毕业论文（设计）文献综述

微服务划分工具的设计与实现

摘　　要：微服务架构的兴起是云计算时代的一项重要事件。在将传统架构下的代码迁移至微服务架构下时，不可避免的一个大问题就是微服务划分。当下企业主要使用的划分方案为人工划分，有较强的个体性，难以将经验直接复制使用。因此对自动化的普适划分方案有着强大的需求。现有的研究大都将该问题视作精细化的传统模块划分，而没有考虑到微服务架构带来的物理优势。同时现有方案的实验步骤也都过于理论化，没有进行实际的部署与测试。本文利用代码实际运行情况记录，自动分析系统主要特征。再抽象为图结构，设计独特的算法以获得优质的微服务划分方案。划分算法将考虑到云环境下微服务架构的逻辑层面特征和物理层面特征，尽可能使划分后系统具有高内聚低外耦合的特点，同时挖掘对高性能、高可靠有需求的模块，便于实际运行过程中的独立弹性伸缩与安全可靠易维护。最后使用将结果作为微服务部署，用实际情况来验证了优秀的效果。可以说是做到了领域内的全新突破。

关键词： 微服务；　运行路径；　基于图的划分；　物理特征识别

Design and Implementation of Microservices Extraction Tool

Abstract: The rise of microservice architecture is an important event in the era of cloud computing. When migrating the code under the traditional architecture to the microservice architecture, one of the inevitable big problems is the division of microservices. At present, the division scheme mainly used by enterprises is manual division, which has strong individuality and is difficult to directly copy and use. Therefore, there is a strong demand for automated universal division schemes. Existing studies mostly regard this problem as a refined traditional module division, without considering the physical advantages brought by the microservice architecture. At the same time, the experimental procedures of the existing schemes are too theoretical, without actual deployment and testing. This article uses the actual operation records to automatically analyze the main features of the system. It is then abstracted into a graph structure, and a unique algorithm is designed to obtain a high-quality microservice partitioning scheme. The partition algorithm will take into account the logical and physical characteristics of the microservice architecture in the cloud environment, and try to make the partitioned system have the characteristics of high cohesion and low external coupling. At the same time, it will mine the modules that require high performance and high reliability to facilitate Independent elastic scaling and safety, reliability and easy maintenance during actual operation. Finally, the result was deployed as a microservice, and the actual situation was used to verify the excellent effect. It can be said that a new breakthrough in the field has been achieved.

Keywords: Microservices; Execution Trace; Graph-based Extraction; Physical-feature Recognition

文献综述正文

一、 微服务架构

科技在不断进步，5G 和新材料的发展让我们看到一幅构建在“云”上的未来图景——将占用空间的负责计算的“重”物理机器放在服务端，把只有操作功能的“轻”机器留给客户端。这样的云环境使得用户能够拥有更多的更灵活的选择，同时也能通过购买服务、资源而获得更强大的计算能力。随着“云上计算”概念的普及，如何在云上开发这个问题也走入了研究者们的视野[1]。微服务架构概念正是在解决这个问题的过程中变得非常有热度，因为它不再按传统将系统视为一体式的代码，而是视为一个个独立的微服务[2]。每个微服务都足够独立，能够部署在完全独立的机器上，使用不同的技术栈，无论是编程语言还是数据库，都能够根据每种服务的需求而独立更改。同时各个微服务间使用轻量级机制进行通信，例如使用 HTTP（超文本传输协议）资源 API（应用编程接口）。这使得各服务本身的独立调度、独立伸缩变得更加容易，各服务间的界限更为明确。其思路完全适应于完全分布式的环境，恰好能充分利用云环境的优势，因此被认为是一种最适应云生态环境的新架构。

