计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2020.20190460
57(5): 1103-1122, 2020

微服务技术发展的现状与展望

冯志勇 徐砚伟 薛 霄 陈世展
(天津大学智能与计算学部 天津 300350)
(xuyanwei@tju.edu.cn)

Review on the Development of Microservice Architecture

Feng Zhiyong, Xu Yanwei, Xue Xiao, and Chen Shizhan
(*College of Intelligence and Computing, Tianjin University, Tianjin* 300350)

Abstract With the rapid development of cloud computing and Internet of things, users' demand for software systems tends to be diversified. Service oriented architecture (SOA) needs to strike a balance between stable service integration and flexible adaptation of requirements. Based on this situation, the microservice technology, which goes with independent process as well as independent deployment capability, emerges as the times require. It has a slew of advantages, such as distributed storage, high availability, scalability, and intelligent operation maintenance, which can make up for the shortcomings of the traditional SOA architecture. From the perspective of system integration, the paper firstly describes the application background of microservice, which include the core components of microservice, software technology development and architecture evolution to ensure the availability of microservice infrastructure. Secondly, in view of problems existing in practical applications, the paper analyzes the key technologies utilized in the specific application of the microservice architecture through the aspects of distributed communication, distributed data storage, distributed call chain, and testing complexity; then, a specific application case is given to confirm the technical feasibility of microservice. Finally, this paper intends to explore the challenges by microservice through the aspects of infrastructure, information exchange, data security, and network security. Meanwhile, the future development trend is analyzed so as to provide valuable theoretical and technical reference for the future innovation and development of microservice.

Key words service oriented architecture; microservice; system integration; microservice architecture; key technologies

摘 要 随着云计算、物联网等技术迅速发展,用户对软件系统的需求趋于多样化,面向服务的体系架构(service oriented architecture, SOA)需要在服务稳定集成与需求灵活适配之间寻求平衡.基于此,拥有独立进程、具备独立部署能力的微服务技术应运而生,它具有分布式存储、高可用性、可伸缩性、运维智

收稿日期:2019-07-01;**修回日期**:2019-11-13
基金项目:国家重点研发计划项目(2017YFB1401200);国家自然科学基金项目((61972276,61572350,41701133,61832014);河南省教育厅杰出青年科学基金项目(174100510008);河南省基础与前沿技术研究计划面上基金项目(162300410121)
This work was supported by the National Key Research and Development Program of China (2017YFB1401200), the National Natural Science Foundation of China (61972276, 61572350, 41701133, 61832014), the Science Fund for Distinguished Young Scholars of Henan Provincial Education Department (174100510008), and the General Program of Fundamental and Frontier Technology Research Plan of Henan Province (162300410121)
通信作者:薛霄(jzxuexiao@tju.edu.cn)

能化等优势,能够弥补传统 SOA 的缺陷.首先,从系统集成角度的出发,阐述微服务出现的应用背景,利用微服务的核心组件、软件技术发展、架构演化等基础技术,以保证微服务基础设施的可用性;其次,基于微服务体系架构在实际应用中的问题,从分布式通信、分布式数据存储、分布式调用链、测试的复杂性等方面,分析微服务体系架构具体应用中采用的关键技术,并给出具体应用案例,以保证微服务的技术可行性;最后,从基础设施、信息交互、数据安全与网络安全等方面探寻微服务所面临的诸多挑战,并分析未来发展趋势分析,以期为微服务未来的创新和发展提供有价值的理论与技术参考.

关键词 面向服务的体系架构;微服务;系统集成;微服务体系架构;关键技术

中图法分类号 TP303

随着云计算、物联网、服务计算等技术快速发展,社会生产生活对软件系统的需求量与日俱增,用户需求的多样性、个性化趋势日趋凸显.为了满足不断演化的用户需求,许多软件企业开放其软件产品线,允许上下游相关企业、外部开发者甚至用户参与到软件开发维护工作之中,进而有效加快软件产业的垂直分工和水平整合,促进了软件生态系统(software ecosystems, SECO)[1]的丰富和完善.

随着软件功能不断扩展、用户负载量逐渐增长等发展问题,软件生态系统中模块与组件之间的调用依赖关系也变得越来越复杂.在这种背景下,新组件的引入与迭代、数据的快速更新,无疑会造成软件生态系统的不稳定、不平衡.为了确保系统满足高可用、高并发的要求,系统架构需要根据用户需求合理配置资源,以保证资源利用率最大化.具体而言,从系统集成的角度,软件生态系统的演化过程可以大致分为 3 个阶段,各阶段特点如图 1[2]所示.

1) 点对点集成

点对点集成阶段通常产生在系统发展初期,其演化和扩展的方式为 1 对 1 集成.在企业应用系统个数较少的场景下,系统通过提供的对应接口,实现系统间的数据传输.当应用系统数量达到一定程度时,系统间的接口众多,会导致大量重复开发和联调工作,接口的开发难易程度和维护成本会随之增大.因此,整个软件生态系统的扩展能力较弱,其演化方式也显得笨重且僵硬.在此种情况下,面向服务体系架构(service-oriented achitecture, SOA)[3]的集成理念应运而生.

2) 平台集成

SOA 是一种软件系统的设计方法,以松耦合的方式,将应用系统的不同服务功能进行拆分,通过服务之间定义良好的接口实现服务集成.SOA 理念[4]的出现使企业开始从整体角度看待 IT(information technology)架构的建设,更加强调自上至下的整体架构.同时,企业服务总线(enterprise service bus, ESB)[5]的兴起,为服务整合提供行之有效的解决方案,ESB 在数据集成、应用集成、流程集成、门户集成等方面具有独特的优势,可以有效地简化 IT 系统结构,提高系统的灵活性和可扩展能力.

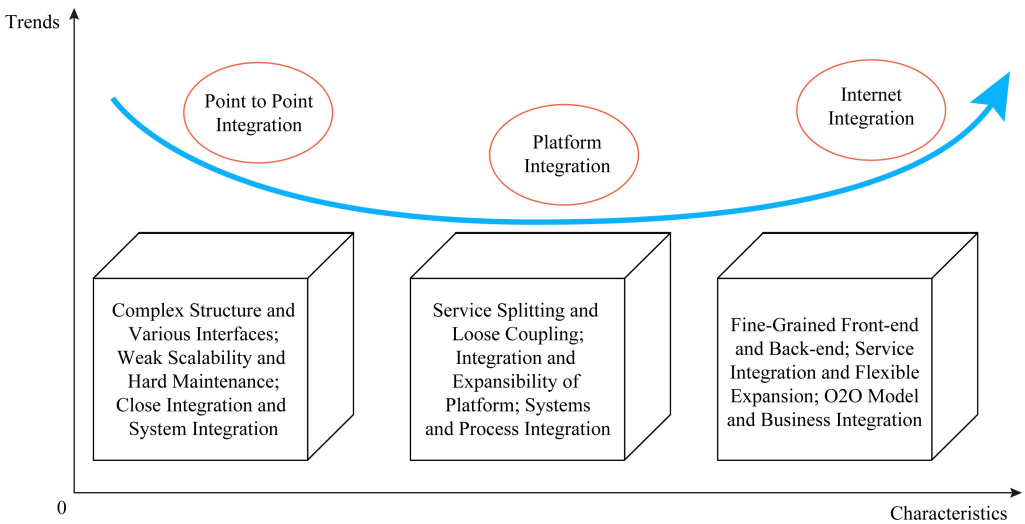


Fig. 1 Integrated evolutionary graph of software ecosystems

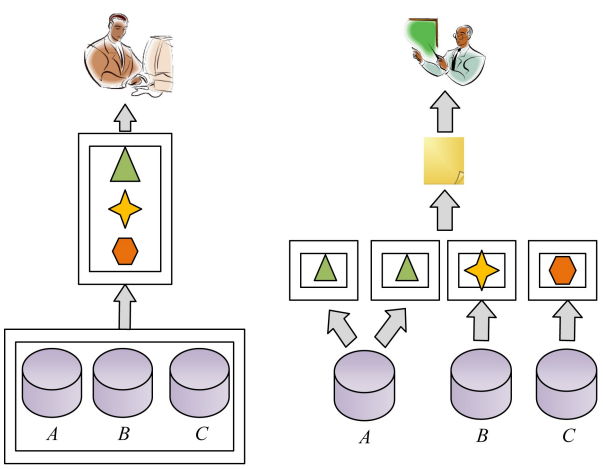
图 1 软件生态系统集成进化图

3) 互联网集成

随着移动互联网与互联网+的发展,原有的SOA体系架构遇到了4个问题:①缺乏有效的服务治理,服务资产混杂不清,没有有效的服务管控手段;②业务支撑响应慢,系统尾大不掉,无法做到实时更新和模块化发布;③系统可用性差,无法做到7×24 h无间断提供服务;④创新业务难以支撑,特别是带有互联网特点的创新业务.

基于此,开发人员以体系结构优化为出发点,提出了基于微服务的SOA体系架构[6];其核心理念是将复杂的应用系统以独立业务单元的形式分解为多个服务,每个服务可以采用不同的实现技术,以轻量级、更灵活的模式进行独立设计、开发、部署,运行于独立的进程中,形成高度内聚的自治单元[7].

具体而言,SOA把系统划分成不同的服务,使用接口进行数据交互,服务之间通过相互依赖、有效整合,以达到系统的整体功能[8].而在体系架构方面,微服务架构是基于SOA架构基础上的改进,并且融入组件化思想与领域建模思想.首先,系统被拆分为多个微服务,以松耦合的方式被独立部署;其次,每个微服务仅需高质量地完成本身任务,且每个任务代表着一项细粒度业务能力;由此,各项业务被彻底的组件化和服务化[9];最后,提供领域服务能力的模块在底层微服务架构中实现服务的组合和组装.如图2(a)所示,整体架构将所有软件功能放在一个进程中,由多个服务器共同支持运行计算任务,最后将运行结果返还给用户;而微服务架构(图2(b))将软件整体功能分解成多个服务,分别由不同类别的服务器进行支持;然后,将数据反馈给数据库,用户可以从数据库中获取数据,既加快了系统的整体响应速度,又满足互联网化环境下前后端分离的业务需求.由此可见,微服务体系架构的优势[10]主要体现在:分布式(物理部署、服务部署、数据存储)、高可用(分布式架构、集群化部署、服务自动注册)、可伸缩(按需分配资源)、运维智能化等方面.



(a) Holistic architecture (b) Microservice architecture

Fig. 2 Holistic architecture and microservice architecture

图2 整体架构与微服务架构

随着软件系统功能的日益扩展复杂化,基于微服务的SOA架构开始日益兴起.具体而言,微服务在前台支撑业务的快速响应与个性化,是采用面向服务开发的SOD(service-oriented development),具有开发快速、响应及时、易于实现等特点[11];ESB服务总线作为后端支撑各系统、技术、平台的集成,而面向服务的基础设施(service-oriented infrastructure, SOI)[12],具有稳定、高度集成等特点.微服务技术与ESB技术二者相辅相成,分别在SOA架构中发挥不同的核心作用,以期达到软件系统的自服务效果.

本文首先以系统集成角度,阐述微服务出现的特定背景,并逐渐深入探讨微服务核心组件以及微服务技术与架构发展;其次,详细介绍近年来微服务系统所采用的关键技术;最后,对微服务所面临的挑战与未来发展趋势进行了分析.

1 微服务的核心组件

根据Lewis等人[13]的阐述,软件架构研讨会(Seminar on Software Architecture)于2014年5月首次定义了"微服务"这一术语,用来表示诸多学者一直探索的系统共同架构方法.此前,部分学者已经使用不同的方法实现了该框架.例如亚马逊公司Vogels等人[14]将该种架构方法描述为"将直接操作数据的业务逻辑来封装数据,通过已发布的服务接口进行唯一访问",Netflix公司Cockcroft[15]将其定义为"松散耦合的面向服务的体系结构与有限制的信息交换".工业界中相关概念有"fine-grained SOA"和"SOA done right"等方法.

