(1): 113—135

[67] Hao Tingyi, Wu Heng, Wu Guoquan, et al. Elastic resource provisioning approach for container in Micro-service architecture [J]. Journal of Computer Research and Development, 2017, 54(3): 597—608 (in Chinese)
(郝庭毅, 吴恒, 吴国全, 等. 面向微服务架构的容器级弹性资源供给方法[J]. 计算机研究与发展, 2017, 54(3): 597—608)

[68] Wan Xili, Guan Xinjie, Wang Tianjing, et al. Application deployment using Microservice and Docker containers: Framework and optimization [J]. Journal of Network and Computer Applications, 2018, 119: 97—109

[69] Li Chao, Hua Lei, Song Yunkui. OpsFlow: Automatic application deployment engine for DevOps [J]. Computer & Digital Engineering, 2019, 47(1): 190—194 (in Chinese)
(李超, 花磊, 宋云奎. OpsFlow: 一种面向 DevOps 的应用自动化部署引擎[J]. 计算机与数字工程, 2019, 47(1): 190—194)

[70] Zheng Tianlei, Zheng Xi, Zhang Yuqun, et al. SmartVM: A SLA-aware Microservice deployment framework [J]. World Wide Web, 2019, 22(1): 275—293

[71] Gabbrielli M, Giallorenzo S, Guidi C, et al. Self-reconfiguring Microservices [G] //LNCS 9660: Theory and Practice of Formal Methods. Berlin: Springer, 2016: 194—210

[72] Guo Dong, Wang Wei, Zeng Guosun, et al. Microservices architecture based cloudware deployment platform for service computing [C] //Proc of the 9th IEEE Symp on Service-Oriented System Engineering. Piscataway, NJ: IEEE, 2016: 358—363

[73] Chen Lianping. Microservices: Architecting for continuous delivery and DevOps [C] //Proc of the 15th IEEE Int Conf on Software Architecture. Piscataway, NJ: IEEE, 2018: 39—46

[74] Ebert C, Gallardo G, Hernantes J, et al. DevOps [J]. IEEE Software, 2016, 33(3): 94—100

[75] Xin Yuanyuan, Niu Jun, Xie Zhijun, et al. Survey of implementation framework for Microservices architecture [J]. Computer Engineering and Applications, 2018, 54(19): 16—23 (in Chinese)
(辛园园, 钮俊, 谢志军, 等. 微服务体系结构实现框架综述[J]. 计算机工程与应用, 2018, 54(19): 16—23)

[76] Richards M. Microservices vs. Service-Oriented Architecture [M]. Sebastopol, CA: O'Reilly Media, 2015

[77] Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software [M]. Reading, MA: Addison-Wesley Professional, 2004

[78] Wan Changlin, Shi Zhongzhi, Hu Hong, et al. QOS-aware semantic Web service modeling and discovery [J]. Journal of Computer Research and Development, 2011, 48(6): 1059—1066 (in Chinese)
(万长林, 史忠植, 胡宏, 等. 基于本体的语义 Web 服务 QoS 描述和发现[J]. 计算机研究与发展, 2011, 48(6): 1059—1066)

[79] Ye Lei, Zhang Bin. A method of Web service discovery based on functional semantics [J]. Journal of Computer Research and Development, 2007, 44(8): 1357—1364 (in Chinese)
(叶蕾, 张斌. 基于功能语义的 Web 服务发现方法[J]. 计算机研究与发展, 2007, 44(8): 1357—1364)

[80] Zhang Guangsheng, Jiang Changjun, Ding Zhijun. Service discovery framework using fuzzy Petri net [J]. Journal of Computer Research and Development, 2006, 43(11): 1886—1894 (in Chinese)
(张广胜, 蒋昌俊, 丁志军. 基于模糊 Petri 网的服务发现框架研究[J]. 计算机研究与发展, 2006, 43(11): 1886—1894)

[81] Conway M E. How do committees invent [J]. Datamation, 1968, 14(4): 28—31

[82] Wolff E. Microservices: Flexible Software Architecture [M]. Reading, MA: Addison-Wesley Professional, 2016

[83] Zhang Limin, Gao Jing, Li Wubin, et al. Visualization of container orchestration in Microservice environment [J]. Computer Engineering & Science, 2019, 41(8): 1366—1373 (in Chinese)
(张丽敏, 高晶, 李务斌, 等. 微服务环境下容器编排可视化实践研究[J]. 计算机工程与科学, 2019, 41(8): 1366—1373)