二、 微服务划分工具意义

微服务架构下代码结构特殊，研究人员认为从传统架构下的现有系统进行迁移会比直接构建微服务系统更加有效[3]。为充分发挥微服务架构的优势，迁移过程中就需要对代码进行合理划分。也正是因此，对一体式系统进行划分构成微服务已经被认为是一个重要需求[4]。在工业实践上，常常被选择的方案是人工划分，这对划分者的专业性要求极高，同时划分的结果、步骤也没有的普适性，据调查显示目前许多已划分的微服务都是功能小众的、适用范围狭窄的[5]，这导致了划分方案难以普及，划分过程也难以为他人所借鉴。显然，只有发现了普适化的划分方案，微服务架构才能在每个人手上发挥其作用，才能真正使云环境的优势得到展现。而普适化且自动化的工具更是直戳用户的需求，大大降低了使用微服务架构的门槛。因此，可以说普适化自动化划分工具是微服务架构推广的基石，其能够在整个微服务架构体系下发挥出重要的作用。

三、 代码划分问题

在计算机历史上发挥重要作用的单体式架构仍然是一种非常有效的架构，但其一直以来都有一个巨大的问题：随着代码量的增加与功能的复杂化，单体式架

构下的代码常常会变得非常臃肿。为了解决其结构混乱、复杂性高的问题，代码划分早就为人所讨论。所有划分的核心思想都是分而治之，即目的为对一个完整而庞大的系统进行拆分，使得拆分后的每个部分逻辑简单而有效，以此来实现逻辑层面的简化。

众多划分问题中，“模块划分”问题已经有长久的历史。模块划分常常出现在开发者口中，通常来说，模块划分最重要的意义在于方便开发者梳理逻辑、简化开发过程，从而对于开发者能够节约时间、提高效率，对于系统能够使其逻辑清晰，便于迭代升级或安全维护。

模块划分的主要切入点有两个：复用和解藕。研究者与开发者发现，系统的不同功能间常常会出现逻辑重复的情况，当大段代码重复出现时，系统整体自然会显得非常臃肿、混乱与复杂。这时候一个简单的处理方法就是按功能提取公共部分作为一个新模块，等价于将大段代码块更换为简短而明了的接口，同时在未来实现其他功能时，也可以通过对已有模块的组合来简单地实现，这也就是所谓的复用。而所谓的解藕则是将各功能的逻辑梳理分组，使得不同的组尽可能独立，尽可能没有逻辑上的功能依赖关系。这样就能分离各部分的责任，划清边界，便于组织管理以及维护和独立升级。

微服务划分也包含逻辑层面划分，也需要考虑如何划分能够便于使逻辑清晰、使管理方便，因此过往的划分思路对微服务划分也完全有借鉴意义。同时，微服务划分的主要思路是从业务层面出发，按业务的功能流程进行划分。另外，分布式环境带来的全新模式也给微服务划分带来了新的思路。微服务架构下服务的独立部署导致各服务间的通信代价增大，因此解藕对于微服务而言就更加有意义。同时微服务架构对提升了各服务部署、调度、运维的独立性，微服务划分时也完全可以充分利用这些特点，扬长避短。

由于分布式基础设施的特点，微服务划分也比传统划分多了一个问题：粒度，也就是在微服务大小和数量上的权衡。在传统架构下，划分的粒度对于系统的运行并无影响，因此粒度问题不需要被考虑，即便在系统中粒度有大有小也毫无问题。但对于微服务架构而言，微服务的架构对于系统的表现有着重要的影响。若粒度太大，则会使系统与传统架构下的一体式代码相似，无法发挥出分布式基础设施的优势，也就无从说起微服务架构比传统一体式架构有什么优点。若粒度太小，则会由于微服务间的壁垒而增加数据传输代价，同时在微服务大量重用的情况下会使传输体系对系统的影响大大增加。但在一些研究者看来，微服务架构在实践中的重用远远小于预期，因此微服务的粒度应当足够小，从而能够在修改时足够迅速、对其他部分影响足够小，发挥其在快速开发，快速迭代和快速运维上的优势[6]。