从字面去理解微服务,即什么是"微"、什么是"服务","微"狭义上来讲就是体积小、架构小,而"服务",区别于整体系统的概念,是一个或者一组相对较小且独立的功能单元,是用户可以感知到的最小功能集;每个微服务具有自己的轻量处理和通信机制,并且自动化部署在单个或多个服务器上.其中,微服务概念来源领域驱动设计(domain-driven design, DDD)[16],即是一种基于模型的开发方法,

由有限组织环境和持续软件集成等原则进行指导.该设计统一了分析和设计编程,使得软件能够更灵活快速跟随需求变化.解决了分布式 Web 规模应用程序(Facebook,Spotify 等)中存在的挑战以及大型科技公司面临的组织问题.

微服务为分布式软件系统提供了良好的解决方案.与传统面向服务体系结构的实施方案相比,微服务体系结构除了提供服务注册与发现,服务组合等基本组件外,还加入了负载均衡、服务网关和容错机制等,同时对已有模块进行了扩展和优化;图 3 是对微服务框架相关元素[17]及元素内容的具体展示.将结合图 3,从 5 个方面介绍微服务核心组件.

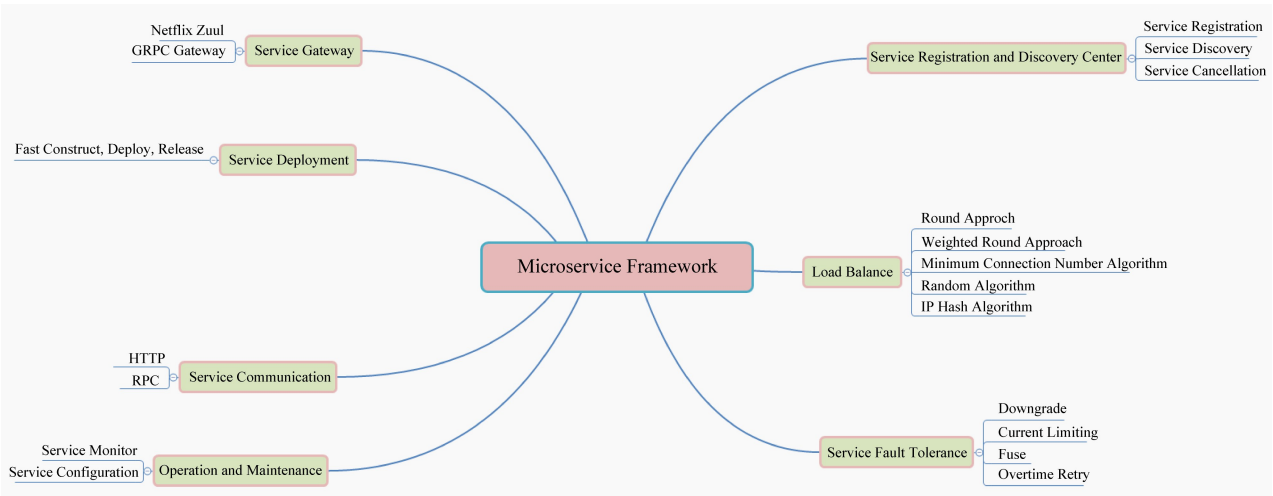


Fig. 3 Microservice core components

图 3 微服务核心组件

1.1 服务发现机制与注册中心

微服务遵循轻量级通信原则,单个微服务一般部署在轻量级容器如 Docker 中.然而,在运行过程中,服务实例随时可能被销毁、克隆或者重新定位;由此,服务实例在动态变化中,创建一种服务发现机制,有利于服务之间感知彼此的存在.其中,服务注册中心[18]是服务发现机制中重要的一环,即服务启动时会将自身的网络地址与数据提交到注册中心,并订阅自己需要消费的服务.

服务注册中心是服务发现的核心,必须具有高可用性和实时更新功能;主要存储服务提供者和消费者的统一资源定位器(uniform resoure locator,URL)地址及路由转发信息;实现服务注册、发布、健康检查和故障检测等功能.表 1 对目前较为流行的服务注册中心进行了分析与归纳.其中,在 Key-Value,Kubernetes,Cloud Foundry 等应用平台中使用的 Etcd 具有分布式、高可用性以及强一致性特征,适合于少量数据的情形.而由 Google 公司开发和维护注册中心 Consul 功能更为全面,作为一个用于发现和配置的工具,Consul 提供了允许客户端注

Table 1 Service Registry Comparisons

表 1 服务注册中心比较

Service Registry	Etcd	Consul	Zookeeper	Euerka
Main Algorithm	Raft	Raft	Paxos	
Health Examination		Support	Support	Support
CAP Theory	CP	CA	CP	AP
Support Framework	GRPC	Motan/GRPC..	Dubbo/Motan/GRPC..	Spring Cloud
Multiple Data Centers		Support		
Self Monitoring	Metrics	Metrics		Metrics
Security	HTTP (weak)	ACL/HTTPS	ACL	
Spring Cloud Integration	Support	Support	Support	Support

册和发现服务的 API,而其提供的服务健康检查,可以帮助确定服务可用性.此外,具有高协调能力的 Zookeeper 在分布式系统中应用较为广泛[19],通过提供统一命名服务、集群管理、状态同步、分布式应用配置与管理等功能,解决了分布式系统中数据一致性问题.

1.2 负载均衡

为了保证服务具有高度可用性,微服务需要部署多个服务实例来提供业务支持;当请求面对同一服务的多个实例,如何合理选择服务实例以减少业务等待时间成为一个亟待解决的问题,由此负载均衡可以分为客户端负载均衡与服务端负载均衡,以选择合理的服务负载均衡策略.

负载均衡具有多种实现算法,其中应用最广泛来自著名的 Round Robin 算法,即轮询法[20];其基本思想是将多个可用服务实例组织成一个循环队列,然后根据实例顺序轮流分配给内部的服务器,从 1 到 N(N 个服务器)依次循环;该方法适用于基本配置相同的服务且服务平均请求相对均衡的情境;然而,在面临性能方面差异较大的服务实例时,一般采用加权轮询法[21],即根据服务器的不同处理能力,给每个服务器分配不同权重,响应具有相应权值数的服务请求;该均衡算法可以提升高性能服务器的利用率,降低低性能服务器负载过重的概率,避免负载不均衡的情况.但在实际应用中,客户端每一次请求服务,服务器响应时间都具有较大差异;因此,若采用简单轮询或随机均衡算法,并不能达到真正的负载均衡.鉴于这一缺点,可采用最小连接数算法(least connections scheduling, LCS)[22];该算法记录当前服务器可负载实例数量,以及该服务器正在处理的进程数量;具体而言,当产生新服务连接请求时,将把当前请求分配给连接数最少的服务器,以提升服务实例利用率以及服务器负载能力.

微服务架构均支持以上负载均衡算法.与传统整体架构负载均衡不同的是,传统整体架构使用负载均衡器分发高并发的网络请求[23].在微服务架构中,服务端的软件模块维护一个可用的服务端清单;客户端节点也需维护本身所访问的服务端清单,而这份服务端清单来自于(微服务架构中独有)服务注册中心,例如 Eureka 服务注册中心;同时,客户端需要维护服务端清单的健康性,也需与服务注册中心配合完成[24].其中,SpringCloud Ribbon 是微服务架构中基于客户端的负载均衡工具,将面向服务的 REST(representational state transfer)模板请求自动转换成客户端负载均衡的微服务调用[25].

此外,微服务架构支持的负载均衡算法还包括随机分配服务器算法、生成请求源 IP Hash 值方式,精确找到服务器的 IP Hash 算法等相关算法.这里不再一一赘述.

1.3 服务容错

容错,即将系统错误产生的影响限制在一定边界内.在微服务体系结构中调用集群服务时,若单个微服务调用异常,产生如连接超时、请求失败、流量突增或负载过高等问题,则需要制定容错策略进行容错处理,使微服务具有自我恢复功能.

服务容错分为 2 种情况:1)若产生超时异常,可采用超时重试机制[26],通过设置服务请求超时响应时间;或者服务的响应时间和次数,进而决定是否采用超时重试机制.2)若服务因负载过高引起异常,可采用限流和熔断器 2 种容错策略.其中,限流是以限制服务的最大访问量或者访问速率的方式,对服务进行容错处理,熔断器会记录和监测服务执行情况;若监测到某个服务实例超过阈值,可拒绝接收服务请求将其直接返回.目前,微服务框架支持的容错策略还有控制并发、线程隔离等策略;如果连续失败多次则直接熔断,不再发起调用,避免单个服务异常影响系统中整体服务的运行.

1.4 服务网关

服务网关(service gateway)作为微服务架构中的重要组件,其关键思想是:将轻量级网关作为所有客户端/消费者的主要入口点,并在 Gateway(网关)级别实现常见的非功能需求.服务网关的基本功能有:统一接入、安全防护、协议适配、流量管控、长短链接支持、系统容错能力等.目前已有许多成功的应用案例.例如,由 Netflix 公司开发的 Netflix Zuul[27]是目前较通用的服务网关组件;其主要作用是协调客户端与微服务的中间层,提供权限验证、压力测试、负载分配、审查监控等较为全面的服务网关功能.其中,Zuul 主要负责处理 RESTful 的服务请求及调用.然而,在部分微服务业务场景下,仍存在“外部客户端是 RESTful 的接口请求,而内部服务之间却是 RPC 通信”的情况.因此,产生同一系统具有 2 套不同类型的 API 接口,无疑增加了通信的复杂度.由此,GRPC Gateway 通过读取 GRPC 服务请求并为其生成反向代理服务器,将 RESTful 的 HTTP/JSON API 接口转化为内部 GRPC 的形式,从而解决了服务内外接口不兼容这一问题[28].其他网关解决方案这里不再赘述.

1.5 服务部署和服务通信

作为微服务框架核心部件之一,微服务部署和

服务通信具有至关重要的作用.微服务部署中关键问题之一是如何做到独立于其他微服务部署,使每个微服务级别都可以进行部署与扩展.从而在单个微服务的故障不影响任何其他服务前提下,快速构建和部署微服务,Docker[29]作为一种开源应用容器引擎,以应用打包以及依赖包到一个可移植容器中的方式,达到上述功能需求.其具体理念是:将每个服务实例部署为容器,微服务作为 Docker 容器镜像打包,根据容器实例数量变化进行缩放.在 Docker 容器部署过程中,使用 Kubernetes 应用平台组件,将一组 Linux 容器作为单个系统进行管理,在多个主机上管理和运行 Docker 容器;根据容器的部署位置,进行服务发现和复制控制等机制,以期对 Docker 功能进行扩展与伸缩.基于此种方式,构建、部署和启动微服务的速度将会大大提升.其中,Kubernetes 技术为 Docker 容器部署微服务提供了强有力的支持.

服务通信是指网络传输过程中,服务之间进行信息交互或消息传递.其关键是保证具有良好的通信机制,实现准确、高效的信息交换.在微服务框架中,微服务通信方式分为同步和异步 2 种模式.在同步通信模式下,客户端发出请求,服务器即时响应.这种通信模式实现较为简单,没有中间件做代理,一般采用 REST 和 Thrift 两种协议.其缺点是:通信机制较为单一,制约了进程的速度,在一定程度上降低了系统的可用性.在异步通信模式中,客户端请求不会阻塞进程,服务端中响应可以是非即时.其优点是实现了客户端与服务器之间的松耦合,通过中间件进行消息缓冲,实现灵活交互,提高了系统可用性;缺点是基于请求/响应交互模式的复杂性大大提高,为开发人员带来大量额外工作.在主流系统框架中,一般采用同步、异步混合通信模式以提高系统可伸缩性以及系统可用性.

2 微服务技术的发展过程

从微服务最初定义到逐步被软件系统平台应用,微服务技术在不断发展进步,以适应平台中数据量大、更新迭代快的特点.下面以微服务软件技术发展与微服务架构发展 2 个方面,揭示微服务技术的不断演变.