[84] Savchenko D I, Radchenko G I, Taipale O. Microservices validation: Mjolnirr platform case study [C] //Proc of the 38th Int Convention on Information and Communication Technology, Electronics and Microelectronics. Piscataway, NJ: IEEE, 2015: 235—240

[85] Wu Na, Zuo Decheng, Zhang Zhan. An extensible fault tolerance testing framework for Microservice-based cloud applications [C] //Proc of the 4th Int Conf on Communication and Information Processing. New York: ACM, 2018: 38—42

[86] Heorhiadi V, Rajagopalan S, Jamjoom H, et al. Gremlin: Systematic resilience testing of Microservices [C] //Proc of the 36th IEEE Int Conf on Distributed Computing Systems. Piscataway, NJ: IEEE, 2016: 57—66

[87] Camargo A, Salvadori I, Mello R S, et al. An architecture to automate performance tests on Microservices [C] //Proc of the 18th Int Conf on Information Integration and Web-based Applications and Services. New York: ACM, 2016: 422—429

[88] Rahman M, Chen Zehua, Gao Jerry. A service framework for parallel test execution on a developer's local development workstation [C] //Proc of the 9th IEEE Symp on Service-Oriented System Engineering. Piscataway, NJ: IEEE, 2015: 153—160

[89] Avritzer A, Ferme V, Janes A, et al. A quantitative approach for the assessment of Microservice architecture deployment alternatives by automated performance testing [G] //LNCS 11048: Proc of the 2018 European Conf on Software Architecture. Berlin: Springer, 2018: 159—174

[90] Ma Shangpin, Fan Chenyuan, Chuang Yen, et al. Using service dependency graph to analyze and test Microservices [C] //Proc of the 42nd IEEE Annual Computer Software and Applications Conf. Piscataway, NJ: IEEE, 2018: 81—86

[91] Meinke K, Nycander P. Learning-based testing of distributed microservice architectures: Correctness and fault injection [G] //LNCS 9509: Proc of the 2015 Int Conf on Software Engineering and Formal Methods Collocated Workshops. Berlin: Springer, 2015: 3—10

[92] Kargar M J, Hanifizade A. Automation of regression test in Microservice architecture [C] //Proc of the 4th Int Conf on Web Research. Piscataway, NJ: IEEE, 2018: 133—137

[93] Rahman M, Gao Jerry. A reusable automated acceptance testing architecture for Microservices in behavior-driven development [C] //Proc of the 9th IEEE Symp on Service-Oriented System Engineering. Piscataway, NJ: IEEE, 2015: 321—325

[94] Winzinger S. Mutation testing for Microservices [C/OL] //Proc of the 10th ZEUS Workshop. 2018: 17—19 [2019-12-12]. http://ceur-ws.org/Vol-2072/paper3.pdf

[95] Zhou Xiang, Peng Xin, Xie Tao, et al. Fault analysis and debugging of Microservice systems: Industrial survey, benchmark system, and empirical study [OL]. [2019-08-01]. https://ieeexplore.ieee.xilesou.top/abstract/document/8580420

[96] Zhou Xiang, Peng Xin, Xie Tao, et al. Delta debugging Microservice systems [C] //Proc of the 33rd ACM/IEEE Int Conf on Automated Software Engineering. New York: ACM, 2018: 802—807

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

[98] Li Wenhai, Peng Xin, Ding Dan, et al. Method of Microservice system debugging based on log visualization analysis [J]. Computer Science, 2019, 46 (11): 145—155 (in Chinese) (**李文海, 彭鑫, 丁丹, 等. 基于日志可视化分析的微服务系统调试方法**[J]. **计算机科学**, 2019, 46 (11): 145—155)

[99] Rajagopalan S, Jamjoom H. App-bisect: Autonomous healing for Microservice-based apps [C] //Proc of the 7th USENIX Workshop on Hot Topics in Cloud Computing. Berkeley, CA: USENIX Association, 2015

[100] Furda A, Fidge C, Zimmermann O, et al. Migrating enterprise legacy source code to Microservices: On multitenancy, statefulness, and data consistency [J]. IEEE Software, 2017, 35(3): 63—72

