软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2021,32(5):1284−1301 [doi: 10.13328/j.cnki.jos.006233]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

一种优化的数据流驱动的微服务化拆分方法*

李杉杉[1,2], 荣国平[1,2], 高邱雅[1,2], 邵 栋[1,2]

[1](南京大学 软件学院,江苏 南京 210023)

[2](计算机软件新技术国家重点实验室(南京大学),江苏 南京 210023)

通讯作者: 荣国平, E-mail: ronggp@nju.edu.cn

摘 要: 近年来,微服务架构已经成为软件工程领域比较流行的架构风格,其天然支持 DevOps 和持续交付以及可伸缩性、可扩展性好等特性,驱动着业界实践者纷纷向微服务架构迁移.然而,采用微服务架构也面临诸多挑战,其中最关键的是缺乏自动化、一体化的解决方案来高效支持面向微服务的拆分设计以及候选微服务架构的评估.为了应对该挑战,对已有的数据流驱动的微服务化拆分方法的局限问题(例如效率和灵活性)进行改进,在此基础上,提出了一种优化的微服务化拆分方法(DFD-A).该方法通过动、静态分析相结合的方式,实现了更加高效的数据流信息自动化收集,同时,采用两阶段的聚类算法来取代完全基于自定义规则的微服务化拆分算法.同时实现了原型工具来支持从数据收集分析、服务拆分到候选微服务架构评估的完整且自动化的过程.案例研究结果表明,该优化方法 DFD-A 及其原型工具在保证拆分结果有效性的基础上,可以更加高效、灵活地支持面向微服务的自动化拆分与评估.

关键词: 微服务;DevOps;可伸缩;拆分;评估;数据流

中图法分类号: TP311

中文引用格式: 李杉杉,荣国平,高邱雅,邵栋.一种优化的数据流驱动的微服务化拆分方法.软件学报,2021,32(5):1284−1301. http://www. jos.org.cn/1000-9825/6233.htm

英文引用格式: Li SS, Rong GP, Gao QY, Shao D. Optimized dataflow-driven approach for microservices-oriented decomposition. Ruan Jian Xue Bao/Journal of Software, 2021,32(5):1284−1301 (in Chinese). http://www.jos.org.cn/1000-9825/6233.htm

Optimized Dataflow-driven Approach for Microservices-oriented Decomposition

LI Shan-Shan[1,2], RONG Guo-Ping[1,2], GAO Qiu-Ya[1,2], SHAO Dong[1,2]

[1](Software Institute, Nanjing University, Nanjing 210023, China)

[2](State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210023, China)

Abstract: In recent years, microservices architecture (MSA) has become a prevalent architectural style in the field of software engineering. The natural characteristics of MSA, e.g., supporting DevOps andcontinuous delivery, scalability and extensibility, motivate practitioners to migrate their legacy systems to this new architectural style. However, the migration to MSA also causes many challenges, among which the most critical one is lacking an automated and integrated solution for the microservices-oriented decomposition and the evaluation of candidate microservices. To address this challenge, an optimized approach (DFD-A) is proposed through overcoming two limitations of an existing data flow-driven decomposition solution (DFD), i.e. efficiency and flexibility. The proposed DFD-A approach realizes the automatic data flow information collection through combining the dynamic and static analysis technology and identifies

* 基金项目: 国家自然科学基金(62072227, 61802173); 国家重点研发计划(2019YFE0105500); 江苏省政府间双边创新项目(BZ2020017); 计算机软件新技术国家重点实验室(南京大学)创新项目(ZZKT2019B01)

Foundation item: National Natural Science Foundation of China (62072227, 61802173); National Key Research and Development Program of China (2019YFE0105500); Intergovernmental Bilateral Innovation Project of Jiangsu Province (BZ2020017); Innovation Project of State Key Laboratory for Novel Software Technology (Nanjing University) (ZZKT2019B01)

本文由“面向持续软件工程的微服务架构技术”专题特约编辑张贺教授、王忠杰教授、陈连平研究员和彭鑫教授推荐.

收稿时间: 2020-09-15; 修改时间: 2020-10-26; 采用时间: 2020-12-15; jos 在线出版时间: 2021-02-07

microservices using a more flexible two-phase clustering algorithm. A prototype tool is also implemented to automatically support the whole process of the data collection, the decomposition, and even the evaluation of microservice candidates using some typical metrics. The results of a case study demonstrate the effectiveness, efficiency, and flexibility of the proposed DFD-A method for microservices-oriented decomposition and evaluation.

Key words: microservice; DevOps; scalability; decomposition; evaluation; data flow

微服务架构(microservices architecture,简称 MSA)是近年来软件工程领域中的一个热门话题.作为面向服务架构(service-oriented architecture,简称 SOA)之后的分布式系统解决方案,该架构风格提倡将系统拆分为细粒度的服务,通过轻量级的通信机制比如 RESTFul API 等进行协同工作[1].微服务架构中,业务服务因拆分而变得独立,且可以带来诸多好处,帮助解决单体架构或面向服务架构中的一系列问题.具体来讲,小而自治的微服务通常承担“单一职责”,且可以运行在单独的进程中,具备可独立开发部署的特性,以较好地支持 DevOps 和持续交付实践;拆分后的微服务可进行独立伸缩和功能扩展;技术异构,微服务的开发团队可根据其以往经验或微服务功能特点选择不同的技术栈进行开发.这些特点和好处驱动企业纷纷采用微服务架构来开发新系统或从单体架构向微服务架构迁移,其中比较典型的是 Netflix、Amazon 以及 eBay 等世界领先的互联网公司.

尽管微服务架构解决了传统单体架构以及面向服务架构设计中存在的诸多问题,使得架构设计能够更快地对业务的变化做出响应,IT 能力能够更好地满足业务发展的需求等等.然而,这些益处的先决条件是合适粒度的微服务化拆分,一旦所设计的微服务粒度不合适,会给系统带来很多负面影响,比如性能和可维护性等.我们的调研发现[2],微服务化拆分是该领域中最为关键的挑战之一.企业在现阶段的实践中仍旧主要依赖架构师或者项目管理者的个人经验,然而当面临复杂的系统时,这种方法低效且无法保证实践者期望快速享受微服务架构诸多好处的需求.反观学术领域,现有的拆分方法研究中,一部分需要手动构建满足拆分方法需求的数据信息或人工干预分析以供决策,同样面临效率低下的问题;另一部分自动化或半自动化的方法在一定程度上可以帮助解决效率的问题,但除了数据流驱动的微服务化拆分方法[3],其他相关工作均未真正地实现微服务架构倡导的 Database per Service 原则,即每个服务拥有自己的数据库.此外,在设计阶段,大多依靠架构师来主观判断所拆分出的候选微服务是否合适,缺乏系统化的指标体系来帮助快速实现候选微服务的合理性评估,从而帮助降低因评估不当或滞后至代码实现阶段而带来的系统重构代价.

为了应对上述挑战,探索和研究更加客观、高效且合理的微服务化拆分方法,对于期望使用微服务架构的实践者而言是至关重要的.本文旨在在现有的拆分方法基础上,研究一种更加高效的微服务化拆分方法 DFD-A.具体来讲,针对已有的数据流驱动的微服务化拆分方法的瓶颈问题进行优化改进,使用动静态分析技术相结合的方式改进其数据流信息收集、分析和构建流程,同时实现了两阶段的聚类算法来优化其基于自定义规则的微服务化拆分算法,最终实现原型工具来支持从数据收集分析、服务拆分到候选微服务评估的整个全自动过程,在保证所拆分微服务合理性的基础上,大幅度提升微服务化拆分的效率和灵活性.

作为面向软件过程中设计阶段的应用方法,本文的贡献点在于:所提出的微服务化拆分优化方法和原型工具可以帮助企业以及其他组织在进行微服务架构设计过程中,做出更加合理和客观的微服务化拆分决策,降低微服务化拆分设计过程的复杂度,帮助软件架构师以及项目管理者等更好地实践微服务架构设计.

本文第 1 节介绍背景及相关工作,尤其是本文所优化的数据流驱动的微服务化拆分方法的基本原理及缺陷问题.第 2 节阐述本文所提出的数据驱动的微服务化拆分优化方法 DFD-A,包括优化思路、整体流程和基于方法所实现的原型工具.第 3 节通过案例研究,验证并评估本文提出的方法对于优化目标的实现效果.第 4 节讨论本文的贡献点和不足之处.第 5 节总结本文工作,并给出针对不足之处的下一步工作展望.

1 背景及相关工作介绍

微服务架构目前受到了企业界和学术界的广泛关注,其在实践中主要有两种类型,即新的微服务系统设计和遗留系统通过重构向微服务架构迁移,二者皆避不开合适粒度的微服务化拆分问题.但目前,该问题的研究尚不成熟,仍处在探索发展过程中.

1.1 不同类型的微服务化拆分方法

当前,在微服务的拆分方面,更多地依赖主观判定和手动实施,缺乏系统化且高效的方法帮助实践者完成整个拆分评估流程[4].Newman 主张通过增量式地寻找服务边界进行微服务化拆分,并提出了寻找边界、循序渐进地将服务从杂乱依赖中剥离的一系列方法[5].微服务倡导的是根据业务能力来确定边界,这一点与 Evans 的领域驱动设计(domain-driven design,简称 DDD)思想相契合,即“围绕业务能力构建”[6].DDD 中的核心是领域模型,领域(domain)被定义特定范围内的知识、影响或行为,本质上包含问题空间和解决方案空间,与系统的核心业务功能相关.其中:问题空间包括核心域和其他可能的子域;解决方案空间包括一个或者多个限界上下文(bounded context),即一组特定的软件模型.限界上下文对于识别明显的微服务边界很有用[4],Richardson 给出了基于限界上下文的拆分模式[7],即按业务能力拆分、通过领域驱动设计的子域拆分、通过动词或用例拆分、通过名词或资源拆分,其中,前两种模式较为抽象,且需要不同类型的专家,分别是业务架构建模和领域驱动设计专家.这 4 种模式的提出仅停留的概念阶段,缺乏系统化的指导来帮助实践.Kecskemeti 等人[8]宣称通过对镜像(image)分析对微服务进行拆分,但该研究没有对与微服务相关的拆分原理和方法进行详细介绍,且结果缺乏可靠的验证.上述研究均未考虑粒度的问题.Hassan 等人[9]的研究对微服务的粒度有所关注,对架构中微服务的大小和数量、系统的整体非功能性需求满意度和个人的非功能性需求满意度作出的权衡,以此为依据对微服务进行拆分.其研究成果仅处于自然语言描述阶段,涉及的因素不够详细,划分机制还不够完善.