2.1 微服务软件技术发展

早期微服务应用程序受到新一代软件开发、部署和管理工具的影响较大.然而,随着微服务架构应用越来越广泛,管理工具不断发展创新,目前微服务可以支持更庞大、更加多样化的用户数据群.图 4 显示了 10 种“waves”软件技术发展的时间表[30];其中包括应用较为广泛的工具,这些工具在过去 10 年中极大地影响了微服务应用程序开发、部署和运行等方面.

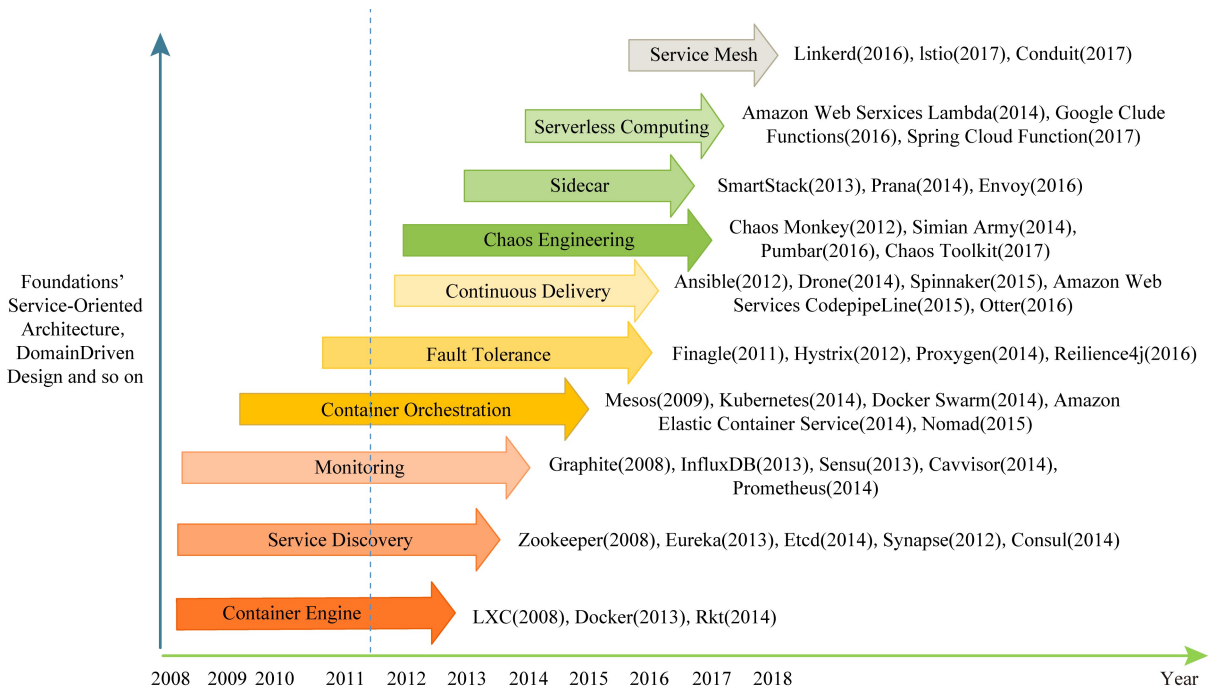


Fig. 4 Development of microservice technology

图 4 微服务技术发展历程

在“微服务”概念提出之前,业界已经诞生5次相关新技术[30].

1) 轻量级容器(lightweight container engine)技术(例如LXC和Docker),允许在系统运行时有效地对各个服务打包、部署和管理.

2) 服务发现(service discovery)技术(例如Eureka和Consul),允许服务彼此通信而无需明确地参考服务网络位置.

3) 监控(monitoring)技术(例如Graphite和Sensu),能够在不同层次程度上监控和分析微服务资源的行为.

4) 容器编排(container orchestration)技术(例如Mesos和Kubernetes),自动化分配容器和管理任务,开发人员从底层架构中抽象出底层物理或虚拟基础架构,为开发人员提供一个抽象层,以确定应用程序部署在服务器以及数据中心中.

5) 建立延迟和容错(default tolerance)通信库(例如Finagle和Hystrix),使服务进行更有效地执行,更可靠地通信.

后5次系统软件管理方法的提出,为微服务系统实施提供了底层技术支撑.

6) 包括持续交付(continuous delivery)技术(例如Ansible和Drone)即指通过软件自动化的方式,快速迭代发布软件;允许团队频繁地交付快速更新的软件产品.其特征是持续整合、内置测试、持续监控和分析反馈.为系统提供了通用集成的解决方案[31],并且在Web级微服务生产环境中已经进行DevOps实践.

7) 混沌工程(chaos engineering)技术(例如Simian Army和Chaos Toolkit)[32]可自动解决系统中出现的故障,具有高可靠性与高可用性等特性.

8) 边车(sidecar)技术(例如Prana和Envoy)[33],封装与通信相关的功能.例如服务发现以及特定协议和容错通信库的使用功能,使服务开发人员快速获得通信消息.

9) 包括“无服务器”计算(serverless computing)技术(例如AWS Lambda,OpenWhisk,Amazon Web Services)[34],实现了“功能即服务”(functions as a service, FaaS)云模型.这种模式允许云用户开发、部署及交付更精细的服务功能,而无需创建和管理(如应对不一致的流量模式)复杂基础模块执行所需的资源.

10) 服务网格(service mesh)技术(例如Linkerd和Istio)[35],以边车技术为基础,提供完全集成的服务通信监控.

图4中大部分工具都源于工业应用领域.作为开源项目提供给用户下载与应用.其中,表2给出了每个工具的URL网址[30].

Table 2 The Web Site of the Microservice Tool in Figure 4

表2 在图4中微服务工具的网址(URLs)

Implementation Tool Category	Implementation Tool	URL
Container Engine	LXC	linuxcontainers.org
	Docker	www.docter.com
	Rkt	coreos.com/rkt
Service Discovery	ZooKeeper	zookeeper.apache.org
	Eureka	github.com/Netflix/eureka
	Etcd	coreos.com/etcd
	Synapse	github.com/airbnb/sysnapse
	Consul	www.consul.io
Monitoring	Graphite	graphiteapp.org
	InfluxDB	github.com/.influxdata/influxdb
	Sensu	sensuapp.org
	Cavvisor	github.com/google/cadvisor
	Prometheus	prometheus.io
Container Orchestration	Mesos	mesos.apache.org
	Kubernetes	kubernetes.io
	Docker Swarm	docs.docker.com/engine/swarm
	Amazon Elastic Container Service	aws.amazon.com/ecs
	Nomad	www.nomadproject.io

Continued (Table 2)

Implementation Tool Category	Implementation Tool	URL
Fault Tolerance	Finagle	twitter. github.io/finagle
	Hystrix	github.com/Netflix/Hystrix
	Proxygen	github.com/facebook/proxygene
	Reilience4j	github.com/resilience4j/resilience4j
Continuous Delivery	Ansible	www.ansible.com
	Drone	drone.io
	Spinnaker	www.spinnaker.io
	Amazon Web Services CodepipeLine	aws.amazom.com/cedepipeline
	Otter	inedo.com/otter
Chaos Engineering	Chaos Monkey	github.com/Netflix/chaosmonkey
	Simian Army	github.com/Netflix/SimianArmy
	Pumba	github.com/alexei-led/pumba
	Chaos Toolkit	chaostoolkit.org
Sidecar	SmartStack	nerds.airbnb.com/smartstack-service-disconvery-cloud
	Prana	github.com/Netflix/Prana
	Envoy	www.envoyproxy.io
Serverless Computing	Amazon Web Serxices Lambda	aws.amazong.com/lambda
	Azure Function	azure.microsoft.com/services/functions
	Google Clude Functions	cloud.google.com/functions
	OpenWhisk	openwhisk.apache.org
	Spring Cloud Function	cloud.spring.io/spring-cloud-function
Service Mesh	Linkerd	linkerd.io
	Lstio	istio.io
	Conduit	conduit.io

2.2 微服务架构的发展

在工业应用中,一个完整的微服务架构由多个元素构成,它具有独立的生态圈;不但需要基础设施配合,还需生产环节部门共同协作;不但在系统运行时进行管理控制,而且还要具有安全保障的服务治理.其中,微服务治理从 4 个步骤[36]进行:1)请求网关.常用有 Zuul,Spring Cloud Gateway 等组件.2)信息采集.具有服务注册发现、服务日志、链路追踪等功能.3)信息分析.具有时序性统计、监控平台以及监控报警等功能.4)治理策略.具有负载均衡、请求限流、服务容错与服务配置等功能.在传统微服务架构中,可能需要链接不同的容器和主机,以使多个微服务共同协作[37]完成一项业务.在此种架构中,若使用传统的日志定位出现问题的容器和主机,不仅会耗费大量的人力与物力,而且可能导致系统运行失误.

在此种情境下,需要在技术层面提供分布式调用链跟踪能力,以能够快速定位出现问题的节点.同样,在微服务架构中,原来整体式系统分解成多个微服务,在此种情况下,传统的逐步式部署方式已不再适合.自动化部署方式成为软件系统所面临的必然选择.此外,出于服务治理和系统安全考虑,需要对分散的微服务进行集中管控,因此,服务网关也是必不可少的组件.这些组件构成是由微服务特点所决定的.具体而言,微服务架构生态图[38]如图 5 所示.同时,由图 5 中微服务相关技术逐步发展,这些技术革新的影响在微服务应用程序架构发展中有明显体现.

图 6 概括了 4 代微服务体系架构[30].其中,在第 1 代结构中,如图 6(a)所示,使用轻量级容器技术(如 LXC)打包每个服务,然后使用容器编排工具(如 Mesos)在运行时进行部署和管理.每个服务都负责跟踪其他服务的位置,并且根据特定的通信协议调用其他服务;任何故障处理机制(例如重试和回退)都直接在服务的源代码中产生深远的影响.随着

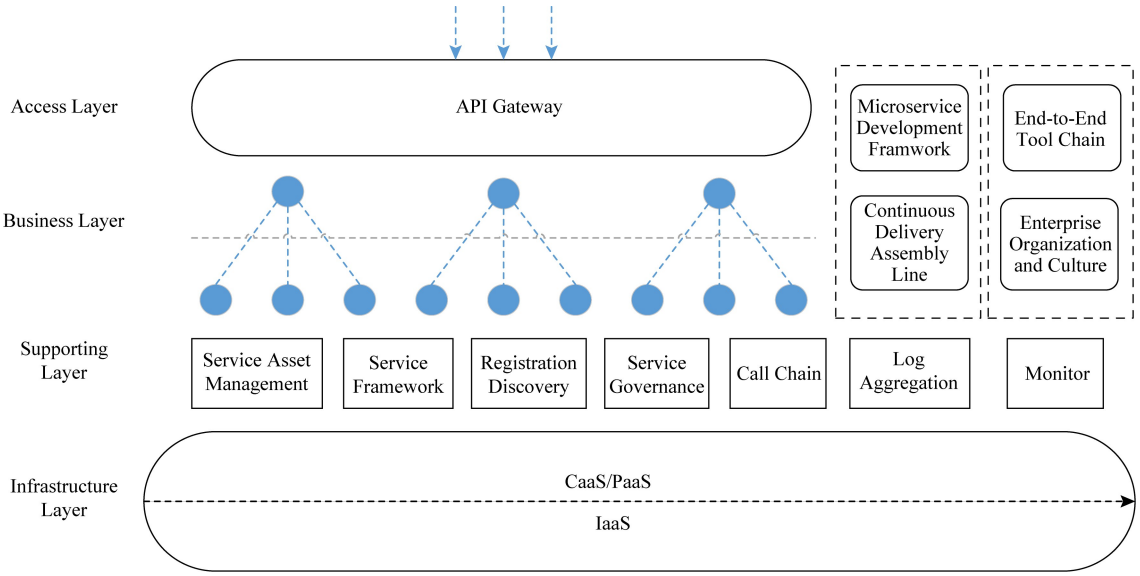


Fig. 5 Microservice architecture ecological graph

图 5 微服务体系架构生态图

应用程序中服务数量的增加,应对不同执行环境中部署和重新部署服务日益增长的需求,调用正确的服务实例成为一个难题.此外,新服务的实现语言不尽相同,因此重用现有代码以及快速有效地处理故障变得愈加困难.

