计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.210500078

一种基于微服务架构的服务划分方法

江 郑 王俊丽 曹芮浩 闫春钢
嵌入式系统与服务计算教育部重点实验室(同济大学) 上海 201804
(zhengjiang@tongji.edu.cn)

摘 要 对单体系统进行微服务划分能有效缓解单体架构中系统冗余、难以维护等问题,但是现有的微服务划分方法未能充分利用微服务架构的属性信息,导致服务划分结果的合理性不高。文中给出了一种基于微服务架构的服务划分方法。该方法通过系统服务与属性的关联信息来构建实体-属性关系图,然后结合微服务架构的特征信息与目标系统的需求信息制定服务划分规则,量化两类顶点之间的关联信息,生成实体-属性加权图,最后应用加权的GN算法自动地实现系统的微服务划分。实验结果表明,该方法在服务划分的时效性上有较大提升,并且生成的微服务划分方案在评估指标上的表现更好。

关键词:微服务架构;服务划分;问题建模;划分规则;GN算法
中图法分类号 TP311

Method of Service Decomposition Based on Microservice Architecture

JIANG Zheng, WANG Jun-li, CAO Rui-hao and YAN Chun-gang
Key Laboratory of Embedded System and Service Computing of Ministry of Education(Tongji University), Shanghai 201804, China

Abstract The microservice decomposition of the monolithic system can effectively alleviate the problems of system redundancy and difficulty in maintenance of the monolithic architecture. However, the existing microservice decomposition methods fail to make full use of the attribute information of the microservice architecture, which leads to the low rationality of service decomposition results. This paper proposes a service decomposition method based on microservice architecture. The method constructs an entity-attribute relationship graph through the association information of system services and attributes. Then the service decomposition rules are formulated by combining the feature information of the microservice architecture with the demand information of the target system, the association information between the two types of vertices is quantified, and a weighted entity-attribute graph is generated. Finally, the weighted GN algorithm is applied to realize the microservice decomposition of the system automatically. The experimental results show that the method greatly improves the timeliness of service decomposition, and the generated microservice decomposition scheme performs better in terms of various evaluation metrics.

Keywords Microservice architecture, Service decomposition, Problem modeling, Decomposition rules, GN algorithm

1 引言

单体架构[1]系统中存在启动时间长、持续部署成本高等问题,通过对单体系统进行微服务划分,能够充分发挥微服务[2-3]架构的高可维护性、高可复用性以及自动部署等优势。

现有的微服务划分方法通过分析系统并抽取特定数据信息的联系来抽象化系统,然后将满足要求的数据信息划分到同一服务,从而实现微服务的划分。Mazlami等[4]提出以类文件为顶点构建无向图,进行微服务的划分。但是,这类方法在问题建模中只考虑了同类型顶点之间的交互,缺乏对不同类型顶点之间的归属与合作关系的描述,造成系统信息的利用不足。另外,Levcovitz等[5]根据需求入口、业务需求以及数据库表之间的关系识别出微服务。但是,这类基于服务需求驱动的微服务划分方法,在微服务特性、实际系统内部的关联以及业务需求等综合层面的考虑不够充分,以致微服务划分方法表现出一定的不合理性。

本文针对建模中系统信息利用不足的问题,通过系统和需求分析来构建问题模型,建立包含两类顶点的实体-属性关系图,用于刻画系统的内在联系。针对微服务灵活性高、可维护性强等特性和实际系统需求信息考虑不足的问题,通过制定微服务划分规则,量化顶点之间的关联关系,对图中的边进行权重赋值。最后基于实体-属性加权图进行系统划分,生成相应的微服务划分方案。

本文的主要贡献如下:1)通过给出微服务划分的问题建模方法,充分考虑系统中两类顶点的归属和并列关系,提升系统分析的完整性和效率;2)综合考虑微服务核心原则与业务

到稿日期:2021-05-11 返修日期:2021-08-09
基金项目:国家重点研发计划(2018YFC0831403)
This work was supported by the National Key Research and Development Program of China(2018YFC0831403).
通信作者:闫春钢(ycg2@163.com)

功能逻辑制定划分规则，提高划分的合理性与准确度；3）通过实际系统验证划分方法的有效性，实验结果表明，这种方法可以加快微服务的划分速度，降低微服务划分的难度与成本。

本文第 2 节介绍了微服务划分方法的相关工作；第 3 节介绍了本文提出的微服务划分方法；第 4 节结合案例研究，阐述了实验细节和结果分析；最后总结全文并提出未来工作的方向。

2 相关工作

近年来，微服务划分方法与应用受到了越来越多研究人员和工业应用领域的关注。表 1 列出了具有代表性的划分方法。微服务的划分方法大致可分为基于机器学习的方法与基于图的方法[6]。