学术领域研究目前针对微服务的拆分问题提出的较为系统的方法主要分为 3 类.

1) 基于元数据辅助分析.使用抽象的软件架构描述作为输入数据来分析并进行拆分,例如 UML 图、用例或接口描述、代码版本控制库中的数据以及非功能性需求等内容.Gysel[10]等人提出了一种基于 16 种从文献及行业经验中提取出来的耦合标准的微服务化拆分方法——Service Cutter.该方法基于这 16 种耦合标准,使用聚类算法得到微服务候选集.该方法中的耦合标准优先级设定是人为设定,具有很大的主观性,不同的优先级设定可能将会产生不同的微服务候选集.Ahmadvand 等人[11]提出了基于软件的安全性与可拓展性需求进行微服务化拆分,他们使用具有安全性评估和可拓展性评估的 UML 用例图作为输入,从而计算出微服务候选集.Baresi 等人[12]基于 API 规范的语义相似性,计算接口间的语义相似度,从而计算出适合的微服务候选集.Mazlami 等人[13]提出了一种形式化的微服务提取模型,以支持在重构和迁移场景中进行微服务候选集推荐.
2) 基于静态源码数据分析.依赖应用程序源码的静态分析来进行拆分.Escobar 等人[14]基于源代码静态分析,通过 Java 注释识别数据类型,使用聚类算法得到微服务候选集.Levcovitz 等人[15]同样基于源代码静态分析,不同的是,他们是基于业务功能之间的关联以及类与类之间的静态关系和数据库表两者之间的依赖构建微服务候选集.
3) 基于动态运行数据分析.通过在模块或功能级别上测量软件系统的性能来找到合适的微服务化拆分方案,比如:Hassan 等人[16]定义了具有适应性边界的架构元素(环境),使用工作负载数据在运行时调整粒度,得到微服务组合;Klock 等人[17]使用遗传算法,基于工作负载计算最佳微服务部署和粒度.

综上所述,在单体系统向微服务迁移的过程中,可以立足于不同的视角、通过不同的数据分析方法进行微服务的拆分工作,但现有的方法都有其各自的局限性.基于元数据辅助分析的方法更加适用于软件系统架构设计阶段相关制品比较完善的情况,尤其是新的微服务系统设计,但该类方法的复杂、主观判断不准确等问题有待进一步解决;另一方面,很多遗留系统面临架构设计阶段相关制品不全、只有源码和运行时数据的窘境,这种情况下,基于静态代码分析和负载数据分析的方法更加适合.然而,单一视角下的数据分析无法全面获取系统的实际特征,比如单纯地通过静态源码数据分析无法获得系统运行时的依赖关系,如与多态、依赖注入和控制反转相关的动态特征,也无法真正获取用户使用规律不同而影响的系统模块之间的依赖和耦合关系.而动态分析可能会出现代码覆盖率低的问题,而无法获得应用的全部特征信息.不够全面的数据分析可能会造成微服务识别的偏差,从而影响微服务化拆分结果的质量.当前,在面向微服务的拆分设计问题的挑战,驱动本文探究针对该问题更加系统且有效的解决方案.

1.2 数据流驱动的微服务化拆分方法(DFD)

微服务架构提倡根据 Database per Service 原则,为每个微服务设计自己的私有数据库,其他微服务需要通过微服务暴露的接口来访问数据库中的数据,从而实现数据的隔离性.这有助于解决单体系统中存在的问题,其通常使用的共享数据库的缺点是容易造成服务的紧耦合,使得任何模式更改都需要在相关服务间协调,可能会增加部署更改的复杂性.单个数据库的设计也使得每次服务在扩展的时候都需要扩展整个数据库,降低了服务的可扩展性.此外,复杂单体系统中的单一共享数据库往往很庞大,将会对系统性能的提升造成一定的挑战.

然而,简单粗暴地将数据拆分到不同的微服务中,在实践中被认定为一种反模式.一方面,数据间存在的耦合性以及服务对于数据访问的频率和类型不同等因素,是微服务化拆分过程中需要考虑的因素,否则将可能会造成拆分后的微服务频繁的数据交互而降低整个系统的性能;另一方面,一旦共享数据库被拆分到不同的微服务所拥有的独立数据库中,服务粒度的调整可能会造成较高的数据迁移成本.因此,在向微服务迁移过程中,需要综合考虑业务操作与数据之间的交互关系来支持微服务的拆分.

现有的拆分问题研究中,数据流驱动的微服务化拆分方法(以下简称 DFD)[3]给出了针对上述问题的易理解操作且结果合理的解决方案.其方法简单来讲是分析业务操作与数据存储之间的交互特点,并构建仅关注这二者之间交互关系的精简数据流图(DFDPS),在此基础上,通过预设的规则和算法进一步聚合和数据库表关系紧密的业务操作,从而实现在减小微服务粒度与数据隔离性的同时,尽可能降低对传输性能和数据一致性造成的影响.该方法的整个过程分为 4 步,用到该方法预先设定的规则和算法,具体如图 1 所示.

1) 需求分析:由需求分析人员对待拆分的软件系统需求进行分析.根据系统用例或自然语言描述的业务逻辑,识别领域上下文和实体,为数据流图的构建做准备.
2) 精简数据流图构建:基于第 1)步中的用例和业务逻辑分析,由数据分析人员手动构建详细版本的分层数据流图(第 0 层和第 1 层)和精简的数据流图 DFDPS.详细版本的数据流图构建需要遵循方法预设的分解规则 1~规则 3;而精简数据流图(DFDPS)的构造过程需要遵循规则 4.所构造的 DFDPS 仅关注操作 *P* 与相关数据存储 *DS* 之间的关系,而排除了诸如外部实体以及数据流传输的具体数据之类的无关信息.
3) 可拆分数据流图构建:从 DFDPS 中提取操作 *P* 和数据存储 *DS* 之间数据交互信息,将精简数据流图 DFDPS 通过算法自动转化可拆分的数据流图.之所以关注操作 *P* 和数据存储 *DS* 之间的数据交互,目的是为了避免数据存储在拆分过程中被分到不同的微服务中,这样可以保证在减小微服务粒度的同时,尽可能减少不必要的数据一致性的问题.
4) 候选微服务识别:对第 3)步导出的可拆分的数据流图,通过设计的算法进行拆分,具体是根据规则 5 对同一数据存储的相关操作进行聚合,然后根据规则 6 对出现重复操作的聚合结果进行合并,合并后的结果作为候选微服务.

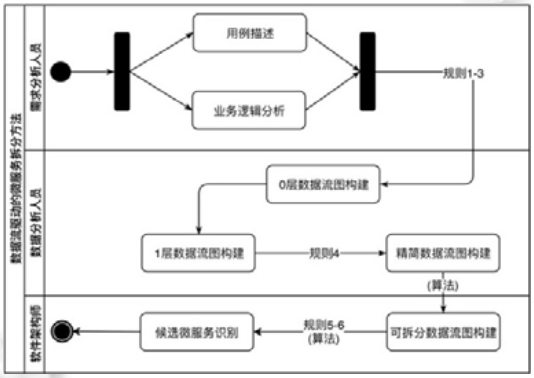

Fig.1 Process of the dataflow-driven decomposition method for microservices[3]

图 1 数据流驱动的微服务化拆分方法过程[3]

尽管通过案例研究证实了 DFD 方法可以产出合理且易于理解的微服务化拆分结果,但仍然存在两方面的问题有待改进.

- 一方面是数据收集过程的效率较低.由于 DFD 方法目前仅实现了半自动化拆分,拆分的前提是需要依赖人工分析系统需求,并根据系统用例或业务逻辑手动构建数据流图,因此其更适用于中小型软件系统,对于大型复杂系统,这种手动的数据收集方式在一定程度上降低了整个 DFD 方法实施的效率.
- 另一方面是拆分算法对于结果可修改性的支持度不高.DFD 方法所使用的拆分算法完全基于生成的可拆分的数据流图和预设的规则 5、规则 6 来进行实现的,前期人工分析的准确性将在很大程度上影响拆分结果的可靠性.而 DFD 方法给出的微服务化拆分建议是唯一的且不支持调整,这也限制了该方法在实际应用过程中的灵活性.

2 优化的数据流驱动的微服务化拆分方法

本文针对上述 DFD 方法的瓶颈问题(效率和灵活性)进行改进,在保证拆分结果有效性的基础上,提出了一种更加优化的数据流驱动的微服务化拆分方法 DFD-A.本节详述所提出的 DFD-A 方法的优化思路、整体流程以及基于该方法实现的原型工具.

2.1 优化思路

整体来讲,本文所关注的数据流信息参考 DFD 方法中的简化数据流图(不关注外部实体和数据流上的具体数据),概念化表示如图 2 所示.