为解决上述问题,第 2 代引入了服务和可重复利用的容错通信库,如图 6(b)所示服务使用公共发现服务(例如 Consul)来注册其提供的功能.客户端服务可以动态发现和调用这些功能,而无需明确被调用服务的位置.在服务调用期间,所有特定协议和故障处理功能被委托给相对应通信库;例如 Finagle;该策略不仅简化了服务实现和测试,还允许跨服务重复利用的样板通信代码.

然而,随着通信库功能愈加复杂,利用不同编程语言对通信功能重新实现变得越来越困难.开发人员经常被迫使用当前程序编程语言来实现新服务,无疑增加服务接口的难度.由此,微服务可以使开发人员自由编写程序,开发人员可以自主选择任何编程语言,或以适合特定服务的需求选择相对应开发编程语言,这是之前技术所不具备的优点.

因此,第 3 代引入标准服务代理或 sidecars 边车模式,如图 6(c)所示,例如 Envoy 作为可检测的服务中间体.基本思想是通过边车封装所有服务发现和通信功能,从而提高软件可重用性.由于每个边车都是一个独立的服务,因此该策略将现有容错通信库的优势带到新的编程语言,从而大大提高了自主开发性.

当作为网络中介时,边车成为监控微服务应用程序中所有服务信息交互行为的场所.这些工具扩展自包含边车理念,以提供更加集成的服务通信解决方案.应用运营商可以集中控制平面动态监控和管理多个分布式边车的行为.基于此种方式,运营商可以对各种服务及服务通信功能进行更加细粒度的控制,包括服务发现、负载平衡、容错、消息路由和安全性等功能.

第 4 代旨在将微服务应用程序引入一个全新的应用领域,如图 6(d)所示基本思想是利用最新的 FaaS 技术和无中断计算技术,简化微服务底层开发和持续交付[39]操作.这种无服务器架构,其基本思想是将微服务应用程序变成“短暂”函数的集合,每个函数根据特定需求,进行快速、创建、更新、替换和删除等操作,从而极大地提高微服务的运行、部署等操作效率.当然,这种无服务器架构仍然需要以通信为中心的技术,例如边车技术和服务网格.现有 FaaS 平台并未提供融合 2 种技术的通信和流量管理功能.因此,若在无服务器应用程序中创建函数到函数的交互,一个具有更全面控制功能的服务(或功能)网格应被创建,以便监控和管理这些边车功能的行为.

3 微服务关键技术

微服务强调服务功能的大小,但在实际应用中并没有统一标准.一个功能相对简单的微服务在实际需求与扩展中会变得更加复杂,并且同一业务由

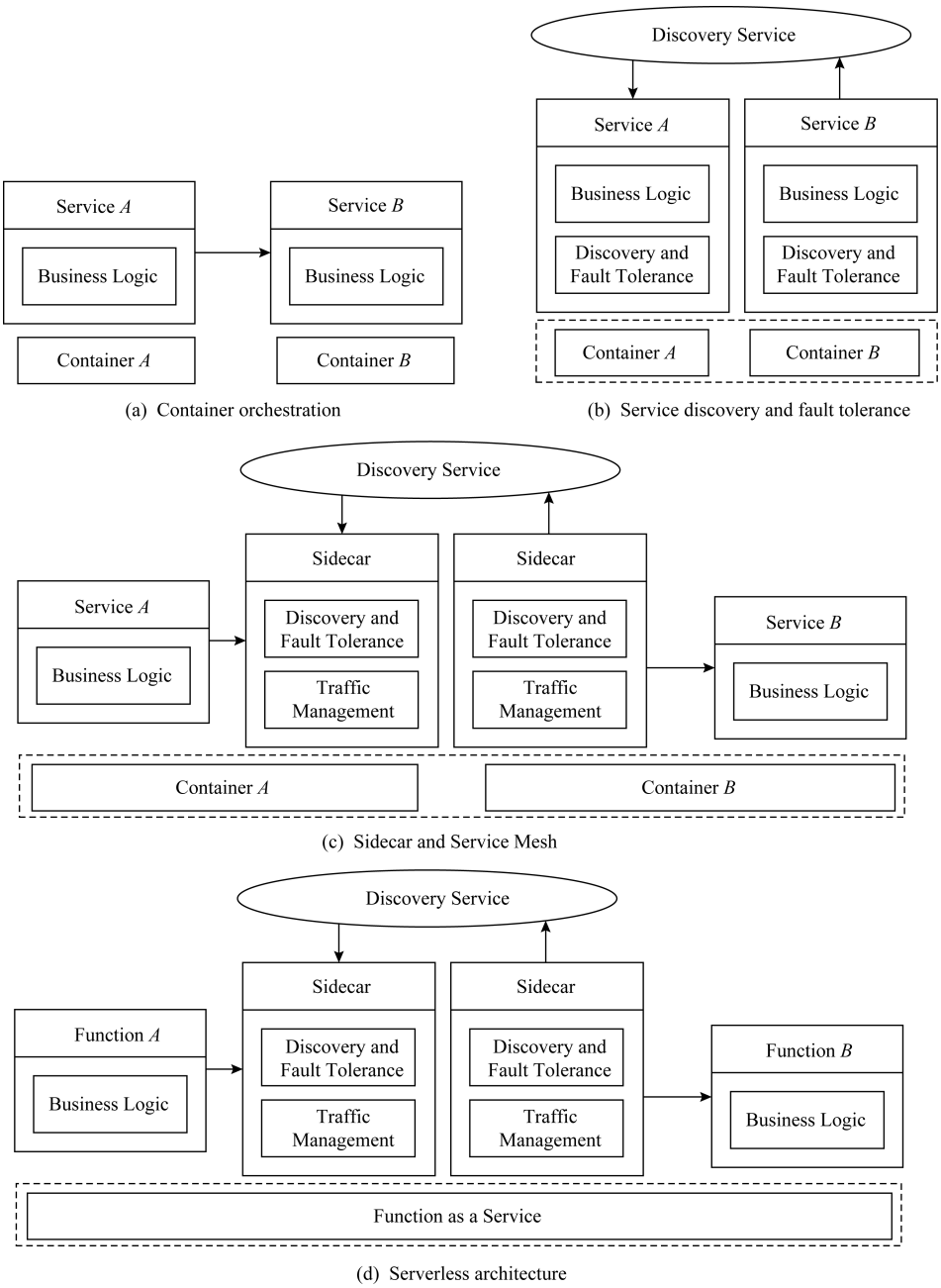


Fig. 6 Four microservice architectures

图 6 4 种微服务体系架构

多个微服务共同协作完成.由此,微服务存在 3 个问题:1)在分布式协作方面,存在微服务之间通信、分布式数据存储一致性等问题;2)在微服务定位方面,如何快速定位出现的性能瓶颈.3)在测试方面,众多的微服务如何协调一致,保证测试的准确性.基于这 3 个问题,微服务的顺畅运行主要采用一些关键技术.

3.1 微服务分布式通信

首先,在功能相互依赖的大型网络系统中,服务之间的实时通信显得尤为重要.在微服务架构中,每 1 个服务都是 1 个进程,使用进程间通信(inter-process communication, IPC)技术,以实现进程间信息交互.IPC 将进程间的交互模式分为 2 种:1)1 对 1 交互模式与 1 对多交互模式.简而言之,1 对 1 模式即是单个客户端向服务器端发出请求,客户端期望此响应即时到达;然而在线程应用中,请求传送过程可能造成线程阻塞.2)1 对多交互模式即是一个客户端发出多个通知,被多个相关联服务消费模式.进程间的通信统分为以上 2 种模式,而在实际技

术应用层面,不同编程语言对应着不同的通信技术.以 Java 底层编程语言为基础的微服务架构通信为例,阐述相关技术及协议的应用.在早期分布式初步通信时,采用 RPC(remote produce call),即远程过程调用协议[40],是基于客户端/服务器(Client/Server,C/S)模式调用机制,客户端向服务器端发送调用请求等待服务器应答,是一种典型的请求应答机制;其基本过程是本地分布式对象向本机发出请求,不用开发人员编写底层通信代码,直接向服务器发送请求;服务器对象接受请求信息后,经过计算将处理后的结果返还回客户端.然而,由于 RPC 不支持面向对象,使用的场景大幅度降低;进而促进远程方法调用(remote method invocation, RMI)协议[41]的发展,RMI 协议支持面向对象;采用客户机(stubs)和框架(skeletons)进行远程对象(remote object)的通信.stubs 模拟成远程对象的客户端代理,具有与远程对象相同的远程接口;远程对象调用操作实际是通过调用该对象的客户端代理对象 stub 来完成,实现效果与调用本地对象相同.其优点是支持分布式对象、跨平台高速率的数据传输;缺点是不能跨越编程语言进行数据传输.

随着微服务的架构广泛应用,相对应的通信协议也在成熟发展以适应层次不同的通信要求,如 JMS(Java message service),Web Service 等通信标准已被较为成熟地应用.

3.2 分布式的数据存储一致性

传统整体架构是基于数据库提供的事务一致性[42]标准,即通过数据库提供的提交以及回滚机制中相关操作的原子性、一致性、隔离性、持久性(atomicity, consistency, isolation, durability, ACID),保证数据库中的数据一致性.在微服务架构中,每个微服务具有独立的数据库,以保证微服务进程独立演进、独立开发.然而在分布式环境中,每个服务访问数据是相互分隔,服务之间不能依靠传统数据库中 ACID 保证事务一致性.由此,基于对微服务分布数据一致性要求,开发人员在初始阶段采用 2 阶段 2PC(two phase commit)提交协调机制,该机制为分布式事务提供了强一致保障.然而该机制存在隔离性互斥的特点.在事务执行过程中,所有的资源都是被锁定的,该机制只适合执行时间确定的短事务,并降低了系统的吞吐率.基于此种情况,开发人员通过业务逻辑将互斥锁操作从资源层面移到业务层面,即提出 TCC(try,confirm,cancel)方式.通过 Try(尝试执行业务)、Confirm(确认执行业务)及 Cancel(取消业务)三种操作方式,达到数据最终一致性,以提高系统整体性能.但是 TCC 方式中,微小事务变动,牵动整个业务逻辑,复杂性较高.Saga 事务[43]正好弥补该缺点.Saga 事务就是一个长期运行在线的事务,由多个本地事务所组成,每个本地事务具有相应的执行模块和补偿模块.当 Saga 事务中任意一个本地事务出错,可以通过调用相关事务对应的补偿方法进行恢复,达到事务最终一致性目标.然而 Saga 只提供 ACD(atomicity,consistency,durability),不能保证事务的隔离性.由此,产生诸如 2 个 Saga 事务同时操作一个资源出现语义不一致、2 个 Saga 事务操作同一个订单出现更新丢失等问题,但可以采用在应用层面加入逻辑锁逻辑或者在 Session 层面隔离来保证串行化等操作,以实现数据分布一致性.Saga 的实现方式可以分布 2 种:1)集中式实现方式,即协调器负责服务调用以及事务协调.2)分布式实现方式,即以事件驱动的底层方法进行事务协调.在 2017 年 5 月,Saga 模式在华为开源微服务框架[44]中已被应用,并且在不断地发展进步.

在微服务分布式数据存储一致性方面,不同系统具有不同性能、环境以及对事务协调一致性的要求,采用不同模式事务分布机制以达到分布式数据存储一致性的要求.

