毕业论文（设计）开题报告

毕业论文（设计）题目：微服务划分工具的设计与实现

一、综述本课题国内外研究动态，说明选题的依据和意义

（一）微服务架构

随着“云上计算”概念的普及，微服务架构概念也变得热火朝天，因为它使每个微服务部署在完全独立的机器上，各服务本身的独立调度、伸缩变得更加容易，各服务间的壁垒也更加坚固，这能充分利用云的优势，是全新的适应云生态环境的架构。

（二）微服务划分工具意义

微服务架构下代码结构特殊，为充分发挥优势，需要对代码进行合理划分。在工业上，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、步骤也通常没有的普适性。这导致了这种划分方案难以普及。自动化普适化划分方案是微服务架构推广的基石，因此对其有着较为急切的需求。

（三）代码划分问题

代码划分问题一直以来都被人所讨论，特别是传统架构下的“模块划分”问题已经有长久的历史。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。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。

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、修改人员等等。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。

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，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；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。另外一小部分文章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些。例如[8]中按照自行设计的六条划分规则进行划分，就容易造成情况覆盖不全面的问题。

（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。

二、研究的基本内容，拟解决的主要问题：

微服务划分工具面对的最基本的问题为自动化解决划分，即面对不同的系统都能够在尽量少的人工干预下完成对系统的分析、划分。这就要求工具选择足够有效的特征，使得对不同目标系统都能从中反映自身特点，从而便于划分。

另外一个问题是需要尽可能多地发挥微服务架构与云基础设施的优势，而这一点是目前研究所忽略的。

针对上述问题，本项目旨在构建一套微服务划分工具。其能够使用户几乎全自动化地获得传统架构下以 Java 语言构建的系统在微服务架构下的分割方式，即原始代码的哪些部分应当分别组合成微服务。同时，工具所提供的分割方式，应当使划分后的微服务在逻辑层面和物理层面都能够发挥微服务架构的优势。

具体而言，本项目沿用目前研究的整体思路，将微服务划分问题转化为图结构的划分问题，而在细节上对其进行继承与发展。

在建图阶段，将以代码的动态信息为依据，也就是监控代码实际运行过程中的调用情况，获取其记录文件，并以此为分析代码各部分关联度的基石。如果考虑到系统设计者在设计过程中会因为不断变更的需求而影响其对系统运行情况的把握，用实际运行情况的记录就能被视为一个足够客观的方案。同时，受物理机影响，被划分成的微服务在云架构下相互调用时会有更大的代价，因此参考实际运行情况更能避免在这方面的误判所导致的代价。

这一步的监控过程需要使用到 Kieker 工具，其拥有较长久的历史并享有不错的口碑，能够将代码运行过程以文本文件形式输出。用户进行监控时，需要面向尽可能长的一段时期或对尽可能丰富而完善的用户案例，这样才能够保证不会因为原系统在时间尺度上的极端案例导致划分结果的偏颇。

在划分阶段，设计了两种度量以验证划分结果的质量。

第一种度量从逻辑角度出发，考虑划分结果的内聚度与外耦合度。各个微服务的内聚度越高，微服务间的外耦合度越低，就越认为划分结果的质量好，这也是传统模块划分下考虑的一个重要因素。

第二种度量从物理角度出发，考虑划分结果内是否存在明显需要高性能或高可靠的部分。当一个模块的流量很大时，我们认为其是该微服务的性能瓶颈，是需要保证高性能的。将其从微服务中划分出来可以便于对其进行单独的弹性伸缩，这样既能保证整体系统的性能，又能降低系统的成本。当一个模块受到很多其他模块的依赖时，我们认为其是该微服务的核心模块，是需要保证高可靠的。将其从微服务中划分出来可以便于对其设置额外的安全措施，保证当其出现问题时能够快速处理，既提高了系统整体的可靠性，又不会花费过于高昂的代价。

三、 研究步骤、方法及措施：

1. 调查研究云环境与传统环境的主要区别；调查研究微服务架构特点及优势，对比传统架构。从大环境背景出发，拓展思路。针对新问题与传统问题的矛盾点，得出设计方案过程中应当注意的关键处。

2. 查阅相关文献，梳理并对比目前研究中不同方案的关注点、亮点及对关键问题的解决措施。拆解出算法涉及的不同阶段：确定最小划分单位；抽象并构建图结构；对图结构划分或聚类。

3. 对现有方案分析与总结，针对关键问题设计本项目核心算法，筛选出不同阶段下最为合理的解决方案。

4. 搜集实验数据，对已有的具有特点的实验数据进行保存。便于后续快速获得反馈，检测实验效果。

5. 基于 Angular 和 Bootstrap 框架进行前端部分的设计和实现，采用基于 VisJS 可视化库的数据可视化方案。基于 Spring 框架、PostgreSQL 数据库开发服务端部分。

6. 针对已有实验数据确定所需数据具体格式，依此完成文本处理模块，便于核心算法中构建图结构阶段的操作。

7. 设计图上划分阶段的两个核心度量，具体考虑到各个部分的物理意义以及对于结果的影响情况是否合理。

8. 设计系统核心算法的实验部分，完成对算法效果的评估，针对不足之处进行进一步的改进与修正。以此流程反复迭代打磨，最终获得理想的效果。

四、主要参考文献：（所列出的参考文献不得少于 10 篇，其中外文文献不得少于 3 篇。）

[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.