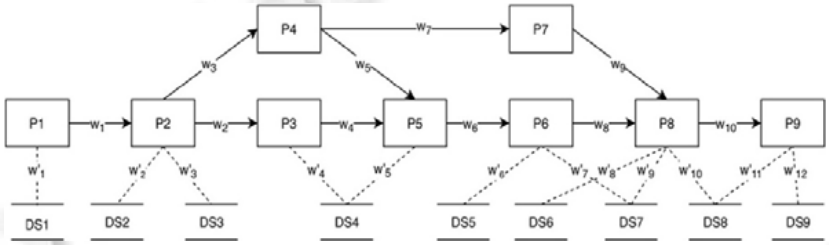

Fig.2 Conceptual dataflow diagram of a legacy system

图 2 遗留系统数据流图的概念化表示

除了业务操作结点 $P=\{P_1,P_2,\ldots,P_7\}$和数据存储结点 $DS=\{DS_1,DS_2,\ldots,DS_9\}$,还收集了业务操作 P 之间的交互频率 $W=\{w_1,w_2,\ldots,w_{10}\}$以及业务操作和数据存储之间的交互频率 $W'=\{w'_1,w'_2,\ldots,w'_{12}\}$.

有效性方面,基于 DFD 方法的基本原理,交互越频繁的模块关系越紧密,如果被拆分到不同的微服务中,会导致分布式环境下信息交互的成本增加,从而降低整个系统的性能.考虑到操作之间的交互频率信息反映了业务操作之间的耦合程度,本文提出的 DFD-A 方法主要依据该信息来作为权重来实现基于图聚类算法的业务操作子图拆分,可采用的典型算法如基于最小生成树(minimum spanning tree,简称 MST)的图聚类算法[13].该算法的优点是不用计算相似度,可以通过生成关于业务操作结点的最小生成树,并按最大距离原则依次删除一条最小生成树的边(距离越大,说明耦合性越低)来达到拆分的目的,直到划分为符合个数预期的业务操作集合;与此同时,微服务架构倡导每个微服务拥有自己的数据库(database per service 原则),这在 DFD 方法中有所关注.本文对业务操作进行聚类,得到的集合中可能包含与数据库有交互关系的业务操作结点,二者之间的交互频率信息反映了数据流图中业务操作集合和数据存储之间的耦合关系.鉴于操作之间、操作与数据存储之间的交互关系紧密程度不同,本文改进了传统的 *K*-means 算法来保证数据存储被有效拆分到相应的服务中.具体来讲,针对传统 *K*-means 算法采用欧式距离来度量样本间相似度的方法对图结构数据集失效问题,本文设计了计算图结点之间的最短距离作为新的度量方式;其次,支持用户输入中心点来避免 *K*-means 随机选择聚类中心带来的效果不稳定问题,*K*-means 的聚类数据对象是初步拆分得到的操作集合与数据库表,聚类中心是用户根据对被拆分

系统业务需求的了解所确定的数据库中心表.

效率方面,优化前的 DFD 方法中,通过手动构建数据流图的方式来分析用于微服务化拆分的数据流信息,包括业务操作之间和业务操作与数据存储之间的交互信息.数据流图的手动构建与分析耗时较长,本文探究了一种自动化的数据流信息收集与分析方法,将有助于大幅度提升 DFD 方法的应用效率.

现有研究中,动态链路追踪技术[18]和静态源码分析技术[16,17]等都可以帮助实现本文所需要的数据流信息的自动收集,从而达到效率优化的目的.动态链路追踪本身是为故障定位而提出的一种系统行为特征的动态分析技术,为了锁定故障根源,需要获取业务操作的执行链路信息,包括操作与操作、操作与数据库之间的交互信息[18],而这些信息正是应用 DFD 方法进行拆分所需要分析的;类似地,静态分析技术可以帮助获得软件系统源码的静态结构特征,包括类、方法之间的静态关联关系.前文已经提到单一视角下的数据分析无法全面获取系统的实际特征的问题,有鉴于此,本文采用动静态分析相结合的方式来帮助从遗留系统中更加客观高效地收集数据流信息(业务操作之间和业务操作与数据存储之间的交互信息),旨在保证数据分析全面性的基础上,解决 DFD 方法中人工的数据流图构建与分析效率不高的问题.

灵活性方面,本文依据自动化收集的数据流信息生成带权图,继而选用成熟的图聚类算法来优化 DFD 方法中基于自定义规则、不支持调整的微服务化拆分,基于 Kruskal 的最小生成树聚类算法支持通过参数的设置来定制服务的拆分方案,可以在保证所拆分微服务合理性的基础上,大幅度提升微服务化拆分的灵活性.此外,可以通过实现原型工具的方式来支持从数据收集分析、服务拆分到候选微服务评估以及候选微服务再调整的整个全自动过程,帮助进一步提高微服务化拆分的效率和灵活性.

2.2 整体流程

本文针对前面讨论的现有数据流驱动的微服务化拆分方法 DFD 的缺陷问题进行改进,提出了更加优化的微服务化拆分方法 DFD-A.改进后的 DFD-A 方法包括 3 个阶段,即基于动静态分析技术相结合的数据流信息收集、基于最小生成树的业务微服务化拆分和基于 *K*-means 的数据存储拆分.方法的整体流程如图 3 所示.

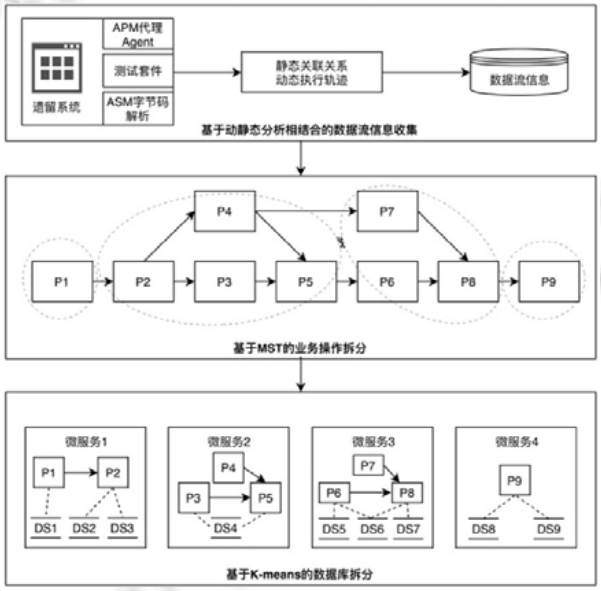

Fig.3 Process of the optimized microservices-oriented decomposition method driven by dataflow

图 3 优化的数据流驱动的微服务化拆分方法过程

具体来讲,首先通过动态链路追踪技术与静态字节码解析技术相结合的方式来更准确高效地收集能客观反映内聚耦合程度的动静态数据交互信息(包括操作与操作、操作与数据存储之间的交互).在此基础上构建带

权图,并实现第 2 和第 3 阶段的图聚类算法以来进行业务微服务和数据存储的拆分:第 2 阶段中,对于第 1 阶段追踪到的动静态交互信息中操作与操作的调用关系生成带权图,使用基于最小生成树 MST 的算法进行聚类,得到业务操作的聚类集合,即业务服务的拆分方案;第 3 阶段以第 2 阶段聚类后的业务操作集合为结点,并结合第一阶段追踪的动静态交互信息中操作与数据库表的交互关系生成带权图,使用 *K*-means 算法进行聚类,得到数据库表对应操作结合的拆分结果.综合第 2 和第 3 阶段的聚类结果,输出微服务的候选,包括业务服务集合和相应的数据库表分配方案.

2.2.1 数据流信息收集过程与分析过程优化

本文首先从效率的角度对现有的数据流驱动的微服务化拆分方法 DFD 进行了优化,基于动态链路追踪技术与静态字节码解析技术相结合的自动化方式来代替手动的数据流图分析与构建,实现数据流信息的自动收集,以供后期分析业务操作的数据交互关系信息进行聚类分析,以支持候选微服务的识别.

1) 基于 Pinpoint 的动态链路追踪

本文选择了开源工具 Pinpoint(https://github.com/naver/pinpoint),并对其进行封装和改造,以支持单体系统运行时的动态链路分析.Pinpoint 本身是一个应用性能管理(application performance management,简称 APM)工具,其具备分析系统的整体结构以及追踪其内部不同服务之间通信链路的功能.Pinpoint 可以通过分布式事务跟踪,实现跨应用的消息定位,同时能够自动检测系统的拓扑结构,提供代码级别的可见性;此外,该工具主要通过 Java 探针技术实现客户代码无侵入的链路分析,即在使用 Pinpoint 收集运行时信息时,无需改写应用代码,无需增加锚点.

基于 Pinpoint 进行数据流信息的过程如图 4 所示,流程中主要涉及了代理 Agent、日志收集器 Collector 和控制台 Web 这 3 个组件.其中,代理 Agent 组件用于数据采集,通过打成 JAR 包的形式和软件系统一起启动且与其共享 JVM,并且可以将运行时代码级别的调用关系数据定时发送给日志收集器 Collector,以供后期的链路特征分析;Collector 组件接收到代理 Agent 发送的数据,会将其存储到 Hbase 中;最后由控制台 Web 组件从 Hbase 中读取数据,并展示到自带的前端页面.

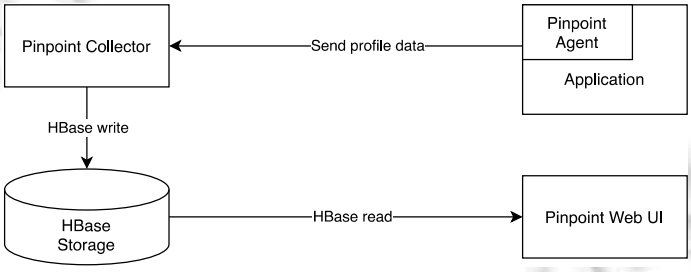

Fig.4 Process of dynamic tracing using Pinpoint

图 4 基于 Pinpoint 的动态链路追踪过程

通过 Pinpoint 进行服务链路日志采集需要自行实现代理 Agent 插件,为了降低该步的复杂度并减少出错的可能性,本文使用 pinpoint-plugin-generate(https://github.com/bbossgroups/pinpoint-plugin-generate)帮助快速生成目标系统的 Pinpoint 代理插件;然后将目标系统生成的代理插件加入到 Pinpoint plugins 包中,并在目标系统启动时,通过“java agent”启动 pinpoint 代理;最后,Pinpoint 会根据插件所指定的数据进行跟踪,从而实现对目标项目数据的采集.为了在本文实现的方法中无缝集成 Pinpoint 的分析结果,本文改造了 Web 模块,使其暴露出以供本文基于优化方法实现的原型工具平台查询的接口,实现了该平台基于 Pinpoint 对于待拆分系统动态运行数据流信息的自动收集(详见第 2.2 节).