3.3 分布式调用链

在微服务架构下,服务按照不同维度进行拆分.由此,一次业务请求需要调用到多个微服务.系统应用构建在不同软件模块中,存在不同模块由不同的团队、编程语言实现.同时,可能部署在几千台服务器,横跨成百个不同的数据中心.因此,亟需理解系统行为、用于分析性能问题的工具,以便发生故障时能够快速定位和解决问题.分布式调用链应运而生,即可以监控那些横跨不同应用、不同服务器之间的关联动作,进而快速定位与解决故障.

Google 公司首先开发其分布式跟踪系统并且发表相关论文[45],Twitter 参照该论文设计思想开发 zipkin 分布式跟踪系统,以期为微服务架构提供参考.zipkin 通过采集跟踪数据,从而使开发人员深入了解在分布式系统中某一个特定请求如何执行.例如,存在一个用户请求超时,跟踪系统可以将这个超时的请求调用链展示在 UI 当中.开发人员可以快速定位出现故障的服务,具体可定位到服务中导致超时的具体位置.同时,通过服务调用链路能够快速定位并解决系统的性能瓶颈.同时,针对不同的应用系统,存在其他的应用程序性能管理(application

performance management, APM)工具,如 Pinpoint 是一款针对 Java 编程语言大规模分布式系统的 APM 工具,通过 JavaAgent 机制字节码代码植入,实现加入 traceid 和抓取性能数据目标;Central Application Tracking 是由国内大众点评网开源出来的追踪系统,已被成熟应用[46].

在分布式调用链方面,不同的系统平台根据其特有的功能属性,开发符合其特点的追踪系统以及调用链.

3.4 测试方面的复杂性

随着平台系统开发模式逐渐从传统的整体式产品向快节奏的微服务架构迁移,软件测试人员也必须相应地调整测试方法和工具,才能快速便捷地提高测试覆盖率.传统整体应用只需测试单一的 REST API(application programming interface)测试,需要启动相关联所有其他服务,使得测试复杂性较高.在开发-测试-部署方面,业界已经具备一套较为成熟的解决方案.然而微服务架构是一种演进式架构,系统最初划分独立服务的数量可能极少;随着业务复杂功能分解与扩展,微服务数量不可避免地增加.同时,微服务数量众多,功能相对独立;在执行业务过程中,难免需要跨服务调用.如何保证跨服务调用的可靠性以及不同阶段的开发测试?

在微服务测试方面,系统被分解成独立的微服务,即每个微服务都是一个功能完整的小型系统.首先需要保证服务自身业务功能的正确性,即通过单元测试保证每个微型系统正确执行.其次,由众多微服务构成完整的系统,需要集成测试来保证整体系统的良好质量.然而,在一个微服务架构中,微服务的个数达到一定阈值时,开发团队面临如网络环境不稳定、运行速度慢、反馈周期长等问题,因此对集成测试望而却步.进而,开发人员采用 Pair 集成测试,即将集成范围缩小,保证两两微服务集成的可靠性.但是这种 Pair 测试方法存在模拟(Mock)其他服务,反复测试已经测试过的服务等问题,增加了额外的工作量.由此,引入 Contract 概念的集成测试[47],即前端与后端开发人员基于业务共同定义 API 协议(Contract);该协议以 JSON 文件形式存储于代码库的测试资源目录中.前端在开发过程中以 JSON 文件作为测试正确依据,后端开发人员则参照该协议编程实现相关 API.这种方式具有环境简单、运行快等优点.但是若存在任意一方修改协议内容,测试便会失败,缺乏自动化监控机制[48]以及不能提前发现问题等缺点.在基于 Contract 概念的基础之上,消费者驱动契约测试(consumer-driven contract testing)将 Contract 契约中 JSON 文件转换成可工作软件时,文档的细微修改便会导致任意一方测试失败;并且通过可工作测试套件保证契约一致性与实时性.由此,成为双方共同遵守的契约.

3.5 微服务应用案例

在现代企业中,微服务所具有的高可用性、容错性、可管理性、可替代性[49]等优点,满足企业用户的主要业务诉求;由此,越来越多的软件架构采用微服务架构.基于微服务以上关键技术,以教育网站的部分功能为案例[50],讲解微服务技术架构的具体应用.

教育网站具有 3 部分功能:1)用户可以登陆注册、获取用户基本信息的功能;2)向用户发送邮件、短信的功能;3)可以查看课程列表和对课程的基本增加、读取、更新、删除等功能.如图 7 所示,展示教育网站部分功能整体架构.

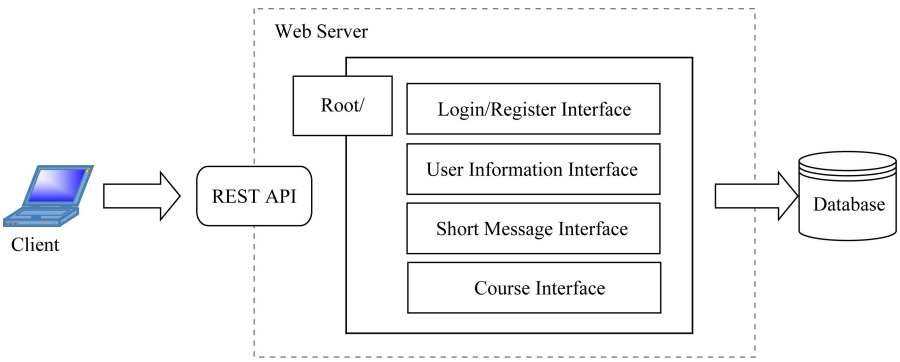


Fig. 7 The overall framework of some functions of education website

图 7 教育网站部分功能整体架构

网站采用 Java 为底层基础语言,部署在轻量级容器 Docker 中进行实现.根据教育网站部分功能的划定与分析,进而将整体架构进行分解为登录注册微服务、用户信息微服务、邮件短信微服务以及课程

微服务等部分.对于微服务之间的通信,从通信协议角度,采用 RPC 协议;该框架具有系统可扩展性、持续交付能力与高可用性等优点[51].常用的 RPC 框架有 Dubbo, Motan, Thrifty, GRPC 等,采用 Dubbo 框架实现教育网站中部分功能的通信功能.如图 8 所示,展示 Dubbo 框架的架构[52].其中,短虚线表示系统启动过程中的初始化阶段,长虚线表示异步通信模式,实线表示同步通信模式.在注册服务部分,服务提供者向注册中心注册所提供的服务;在订阅服务部分,服务消费者向注册中心订阅所需的服务;在通知部分,注册中心返还服务提供者地址列表给消费者;在调用部分,采用同步通信,基于负载均衡算法,服务消费者从提供列表中选择一台提供者进行调用,若调用失败,则重新选择.

在微服务架构方面,SpringCloud 是一系列框架的有序集合,具有服务发现与注册功能的 Netflix Eureka、客服端负载均衡功能的 Netflix Ribbon、服务网关功能的 Netflix Zuul 与分布式配置功能的 Spring Cloud Config 等核心组件.网站中的服务网关采用具有易于认证、易于监控的 Netflix Zuul 组件.如图 9[50] 所示,客户端请求服务调用,通过 API Gateway,转发到后端微服务的 REST API,以期调用网站中各个微服务;每个微服务具有独立的数据库,进而查询每个微服务基本信息,实现用户所需功能.

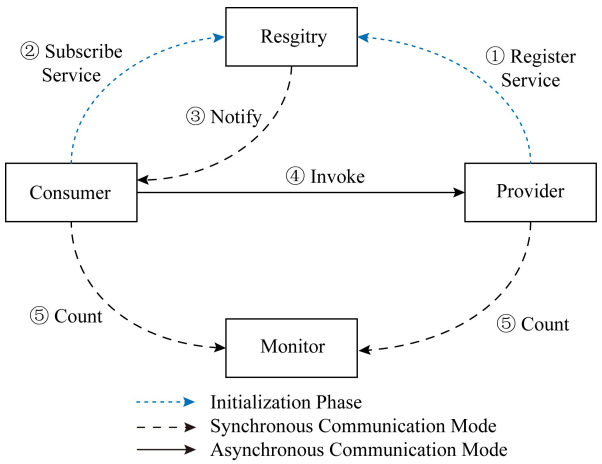


Fig. 8 Dubbo framework in RPC protocol

图 8 RPC 协议中 Dubbo 框架

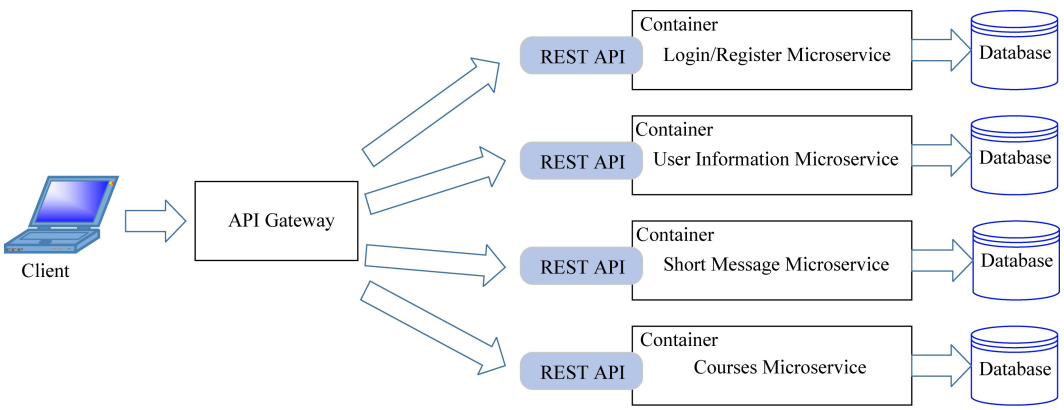


Fig. 9 Netflix Zuul service gateway

图 9 Netflix Zuul 服务网关

SpringCloud 框架是具有高质量、稳定性与持续性等特性的一系列核心组件,可以完成以 Java 编程语言为基础的微服务架构的核心功能,这里不再一一赘述.

基于对微服务核心组件与关键技术的介绍,微服务架构具有 4 个优势:1)整体式应用分解为具有独立功能的微服务,提供多种服务方法,解决系统整体功能复杂性问题;2)每个微服务可以由专有团队开发,对编程语言具有宽泛的选择性;3)每个微服务可以独立部署,提升系统部署的效率;4)由于每个微服务具有独立性,易于在原有微服务基础上进行功能扩展,提升系统的整体可扩展性.然而,微服务应用采用的分布式架构,具有其固有的复杂性,因此,面临以下挑战.

4 微服务面临的挑战

虽然微服务相关技术在逐步发展创新,但由于微服务相当于是相对独立的小型软件系统,在一个具有多种功能、多样性复杂的大型系统平台中,如何在一定人力物力成本中,快速部署微服务以及大量底层应用组件,使其具有自动化部署功能;微服务之间如何迅速准确地通信,以满足用户快速响应的需求;如何在正常通信中,保证用户的数据信息安全;

在数量众多的微服务中,如何保证数据传输中的网络安全.基于这些问题,微服务面临 4 种挑战.

4.1 基础设施

随着分布式系统中应用程序需求迅速增长,传统依赖于客户端向服务器处理端发起请求的整体体系结构开始发生转变;这种体系结构不足以应付不断增长的快速请求;面向服务的体系结构发展成为客户端-服务器体系结构中最成功表示形式之一,然而,从功能角度,SOA 架构中服务之间通过相互依赖最终形成一系列的功能,服务组件大小既可以是小型应用程序服务,也可以是大型企业应用程序,部分组件可以实现多项功能;由此,SOA 仍然属于整体式系统,不能达到客户在弹性、可扩展性、快速软件交付等方面的期望;由此,微服务体系结构利用其本身灵活性、占用更少资源等特性,满足了商业系统自身特性发展的需要[53].Newman[54]从服务平台的角度,阐述了微服务架构存在必要性;系统可以通过微服务使用不同的技术、不同编程语言进而满足不同的应用需求;与此同时,微服务之间相互独立与更新,可以通过大型虚拟机运行,如使用 VMW 和 AWS(amazon Web services)等虚拟技术;然而一方面,配置大型虚拟机耗费较长时间;另一方面,管理程序层具有很大的运载负荷;基于此,容器技术被提出,如 Linux Containers[55],容器技术为程序提供了分割的空间,而不需要管理程序层控制整个虚拟机.随着容器技术的发展,Docker 创建了轻量级容器技术,具有将服务及其依赖项打包成单个映像的特性,使其具有代码移植性[56].每个微服务都位于本身容器内,因此需使用调度程序协调微服务之间事务一致性,由此产生“集群管理器”,其基本任务是确认主机与之相对应的容器[57].同时,微服务分布在不同容器内,使之只能使用轻量级通信.利用其 HTTP 特性,REST API 成为最适合微服务的消息交换机制,可以使用多种语言格式,如 XML 与 JSON[58].