基于机器学习的方法中，Al-Debagy 等[7]提出使用 code2vec 从整体应用程序源代码创建代码嵌入，从而将单体系统转化为微服务系统。

基于图的方法需要构建不同数据之间的关系，根据依赖关系是单向或者是双向又可以分为基于有向图的方法与基于无向图的方法。Ren 等[8]结合静态和动态分析来构建有向图并识别出微服务。但有向图描述双向的数据依赖比较复杂，因此 Gysel 等[9-10]基于文献与行业经验设计生成无向加权图，并应用聚类算法识别微服务。但该方法无法从系统本身抽取必要的结构信息，在此基础上，Mazlami 等[4]提出了一种微服务提取模型，进行微服务划分。但该模型以类为最小单位进行拆分，没有对数据库进行彻底拆分，因此 Ding 等[11]获取系统运行时的调用方法和数据库操作信息，并基于数据表之间的关联构建系统无向图，从而完成相应代码模块的拆分。上述方法都是基于同类型的顶点来构建图，对顶点之间的差异性和层次性考虑得不够充分，以致系统信息的描述具有一定的不完整性。

在微服务划分依据设计方面，Newman[12]指出微服务划分的核心原则是高内聚与低耦合。在此基础上，Ahmadvand 等[13]指出微服务划分时应该考虑系统要求、安全性和可扩展性等。

同时，在指导微服务划分的策略上，Richardson[14]介绍了 4 种策略，分别按业务能力、领域驱动设计子域、动词或用例、名词或资源进行划分。上述研究给出了微服务划分依据的理论指导，在具体方法上可以依据系统特定的资源进行微服务划分，Baresi 等[15]提出基于 OpenAPI（Application Programming Interface）规范分析自动查找微服务的划分流程，将具有高相似度的操作与数据划分到相同的微服务。Abdullah 等[16]提出使用应用程序访问日志和无监督的机器学习方法，自动将应用程序分解为能映射到具有相似性能和资源需求的统一资源定位符分区的微服务。这些划分依据的设计一方面需要结合大量的专家经验，另一方面在微服务特性上考虑不充分，以致划分方法具有一定的局限性。

表 1 划分方法分析

Table 1 Analysis of decomposition method

方法	类型	适用范围	操作单元
基于微服务抽取模型划分方法[4]	基于版本控制等元数据	单体系统迁移	类
基于代码静态依赖图划分方法[5]	基于源代码分析	单体系统迁移	业务入口、功能、数据表
Service Cutter 划分方法[9-10]	基于抽取的耦合信息等元数据	单体系统迁移与微服务开发	数据、操作、软件制品
场景驱动、自底向上的划分方法[11]	基于工作流程等动态数据	单体系统迁移与微服务开发	场景、方法、数据表
可伸缩和安全协调的划分方法[13]	基于安全性和可伸缩性等元数据	单体系统迁移与微服务开发	系统需求
API 分析划分方法[15]	基于 API 语义相似度等元数据	单体系统迁移与微服务开发	符合 API 规范的单个操作
基于系统日志的机器学习划分方法[16]	基于统一资源定位符等资源数据	单体系统迁移与微服务开发	系统日志
本文方法	基于系统架构与业务功能等元数据	单体系统迁移与微服务开发	架构元素、系统需求

3 微服务划分方法

基于微服务架构的服务划分方法包括 3 个部分：问题建模、规则制定和系统划分。将单体系统进行分析并构建实体-属性关系图；结合微服务特征属性与系统架构信息制定微服务划分规则，量化图中顶点之间的关联程度，对图中的边赋予权重；分解生成的加权图，给出相应微服务划分方案。

3.1 问题建模

单体系统中包括众多实体，实体拥有若干属性，部分实体之间存在着关联。但是现有基于图的划分方法主要采用单类顶点的图进行系统分析，造成实体与属性之间关联信息的利用不够充分，导致微服务划分方法存在一定的不足，为此我们建立包含两类顶点的实体-属性关系图。