2) 基于 ASM(https://asm.ow2.io)的静态字节码解析

为了解决动态链路追踪覆盖率不全、无法获得未执行到的链路问题,本文结合使用了基于 ASM 的静态字节码解析技术来补充系统的类、方法等的静态关联信息,保证所收集的数据流信息的全面性.ASM 是一个通用

的轻量级 Java 字节码处理框架,它既可以通过直接产生字节码文件的方式动态生成类,又可以在虚拟机加载类前,通过修改类行为的方式来增强类的功能.Java 字节码.class 文件通过严格格式定义来描述 Java 类的类名/属性/方法和指令.ASM 从.class 文件中读入字节码信息后,可以分析类结构,改变类行为,甚至能够生成新的类.

ASM 通过访问者模式实现类信息的获取.访问者模式是一种将数据操作和数据结构分离的设计模式,有一个元素和一个访问者.元素通过 *accept*(·)方法接受访问者作为参数,并将元素自身传给访问者,在 *accept*(·)方法中调用访问者的 *visit*(·)方法.在 ASM 中,ClassReader,MethodeNode 等作为被访问的元素,ClassVisitor,MethodVisitor,FieldVisitor 等是访问者接口.开发者可以使用 ClassReader 类用来解析编译过的 class 字节码文件,使用 ClassWriter 类用来重新构建编译后的类,通过实现访问者接口的 *visit*(·)方法实现自定义访问.

2.2.2 微服务化拆分算法优化

前面提到,现有的数据流驱动的微服务化拆分算法存在特征分析单一且灵活性不高的问题,本文对于这两方面的问题从算法层面进行了改进优化.不同于 DFD 方法仅通过交互关系的有无来构建数据流图,并使用有限的规则来进行拆分,本文增加了两个能反映耦合程度的数据流信息特征收集,即操作之间以及操作和数据存储之间的交互频率,在此基础上,设计并实现了两阶段的聚类算法(如图 5 所示)来分别支持业务操作拆分和相应数据库表的分配,两者结合,从而生成候选微服务(业务操作+数据库)的拆分方案.

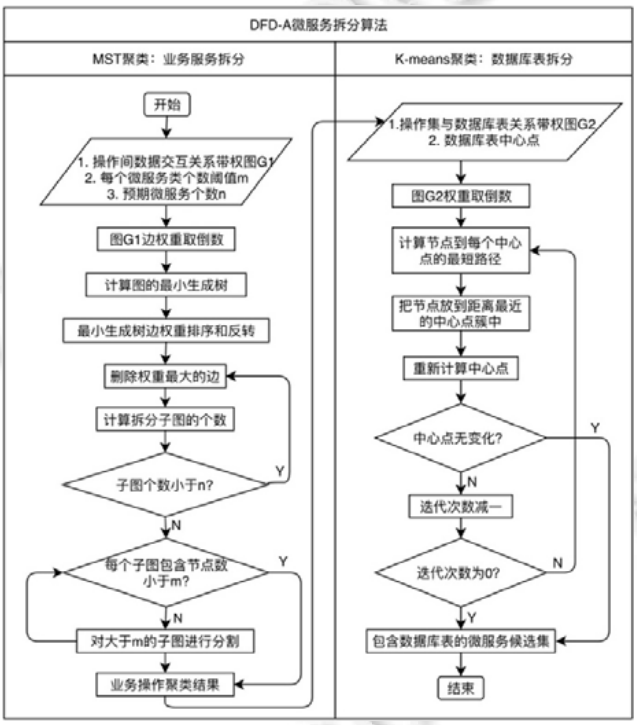

Fig.5 Microservices-oriented decomposition algorithm of the DFD-A method

图 5 DFD-A 方法的微服务化拆分算法

1) 基于 MST 的业务操作拆分

本文实现了基于最小生成树 MST 的算法来完成业务操作的聚类拆分,该算法依赖的参数是根据第 2.1.1 节收集的操作间的数据流信息而生成的带权图 *G*1、预期的微服务个数 *m* 和每个微服务的类个数阈值 *n*.其中,图 *G*1 的结点表示的是类级别的业务操作;边权重表示的是操作结点之间的交互频率,边权重值越大,表示操作之间的交互越频繁,耦合程度越高,即越倾向于被分到同一个微服务中.该阶段算法的基本原理是:首先,通过 Kruskal 算法生成图 *G*1 的最小生成树 MST;然后对最小生成树的边权重进行分析和删除,以实现图的拆分.由于

*G*1 的边权重越大表示结点间的距离越近,因此在计算最小生成树之前需要对边权重值取倒数;所生成的 MST 中包含了关系最为紧密的业务操作结点和边,对 MST 涉及的边权重值进行排序和反转操作,依次删除权重值越高的边来实现耦合度较低的业务操作的拆分,并通过深度优先搜索算法(depth-first-search,简称 DFS)遍历剩余边组成的子图.本文通过预期的微服务个数和每个微服务类个数阈值这两个参数作为算法的终止条件,不同的参数设置可以产生不同的微服务化拆分结果,用户可以根据自己的需求进行设置,一定程度上保证了方法在使用过程中的灵活性.

2) 基于 *K*-means 的数据库表拆分

Database per Service 原则倡导每个服务维护自己的数据库,其他的微服务通过向外暴露的接口来访问服务私有的数据,因此在向微服务架构迁移时,需要对数据库表进行拆分.现有的 DFD 方法中涉及了数据库表的拆分,但与业务操作拆分类似,其仅考虑了操作与数据存储之间有无数据交互关系,使用预先定义的简单规则来实现拆分;其他工作集中在通过设计阶段数据库实体的结构分析以及其被系统访问的特征来进行拆分.然而一般情况下,单体系统,尤其是复杂单体系统中使用的集中式数据库中,数据库表之间存在耦合和粘连性,设计阶段单一特征分析和简单的规则无法全面解决该问题,有可能会造成拆分的粒度过细或过粗,从而导致后期微服务之间的频繁交互而影响性能.

本文在前一阶段业务拆分结果的基础上,依据第 2.1.1 节收集的类级别的业务操作与数据存储之间的交互关系来分析其在运行时对数据的访问特征,通过 *K*-means 算法来实现数据库表的有效拆分.*K*-means 算法依赖的参数是上一阶段聚类产生的业务操作集合为结点、集合中操作与数据库表交互关系为边所生成的带权图 *G*2,以及根据数据库实体关系分析初步设定的数据库表中心点集合.具体实现过程中,由于操作对数据的访问频率越高同样代表其之间的关系越紧密,因此以图 *G*2 边权重值倒数计算结点到中心点的 Dijkstra 最短路径,将结点放到与之路径最短的中心结点簇中;然后在每个中心点的簇中选出结点度最大的结点为该簇的中心点(如果度相同,则选取权重最大的),直至中心点不再发生变化或迭代次数为 0;最后输出中心点与其簇中结点,其代表了包含数据库表的微服务候选集.

2.3 原型工具

本文设计并实现了原型平台来支撑优化的数据流驱动的微服务化拆分方法,平台的架构设计如图 6 所示.平台采用了前后端分离的分层架构,包含了与用户直接交互的展示层、提供业务接口的服务层和存储数据的数据存储层.展示交互层使用 Vue 和 Element 实现,给用户提供方便易用的可视化页面.用户可以在系统管理页面完成对拆分项目和项目执行任务的管理,在数据收集页面对拆分项目的系统数据进行收集,在服务拆分页面执行拆分算法生成微服务候选集并查看该拆分方案的详情,在评估对比页面比较各个划分方案的指标变化.服务层通过 SpringBoot 来实现,向展示层暴露 RESTful API 来提供服务.在服务层中又分为核心业务层和基础服务层:核心业务层是用来实现平台的主要功能,包括系统管理模块、数据收集模块、服务拆分模块和结果评估模块;而基础服务层是由职责较为单一的基础功能模块组成,考虑到平台后期的可扩展性,这些基础功能模块以插件的形式提供服务,平台用户可根据所收集的信息或算法的不同来进行功能扩展.目前包含的插件服务中,数据流信息收集服务主要为微服务化拆分提供了基于 Pinpoint 的项目运行时数据信息收集和基于 ASM 的静态源码数据信息收集的功能,基于数据流的拆分服务是基于本文优化提出的两阶段的聚类算法的实现,指标评估服务给出了微服务化拆分方案的评判指标,任务管理服务负责调度系统中的插件运行,插件管理服务帮助系统组织具体插件的注册和卸载.其中,平台通过数据流信息收集服务插件远程连接经过本文改造后的 Pinpoint Web 暴露的接口获取系统的数据流信息.具体来讲,改造后的接口首先通过 appName 找到与目标项目有关的运行时的信息,然后使用 SpanService 获取 SpanResult 列表,每一个 Span 代表事务经过单个逻辑结点的跟踪,Span 记录了重要的类调用及其相关数据,最后根据需要的项目业务结点对 SpanResult 的数据进行筛选和整理,返回目标系统类级别的调用关系.数据存储层使用了 MySql 数据库,通过与服务层 MVC 模型中的 Dao 层通信进行数据持久化的工作.