在网格和集群计算时代,通用监控工具只能关注监控性能与数据中心资源级别的层次,但不能监控到微服务层次,如 Amazon EC2 容器的监控技术框架并不能监控到微服务级别的服务表现.由此,收集、整合所有微服务和数据中心资源的整体技术作为新课题被提出[59-60].另一方面,在部署阶段跟踪通过网关的微服务请求流,增加了监控成本[61].Kang 等人[62]基于对容器中管理服务状态面临挑战的研究,对于容器服务管理和监控,则需动态服务发现和注册等组件.

追根究底,微服务目的是使系统在水平扩缩容和弹性伸缩方面更加敏捷、迅速,但前提是仍需基础设施自动化,如何实现微服务基础设施自动化?就目前发展来看,是微服务发展过程中不可避免的挑战.

4.2 信息交互

在微服务信息交互方面,若单个服务由攻击者掌控,该服务可能会恶意影响系统其他服务.由此,微服务架构应该验证微服务之间信息交流的真实性与正确性.从微服务信息交互的认证权限角度,Gegick 等人[63]只将最低和必要权限分配给相对应的用户,并且在最短时间生效.通过谨慎授权访问权限方式,限制攻击者破坏系统.从授权协议的角度出发,轻量级目录访问协议(lightweight directory access protocol, LDAP)[64]是在 SaaS 应用程序中常用的认证协议;通过 LDAP,服务可以存储相对静态授权信息,适合于一次记录,多次读取的应用场景.在数据结构方面,基于树结构的 LDAP 有效、清晰地描述组织的结构信息,简化了目录访问协议(directory access protocol, DAP),以提供基于 TCP/IP 网络的轻量级协议,降低管理维护成本.

针对系统内部的共享资源,一些学者认为授权策略应该以层次结构方式构建,以实现可伸缩性与可表达性.为达到层次性访问控制,网格安全基础设施(grid security infrastructure, GSI)[65]被广泛应用,其包括私钥保护和通信加密等步骤.与传统的授权系统相比,基于分布式管理机制的社区授权服务(community authorization service, CAS)模型[66],允许资源提供者授予部分权限,既可以对社区细粒度访问控制进行维护,又对资源保持最大控制,提高系统的可扩展性与灵活性.Patanjali 等人[67]提出了基于模块化的微服务架构;通过动态评级、计费模式为云服务提供商验证相关接口.基于对服务提供商在整个应用程序生命周期中权限安全性要求,Thanh 等人[68]利用包含网络功能虚拟化等新兴技术的微服务框架 DevOps[69],该框架具有 2 个重要的安全优势:1)是虚拟化安全功能/服务,框架为用户选择和集成多个供应商提供安全解决方案;2)对于网络层访问控制,利用防火墙即服务(firewall as a service, FWaaS)[70]技术,使用户对进入和离开应用程序的网络流量采取不同访问控制策略.Li 等人[71]采用微服务框架 OAuth 保证相关权限的安全性,OAuth 框架授权第三方访问用户帐户以及用户身份,但是第三方网站可以以访问用户帐户临时密钥方式得到访问用户身份信息,由此产生用户私有

信息泄露问题.Cheung 等人[72]基于微服务的框架,采用第 2 版本的 OAuth 修复用户信息漏洞等问题,采用 HTTPS 协议并简单化授权验证过程,但上述问题仍然存在.在信息交互方面,微服务框架与对应接口认证权限,仍需不断地扩展、改进.以适应大数据时代下用户需求以及隐私信息保护需求.

在微服务信息交互层面,仍存在其他问题,如信息交互用时较长、信息交互准确率较低等问题.基于以上阐述,微服务信息交互在信息安全性、网络稳定性等方面,仍面临巨大挑战.

4.3 数据安全

在微服务数据安全方面,微服务架构在云环境中应用越来越广泛,微服务中分布式数据隐私信息保护以及保密性越来越被重视.部分学者采用密码学中传统加密方式对数据进行加密.基于密码学的加密技术可以分为对称与非对称方法[73-76].对称加密技术,即是通过加密密钥对原始数据进行加密,反之亦然.非对称技术,即是通过公有/私有密钥对原始数据加密,只有相对应私有/公有密钥才能进行解密.一般来说,加密密钥越长,加密后的数据也更加安全.然而,这表示在加密与解密过程中产生更大性能开销[77].基于此,考虑到微服务模型中数据具体应用场景与对部分敏感信息的保护,学者们力求在计算复杂度与数据安全保护取得平衡.Shah 等人[78]从提高数据安全性角度,利用第三方审计协议,允许用户评估风险并提高降低风险的效率;同时还提出了支持在线存储内部服务和外部审计的数据隐私方法.审计目的是监控服务器和网络中用户详细行为信息记录,一旦发现违反安全策略相关行为信息,立即向管理员报告.Wang 等人[79]仅从数据加密角度出发,并没有考虑对审计协议的设计,由此并不能保证数据不会被泄露给第三方平台.

微服务是分布式的,即拥有众多不同的数据服务平台,由此需对不同数据来源加以保护.Callegati 等人[80]从数据可靠性、完整性和真实性角度出发,采取 4 个步骤进行保护数据源:1)提出一个具有可验证性、可问责性和可生产性的数据源管理系统,可验证性是指管理体系应能够验证与操作有关的参与者(或服务)的数据源,可问责性即是指管理系统应能够让参与者(或服务)对其在数据源中的行为负责,可生产性是指管理系统应能够在数据源上生产过程;2)为数据流认证创建一个私有和公共密钥系统;3)使用基于密码学出处验证方法确认数据源主机数据属性与完整性;4)将数据元数据传播与密钥分发传播管理相结合,确保数据源管理系统的可靠性.基于以上 4 个步骤,既可以有效地进行微服务架构信息交流,又可以对数据来源进行有效保护.Subashini 等人[81]从企业数据安全的角度,建议对数据供应商采取额外的安全检查,以确保数据源安全并防止应用程序中安全漏洞.企业数据定期备份使用强加密方案,便于在发生紧急情况时快速进行数据恢复.Saad 等人[82]以对数据源全方位保护角度,建立保障数据源存储与访问事务处理安全的信任模型,提高了服务之间信任度.

基于以上阐述,在微服务分布式结构中,如何有效保证数据源信息的安全以及对第三方平台数据有效认证,是十分值得思考的问题.

4.4 网络安全

在微服务网络安全方面,首先,数量众多的微服务带来网络复杂性,增加监控整个应用程序安全性的难度.其次,微服务通常被设计成彼此完全信任,因此单个微服务的失误可能会导致整个应用程序崩溃.Sun 等人[83]从微服务增加了监控整个应用程序安全性的难度与单一微服务失败会导致整个应用程序崩溃的角度,提出了基于微服务云应用安全即服务(security-as-a-service, SaaS)设计方案.通过为网络管理程序添加一个新的 API 原语流 TAP,进而为网络流量构建了灵活地监控和策略强制基础设施,以保护云应用程序的安全.基于对监控数据中心网络流量和安全威胁模型、方法和设备的描述,Ahuja 等人[84]扩展了安全系统中微服务的层次结构.具体而言,创建第 1 层次结构的新微服务,以配置第 1 层与第 2 层次结构中微服务之间的数据平面连接;配置第 1 层与第 3 层次结构微服务;同时将新加入的微服务包括在第 1 个层次结构的负载平衡决策中,以提高微服务网络安全性.Ross 等人[85]研发分布式微服务中提供漏洞扫描系统.该系统包括多个微分段环境,既与云数据中心服务器相关联,又与云环境的多个微分段环境耦合.云数据中心服务器具有安全控制器与主动探测控制器,前者为每一个微分段环境提供安全策略,后者为主动探测微分段环境设备,并执行漏洞扫描策略.Yarygina 等人[86]基于对目前技术仅对微服务系统特定安全问题的研究;提供微服务安全分类,进而评估微服务架构的安全性;最后提出相关合理解决方案,并对 Docker Swarm 和 Netflix 的安全决策进行分析.

一方面,微服务安全是一个多方应用结合的问题,目前没有系统的分层安全解决方案;另一方面,

如果这些安全挑战得到解决,微服务体系结构的安全性将会大大提高.

5 总结与展望

目前,国内外学者对基于微服务技术的 SOA 架构及其实现机制的研究进展十分重视,微服务具有独立进程、轻量级通信和独立部署环境等显著的特性;将系统整体功能分解到单个微服务中以实现对系统的解耦;这种方式不仅可以降低系统的耦合性,提升系统的内聚性,而且减少服务交互的成本,提供更加灵活的服务支持.本文从微服务概念、核心组件与技术发展过程、面临挑战 3 个方面进行了分析:

1) 论述了微服务体系结构的产生背景、相关概念,指出传统整体架构不足及微服务体系结构发展背景及优势.

2) 论述了微服务的核心组件与技术发展,前者在服务发现机制、服务容错等层面进行研究;后者在技术发展方面,从微服务软件技术层面以及微服务架构层面 2 个方面进行概述.

3) 基于对核心组件的介绍,进而对微服务关键技术进行分析;提出微服务在基础设施层面、信息交互层面、数据安全层面、网络安全面临的四大挑战以及发展趋势.

与传统开发模型相同,选择合适的未来开发平台对于微服务十分重要.微服务如何与 2 个主要的新兴平台进行集成,即云平台[87]和物联网[88].随着科技新兴技术与数据时代的到来,两个平台很可能在互联网行业中占主导地位.由于微服务本身具有移植性和可伸缩性等特性,在物联网上运行存在部分难题,若在云平台中运行微服务似乎是恰当的选择.但从系统安全性角度出发,存在部分功能具有低计算能力且具有较高风险的缺点.因此,微服务与具体应用平台相结合,解决微服务与平台相集成的特定实施方案以及安全方案需求,变得更加迫切.