[101] Di Francesco P, Lago P, Malavolta I. Migrating towards Microservice architectures: An industrial survey [C] //Proc of the 15th IEEE Int Conf on Software Architecture. Piscataway, NJ: IEEE, 2018: 29—39

[102] Taibi D, Lenarduzzi V, Pahl C. Processes, motivations, and issues for migrating to Microservices architectures: An empirical investigation [J]. IEEE Cloud Computing, 2017, 4(5): 22—32

[103] Fan Chenyuan, Ma Shangpin. Migrating monolithic mobile application to Microservice architecture: An experiment report [C] //Proc of the 1st IEEE Int Conf on AI & Mobile Services. Piscataway, NJ: IEEE, 2017: 109—112

[104] Fritzsch J, Bogner J, Zimmermann A, et al. From monolith to Microservices: A classification of refactoring approaches [G] //LNCS 11350: Proc of the 2018 Int Workshop on Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment. Berlin: Springer, 2018: 128—141

[105] Ahmadvand M, Ibrahim A. Requirements reconciliation for scalable and secure Microservice (de) composition [C] //Proc of the 24th IEEE Int Requirements Engineering Conf Workshops. Piscataway, NJ: IEEE, 2016: 68—73

[106] Mustafa O, Gómez J M. Optimizing economics of Microservices by planning for granularity level [C/OL] //Proc of the 2017 Programming Technology for the Future Web. 2017 [2020-02-14]. http://soft.vub.ac.be/~cfscholl/ProWeb17/ProWeb_2017_paper_8.pdf

[107] Hassan S, Ali N, Bahsoon R. Microservice ambients: An architectural meta-modelling approach for Microservice granularity [C] //Proc of the 14th IEEE Int Conf on Software Architecture. Piscataway, NJ: IEEE, 2017: 1—10

[108] Klock S, Van Der Werf J M E M, Guelen J P, et al. Workload-based clustering of coherent feature sets in Microservice architectures [C] //Proc of the 14th IEEE Int Conf on Software Architecture. Piscataway, NJ: IEEE, 2017: 11—20

[109] Procaccianti G. Towards a Microservices Architecture for Clouds [R]. Amsterdam: VU University Amsterdam, 2016

[110] Christoforou A, Garriga M, Andreou A S, et al. Supporting the decision of migrating to Microservices through multi-layer fuzzy cognitive maps [G] //LNCS 10601: Proc of the 2017 Int Conf on Service-Oriented Computing. Berlin: Springer, 2017: 471—480

[111] Levcovitz A, Terra R, Valente M T. Towards a technique for extracting Microservices from monolithic enterprise systems [J]. arXiv preprint arXiv: 1605.03175, 2016

[112] Chen Rui, Li Shanshan, Li Zheng. From monolith to Microservices: A dataflow-driven approach [C] //Proc of the 24th Asia-Pacific Software Engineering Conf. Piscataway, NJ: IEEE, 2017: 466—475

[113] Ren Zhongshan, Wang Wei, Wu Guoquan, et al. Migrating Web applications from monolithic structure to Microservices architecture [C] //Proc of the 10th Asia-Pacific Symp on Internetware. New York: ACM, 2018: 7.1—7.10

[114] Gysel M, Kölbener L, Giersche W, et al. Service Cutter: A systematic approach to service decomposition [G] //LNCS 9846: Proc of the 2016 European Conf on Service-Oriented and Cloud Computing. Berlin: Springer, 2016: 185—200

[115] Mazlami G, Cito J, Leitner P. Extraction of Microservices from monolithic software architectures [C] //Proc of the 24th IEEE Int Conf on Web Services. Piscataway, NJ: IEEE, 2017: 524—531

[116] Abdullah M, Iqbal W, Erradi A. Unsupervised learning approach for Web application auto-decomposition into Microservices [J]. Journal of Systems and Software, 2019, 151: 243—257

[117] Escobar D, Cárdenas D, Amarillo R, et al. Towards the understanding and evolution of monolithic applications as Microservices [C] //Proc of the 42nd Latin American Computing Conf. Piscataway, NJ: IEEE, 2016: 89.1—89.11

[118] Baresi L, Garriga M, De Renzis A. Microservices identification through interface analysis [G] //LNCS 10465: Proc of the 2017 European Conf on Service-Oriented and Cloud Computing. Berlin: Springer, 2017: 19—33