原型工具核心的服务拆分模块的实现效果如图 7 所示,该页面中展示了拆分方案的参数信息、拆分后的评

估信息和微服务候选集关联信息.其中,拆分方案的参数基础信息包含了拆分方案的基本参数、用户填写的拆分算法参数和插件使用的数据集.拆分方案参数信息右侧是该拆分方案的评估指标,按照微服务包含的代码行数进行饼图展示,饼图的占比代表了服务粒度的大小.当鼠标放到某个饼图的微服务块上,将展示出该服务的名称以及所有相关的内聚和耦合指标.系统中微服务候选集的整体指标在右侧显示.微服务候选集关联信息通过Echarts 关系图展示,当选中一个微服务结点,会淡化显示与之无关的微服务结点和关联边,并在右下侧区域详细展示该微服务结点所包含的类结点信息.通过勾选需要调整的类结点、填写需要移动的目标服务分区,可以进行人为手动干预拆分.

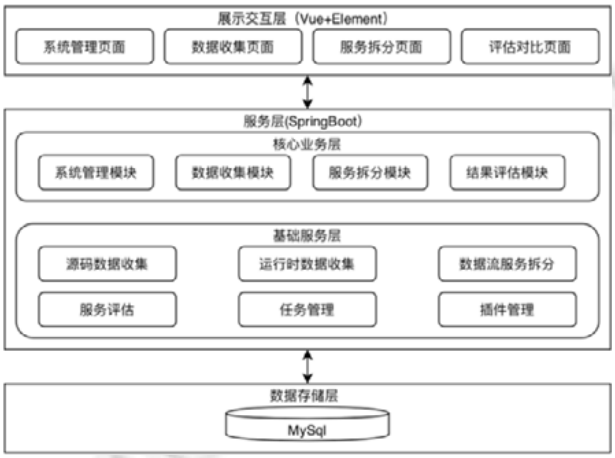

Fig.6 Architecture of the platform prototype based on the DFD-A method

图 6 基于 DFD-A 方法的平台原型架构

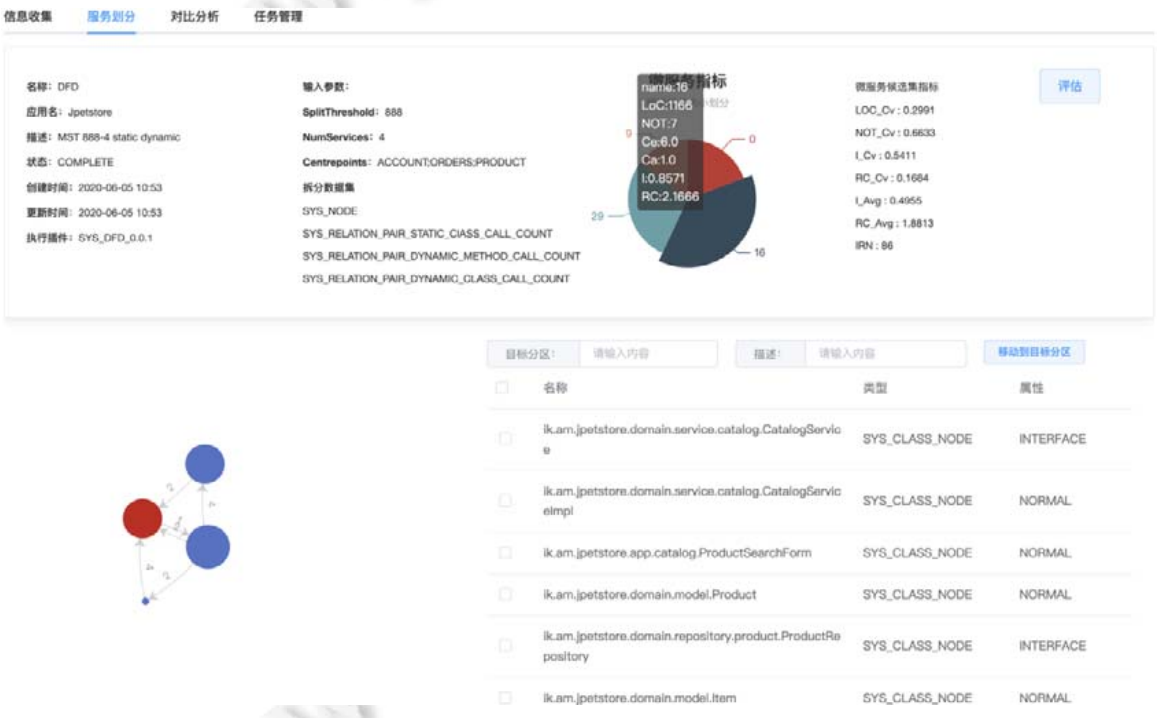

Fig.7 Implementation of the service decomposition module

图 7 服务拆分模块实现

3 案例研究及评估结果分析

3.1 评估案例

为验证本文所提出的优化的微服务化拆分算法 DFD-A 及原型平台,与 DFD 方法[3]的案例研究和方法评估思路类似,本文首先需要选择合适的案例,然后使用优化后的 DFD-A 方法及其原型平台进行拆分,最终对拆分结果进行指标评估,并与相关工作方法的应用结果进行对比.尽管 DFD 方法评估中使用的“Cargo Tracking System”案例(https://github.com/citerus/dddsample-core)较为典型,但其在 Github 上公开的源码仅是基于需求分析实现的简单原型,缺乏必要的业务细节,无法获取有效的数据流信息来评估本文提出的优化方法应用的效果.本文最终选择了同样被微服务化拆分工作广泛应用且实现较为完整的单体系统 Jpetstore(https://github.com/mybatis/jpetstore-6)作为案例进行方法的评估研究.作为一个宠物商店电子商务系统,其主要为用户提供宠物浏览、添加宠物到购物车、下单购买宠物的功能,该案例多次被应用到与微服务化拆分的相关的研究中[19−21].因 DFD 方法的案例研究[3]已初步表明现有的数据流驱动的微服务化拆分方法相对于其他方法的优势,本文的案例研究只对比分析应用优化前的 DFD 方法和优化后的 DFD-A 方法及其原型平台进行拆分后的结果.

本文的案例研究在应用上述两个方法之前都需要对 Jpetstore 案例的业务需求有基本的了解,但二者的目的不同.DFD 方法中,了解业务需求和用例用以构建不同层级的数据流图来支持基于规则的微服务化拆分;而 DFD-A 方法中,则需要根据业务需求设计尽可能覆盖整个系统全部功能的测试用例,并使用测试套件如 Jmeter 来模拟用户请求,为基于 Pinpoint 的链路追踪和数据收集做准备.需要注意的是:相对于数据流图,测试用例在企业级项目中一般比较容易获取.本文通过对 Jpetstore 的需求和业务逻辑进行分析,得到了该系统要实现的 10 个主要的高层次功能需求用例和 14 个数据库表,分别见表 1 和表 2.

Table 1 High-level use cases of Jpetstore

表 1 Jpetstore 高层次用例需求

编号	用例名称	说明
P_1	登录	用户登录系统
P_2	注册	用户注册系统账号
P_3	查询账户信息	用户查询账号相关信息
P_4	修改账号信息	用户查询账号相关信息
P_5	浏览商品信息	用户浏览系统中的宠物信息
P_6	加入购物车	用户将宠物商品加入购物车
P_7	移除购物车	用户将宠物商品从购物车中移除
P_8	更新购物车	用户更新购物车中宠物商品的数量
P_9	下单	用户下单购买宠物
P_{10}	查看订单	用户查看购物订单

Table 2 Database tables of Jpetstore

表 2 Jpetstore 数据库表

编号	数据库表	说明
S_1	Account	存放用户相关数据,如用户的 id、邮箱 email 和所在城市 city 等
S_2	Bannerdata	存放用户最喜爱的类别的 banner 数据,如图片地址 bannername
S_3	Category	存放商品分类数据,如分类的 id、名称 name 和描述 descn 等
S_4	Inventory	存放商品的库存数据,如商品项 id 和库存量 qty
S_5	Item	存放商品项数据,如商品项 id、定价 listprice 和供应商 supplier 等
S_6	Lineitem	存放订单项数据,如订单 id、商品价格 untprice 和购买数量 quantity 等
S_7	Orders	存放订单数据,如订单 id,下单日期 orderdate 和地址信息 shipaddr1 等
S_8	Orderstatus	存放订单状态数据,如订单 id、订单状态 status 和时间 timestamp 等
S_9	Product	存放商品数据,如商品 id、商品名称 name 和商品所属类别 category 等
S_{10}	Profile	存放用户描述数据,如用户 id、最喜爱的类别 favategory 和使用语言 langpref 等
S_{11}	Sequence	存放编号数据,如编号名称 name 和下一个编号 nextid
S_{12}	Signon	存放用户登录信息数据,如用户账号名称 username 和用户登录密码 password
S_{13}	Supplier	存放商品供应商数据,如供应商名称 name,供应商电话 phone 和供应商地址 addr1
S_{14}	Cart	存放购物车数据,如加入的商品项 item 和商品数量 quantity 等

3.2 评估流程

由于所选的案例规模不大,为了分析本文所提出的 DFD-A 方法相对于原本 DFD 方法的优化效果,本文选择了一名熟练掌握软件工程领域基础知识和开发技能的研究人员 *A* 来主要负责两个方法在 Jpetstore 系统上的应用.本文考虑并排除了一些可能会影响方法效率对比公平性的因素,比如选择不了解优化前的 DFD 方法的研究人员;此外,本文安排了熟知并使用过 DFD 方法和 DFD-A 方法的研究人员 *B*,其负责对研究人员 *A* 在应用这两个方法过程中所生成的制品和结果进行检查,以保证方法应用的准确性.