参 考 文 献

[1] Dong Ruizhi, Li Bixin, Wang Lulu, et al. The review of software ecosystem research [J]. Chinese Journal of Computers, 2019, 43(2): 250-271 (in Chinese)
(董瑞志, 李必信, 王璐璐, 等. 软件生态系统研究综述[J]. 计算机学报, 2019, 43(2): 250-271)

[2] Shutongchanglian. Talking on integration [OL]. 2019 [2019-05-26]. https://blog.csdn.net/aeaiesb/article/details/90437876 (in Chinese)
(数通畅联. 漫谈集成[OL]. 2019 [2019-05-26]. http://blog.csdn.net/aeaiesb/article/details/90437876)

[3] Gadgil H, Fox G, Pallickara S, et al. Scalable, fault-tolerant management in a service oriented architecture [C] //Proc of Int Symp on High Performance Distributed Computing. New York: ACM, 2007: 1-15

[4] Feng Hongwei. SOA collaborative software meets the new challenge of informatization [J]. The Window of World, 2005(10): 42-43 (in Chinese)
(冯宏卫. SOA 协同软件应对信息化新挑战[J]. 视窗世界, 2005(10): 42-43)

[5] Fan Jing, Xiong Lirong, Xu Cong. Research and application of data integration in distributed enterprise service bus platform [J]. Computer Science, 2014, 41(2): 212-220 (in Chinese)
(范菁, 熊丽荣, 徐聪. 分布式企业服务总线平台数据集成研究及应用[J]. 计算机科学, 2014, 41(2): 212-220)

[6] Nadareishvili I, Mitra R, McLarty M, et al. Microservice Architecture: Aligning Principles, Practices, and Culture [M]. Sebastopol, CA: O'Reilly Media, 2016

[7] Yang Junwei, Ji Xin, Hu Qiangxin. Electric power cloud service platform based on micro service architecture [J]. Electric Power ICT, 2017, 15(1): 8-12

[8] zposion. The difference between SOA architecture and microservice architecture [OL]. (2018-06-06) [2019-09-16]. https://blog.csdn.net/zpoison/article/details/80729052 (in Chinese)
(zposion. SOA 架构和微服务架构的区别[OL]. (2018-06-06) [2019-09-16]. https://blog.csdn.net/zpoison/article/details/80729052)

[9] Huahua. Understanding microservices, clustering, SOA, distributed [OL]. (2018-11-09) [2019-09-16]. https://blog.csdn.net/qq_34395857/article/details/83780873 (in Chinese)
(花花. 对微服务、集群、SOA、分布式的理解[OL]. (2018-11-09) [2019-09-16]. https://blog.csdn.net/qq_34395857/article/details/83780873)

[10] Li Chunxia. Overview of microservice architecture research [J].Software Guide, 2019, 18(8): 1-3, 7 (in Chinese)
(李春霞. 微服务架构研究概述[J].软件导刊, 2019, 18(8): 1-3, 7)

[11] Balalaie A, Heydarnoori A, Jamshidi P. Microservices architecture enables DevOps: An experience report on migration to a cloud-native architecture [J]. IEEE Software, 2016, 33(3): 42-52

[12] Dongxierwang. Talking about integration [OL]. 2019 [2019-05-25]. http://www.360doc.com/content/19/0519/14/49586_836712947.shtml (in Chinese)
(东西二王. 漫谈集成[OL]. 2019 [2019-05-25]. http://www.360doc.com/content/19/0519/14/49586_836712947.shtml)

[13] Lewis J, Fowler M. Microservices [OL]. (2014-05-25) [2019-07-28]. https://martinfowler.com/articles/microservices.html

[14] Gray J. A conversation with werner vogels [J]. ACM Queue, 2006, 4(4): 14-22

[15] Cockcroft A. Migrating to microservices [OL]. 2016 [2019-08-20]. https://www.infoq.com/presentations/migration-cloud-native/

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

[17] Wang Lei. Microservice Architecture and Practice [M]. Beijing: Publishing House of Electronics Industry, 2016 (in Chinese)
(王磊. 微服务架构与实践[M]. 北京: 电子工业出版社, 2016)

[18] Han Daoqi. Microservice architecture and security system design scheme [J]. Electronic Technology and Software Engineering, 2019, 148(2): 214-216 (in Chinese)
(韩道岐. 微服务架构和安全体系设计方案[J]. 电子技术与软件工程, 2019, 148(2): 214-216)

[19] Hunt P, Konar M, Junqueira F P, et al. ZooKeeper: Waitfree coordination for Internet-scale systems [C] //Proc of Annual Technical Conf. Berkeley, CA: USENIX Association, 2010: 1-14

[20] Anurag U, Hitesh H. Optimization in Round Robin Process Scheduling Algorithm [M]. Berlin: Springer, 2016: 457-467

[21] Li Tong, Baumberger D, Hahn S. Efficient and scalable multiprocessor fair scheduling using distributed weighted round-robin [J]. ACM SIGPLAN Notices, 2009, 44(4): 65-74

[22] Choi D J, Chung K S, Shon J G. An improvement on the weighted least-connection scheduling algorithm for load balancing in Web cluster systems [M] //Grid and Distributed Computing, Control and Automation. Berlin: Springer, 2010: 127-134

[23] liuqing0805. Insufficient single structure [OL]. (2018-09-05) [2019-09-15]. https://blog.csdn.net/liuqing0805/article/details/82422886 (in Chinese)
(liuqing0805. 单体架构存在的不足[OL]. (2018-09-05) [2019-09-15]. https://blog.csdn.net/liuqing0805/article/details/82422886)

[24] Qi Dai. How to implement client load balancing in microservice architecture [OL]. (2018-04-11) [2019-09-15]. https://www.jianshu.com/p/a234bfee26c1?utm_campaign (in Chinese)
(期待. 微服务架构如何实现客户端负载均衡[OL]. (2018-04-11) [2019-09-15]. https://www.jianshu.com/p/a234bfee26c1?utm_campaign)

[25] Mumaren. Load balancing of microservices [OL]. (2019-01-18) [2019-09-15]. http://www.imooc.com/article/274453 (in Chinese)
(慕码人. 微服务之负载均衡[OL]. (2019-01-18) [2019-09-15]. http://www.imooc.com/article/274453)

[26] Zhang Dedong. SYN flood attack defense method based on timeout retransmission mechanism [J].Electronic Science and Technology, 2017, 4(4): 87-90 (in Chinese)
(张德栋. 基于超时重发机制的SYN Flood攻击防御方法[J]. 电子科学技术, 2017, 4(4): 87-90)

[27] zuul. Netflix Zuul [OL]. 2014 [2018-07-24]. https://github.com/Netflix/zuul

[28] Gajek F. API diversity for microservices in the domain of connected vehicles [D]. Stuttgart: University of Stuttgart, 2018

[29] Pang Yu. Application research of Docker technology in software development process [J]. Information Technology, 2019, 43(5): 114-116, 120 (in Chinese)
(庞宇. Docker技术在软件开发过程中的应用研究[J].信息技术, 2019, 43(5): 114-116, 120)

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

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

[32] Rosenthal C, Hochstein L, Blohowiak A, et al. Chaos Engineering [M]. Sebastopol, CA: O'Reilly Media, 2017

[33] Burns B, Oppenheimer D. Design patterns for container-based distributed systems [C] //Proc of the 8th USENIX Workshop on Hot Topics in Cloud Computing. Berkeley, CA: USENIX Association, 2016: 108-113

[34] Roberts M J. Encrypted server-less communication between devices: US, 9325495 [P]. 2016-04-26

[35] Chen Shuyong, Song Shufang, Li Lanxin, et al. Survey on smart grid technology [J]. Power System Technology, 2009, 33(8): 1-7

[36] He Jian. What are we talking about when we talk about microservices [OL]. (2018-12-06) [2019-09-15]. https://www.jianshu.com/p/dd818114ab4b (in Chinese)
(和坚. 当我们在说微服务治理的时候究竟在说什么[OL]. (2018-12-06) [2019-09-15]. https://www.jianshu.com/p/dd818114ab4b)

[37] Fang Yi, Zhu Yongqiang, Gong Xueqing. Distributed transaction processing under microservice architecture [J]. Computer Applications and Software, 2019, 36(1): 158-164 (in Chinese)
(方意, 朱永强, 宫学庆. 微服务架构下的分布式事务处理[J]. 计算机应用与软件, 2019, 36(1): 158-164)

[38] DevOps Era. DevOps-based microservice ecosystem and engineering practice(1)[OL]. 2018 [2019-08-24]. https://cloud.tencent.com/developer/article/1005477 (in Chinese)
(DevOps时代. 基于DevOps的微服务生态系统与工程实践(1)[OL]. 2018 [2019-08-24]. https://cloud.tencent.com/developer/article/1005477)

[39] Jadeja Y, Modi K. Cloud computing-concepts, architecture and challenges [C] //Proc of Int Conf on Computing, Electronics and Electrical Technologies (ICCEET). Piscataway, NJ: IEEE, 2012: 877-880

[40] Chapin P C, Skalka C. SpartanRPC: Remote procedure call authorization in wireless sensor networks [J]. ACM Transactions on Information & System Security, 2014, 17(2): 1-30

[41] Kumar P, Yadav R. Java remote method invocation [J]. Beginning Java Apis Extensions & Libraries, 2018, 18(2): 525-548

[42] Liang Yong. Research on transaction consistency of MySQL database [C] //Proc of the 14th National Youth Communication Conf: New Development of Communication Theory and Technology. Beijing: Publishing of Electronic Industry Society, 2009: 122-127 (in Chinese)
(梁勇. MySQL 数据库的事务一致性研究[C] //第十四届全国青年通信学术会议论文集: 通信理论与技术新发展. 北京: 电子工业出版社, 2009: 122-127)

[43] Liu Chuanjie. Application research of Saga-based advanced transaction model in workflow system [D]. Xi'an: Xidian University, 2004 (in Chinese)
(刘传杰. 基于 Saga 的高级事务模型在工作流系统中的应用研究[D]. 西安: 西安电子科技大学, 2004)

[44] Willem Jiang. Saga distributed transaction solution and practice [OL]. 2019 [2019-05-24]. http://servicecomb.apache.org/cn/docs/distributed-transactions-saga-implementation/ (in Chinese)
(Willem Jiang. Saga 分布式事务解决方案与实践[OL]. 2019 [2019-05-24]. http://servicecomb.apache.org/cn/docs/distributed-transactions-saga-implementation/)

[45] Sigelman B H, Barroso L A, Burrows M, et al. Dapper, a large-scale distributed systems tracing infrastructure [OL]. 2010 [2019-08-19]. https://www.docin.com/p-555529771.html

[46] Wang Lian. Public comment open source distributed monitoring platform CAT depth analysis [OL]. (2016-10-31) [2019-05-18]. https://www.oschina.net/news/78563/cat-depth-analysis (in Chinese)
(王练. 大众点评开源分布式监控平台 CAT 深度剖析[OL]. (2016-10-31) [2019-05-18]. https://www.oschina.net/news/78563/cat-depth-analysis)

[47] Raisehea. Consumer-driven contract testing Spring Cloud Contract introduction [OL]. 2018 [2019-06-11]. https://www.jianshu.com/p/fa981ea8df6e (in Chinese)
(Raisehea. 消费者驱动的契约测试 Spring Cloud Contract 介绍[OL]. 2018 [2019-06-11]. https://www.jianshu.com/p/fa981ea8df6e)

[48] Zhang Yuanbing. Automation testing based on microservice architecture [J]. Electronic Technology and Software Engineering, 2019, 150(4): 119-120 (in Chinese)
(张圆冰. 基于微服务架构的自动化测试[J].电子技术与软件工程, 2019, 150(4): 119-120)

[49] weixin. Talk about huawei microservice solutions and practices (on)[OL]. 2018 [2019-09-15]. https://blog.csdn.net/weixin_34343000/article/details/88032305 (in Chinese)
(weixin. 谈谈华为微服务解决方案与实践(上)[OL]. 2018 [2019-09-15]. https://blog.csdn.net/weixin_34343000/article/details/88032305)

[50] winner. Docker Kubernetes microservice containerization practice (1)[OL]. 2019 [2019-09-15]. https://blog.csdn.net/Sicily_winner/article/details/86750033 (in Chinese)
(winner. Docker Kubernetes 微服务容器化实践(1)[OL]. 2019 [2019-09-15]. https://blog.csdn.net/Sicily_winner/article/details/86750033)

[51] wu1317581750. Advantages and disadvantages of RPC [OL]. 2018 [2019-09-15]. https://blog.csdn.net/wu1317581750/article/details/82380912 (in Chinese)
(wu1317581750. RPC 的优点和缺点[OL]. 2018 [2019-09-15]. https://blog.csdn.net/wu1317581750/article/details/82380912)

[52] zxl. Dubbo+zookeeper basic explanation [OL]. 2018 [2019-09-15]. https://blog.csdn.net/zxljsbk/article/details/81626559 (in Chinese)
(zxl. Dubbo+zookeeper 基础讲解[OL]. 2018 [2019-09-15]. https://blog.csdn.net/zxljsbk/article/details/81626559)

[53] Salah T, Zemerly M J, Yeun C Y, et al. The evolution of distributed systems towards microservices architecture [C] //Proc of the 11th Int Conf on Internet Technology and Secured Transactions (ICITST). Piscataway, NJ: IEEE, 2016: 318-325