[119] Kecskemeti G, Marosi A C, Kertesz A. The ENTICE approach to decompose monolithic services into Microservices [C] // Proc of the 8th Int Conf on High Performance Computing & Simulation. Piscataway, NJ: IEEE, 2016: 591—596

[120] Kalske M, Mäkitalo N, Mikkonen T. Challenges when moving from monolith to Microservice architecture [G] // LNCS 10544: Proc of 2017 Int Conf on Web Engineering. Berlin: Springer, 2017: 32—47

[121] Balalaie A, Heydarnoori A, Jamshidi P. Migrating to cloud-native architectures using Microservices: An experience report [C] //Proc of 2015 European Conf on Service-Oriented and Cloud Computing. Berlin: Springer, 2015: 201—215

[122] Balalaie A, Heydarnoori A, Jamshidi P, et al. Microservices migration patterns [J]. Software: Practice and Experience, 2018, 48(11): 2019—2042

[123] Balalaie A, Heydarnoori A, Jamshidi P. Microservices architecture enables DevOps: Migration to a cloud-native architecture [J]. IEEE Software, 2016, 33(3): 42—52

[124] Zdun U, Navarro E, Leymann F. Ensuring and assessing architecture conformance to Microservice decomposition patterns [G] //LNCS 10601: Proc of the 2017 Int Conf on Service-Oriented Computing. Berlin: Springer, 2017: 411—429

Wu Huayao, born in 1989. PhD, assistant researcher. His main research interests include software testing and analysis.

Deng Wenjun, born in 1995. Master candidate. His main research interest is microservice.

2020 年《计算机研究与发展》专题(正刊)征文通知
——密码学与数据隐私保护研究

如今信息化社会给人们的工作和生活带来极大变革,各种业务平台、各种学习平台、各种购物网站及各种信息资源服务成为人们生活中非常重要的一部分.但同时,各行各业、各种机构积累的数据越来越多.如果不能保证这些数据的安全性和应用的合理性,必将成为社会发展的巨大隐患.为了保障数据的安全与隐私,同时合理有效地对数据进行利用,发挥数据的价值,密码技术具有支撑性作用.面对大数据、物联网、区块链和人工智能应用环境,密码理论与数据隐私保护方法需要新的机制与新的思想,需要更有效的手段.

为推动我国学者在密码学与数据隐私保护领域的研究,及时报道我国学者在密码学与数据隐私保护研究方面的最新研究成果,《计算机研究与发展》将于 2020 年 10 月出版网络与信息安全专题——密码学与数据隐私保护研究,该专题主要聚焦密码学的基础性研究、数据隐私保护方案的设计与分析,以及大数据、物联网、区块链和人工智能等领域的数据隐私保护研究和应用,欢迎相关领域的专家学者和科研人员踊跃投稿.

征文范围　本专题包括(但不限于)下列主题:

- 密码学基础理论
- 数据隐私保护一般理论与方法
- 具体方案的设计与分析
- 密码分析
- 大数据隐私保护
- 物联网、区块链与人工智能安全与数据隐私保护

征文要求

1) 论文应属于作者的科研成果,数据真实可靠,具有重要的学术价值或推广应用价值,未在国内外公开发行的刊物或会议上发表过,不存在一稿多投问题.作者在投稿时,需向编辑部提交版权转让协议.
2) 论文一律用 Word 格式排版,格式体例参考近期出版的《计算机研究与发展》文章的要求(http://crad.ict.ac.cn).
3) 论文请通过期刊网站(http://crad.ict.ac.cn)进行投稿,投稿时提供作者的联系方式,并在作者留言中注明"网络与信息安全 2020 专题"(否则按自由来搞处理).

重要日期

征文截止日期: 2020 年 6 月 11 日　　录用通知日期: 2020 年 7 月 20 日
作者修改稿提交日期:2020 年 8 月 6 日　　出版日期:2020 年 10 月

特邀编委

徐秋亮　教授　山东大学　xql@sdu.edu.cn
张玉清　教授　中国科学院大学、国家计算机网络入侵防范中心　zhangyq@ucas.ac.cn
董晓蕾　教授　华东师范大学　dongxiaolei@sei.ecnu.edu.cn

联系方式

编　辑　部:crad@ict.ac.cn, 010-62620696, 010-62600350
通信地址: 北京 2704 信箱《计算机研究与发展》编辑部　　邮编:100190