实体-属性关系图的构建包含如下步骤：

步骤 1 读取输入的单体系统架构数据，分析数据特征，构建实体集合与属性集合，将每个实体与属性作为图的顶点。

步骤 2 建立实体与属性的联系。针对每个实体，确定与它相关的所有属性，生成实体与对应属性之间的边。

步骤 3 建立实体与实体的联系。对于每个实体顶点，根据原始单体系统中的关联关系，生成实体与实体之间的边。

步骤 4 根据顶点间的邻接关系构建映射。针对图中的任意一个顶点，确定与该顶点相邻的实体顶点和属性顶点，将所有相邻顶点作为该顶点的映射集合。

步骤 5 根据边与顶点的关联关系构建映射。针对图中的任意一条边，确定该边关联的两个顶点，将这两个顶点作为该边的关联映射。

根据上述步骤可以构建实体-属性关系图，我们用一个五元组 $G=(A,E,R,M,I)$ 表示，其中，A 表示属性顶点集合 $\{a_1,a_2,a_3,\cdots\}$；E 表示实体顶点集合 $\{e_1,e_2,e_3,\cdots\}$；R 表示边的集合，包含两种类型的边，分别是实体与属性之间的边 $R(a_i,e_j)$ 和实体与实体之间的边 $R(e_i,e_j)$；M 为某个顶点的映射集合，表示与该顶点相邻的顶点集合，例如对属性顶点 a_i 有 $M(a_i)=\{e_j \mid$ 存在边 r，使得 e_j 与 a_i 通过 r 关联，$a_i\in A, e_j\in E, r\in R\}$；$I$ 为关联函数，I 将 R 中的每个元素映射到 $A\times E$

或者 $E\times E$。如果 $I(r)=(a_i,e_j)(a_i\in A,e_j\in E,r\in R)$，那么称边 r 关联顶点 a_i 与 e_j。

包含两类顶点的实体-属性关系图可以更好地描述不同实体和属性之间的归属和并列关系，而且能够更全面地表示结构特征与信息。另外，系统的用例可以描述系统的实际需求信息，每个用例同样包含两类顶点的信息，并表示为实体-属性关系图的一个子图。用例涉及的功能通过读取输入属性和写入输出属性完成。在动态的微服务场景中，当系统的实际需求发生变化时，用例集合会相应地动态更新，因此微服务划分将具有动态适应性。

3.2　规则制定

本节综合考虑了微服务特性和系统实际需求的信息，制定了内聚、耦合、开销以及相似 4 个划分规则，并给出了制定依据和计算公式。通过实体-属性关系图中权重的赋值，量化两类节点之间的关联关系。

3.2.1　内聚规则

每个独立的微服务要求服务内部功能紧密相关，并且性能更集中。微服务架构的这一特性要求划分出来的服务拥有更多密切相关的实体与用例。因此，我们设计了微服务划分的内聚规则。

内聚性体现在不同属性与实体以及实体与实体之间的关联程度上，我们从用例的角度衡量一个或多个用例之间出现属性的相关度，相比其他属性，如果当前属性与对应实体的依赖概率越大，则属性与实体间的内聚性就越大；如果两个实体的相关属性共现的概率越大，则实体之间的内聚性就越高。我们计算不同属性与实体以及实体与实体之间的结合程度，定义高、中、低 3 个内聚度，赋予不同内聚性的值。内聚关联计算的过程如算法 1 所示。

算法 1　内聚关联

输入：用例集合 UC，两个顶点（e_i 和 e_j 或者 a_i 和 e_j）

输出：内聚度关系值 Cohesion

/ * 计算两个实体的共现概率 * /

1. 初始化 relation，sum_i，sum_j
2. if 两个顶点是实体 e_i 和实体 e_j
3.　for each uc_k in UC
4.　　如果 e_i 的任意属性出现在 uc_k 中，sum_i++
5.　　如果 e_j 的任意属性出现在 uc_k 中，sum_j++
6.　　如果 e_i 的任意属性和 e_j 的任意属性共现在 uc_k 中，relation++
7.　end for
8.　如果 sum_i 和 sum_j 都不为 0
9.　　P(i,j)=max(relation/sum_i，relation/sum_j)
10. end if

/ * 计算属性与对应实体的依赖概率 * /

11. if 两个顶点是属性 a_i 和实体 e_j
12.　for each uc_k in UC
13.　　for each atrribute in uc_k
14.　　　如果 attribute 和 a_i 相等，sum_i++
15.　　　如果 attribute 属于 e_j，sum_j++
16.　　end for
17.　end for
18.　如果 sum_j 不为 0
19.　　P(i,j)=sum_i/sum_j
20. end if

/ * 计算关联程度并返回内聚度 * /

21. if(P(i,j)==1)
22.　return HighCohesion
23. else if(P(i,j) >=0.6)
24.　return MidCohesion
25. else return LowCohesion
26. end if

通过内聚关联计算可以得到两顶点间的内聚度关系值 Cohesion。

3.2.2　耦合规则

微服务架构要求每个用例中的属性具有较高的关联，即需要访问的属性尽可能包含在同一个实体或者服务中，降低服务之间交流信息的频率并减少内容。因此，本节通过计算顶点之间信息交换的程度，来设计微服务划分的耦合规则。

耦合规则的衡量包括属性共享度和实体共享度。属性共享度用于衡量单个属性被不同用例读取与写入的调用程度。属性共享度的计算规则如式(1)所示。为了筛选满足共享度要求的属性，设置一个阈值，只有当达到阈值时才满足共享条件。

$$S_{a_i}=\begin{cases}1, & 0\leqslant\lambda\leqslant\dfrac{2u}{3}\\ \lambda, & \dfrac{2u}{3}<\lambda\leqslant u\end{cases}\tag{1}$$