DFD 方法的应用依据的是第 1.2 节中介绍的流程[3],参与评估的研究人员基于业务逻辑和需求分析构建不同层级(第 0 层和第 1 层)的数据流图.其中,第 0 层数据流图中的操作 P_6 和 P_9 在第 1 层的数据流图中分别被拆分为两个更细粒度的操作 P_{61},P_{62} 和 P_{91},P_{92};然后使用 DFD 方法中基于规则的算法对第 1 层的数据流图进行拆分,最终得到了包含 4 个微服务的方案(见表 3).

Table 3 Decomposition results of the Level 1 data flow diagram of the case Jpetstore using the DFD method

表 3 案例 Jpetstore 应用 DFD 方法对第 1 层数据流图的拆分结果

编号/名称	语句集	操作集	数据存储集
MS_1 AccountService	$\{S_{12}\to P_1,P_2\to S_{12},P_2\to S_1,S_1\to P_3,$ $P_4\to S_1,P_2\to S_{10},S_{10}\to P_3,S_2\to P_3\}$	$\{P_1,P_2,P_3,P_4\}$	$\{S_1,S_2,S_{10},S_{12}\}$
MS_2 ProductService	$\{S_3\to P_5,S_5\to P_5,S_5\to P_{61},$ $S_5\to P_{91},S_9\to P_5,S_{13}\to P_{91}\}$	$\{P_5,P_{61},P_{91}\}$	$\{S_3,S_5,S_9,S_{13}\}$
MS_3 CartService	$\{P_{62}\to S_{14},P_7\to S_{14},P_8\to S_{14},S_{14}\to P_9\}$	$\{P_{62},P_7,P_8\}$	$\{S_{14}\}$
MS_4 OrderService	$\{S_{11}\to P_{91},P_{92}\to S_{11},P_{92}\to S_6,S_6\to P_{10},$ $P_{92}\to S_7,S_7\to P_{10},S_8\to P_{10},P_{92}\to S_8,P_{92}\to S_4\}$	$\{P_{92},P_{10}\}$	$\{S_4,S_6,S_7,S_8,S_{11}\}$

DFD-A 方法应用过程中,首先设计覆盖系统功能的测试用例进行测试,以支持数据收集,然后部署 Pinpoint 监控环境,使用 Pinpoint Agent 对 Jpetstore 进行运行时的数据流信息采集,Pinpoint Collector 接收采集到的信息存入 HBase 中,然后由本文开发的原型平台通过运行时数据收集插件远程连接本文改造后的 Pinpoint Web 接口获取动态信息;在静态源码数据收集部分,使用源码数据收集插件分析上传的 Jpetstore JAR 包,通过 ASM 完成对源码信息的静态特征分析,在源码数据收集时,提供了 Package 参数用于指定需要扫描的包路径;然后,平台会对收集的数据流信息使用基于数据流的拆分服务插件进行业务操作和数据库表的拆分.由于该插件是本文优化提出的两阶段的聚类算法(MST 和 *K*-means)的具体实现,因此需要输入依赖的参数,即 MST 中每个微服务类阈值和预期微服务个数、*K*-means 中的中心点集合.此处由本文的研究者根据经验和对系统的理解设定了相关参数,其中,MST 算法中,微服务类阈值为默认的最大值,预期微服务个数 n 设置为 3,4 和 5 这 3 种情况;*K*-means 算法中,数据库的中心点集合设为 $\{S_1,S_7,S_9\}$.具体实践过程中,架构师或相关决策者可以根据经验、需求以及系统和组织结构的具体情况进行参数的设置和调整.其中,预期微服务个数同样设置为 4 的情况下,应用 DFD-A 方法最终得到的微服务化拆分方案见表 4.

Table 4 Decomposition results of the case Jpetstore using the DFD-A method

表 4 Jpetstore 案例应用 DFD-A 方法的拆分结果

微服务	类	数据库表
ProductService	ik.am.jpetstore.domain.service.catalog.CatalogServiceImpl	Item
	ik.am.jpetstore.app.catalog.CatalogController	Product
	ik.am.jpetstore.domain.model.Product	Category
	ik.am.jpetstore.domain.model.Item	Supplier
	ik.am.jpetstore.domain.model.Category	–
	ik.am.jpetstore.app.catalog.ProductSearchForm	–
OrderService	ik.am.jpetstore.domain.model.Order	Inventory
	ik.am.jpetstore.app.order.OrderController	Sequence
	ik.am.jpetstore.app.order.OrderForm	Orders
	ik.am.jpetstore.app.order.OrderHelper	Orderstatus
	ik.am.jpetstore.domain.service.order.OrderServiceImpl	Lineitem

Table 4 Decomposition results of the case Jpetstore using the DFD-A method (Continued)

表 4 Jpetstore 案例应用 DFD-A 方法的拆分结果(续)

微服务	类	数据库表
OrderService	ik.am.jpetstore.domain.model.Sequence	–
	ik.am.jpetstore.domain.model.LineItem	–
	ik.am.jpetstore.domain.model.UserDetails	–
AccountService	ik.am.jpetstore.app.account.AccountController	Account
	ik.am.jpetstore.app.account.AccountForm	Profile
	ik.am.jpetstore.app.account.AccountHelper	Bannerdata
	ik.am.jpetstore.domain.model.Account	Signon
	ik.am.jpetstore.domain.service.account.AccountServiceImpl	–
	ik.am.jpetstore.domain.service.user.UserDetailsServiceImpl	–
	ik.am.jpetstore.app.account.PasswordEqualsValidator	–
CartService	ik.am.jpetstore.app.cart.CartHelper	Cart
	ik.am.jpetstore.domain.model.Cart	–
	ik.am.jpetstore.app.common.session.HttpSessionEventLoggingListener	–
	ik.am.jpetstore.domain.model.CartItem	–
	ik.am.jpetstore.app.cart.CartController	–
	ik.am.jpetstore.app.cart.CartForm	–
	ik.am.jpetstore.app.common.session.EnableSynchronizeOnSessionPostProcessor	–

3.3 结果分析

本文旨在从优化目标的实现效果来评估所提出的 DFD-A 方法,即在保证拆分结果有效性的基础上,提高方法的效率和灵活性.

1) 有效性

应用优化后的方法对 Jpetstore 案例进行拆分后给出的方案有效性,是本文在评估过程中优先关注的,有效性的评估需要依赖相应的度量指标.参照相关研究中可用于可自动化评估的指标[3,22],本文对于拆分方案有效性的评估主要涉及个体微服务和整体系统两个层级.其中,个体微服务层级的评估延用 DFD 方法研究[3]中的耦合和内聚度量指标——向心耦合(*Ca*)、离心耦合(*Ce*)、不稳定性(*I*)和关系内聚(*RC*).由于 *I* 和 *RC* 与性能存在显著的相关性[23],可以反映特定方案的整体性能表现,因此,本文从系统层级计算个体微服务与这两个指标相关的均值 *Avg* 和变异系数 *Cv*(coefficient of variation)(标准差与均值之间的比率,主要用于识别数据中的异常值,*Cv* 的值越大,表示差异越大)[22]来预测评估拆分后系统相应指标的整体情况和服务间的差异情况.所使用的评估指标具体见表 5.

Table 5 Metrics to evaluate the decomposition results[3,22]

表 5 拆分结果评估指标[3,22]

类型	层级	指标	说明
耦合	微服务	*Ca*	*Ca*((afferent coupling))表示依赖该包(服务)包含类的外部包(服务)包含类的数目,数值越大,表示某个包(服务)的职责越大、越稳定
	微服务	*Ce*	*Ce*(efferent coupling)表示被该包(服务)包含类所依赖的外部包(或服务)的数量,数值越大,表示该包越不独立、越不稳定
	微服务	*I*	*I*(instability)通过计算*Ce*和*Ce*+*Ca*的比例来衡量一个包(服务)的不稳定程度,取值区间是[0,1],其中,*I*=0 说明该包(服务)最稳定,即一个包(服务)不依赖其他任何包(或服务);而 *I*=1 表示该包(服务)极不稳定
	系统	*I_Cv*	整体系统的稳定性的变异系数
	系统	*I_Avg*	系统中微服务指标*I*的均值,衡量系统的整体稳定性
内聚	微服务	*RC*	*RC*(relational cohesion)表示该包(服务)中内部关系数与类型数之间的比率,内部关系包括类之间的继承、方法的调用、对类属性的访问以及诸如创建类实例之类的显式引用,*RC*的数值越大,表示包(服务)的内聚性越高
	系统	*RC_Cv*	整体系统的内聚程度的变异系数
	系统	*RC_Avg*	系统中微服务*RC*指标的均值,衡量系统的整体内聚性

本文首先用表 5 中的指标来评估 Jpetstore 案例的第 1 层数据流图应用 DFD 方法后得到的拆分方案(见表 3)以及应用 DFD-A 方法(*n*=4)拆分得到的拆分方案,指标评估结果分别见表 6 和表 7.