[54] Newman S. Building Microservices: Designing Fine-grained Systems [M]. Sebastopol, CA: O'Reilly Media, 2015

[55] Canonical. Linux containers [OL]. (2016-06-01) [2019-08-12]. https://linuxcontainers.org/

[56] Boettiger C. An introduction to docker for reproducible research, with examples from the renvironment [J]. ACM SIGOPS Operating Systems Review, 2014, 49(1): 71-79

[57] Apirajitha P S, Angayarkanni A. A design of an adaptive peer-to-peer network to reduce power consumption using cloud computing [C] //Proc of Int Conf on Advanced Communication Control and Computing Technologies (ICACCCT). Piscataway, NJ: IEEE, 2012: 198-201

[58] Malavalli D, Sathappan S. Scalable microservice based architecture for enabling DMTF profiles [C] //Proc of the 11th Int Conf on Network and Service Management (CNSM). Piscataway, NJ: IEEE, 2015: 428-432

[59] Ranjan R. Streaming big data processing in datacenter clouds [J]. IEEE Cloud Computing, 2014, 1(1): 78-83

[60] Natu M, Ghosh R K, Shyamsundar R K, et al. Holistic performance monitoring of hybrid clouds: Complexities and future directions [J]. IEEE Cloud Computing, 2016, 3(1): 72-81

[61] Villamizar M, Garcés O, Castro H, et al. Evaluating the monolithic and the microservice architecture pattern to deploy Web applications in the cloud [C] //Proc of the 10th Computing Colombian Conf (CCC). Piscataway, NJ: IEEE, 2015: 583-590

[62] Kang Hui, Le M, Tao Shu. Container and microservice driven design for cloud infrastructure DevOps [C] //Proc of Int Conf on Cloud Engineering (IC2E). Piscataway, NJ: IEEE, 2016: 202-211

[63] Gegick M, Barnum S. Least privilege. 2005 [2019-08-23]. https://www. us-cert. gov/bsi/articles/knowledge/principles/least-privilege

[64] Bulusu P, Shahriar H, Haddad H M. Classification of lightweight directory access protocol query injection attacks and mitigation techniques [C] //Proc of Int Conf on Collaboration Technologies and Systems (CTS). Piscataway, NJ: IEEE, 2015: 337-344

[65] Wang Lizhe, Jie Wei, Chen Jinjun. Grid computing: Infrastructure, service and applications [J]. Educause Review, 2009, 47(8): 107-116

[66] Pearlman L, Welch V, Foster I, et al. A community authorization service for group collaboration [C] //Proc of the 3rd Int Workshop on Policies for Distributed Systems and Networks. Piscataway, NJ: IEEE, 2002: 50-59

[67] Patanjali S, Truninger B, Harsh P, et al. CYCLOPS: A micro service based approach for dynamic rating, charging & billing for cloud [C] //Proc of the 13th Int Conf on Telecommunications. Piscataway, NJ: IEEE, 2015: 1-8

[68] Thanh T Q, Covaci S, Magedanz T, et al. Embedding security and privacy into the development and operation of cloud applications and services [C] //Proc of the 17th Int Telecommunications Network Strategy and Planning Symp (Networks). Piscataway, NJ: IEEE, 2016: 31-36

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

[70] Csubak D, Kiss A. OpenStack firewall as a service rule analyser [C] //Proc of Int Conf on Human Aspects of Information Security, Privacy, and Trust. Berlin: Springer, 2016: 212-220

[71] Li Wanpeng, Mitchell C J. Analysing the security of Google's implementation of OpenID connect [C] //Proc of Int Conf on Detection of Intrusions and Malware, and Vulnerability Assessment. Berlin: Springer, 2016: 357-376

[72] Cheung K C, Peel C, Happe S H C. Method and system for credential management and data encryption for iOS based devices: US, 13/228930 [P]. 2013-03-14

[73] Macdonald I G. Symmetric Functions and Hall Polynomials [M]. Oxford, UK: Oxford University Press, 1998

[74] Fujisaki E, Okamoto T. Secure integration of asymmetric and symmetric encryption schemes [J]. Journal of Cryptology, 2013, 26(1): 80-101

[75] Yassein M B, Aljawarneh S, Qawasmeh E, et al. Comprehensive study of symmetric key and asymmetric key encryption algorithms [C] //Proc of 2017 Int Conf on Engineering and Technology (ICET). Piscataway, NJ: IEEE, 2017: 1-7

[76] Lozupone V. Analyze encryption and public key infrastructure (PKI) [J]. International Journal of Information Management, 2018, 38(1): 42-44

[77] Bellare M, Kane D, Rogaway P. Big-key symmetric encryption: Resisting key exfiltration [C] //Proc of Annual Int Cryptology Conf. Berlin: Springer, 2016: 373-402

[78] Shah M A, Baker M, Mogul J C, et al. Auditing to keep online storage services honest [OL]. 2007 (2019-08-20). https://www. usenix. org/legacy/events/hotos07/tech/full _ papers/shah/shah.pdf

[79] Wang Cong, Wang Qian, Ren Kui, et al. Privacy-preserving public auditing for data storage security in cloud computing [C] //Proc of IEEE INFOCOM 2010. Piscataway, NJ: IEEE, 2010: 1-9

[80] Callegati F, Giallorenzo S, Melis A, et al. Data security issues in maas-enabling platforms [C] //Proc of the 2nd Int Forum on Research and Technologies for Society and Industry Leveraging a Better Tomorrow. Piscataway, NJ: IEEE, 2016: 1-5

[81] Subashini S, Kavitha V. A survey on security issues in service delivery models of cloud computing [J]. Journal of Network and Computer Applications, 2011, 34(1): 1-11

[82] Saad M, Jalil K A, Manaf M. Achieving trust in cloud computing using secure data provenance [C] //Proc of IEEE Conf on Open Systems (ICOS). Piscataway, NJ: IEEE, 2014: 84-88

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

[84] Ahuja R P S, Nedbal M. Dynamic, load-based, auto-scaling network security microservices architecture: US, 9716617 [P]. 2017-07-25

[85] Ross C, Shieh C Y M, Lian J J R, et al. Microsegmented networks that implement vulnerability scanning: US, 9438634 [P]. 2016-09-06

[86] Yarygina T, Bagge A H. Overcoming security challenges in microservice architectures [C] //Proc of IEEE Symp on Service-Oriented System Engineering (SOSE). Piscataway, NJ: IEEE, 2018: 11-20

[87] Wang Binfeng, Su Jinshu, Chen Lin. The review of network design of cloud computing data center [J]. Journal of Computer Research and Development, 2016, 53(9): 2085-2106 (in Chinese)
(王斌锋, 苏金树, 陈琳. 云计算数据中心网络设计综述[J]. 计算机研究与发展, 2016, 53(9): 2085-2106)

[88] Wu Jianjia Zhao Wei. WInternet: From net of things to Internet of things [J]. Journal of Computer Research and Development, 2013, 50(6): 1127-1134 (in Chinese)
(武建佳, 赵伟. WInternet: 从物网到物联网[J]. 计算机研究与发展, 2013, 50(6): 1127-1134)

Feng Zhiyong, born in 1965. Professor in the School of Computer Software, College of Intelligence and Computing, Tianjin University. His main research interests include service computing, knowledge engineering, software engineering and information integration.

Xu Yanwei, born in 1993. PhD candidate in the College of Intelligence and Computing, Tianjin University. Her main research interests include service computing, service recommendation, service ecosystem.

Xue Xiao, born in 1979. Professor in the College of Intelligence and Computing, Tianjin University. His main research interests include service science and service computing, multi-agent modeling & simulation, etc.

Chen Shizhan, born in 1975. Associate professor in the College of Intelligence and Computing, Tianjin University. His main research interests include service computing and the service-oriented architecture.

2018 年《计算机研究与发展》高被引论文 TOP10

排名	论文信息
1	李然，林政，林海伦，王伟平，孟丹. **文本情绪分析综述**[J]. 计算机研究与发展，2018，55(1)：30-52 Li Ran, Lin Zheng, Lin Hailun, Wang Weiping, Meng Dan. Text Emotion Analysis: A Survey [J]. Journal of Computer Research and Development, 2018, 55(1): 30-52
2	吕华章，陈丹，范斌，王友祥，乌云霄. **边缘计算标准化进展与案例分析**[J]. 计算机研究与发展，2018，55(3)：487-511 Lü Huazhang, Chen Dan, Fan Bin, Wang Youxiang, Wu Yunxiao. Standardization Progress and Case Analysis of Edge Computing [J]. Journal of Computer Research and Development, 2018, 55(3): 487-511
3	赵梓铭，刘芳，蔡志平，肖侬. **边缘计算：平台、应用与挑战**[J]. 计算机研究与发展，2018，55(2)：327-337 Zhao Ziming, Liu Fang, Cai Zhingping, Xiao Nong. Edge Computing: Platforms, Applications and Challenges [J]. Journal of Computer Research and Development, 2018, 55(2): 327-337/
4	陈珂，梁斌，柯文德，许波，曾国超. **基于多通道卷积神经网络的中文微博情感分析**[J]. 计算机研究与发展，2018，55(5)：945-957 Chen Ke, Liang Bin, Ke Wende, Xu Bo, Zeng Guochao. Chinese Micro-Blog Sentiment Analysis Based on Multi-Channels Convolutional Neural Networks [J]. Journal of Computer Research and Development, 2018, 55(5): 945-957
5	余永红，高阳，王皓，孙栓柱. **融合用户社会地位和矩阵分解的推荐算法**[J]. 计算机研究与发展，2018，55(1)：113-124 Yu Yonghong, Gao Yang, Wang Hao, Sun Shuanzhu. Integrating User Social Status and Matrix Factorization for Item Recommendation [J]. Journal of Computer Research and Development, 2018, 55(1): 113-124
6	贺海武，延安，陈泽华. **基于区块链的智能合约技术与应用综述**[J]. 计算机研究与发展，2018，55(11)：2452-2466 He Haiwu, Yan An, Chen Zehua. Survey of Smart Contract Technology and Application Based on Blockchain [J]. Journal of Computer Research and Development, 2018, 55(11): 2452-2466
7	齐彦丽，周一青，刘玲，田霖，石晶林. **融合移动边缘计算的未来 5G 移动通信网络**[J]. 计算机研究与发展，2018，55(3)：478-486 Qi Yanli, Zhou Yiqing, Liu Ling, Tian Lin, Shi Jinglin. MEC Coordinated Future 5G Mobile Wireless Networks [J]. Journal of Computer Research and Development, 2018, 55(3): 478-486
8	谢振平，金晨，刘渊. **基于建构主义学习理论的个性化知识推荐模型**[J]. 计算机研究与发展，2018，55(1)：125-138 Xie Zhenping, Jin Chen, Liu Yuan. Personalized Knowledge Recommendation Model Based on Constructivist Learning Theory [J]. Journal of Computer Research and Development, 2018, 55(1): 125-138
9	陈伟利，郑子彬. **区块链数据分析：现状、趋势与挑战**[J]. 计算机研究与发展，2018，55(9)：1853-1870 Chen Weili, Zheng Zibin. Blockchain Data Analysis: A Review of Status, Trends and Challenges [J]. Journal of Computer Research and Development, 2018, 55(9): 1853-1870
10	邓晓衡，关培源，万志文，刘恩陆，罗杰，赵智慧，刘亚军，张洪刚. **基于综合信任的边缘计算资源协同研究**[J]. 计算机研究与发展，2018，55(3)：449-477 Deng Xiaoheng, Guan Peiyuan, Wan Zhiwen, Liu Enlu, Luo Jie, Zhao Zhihui, Liu Yajun, Zhang Honggang. Integrated Trust Based Resource Cooperation in Edge Computing [J]. Journal of Computer Research and Development, 2018, 55(3): 449-477

数据来源：CSCD，中国知网；统计日期：2019 年 12 月 23 日