其中，λ 表示属性 a_i 在用例中出现的总次数，u 表示系统中实现的用例总数。采用类似的方法，可以计算实体共享度 S_{e_j}。

我们通过用例的优先级和属性与实体的调用程度来衡量相应顶点间的耦合性。在两个顶点共现的用例中，包含的属性和实体都参与了需求的实现，需要结合它们共同计算信息的交换程度。另外，当属性和实体达到共享度条件时，我们考虑划分的多个服务都需要调用当前的属性或者实体，这会导致系统的性能和灵活性变差，因此倾向于将当前的关联顶点作为一个服务供其他服务调用。我们用式(2)计算顶点间信息的交换程度。顶点间交流的信息越多，则两顶点应尽量包含在同一服务中，从而降低耦合度。

$$Coupling=\begin{cases}\sum\limits_{t\in V}P_t\left(\sum\limits_{x=1}^{m}\dfrac{1}{S_{a_i}}+\sum\limits_{y=1}^{n}\dfrac{1}{S_{e_j}}\right), & I(r)=(a_i,e_j)\\ \sum\limits_{t\in V}P_t\left(\sum\limits_{x=1}^{n}\left(\dfrac{1}{S_{e_i}}+\dfrac{1}{S_{e_j}}\right)\right), & I(r)=(e_i,e_j)\end{cases}\tag{2}$$

其中，V 表示 a_i 与 e_j 或者 e_i 与 e_j 共现的用例集合；P_t 表示用例 t 在所有用例中的优先级高低，$P_t\in[1,u]$；m 表示用例 t 包含的属性总数；n 表示用例 t 包含的属性对应的实体总数。

3.2.3　开销规则

微服务划分需要衡量当前两个顶点与其他顶点的联系程度，通过拆分联系程度较低的连接来降低开销。因此，本节通过考虑顶点间的关联，来设计微服务划分的开销规则。

微服务拆分开销的衡量可以将相邻的两个顶点看作一个整体，分析它们在拆分过程中与外部顶点的关联，计算与它们相邻的顶点占据总顶点数的比例。整体的关联程度越大，断

开连接顶点的边的开销就越大，此时需要尽量保留两顶点之间的边，从而达到降低拆分开销的目的。拆分开销 *Overhead* 如式(3)所示：

$$Overhead=\begin{cases}\dfrac{|M(a_i)|+|M(e_j)|}{|A|+|E|}, & I(r)=(a_i,e_j)\\ \dfrac{|M(e_i)|+|M(e_j)|}{|A|+|E|}, & I(r)=(e_i,e_j)\end{cases} \tag{3}$$

其中，$|M(a_j)|$表示与a_j相邻的顶点个数，$|A|$表示总属性数，$|E|$表示总实体数。

3.2.4 相似规则

微服务划分需要充分利用系统自身信息以及挖掘信息之间的关联，系统中的关键词可以描述整体特征。实体与属性对应的关键词相似度越高，两者之间的关联越紧密。因此，本节考虑不同顶点之间的关键词向量相似程度，设计微服务划分的相似规则。

系统中每个属性和实体都有特殊的含义，并且假设命名规则一般具有良好规范。通过将所有属性和实体的名称预处理后，得到一个代表整个系统的关键词库 $K=\{k_1,k_2,\cdots,k_n\}$，k_m代表第 m 个关键词。对于每个顶点，在当前顶点出现的用例中进行遍历，统计每个关键词k_m出现的次数，记为$n(k_m)$。由此得到描述当前顶点的相关特征的关键词向量$\boldsymbol{X}_i=\{n(k_1),n(k_2),\cdots,n(k_m)\}$。式(4)通过计算两个关键词向量$\boldsymbol{X}_i$和$\boldsymbol{X}_j$的余弦值，来衡量两个顶点间的相似关系。

$$Similarity=\cos(\boldsymbol{X}_i,\boldsymbol{X}_j) \tag{4}$$

3.2.5 权重计算

综合制定的划分规则，在 3.1 节中构建的实体-属性关系图的基础上，对图中的边赋予权重，给出系统的实体-属性加权图。一个实体-属性加权图用一个二元组 $WG=(G,W)$表示，其中，G 表示实体-属性关系图；W 表示图中边权重的集合，W_{ij}表示图中两个顶点间的边权重。