Table 6 Evaluation of the decomposition results of the Level 1 dataflow diagram of the case Jpetstore using the DFD method (n=4)

表 6 Jpetstore 案例应用 DFD 方法对第 1 层的数据流图拆分方案的评估结果(n=4)

微服务	*Ce*	*Ca*	*I*	*RC*
ProductService	1	7	0.125	1.444 4
AccountService	3	3	0.4	2.2
OrderService	6	1	0.857 1	2.166 6
CartService	3	2	0.6	1.714
系统(*Cv*)	–	–	0.541 1	0.168 4
系统(*Avg*)	–	–	0.495 5	1.881 3

Table 7 Evaluation of the decomposition results of the case Jpetstore using the DFD-A method

表 7 Jpetstore 案例应用 DFD-A 方法拆分方案的评估结果

微服务	*Ce*	*Ca*	*I*	*RC*
ProductService	1	7	0.125	1.444 4
AccountService	2	3	0.5 (–)	2.0 (–)
OrderService	5	3	0.555 5 (+)	2.153 8 (–)
CartService	3	2	0.6	1.714 2
系统(*Cv*)	–	–	0.422 (+)	0.148 7 (+)
系统(*Avg*)	–	–	0.445 1 (+)	1.828 1 (–)

“+”:指标效果较优;“–”:指标效果较劣

通过对比表 6 和表 7 可以发现,本文提出的优化方法 DFD-A 以及原有的 DFD 方法都将 Jpetstore 案例划分为了 4 个微服务,分别是 Product 服务、Order 服务、Account 服务和 Cart 服务.其中:Product 和 Cart 这两个服务在划分结果上是一致的,且 *CE*,*CA*,*I*,*RC* 指标上表现相同;两个方法得到的结果差异主要是 Account 服务和 Cart 服务,从个体服务层面来看,DFD-A 方法仅在 Order 服务的稳定性上优于 DFD 方法.但是比较每个服务的在稳定性和内聚性上的数值可以发现:DFD-A 方法中表现稍劣于 DFD 方法的指标,即 Account 服务的 *I*(差值 0.1)、Account 服务的 *RC*(差值 0.2)和 Order 服务的 *RC*(差值 0.01).而 DFD-A 方法中优于 DFD 方法的指标是 Order 服务的 *I*,其与 DFD 中 Order 服务的不稳定性差值为 0.3.从拆分后整体系统的指标来看,DFD-A 方法则在 3 个指标上表现出优势:首先,各个服务之间的稳定性和内聚性差异变得更小;其次,在稳定性的均值上,表现得比 DFD 方法更低,表明 DFD-A 拆分后整体系统的稳定性更优.整体来讲,指标结果对比与分析说明了本文所提出的优化的数据流驱动的微服务化拆分方法 DFD-A 的有效性.

Table 8 Evaluation of the nine decomposition results of Jpetstore using the DFD-A method (system level)

表 8 DFD-A 方法 9 种拆分方案的系统层级指标计算结果

方法	方案	*I_Cv*	*RC_Cv*	*I_Avg*	*RC_Avg*	排名总和
DFD	方案 0-DFD	0.541 1 (9)	0.168 4 (5)	0.495 5 (7)	1.881 3 (2)	23
DFD-A (n=3)	方案 1(动)	0.340 7 (4)	0.098 (2)	0.381 4 (2)	1.815 8 (4)	12
	方案 2(静)	0.323 7 (3)	0.627 7 (8)	0.627 7 (10)	1.316 (8)	29
	方案 3(动+静)	0.229 2 (1)	0.076 1 (1)	0.477 7 (5)	1.919 9 (1)	8
DFD-A (n=4)	方案 4(动)	0.613 4 (10)	0.128 3 (2)	0.381 2 (1)	1.763 3 (5)	18
	方案 5(静)	0.318 8 (2)	0.660 6 (9)	0.495 8 (8)	1.147 4 (9)	28
	方案 6(动+静)	0.422 (5)	0.148 7 (4)	0.445 1 (3)	1.828 1 (3)	15
DFD-A (n=5)	方案 7(动)	0.452 2 (8)	0.302 4 (7)	0.49 (6)	1.495 2 (7)	28
	方案 8(静)	0.422 5 (6)	0.681 2 (10)	0.447 7 (4)	1.092 4 (10)	30
	方案 9(动+静)	0.433 8 (7)	0.285 9 (6)	0.505 (9)	1.559 5 (6)	28

2) 灵活性

为了展示 DFD-A 方法的灵活性,同时进一步说明 DFD-A 方法采用动静态分析相结合的方式来支持拆分的有效性,本文在预期微服务个数 n 为 3,4,5 的设置条件下,分别实现了 3 种类型的数据收集:动态数据、静态数据、动态+静态数据(本文方法),并使用所开发的工具原型完成分析与微服务化拆分,得到了 9 个不同条件下的候选

微服务方案(方案 1~方案 9).

本文继而使用有效性评估部分介绍的指标对方案 1~方案 9 进行评估,其系统层级的指标计算结果如表 8 所示.为方便对比,表中同时列出了 DFD 方法应用的拆分方案(方案 0)的评估结果.表格中括号内的数字代表该指标得分值在所有方案中的优劣排名,本文通过计算每个方案所涉及的 4 个指标得分值的排名总和来衡量该方案对于性能的权衡效果好坏,排名总和越小,代表该方案得到的候选微服务对于性能的权衡表现越好.由评估结果可知:相同的预期微服务个数 n 的情况下,依靠动态和静态两种数据源的分析相结合的方式所得到的拆分方案效果优于单一的数据源分析,比如 n=3 的方案中,方案 3 的排名总和小于方案 1 和方案 2;而单一数据源分析中,仅依赖动态数据支持拆分的效果要优于静态数据分析,同样是 n=3 的方案中,方案 1 的排名总和小于方案 2.这在一定程度上说明了本文所采用的动静态分析相结合的方式支持拆分的合理性和有效性.

灵活性方面,通过上述过程和结果对比可以发现:原有的 DFD 方法基于预先定义的规则对复杂的数据流图进行分析和拆分,得到的拆分方案是唯一且不支持调整的;本文优化后的 DFD-A 方法可以根据不同的算法参数(如微服务个数 n)设定给出不同的拆分方案,例如表 8 中在预期微服务个数设置为 3 和 4 的情况下,得到的拆分方案的性能权衡效果要好于 DFD 方法;而服务个数为 5 的拆分方案效果要劣于 DFD 方法,这种情况下,实践者可以根据对于服务可伸缩性和性能的需求来灵活选择合适粒度的方案.同时,本文所实现的原型平台实现了根据实际需求对拆分结果进行手动调整和调整后的再评估,原型平台中的基础服务通过插件的形式来实现,可以很方便地进行功能的扩展,比如其他类型数据信息的收集、拆分算法以及后期拆分结果的评估指标计算等等;此外,系统在完成微服务化拆分及实现后,依然还可以继续使用本文的原型平台收集微服务运行时的数据流信息分析,以进一步调整服务粒度.这一系列的特点都有助于大幅度提高基于 DFD-A 方法进行微服务化拆分的灵活性.

3) 效率

除了灵活性,效率是所提出的 DFD-A 方法的重要的优化目标,本文通过完成案例的拆分工作所需的时间成本来衡量两个方法的效率.优化前的 DFD 方法和优化后的 DFD-A 方法应用过程及对应的耗时结果对比如图 8 所示.

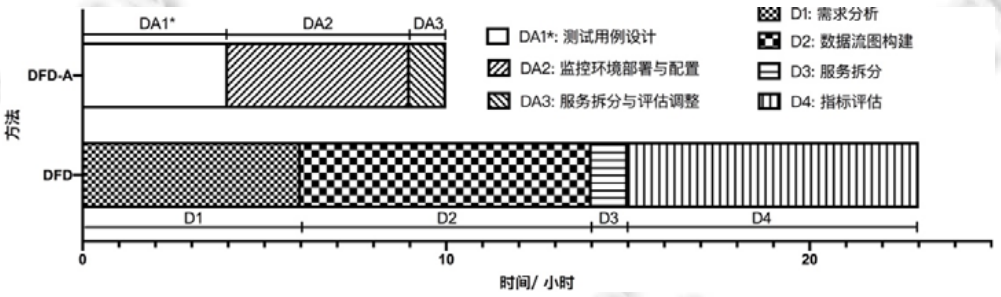

Fig.8 Process and the efficiency of applying the DFD and DFD-A method to Jpetstore

图 8 Jpetstore 案例中 DFD 和 DFD-A 方法应用过程及效率对比

DFD 方法应用主要包括 4 个阶段,分别是需求分析(D1)、数据流图构建(D2)、服务拆分(D3)以及拆分后的指标评估(D4).对于两千行左右代码量的小型案例 Jpetstore,参与评估的一名研究人员约耗费了 3 天(8h 工作制)来应用 DFD 方法完成该系统的拆分与评估,各阶段的耗时分别是 6h,8h,1h 和 8h.理解其业务需求并构建所需的数据流图.可以发现,除了其中自动化的服务拆分阶段,其他 3 个阶段由于手动操作的原因造成该方法的应用效率较低,尤其是基于需求手动构建数据流图(D2)以及对拆分后的结果使用第三方工具进行指标计算和评估(D4),耗费的时间成本较高(这里未包含数据流图和第三方评估工具的学习时长).DFD-A 的应用需要设计测试用例(DA1*)以支持数据收集、部署并配置 Pinpoint 监控环境(DA2)收集数据信息、通过本文开发的原型平台进行数据流信息自动化分析和服务拆分与评估调整(DA3)这 3 个阶段,耗时分别是 4h,5h 和 1h.该部分的对比结

果说明:与半自动化数据流驱动的微服务化拆分方法相比,本平台提供的拆分方法在保证拆分结果有效性的同时,极大地提升了拆分效率.另外需要强调的是:一般情况下,在企业中都会具备测试用例相关文档和测试基本环境,因此,DA1*阶段在理想情况下耗时可忽略估计.

4 讨 论

本文的贡献在于:在既有的数据流驱动的微服务化拆分方法基础上,针对缺陷问题进行了进一步的改进优化,在保证微服务化拆分有效性的前提下,提高了原方法的效率和灵活性.此外,本文基于优化的数据流驱动的微服务化拆分方法实现了可扩展的原型系统来高效支持微服务的拆分、评估及自定义调整工作.为验证优化后方法及原型系统的有效性和实用性,本文开展了案例研究.结果显示:本文基于优化后的方法实现的原型系统平台在功能上符合预期,且在提升微服务化拆分工作效率的基础上,取得了较为满意的拆分结果.