微服务划分的思路还可以考虑到未来云环境的发展趋势。从 IaaS（基础架构即服务）发展到 PaaS（平台即服务）再发展到 SaaS（软件及服务），云环境向用户所提供的服务越来越高层。可以设想，在未来假若能够直接向用户提供带有接口的微服务，交由用户进行微服务的组合，将使编程成为更亲民的事情。用户不需要了解每个服务中的功能是如何实现的，只需要选择自己所需的功能，利用各个微服务向外暴露的接口进行数据传递，即可完成程序的实现。在这种情况下，用户即便不懂得编程知识，也能够通过组合微服务，按照拼图的思路将一个个简单模块构建成理想的系统。在未来，假若能够有这样的技术支持，微服务的划分则需要考虑到每个微服务的功能完整性和足够细的粒度，以此降低用户在组合时能够做出选择的时间成本。

总结而言，微服务划分与模块划分的区别有二：从逻辑层面来说，模块划分是从业务下具体功能的实现角度出发，目的是便于代码功能的实现，相较而言视角更加底层；而微服务划分是从业务的设计角度出发，目的是便于业务逻辑的切割，相较而言视角更加高层。从物理层面而言，微服务划分需要适应并利用分布式环境，而这对于模块划分而言并不需要考虑。总的来说，二者不是对立的，而是各司其职、相辅相成的。

因此，设计微服务划分工具时不能简单地照搬传统模块划分策略，而应当在其基础上有选择地吸收并发展。

四、 目前研究动态

目前，对本课题的研究中，划分的方法通常分为三步：确定最小划分单位；将文件结构抽象并构建成图结构；对图结构进行划分或聚类。

划分的最小单位有两个切入点：源代码信息，API 信息。源代码信息即传统架构下的代码最小单位，例如以文件、类、包、功能块等作为最小单位。在此基础上有的研究者还仿照业界划分模式，通过增加适当的人工介入，提前进行细粒度的组合，也就是对源代码进行域分析（Domain Analysis）[7]。通过这样的手段，可以只将计算各部分间关联度的功能交由自动化工具处理，能够克服自动分析不够精确的问题，不失为一种折中的第三选择。而 API 信息作为最小单位则是出于对“迪米特原则”的考虑，不同服务之间应当只通过接口进行交流，内部的具体信息不应影响结构。API 作为应用程序之间通信和交互的信息特征[8]，能够很好地反映与具体实现无关的特征。从前文“微服务划分的角度”来说，由于有选择性的忽略了具体实现相关的特征，选择 API 作为最小单位恰好符合“关注业务的设计”角度。但这种方案要求系统在设计过程中足够符合规范，或需要人工介入

对不规范的信息进行修正，以保证接口足够明确并且接口名称足以显示功能，方便后续划分时有充分且正确的信息。

在构建图结构步骤中，过往工作的主要关注点集中在两个方面：静态信息，动态信息。其中，静态信息为不需要实际运行就能获得的信息，例如代码内容、（在 git 等版本控制系统中）修改记录[9]、修改人员等等。借助静态信息时，研究者们常常将各模块关联度计算问题转化为语义分析问题，例如将函数或文件中去除停用词后的单词表作为多维特征向量[9]，或直接取 API 信息中的单词作为多维特征向量[8]，以及将单词表按相似度映射到公开词典并取映射后词表作为聚类依据[6]。除此之外，也有研究者根据单一责任原则，认为可以通过文件间修改关系以及各文件修改者之间的关系，认为需要一起修改的文件应当被放在同一部分[9]。动态信息则是在系统的运行过程中所获得的实际关联信息，例如模块间的调用关系，用户运行路径等等。部分研究者认为，由于系统随着时间需求不断变化，后期系统的实际思路与最初设计时的思路已经有了偏差，因此使用设计方面的信息为依据是不足够可靠的[10]。因此他们提出，可以使用动态信息作为划分依据。当监控用户执行路径时，我们所能获得的原始信息是各个函数之间的调用情况，根据这些信息可以构建两种图：由原始调用信息所构建的调用图，这是一种有向有边权图，可以展现各个函数间的直接关联[10]；将动态信息按执行者分组，在同一组出现的函数认为是有关联的，也就是以一个用户的一次完整执行路径为单位，将每次执行过程中涉及到的函数都认为是有关联的，这是一种无向图，可以展现各个函数之间的传递关联[11]。