在进行权重计算时，由于多个划分规则拥有不同的取值范围，我们对所有规则的值都进行了归一化处理。利用当前边的规则取值除以所有边上对应规则的最大值，将结果作为最终规则的值。而且，根据特定场景的需求，可以通过设置内聚、耦合等规则的优先级P_i，来生成不同的微服务划分方案，以适应复杂和动态的应用场景。顶点间的边权重W_{ij}如式(5)所示：

$$W_{ij}=P_1*\frac{Cohesion}{Cohesion_{\max}}+P_2*\frac{Coupling}{Coupling_{\max}}+P_3*\frac{Overhead}{Overhead_{\max}}+P_4*\frac{Similarity}{Similarity_{\max}} \tag{5}$$

3.3 系统划分

在得到系统的实体-属性加权图后，本节将系统服务划分转化为图的划分问题。图有许多聚类算法[17-19]，本文采用了一种基于边介数的无向图聚类算法——Girvan-Newman 算法，简称 GN 算法[20-21]。GN 算法不断进行计算边介数与删除边介数高的边这两个操作，直到图中所有的边均被移除。

但是，传统的 GN 算法一般适用于无权图，对于加权图，在 GN 算法中使用边权比[22]来替代边介数，边权比是边介数与边的权重的比值。本文将 3.1 节与 3.2 节得到的加权图邻接矩阵作为输入，矩阵中的元素为边的权重，于是两点 i,j 间使用边介数B_{ij}除以对应边的权重W_{ij}来得到边权比EW_{ij}，即：

$$EW_{ij}=\frac{B_{ij}}{W_{ij}} \tag{6}$$

将边权比大小作为删除边的依据对图进行划分，同时 GN 算法中引入模块度 Q[23]来衡量图的聚类特性，从而找出微服务划分中最合适的方案。模块度 Q 的定义如式(7)所示：

$$Q=\frac{1}{2m}*\sum_{ij}\left[W_{ij}-\frac{k_i*k_j}{2m}\right]\delta(C_i,C_j) \tag{7}$$

其中，m 为图中所有权重之和；W_{ij}为邻接矩阵的元素，若 i 和 j 相连，则W_{ij}为权重，若 i 和 j 不相连，则W_{ij}为 0；k_i是顶点 i 的度，通过对邻接矩阵的第 i 行求和得到；δ 为隶属函数；C_i和 C_j分别表示 i 和 j 顶点隶属的两个子图，当属于同一子图时函数值为 1，否则为 0。

划分算法的默认输出是一系列子图，每个包含实体的子图都可以对应为一个服务，从而将单体系统划分为微服务候选对象，生成相应的微服务划分方案。

4 实验结果与分析

本节将我们提出的微服务划分方法应用于划分货物追踪系统(Cargo Tracking System，CTS)，它是一个被广泛使用的单体应用系统。通过分析，实验结果验证了本文方法的有效性。

4.1 CTS 案例分析

CTS 是 Evans 等[24]用于说明领域驱动设计(Domain-Driven Design，DDD)的典型案例。CTS 主要包括 9 个实体对象，表 2 列出了这些实体对象及其对应属性。另外，该系统还给出了 9 个用例：ViewTracking，ViewCargos，BookCargo，ChangeCargoDestination，RouteCargo，CreateLocation，CreateVoyage，AddCarrierMovement，HandleCargoEvent。

表 2 CTS 的实体对象及属性

Table 2 Entity objects and attributes of CTS

Entity	Attributes
Cargo	trackingId
RouteSpecification	origin destination arrivalDeadline
Itinerary	itineraryNumber
Leg	loadLocation unloadLocation loadTime unloadTime
Location	unLocode name
HandlingEvent	type completionTime registrationTime location
Delivery	transportStatus misdirected estimatedArrivalTime isUnloadedAtDestination routingStatus
Voyage	voyageNumber
CarrierMovement	departureLocation arrivalLocation departureTime arrivalTime

4.2　实验结果

4.2.1　划分结果分析

在 CTS 上利用微服务划分方法,输出每次划分的模块度与最终微服务的划分结果,模块度的变化情况如图 1 所示。随着迭代次数的增加,模块度不断增大,迭代 20 次左右时,模块度达到 0.7 左右,此时图的聚类特性较强。表 3 列出了依据实体拆分对应的微服务划分方案。在之后的迭代中,模块度整体呈减小的趋势,图的聚合程度下降。

图 1　模块度 Q 的变化

Fig. 1　Change of modularity Q

表 3　CTS 的划分方案

Table 3　Decomposition scheme of CTS

MS	MS name	Entities	Use cases
MS1	CargoService	Cargo RouteSpecification Delivery HandlingEvent	ViewTracking ViewCargo BookCargo HandleCargoEvent ChangeCargoDestination
MS2	LocaionService	Location	CreateLocation
MS3	LegService	Leg Itinerary	RouteCargo
MS4	VoyageService	Voyage CarrierMovement	CreateVoyage AddCarrierMovement

图 2 给出了 CTS 的模型和利用本文方法得出的划分方案。我们的划分方法综合考虑实体和属性两类顶点,但是由于某些属性与对应实体的联系程度很低,会出现属性被单独划分出来的情况,因此我们把它们封装起来以供其他微服务调用。