本文的不足之处在于:首先,本文提出的优化方法对于系统有一定的要求,因该方法的数据流信息收集与分析建立在源码、运行时数据收集的基础上,因此适用于遗留系统向微服务迁移的情况;其次,提出的基于数据流的拆分算法上依然存在一些需要根据主观经验设置的参数,比如数据库中心表的确定;对于候选微服务的评估,目前缺乏系统化的研究成果来支持本文研究工作,本文虽然使用了耦合、内聚指标等微服务设计中反映模块度和独立性的、可自动化计算的基本指标,其一定程度上可以反映并预测与可伸缩性权衡的质量属性的效果,如性能,但其仅仅是设计阶段衡量部分质量属性指标体系的子集,未来考虑将运行时的指标以及其他相关质量属性的指标(可维护性等)纳入到评估体系中;最后,尽管本文使用了相关工作中广泛应用且实现较为完整的单体系统 Jpetstore 来进行方法的评估研究,但当前阶段所选的评估案例并非是企业中的实际系统,且覆盖范围较小,可能会对方法的普适性造成威胁,未来将针对 DFD-A 方法开展大规模的企业级案例研究及评估.

5 总结与展望

本文提出了一种优化的数据流驱动的微服务化拆分方法,以应对现有的相关方法中存在的效率和灵活性方面的问题.本文所提出的方法通过 Pinpoint 追踪系统业务操作的执行轨迹和 ASM 解析系统的静态关联关系,来更准确高效地收集能客观反映内聚耦合程度的数据交互信息,在此基础上构建带权图,并实现两阶段的聚类算法(基于 MST 的业务操作聚类和基于 *K*-means 的数据库表聚类)来进行微服务化拆分.本文介绍了优化的 DFD-A 方法的整体流程和优化点,包括数据流信息收集与分析过程优化、拆分算法优化的具体实现;此外,本文基于 DFD-A 方法还设计并实现了一个原型平台.本文选择了被微服务化拆分工作广泛应用且实现较为完整的单体系统 Jpetstore 作为案例对 DFD-A 进行评估,展示了原有的 DFD 方法和优化的 DFD-A 方法在该案例中的应用过程和评估流程,3 个优化目标的实现效果来对两个方法的拆分方案进行对比和评估,结果说明了本文所提出的优化的数据流驱动的微服务化拆分方法 DFD-A,在保证拆分的候选微服务有效性的基础上更加高效、灵活.

未来我们将从如下几个方面展开深入研究.

1) 算法优化.针对依赖先验知识来进行算法参数设置的问题,本文未来考虑实现通过对系统数据关系进行排序分析给出建议的中心表的功能.
2) 候选微服务评估体系的构建.未来将全面探究微服务评估体系的构建,把其他质量属性的相关评估指标纳入到评估体系中,如性能、可靠性、可维护性等等,为候选微服务的评估提供一种结构化、系统化的解决方案.
3) 原型工具平台的完善.本文基于提出的优化的数据流驱动的微服务化拆分方法开发了原型系统平台以支持本文方法的微服务自动拆分,目前其核心功能已经完成.为了能够更好地帮助研究者们分析系统结构实现微服务化拆分,该平台仍然需要进一步的改进和扩展,比如以插件的方式将微服务化拆分各个阶段现有的其他解决方案融入平台,尤其是将兼顾新系统设计阶段制品和遗留系统运行阶段的多源数据的全面收集与分析,从而给拆分项目提供多角度的拆分方案,供实践者根据实际情况和需求

选择合适的方案执行.此外,还可以添加诸如生成方案对比报告等辅助功能来完善平台,让用户可以在本平台完成与微服务化拆分相关的大部分工作.完善后的工具平台将为实践者的微服务化拆分与评估工作提供全方位、一站式的支持.

4) 企业级案例研究与有效性评估验证.未来考虑将本文提出的方法对企业级的单体系统进行拆分,并围绕方法是否可以简化微服务识别、是否有效、实践者是否愿意使用该方法等方面展开问卷调查和方法的评估验证.

References:

[1] Fowler M, Lewis J. Microservices—A definition of this new architectural term. 2014. https://martinfowler.com/articles/microservices.html

[2] Zhang H, Li S, Zhang C, Jia Z, Zhong C. Microservice architecture in reality: An industrial inquiry. In: Proc. of the 2019 IEEE Int'l Conf. on Software Architecture (ICSA). 2019. 51–60.

[3] Li S, Zhang H, Jia Z, Li Z, Zhang C, Li J, Gao Q, Ge J, Shan Z. A dataflow-driven approach to identifying microservices from monolithic applications. Journal of Systems and Software, 2019,157:Article 110380.

[4] Jamshidi P, Pahl C, Mendonça NC, *et al*. Microservices: The journey so far and challenges ahead. IEEE Software, 2018,35(3): 24–35.

[5] Newman S, Wrote; Cui LQ, Zhang J, Translate. Building Microservices: Designing Fine-grained Systems. 2nd ed., Beijing: People's Posts and Telecommunications Press, 2016 (in Chinese).

[6] Evans E, Wrote; Zhao L, Sheng HY, Liu X, Translate. Domain-driven Design: Tracking Complexity in the Heart of Software. Beijing: People's Posts and Telecommunications Press, 2016 (in Chinese).

[7] Richardson C. Pattern: Microservice architecture. 2018. http://microservices.io/patterns/microservices.html

[8] Kecskemeti G, Marosi AC, Kertesz A. The ENTICE approach to decompose monolithic services into microservices. In: Proc. of the 2016 Int'l Conf. on High Performance Computing Simulation (HPCS). IEEE, 2016. 591–596.

[9] Hassan S, Bahsoon R. Microservices and their design trade-offs: A self-adaptive roadmap. In: Proc. of the 2016 IEEE Int'l Conf. on Services Computing (SCC). IEEE, 2016. 813–818.

[10] Gysel M, KöLbener L, Giersche W, *et al*. Service cutter: A systematic approach to service decomposition. In: Proc. of the European Conf. on Service-oriented and Cloud Computing. Springer-Verlag, 2016. 185–200.

[11] Ahmadvand M, Ibrahim A. Requirements reconciliation for scalable and secure microservice (de) composition. In: Proc. of the 2016 IEEE 24th Int'l Requirements Engineering Conf. Workshops (REW). IEEE, 2016. 68–73.

[12] Baresi L, Garriga M, Derenzis A. Microservices identification through interface analysis. In: Proc. of the European Conf. on Service-oriented and Cloud Computing. 2017. 19–33.

[13] Mazlami G, Cito J, Leitner P. Extraction of microservices from monolithic software architectures. In: Proc. of the 2017 IEEE Int'l Conf. on Web Services (ICWS). 2017. 524–531.

[14] Escobar D, Cárdenas D, Amarillo R, Castro E, Garcés K, Parra C, Casallas R. Towards the understanding and evolution of monolithic applications as microservices. In: Proc. of the 2016 XLII Latin American Computing Conf. (CLEI). 2016. 1–11.

[15] Levcovitz A, Terra R, Valente MT. Towards a technique for extracting microservices from monolithic enterprise systems. arXiv preprint arXiv:1605.03175, 2016.

[16] Hassan S, Ali N, Bahsoon R. Microservice ambients: An architectural meta-modelling approach for microservice granularity. In: Proc. of the 2017 IEEE Int'l Conf. on Software Architecture (ICSA). IEEE, 2017. 1–10.

[17] Klock S, Van Der Werf JME, Guelen JP, Jansen S. Workload-based clustering of coherent feature sets in microservice architectures. In: Proc. of the 2017 IEEE Int'l Conf. on Software Architecture (ICSA). 2017. 11–20.

[18] Chen MY, Kiciman E, Fratkin E, Fox A, Brewer E. Pinpoint: Problem determination in large, dynamic internet services. In: Proc. of the Int'l Conf. on Dependable Systems and Networks. IEEE, 2002. 595–604.

[19] Ren Z, Wang W, Wu G, Gao C, Chen W, Wei J, Huang T. Migrating Web applications from monolithic structure to microservices architecture. In: Proc. of the 10th Asia-Pacific Symp. on Internetware. 2018. 1–10.

[20] Saidani I, Ouni A, Mkaouer MW, Saied A. Towards automated microservices extraction using muti-objective evolutionary search. In: Proc. of the Int'l Conf. on Service-oriented Computing. 2019. 58−63.

[21] Jin W, Liu T, Cai Y, Kazman R, Mo R, Zheng Q. Service candidate identification from monolithic systems based on execution traces. IEEE Trans. on Software Engineering, 2019. Early Access. [doi: 10.1109/TSE.2019.2910531]

[22] Cojocaru M, Uta A, Oprescu A. MicroValid: A validation framework for automatically decomposed microservices. In: Proc. of the 2019 IEEE Int'l Conf. on Cloud Computing Technology and Science (CloudCom). 2019. 78−86.

[23] Redin RM, Oliveira MFS, Brisolara LB, Mattos JCB, Lamb LC, Wagner FR, Carro L. On the use of software quality metrics to improve physical properties of embedded systems. In: Proc. of the IFIP Working Conf. on Distributed and Parallel Embedded Systems. Springer-Verlag, 2008. 101−110.

附中文参考文献:

[5] Newman S,著;崔力强,张骏,译.微服务设计.第 2 版,北京:人民邮电出版社,2016.

[6] Evans E,著;赵俐,盛海艳,刘霞,译.领域驱动设计——软件核心复杂性应对之道.北京:人民邮电出版社,2016.

李杉杉(1990－),女,博士,助理研究员,主要研究领域为软件体系结构,微服务架构,DevOps,区块链,经验软件工程.

高邱雅(1993－),女,硕士,主要研究领域为软件体系结构,微服务架构.

荣国平(1977－),男,博士,副研究员,CCF 专业会员,主要研究领域为软件过程,实证软件工程.

邵栋(1976－),男,副教授,CCF 专业会员,主要研究领域为软件过程,高科技市场理论,敏捷软件开发,软件工程教育.