图上划分/聚类步骤则比较偏向于算法设计问题中的图结构划分/聚类问题，可以充分借鉴算法领域过往的研究成果。以往成果大多再上一步骤中构建出无向图，此时图中的边即为各点间的关联程度，边权越大表示所连接的两点关联程度越高。之后再按以下两种主要思路进行：一是作为划分问题，先将整个系统试作整体，再从连接不紧密处入手，逐步划分。此时划分问题与传统图划分问题较为类似，可以选择由边权入手，逐步按边权由小到大将边删除。在这样的删除过程中，图会被逐渐划分成多个联通块，由于每次删除的都是边权最小的边，各个联通块间的紧密程度理论上会逐渐变大。二是作为聚类问题，先将系统按每个最小单位划分成非常细碎的部分，再从连接紧密处入手，逐步聚类。此时聚类问题更接近机器学习中的聚类问题，按照前期所获得的各个特征向量进行聚类。由于聚类时选择的一定是更加有价值的边，因此各个部分在聚类过程中理应是越发紧密的。另外有研究者以有向图为依照进行划分，这种情况下算法的设计会较为复杂且麻烦。例如按照自行设计的数条划分规则进行划分[10]，此时容易造成覆盖不全面或有特殊情况导致遗漏的问题。

五、 目前研究的不足

目前的研究关注的基本都在微服务的逻辑层面特性，而忽略了微服务的物理层面特性。换言之，只考虑如何使划分后的微服务间在逻辑上保持尽量独立，却没有考虑去利用好“云基础设施”的优势。由此导致其产出的结果更像是传统架构下的更精细的模块划分，而非云服务时代新架构下的微服务划分。

文献综述参考文献

[1] Cito J, Leitner P, Fritz T, et al. The making of cloud applications: An empirical study on software development for the cloud[J]. Joint Meeting on Foundations of Software Engineering, 2015: 393–403.

[2] Thönes J. Microservices[J]. IEEE Software, 2015, 32: 116-116.

[3] Newman S. Building microservices: designing fine-grained systems[M]. O'Reilly Media, Inc., 2015.

[4] Alessandra Levcovitz, Ricardo Terra, Marco Tulio Valente. Towards a Technique for Extracting Microservices from Monolithic Enterprise Systems[J]. 3rd Brazilian Workshop on Software Visualization, Evolution and Maintenance, 2015: 97-104.

[5] Schermann G, Cito J, Leitner P. All the services large and micro: Revisiting industrial practice in services computing[J]. International Conference on Service-Oriented Computing. 2015: 36–47.

[6] Baresi L, Garriga M, De Renzis A. Microservices Identification Through Interface Analysis[J]. European Conference on Service-Oriented and Cloud Computing, 2017: 19-33.

[7] Krause A, Zirkelbach C, Hasselbring W, et al. Microservice Decomposition via Static and Dynamic Analysis of the Monolith[J]. IEEE International Conference on Software Architecture Companion, 2020: 9-16.

[8] Al-Debagy O, Martinek P. Extracting Microservices' Candidates from Monolithic Applications: Interface Analysis and Evaluation Metrics Approach[J]. 2020: 289-294.

[9] Mazlami G, Cito J, Leitner P. Extraction of Microservices from Monolithic Software Architectures[J]. IEEE International Conference on Web Services, 2017: 524-531.

[10] Fola-Dami Eyitemi, Stephan Reiff-Marganiec. System Decomposition to Optimize Functionality Distribution in Microservices with Rule Based Approach[J]. IEEE International Conference on Service Oriented Systems Engineering, 2020: 65-71.

[11] Jin W, Liu T, Zheng Q, et al. Functionality-Oriented Microservice Extraction Based on Execution Trace Clustering[J]. IEEE International Conference on Web Services , 2018: 211-218.