根据划分结果,CTS 被划分为 4 个微服务模块,包括 CargoService,LocationService,LegService 和 VoyageService。从业务功能进行分析,4 个微服务模块都能够独立完成相应功能,CargoService 管理所有货物以及跟踪货物的具体状态,LocationService 管理系统的地点数据,LegService 管理货物航段信息,VoyageService 管理所有航线的运输信息。对划分合理性进行分析,CargoService 围绕 Cargo 这个主要实体,实现单一业务功能;由于 Leg 与 Itinerary 之间具有相当高的联系程度,因此被划分在 LegService 模块中;VoyageService 中的 Voyage 和 CarrierMovement 对外部依赖很小。综上,这种划分方案是合理的。

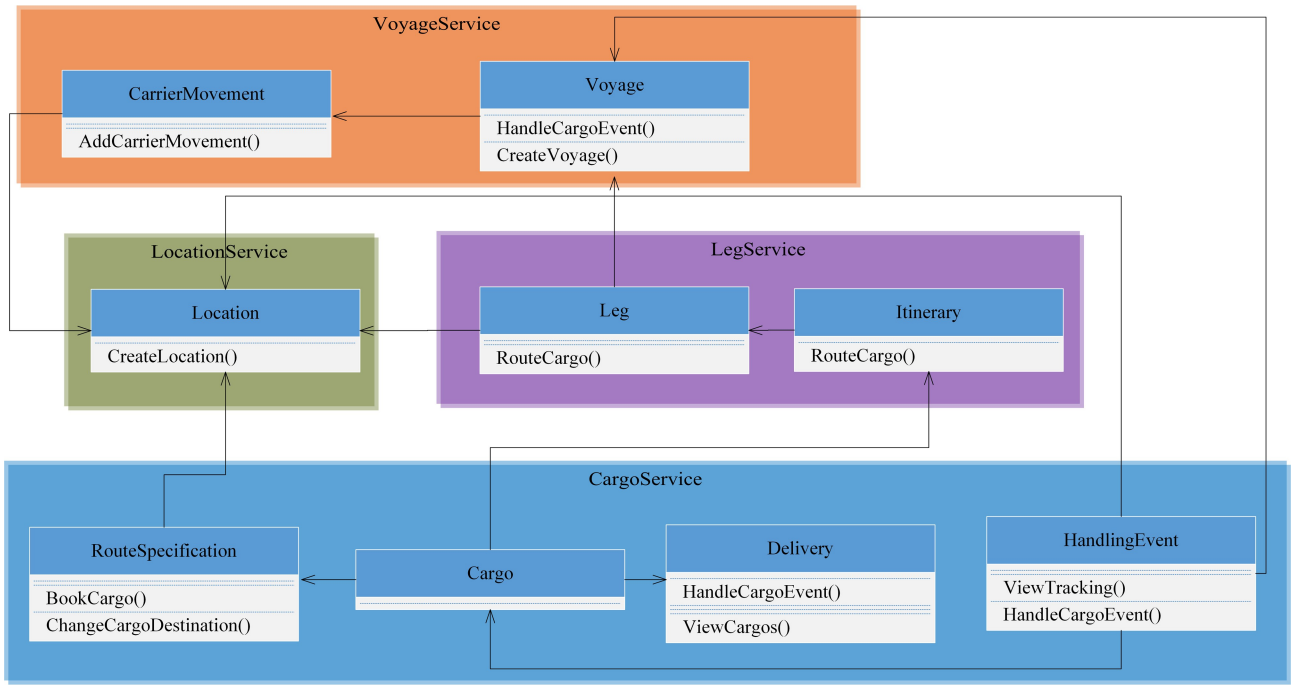

图 2　CTS 的划分结果

Fig. 2　Decomposition result of CTS

4.2.2　划分结果量化评价

为了进一步验证本文微服务划分方法的有效性,我们采用 Fritzsch[25] 推荐的传入耦合(Afferent Coupling,Ca)、传出耦合(Efferent Coupling,Ce)、不稳定性(Instability,I)和关系

内聚(Relational Cohesion, RC) 4 个指标来评估微服务的划分结果。这 4 个指标在表 4 中进行了详细解释,并且由 Sonargraph[1]工具生成。

表 4 评估微服务划分的指标

Table 4 Metrics for the evaluation of microservice decomposition

指标	解释
传入耦合(*Ca*)	*Ca* 定义为其他服务中依赖于本服务的类的数量,因此它表示服务的责任
传出耦合(*Ce*)	*Ce* 度量了本服务中的类所依赖的其他服务中的类数,从而表明了它对其他服务的依赖性
不稳定性(*I*)	*I* 通过计算 *Ce* 和 *Ce*+*Ca* 的比率来度量服务的弹性。*I*=0 表示一个完全稳定的服务,而 *I*=1 表示一个完全不稳定的服务
关系内聚(*RC*)	*RC* 被定义为服务中内部关系的数量与类型的数量之间的比率。*RC* 值越大表示服务的内聚性越高

本文选择划分 CTS 系统的 3 种代表性的划分方法进行对比实验,分别为 Service Cutter 方法[9]、API 分析方法[15]、DFD 驱动方法[26]。将本文的微服务划分方法以及对比方法应用到 CTS 系统,4 种指标得出的评估结果如表 5—表 8 所列。

表 5 基于本文方法的微服务指标

Table 5 Metrics of microservices based on this method

Metrics	Cargo	Locaion	Leg	Voyage	Avg
Ca	15	15	9	14	13.25
Ce	4	1	4	3	3
I	0.21	0.06	0.31	0.18	0.19
RC	9.5	21.5	11.5	21.33	15.96

表 6 基于 Service Cutter 方法的微服务指标

Table 6 Metrics of microservices based on Service Cutter

Metrics	Location	Tracking	Voyage & Planning	Avg
Ca	14	11	15	13.33
Ce	1	3	4	2.67
I	0.07	0.21	0.21	0.16
RC	21.5	4.33	16.67	14.17

表 7 基于 API 分析方法的微服务指标

Table 7 Metrics of microservices based on API analysis

Metrics	Planning	Product	Tracking	Trip	Avg
Ca	16	13	15	8	13
Ce	2	4	5	2	3.25
I	0.11	0.24	0.25	0.2	0.20
RC	20.33	1.83	14.86	0	9.25

表 8 基于 DFD 驱动方法的微服务指标

Table 8 Metrics of microservices based on DFD driven approach

Metrics	Cargo	Planning	Location	Tracking	Avg
Ca	13	10	15	16	13.5
Ce	4	3	1	5	3.25
I	0.2	0.2	0.07	0.25	0.18
RC	1.82	11.47	21.5	14.13	12.23

从实验结果可以看出,4 种方法给出的划分方案具有相似的平均 *Ca*。Service Cutter 方法在平均 *Ce* 的指标上比其他 3 种方法略低,这主要是因为它得到的是 3 个微服务模块,每个微服务比其他方法得到的微服务更大,而相对小的微服务在维护性和实现单一功能方面有更好的表现。但是,该方法在构建模型时需要依赖用户提供详尽的系统规范,此外在耦合标准的定义上对实际模型元素的考量不充分,具有一定程度的主观性。*I* 是使用 *Ce* 与 *Ce*+*Ca* 的比值进行计算的,4 种方法总体上的差别不大。在平均 *RC* 指标上,本文方法的表现优于其他方法,这是因为本文在问题建模中考虑了更多的顶点关系,并且综合了微服务特性与系统信息设计划分的规则,而 API 分析方法的值较低是因为该方法必须依赖良好定义与描述的接口,且只考虑了 API 的相似度,并不能充分反映服务内部的关系。

4.2.3 性能评估

为了分析本文方法在更复杂系统中的划分性能,我们复制 CTS 的所有信息并将其放大数倍来创建一个更大、更丰富的模型,测试问题建模、规则计算、系统划分的运行时间。测试指在拥有 2.6GHz CPU 和 16GB RAM 的 Windows10 计算机上单独运行本文方法,性能测试结果如表 9 所列。

表 9 性能测试结果

Table 9 Result of performance test

Number of nodes	Number of edges	Average time/ms
34	41	344
68	82	605
102	123	1066
170	205	2731
306	369	14226
408	492	38084

对于拥有 408 个顶点和 492 条边的系统,微服务划分方法可以在 40s 内给出结果,对于一个中小型项目的微服务划分动辄几天或者几周的周期来说,这个性能开销可以在很大程度上降低开发的成本,我们认为这是合理的,可以为开发人员的微服务划分提供参考。

结束语 本文提出了一种基于微服务架构的服务划分方法,以解决在单体系统上的微服务划分问题。该方法首先分析输入的单体系统架构数据,根据得到的实体、属性以及关联信息,构建实体-属性关系图;其次制定内聚、耦合、开销、相似 4 个划分规则,通过规则的量化对图中边的权重进行赋值,生成实体-属性加权图;最后基于聚类的 GN 算法对加权图进行划分,得到微服务的划分方案。实验结果表明,本文方法得到的微服务划分结果合理可靠,并且能够降低微服务划分的成本。

针对微服务划分的动态性,一方面在动态应用场景下,根据不同需求动态调整用例集合,生成的微服务结构具有动态适应性;另一方面,在特定场景下,可以通过调整不同规则的优先级,来动态划分微服务结构。未来将进一步优化划分方法,增强适应动态微服务应用场景的能力以及提升微服务结构动态调整的效果。

参 考 文 献

[1] RICHARDSON C. Monolithic Architecture[EB/OL]. [2021-04-02]. https://microservices.io/patterns/monolithic.html.

[2] LU Y F, CAO R H, WANG J L, et al. Method of Encapsulating Procuratorate Affair Services Based on Microservices[J]. Com-

[1] https://www.hello2morrow.com/products/sonargraph/architect9

puter Science,2021,48(2):33-40.
[3] WU W J,YU X,PU Y J,et al. Development of Complex Service Software in Microservice Era[J]. Computer Science, 2020, 47(12):11-17.
[4] MAZLAMI G,CITO J,LEITNER P. Extraction of microservices from monolithic software architectures[C]// 2017 IEEE International Conference on Web Services (ICWS). IEEE,2017: 524-531.
[5] LEVCOVITZ A,TERRA R,VALENTE M T. Towards a technique for extracting microservices from monolithic enterprise systems[J]. arXiv:1605.03175,2016.
[6] WU H Y,DENG W J. Research Progress on the Development of Microservices[J]. Journal of Computer Research and Development,2020,57(3):525-541.
[7] AL-DEBAGY O,MARTINEK P. A Microservice Decomposition Method Through Using Distributed Representation of Source Code[J]. Scalable Computing: Practice and Experience, 2021, 22(1):39-52.
[8] REN Z, WANG W, WU G, et al. Migrating web applications from monolithic structure to microservices architecture[C]// Proceedings of the Tenth Asia-Pacific Symposium on Internetware. 2018:1-10.
[9] GYSEL M,KÖLBENER L,GIERSCHE W,et al. Service cutter:A systematic approach to service decomposition[C]// European Conference on Service-Oriented and Cloud Computing. Cham:Springer,2016:185-200.
[10] GYSEL M,KÖLBENER L. Service Cutter-A Structured Way to Service Decomposition[D]. Rapperswil-Jona: HSR Hochschule für Technik Rapperswil,2015.
[11] DING D,PENG X,GUO X F,et al. Scenario-Driven and Bottom-Up Microservice Decomposition for Monolithic Systems [J]. Journal of Software,2020,31(11):3461-3480.
[12] NEWMAN S. Building microservices:designing fine-grained systems[M]. O'Reilly Media,2015:1-47.
[13] AHMADVAND M,IBRAHIM A. Requirements reconciliation for scalable and secure microservice (de) composition[C]// 2016 IEEE 24th International Requirements Engineering Conference Workshops (REW). IEEE,2016:68-73.
[14] RICHARDSON C. Pattern:Microservice architecture[EB/OL]. [2021-04-02]. http://microservices.io/patterns/microservices.html.
[15] BARESI L,GARRIGA M,DE RENZIS A. Microservices identification through interface analysis[C]// European Conference on Service-oriented and Cloud Computing. Cham: Springer, 2017: 19-33.
[16] 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.
[17] VON LUXBURG U. A tutorial on spectral clustering[J]. Statistics and Computing,2007,17(4):395-416.
[18] PONS P,LATAPY M. Computing communities in large networks using random walks[C]// International Symposium on Computer and Information Sciences. Berlin:Springer,2005:284-293.
[19] TRAAG V A. Faster unfolding of communities:Speeding up the Louvain algorithm[J]. Physical Review E,2015,92(3):032801.
[20] GIRVAN M,NEWMAN M E J. Community structure in social and biological networks[J]. Proceedings of the National Academy of Sciences,2002,99(12):7821-7826.
[21] NEWMAN M E J. Fast algorithm for detecting community structure in networks[J]. Physical review E, 2004, 69(6): 066133.
[22] YANG N. Research and application on service partition and selection strategy in microservice platform [D]. Beijing: Beijing University of Posts and Telecommunications,2019.
[23] NEWMAN M E J,GIRVAN M. Finding and evaluating community structure in networks[J]. Physical review E,2004,69(2): 026113.
[24] EVANS E J,EVANS E. Domain-driven design:tackling complexity in the heart of software[M]. Addison-Wesley Professional,2004:15-31.
[25] FRITZSCH J. From Monolithic Applications to Microservices: Guidance on Refactoring Techniques and Result Evaluation[D]. Reutlingen:Reutlingen University,2018.
[26] LI S,ZHANG H,JIA Z,et al. A dataflow-driven approach to identifying microservices from monolithic applications[J]. Journal of Systems and Software,2019,157:110380.

JIANG Zheng, born in 1996, postgraduate. His main research interests include service computing, service decomposition and microservice.

YAN Chun-gang, born in 1963, Ph. D, professor, Ph. D supervisor. Her main research interests include computer collaboration and service computing, trusted computing and Petri net modeling and analysis.