密级：保密期限：

北京郵電大學

硕士学位论文

题目：微服务平台中服务划分和选择策略研究和应用

学号：2016110779

姓名：杨宁

专业：计算机科学与技术

导师：张海涛

学院：计算机学院

2019 年 1 月 1 日

独创性（或创新性）声明

本人声明所呈交的论文是本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢中所罗列的内容以外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得北京邮电大学或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

申请学位论文与资料若有不实之处，本人承担一切相关责任。

本人签名：_____ 日期：_____

关于论文使用授权的说明

本人完全了解并同意北京邮电大学有关保留、使用学位论文的规定，即：北京邮电大学拥有以下关于学位论文的无偿使用权，具体包括：学校有权保留并向国家有关部门或机构送交学位论文，有权允许学位论文被查阅和借阅；学校可以公布学位论文的全部或部分内容，有权允许采用影印、缩印或其它复制手段保存、汇编学位论文，将学位论文的全部或部分内容编入有关数据库进行检索。（保密的学位论文在解密后遵守此规定）

本人签名：_____ 日期：_____

导师签名：_____ 日期：_____

微服务平台中服务划分和选择策略研究与应用

摘 要

随着云平台业务规模的扩展，大型应用的需求增加，传统的单体服务框架的问题越来越明显，微服务架构成为一种趋势。在单体架构向微服务架构转换的过程中，当前流行的微服务划分方法中只考虑微服务的大小，或者没有考虑原有架构中应用之间的功能联系，无法有效提高服务的复用率，减小平台代码冗余；随着 Internet 中 Web 服务数量的增加，服务选择技术也相继出现。但是传统的服务选择方法大多只考虑服务静态的 Qos，未考虑细粒度的在线服务能力和任务的特征，无法保证应用执行的高效率。

如何有效的划分微服务以及组合微服务实例，实现一种高效的服务路径选择策略，提高整个平台中应用的执行效率是本文的研究重点。首先本文分析当前一些流行的微服务设计思想，提出一种基于领域驱动设计思想的语义耦合的服务划分策略，该策略综合考虑了微服务平台应用功能关联性大的特点以及微服务的划分原则，通过分析微服务平台应用代码的语义耦合关系构建无向加权图，然后利用具有高内聚低耦合原则的 GN 聚类算法得到最优的微服务组合；其次本文分析当前一些主流的服务选择算法，提出一种细粒度的性能预测模型，该性能预测模型综合考虑了微服务实例的在线处理能力、任务的特征以及微服务实例之间的传输条件等，能够预测每个微服务实例的执行时间。然后，在性能预测模型的基础上，提出一种性能感知的服务路径选择策略（PSPAS），该策略包括初始化路径选择阶段和自适应服务路径更新阶段，初始化路径选择阶段通过构建任务执行的单源点有向加权图使用最短路径算法得出最优服务路径；在自适应服务路径更新阶段，通过使用服务路径搜索空间缩减原则重新构建单源点有向加权图并更新最优服务路径，整体提高了任务的执行效率。最后，本文通过实验验证了提出的微服务划分策略和路径选择策略，实验结果表明本文所提出的微服务划分算法能够提高平台中服务的复用率以及提出的服务选择策略可以有效提高平台中应用的执行效率。

关键词：微服务架构 服务划分 语义耦合 服务选择 自适应更新

RESEARCH AND APPLICATION ON EXTRACTION OF SERVICE AND SELECTION STRATEGY IN MICRO-SERVICE PLATFORM

ABSTRACT

With the expansion of the cloud platform business scale and the increase of the demand for large-scale applications, the problem of the traditional monolithic architecture has become more and more obvious, and the micro-service architecture has become a trend. In the process of transforming monolithic architecture into the micro-service architecture, the current popular extraction method of micro-service from monolithic software architecture either considers the size of micro-service, or without considering the functional connection between the applications in the original architecture. Using traditional approach for extracting micro-service cannot effectively improve the reusability of service and cannot reduce the redundancy of code; as the number of web services on Internet is increasing, a lot of service selection methods have emerged. However, most of the traditional service selection methods only consider the static QoS of the service, and do not consider the fine-grained online service capabilities and task features at the same time, and cannot guarantee the high performance of application.

This thesis focuses on how to effectively extract micro-services and how to combine the micro-service instances to realize an efficient service path selection strategy and improve the execution efficiency of the application. Firstly, according to the analysis of the current mainstream micro-service design algorithm, this thesis proposes a domain-driven design idea based semantic coupling strategy. This strategy considers the function characteristics between the application of micro-service platform that the functional relevance of the two application is strong and the extraction principle of micro-service. The strategy comprehensively considers the characteristics of the micro-service platform application

function and the division principle of micro-services. It constructs the undirected weighted graph by analyzing the semantic coupling relationship of the code of the micro-service platform, and then uses the GN algorithm with high cohesion and low coupling principle to get the optimal micro-service combination. Secondly, this thesis proposes a fine-grained performance prediction model that takes into account the online processing capabilities of micro-service instances, the characteristics of tasks, and the transmission conditions between micro-service instances based on the analysis some current mainstream service selection algorithms. It can predict the execution time of each micro-service instance. Then, based on the performance prediction model, a performance-aware service path selection strategy (PSPAS) is proposed, which includes an initial path selection stage and an adaptive service path updating stage. We use the shortest path algorithm to construct an optimal micro-service path at the initial service selection stage. And then we use an adaptive and efficient micro-service path updating method during the task execution, which can narrow the service path search space prior to online performing the micro-service path selection. Finally, we conduct extensive performance experiments for verifying the proposed extraction algorithm of service and service path selection algorithm. The experimental results show that the extraction algorithm of micro-service proposed can improve the reusability of services and the proposed service path selection strategy can effectively improve the execution efficiency of the application.

KEY WORDS**:** microservice architecture service extraction semantic coupling service selection adaptive updating

目录

第一章 绪论

1.1 研究背景和意义

随着 SOA、DevOps、持续交付、虚拟化、分布式系统等各种技术的快速出现和发展，软件系统的业务逐渐复杂，开发体量逐渐变大，传统分层的单体架构模式的不足之处越来越明显，许多系统只是简单的将多个系统的功能和逻辑整合，这样导致系统的规模会随着业务量的进一步增加而急剧地膨胀，进而产生架构臃肿、业务逻辑复杂、数据流向复杂等一系列问题，这些问题会给整个系统的开发、维护、部署以及后期的升级带来巨大的困难。

针对上述问题的解决方法就是对现有的业务系统进行拆分和重组，将原有的系统拆分成独立的模块来降低系统整体的复杂度、代码的冗余率以及各个子系统、功能模块之间的耦合程度，因此微服务架构[1][2]获得关注。微服务是最近非常流行的系统架构解决方案，其核心思想是将大型的、复杂的应用划分成细粒度且内聚的服务，划分后的服务都有各自的服务边界和声明周期，并且便于部署和扩展，各服务间配合工作完成任务。

然而目前对于微服务架构，业界尚无明确定义，而且围绕微服务的工作是有限的。在现有技术水平上，微服务缺乏工具支持，很大一部分工作只是概念性的[3]。如何划分微服务，制定划分策略，满足微服务内部的高内聚性和微服务之间的低耦合性，是我们面临的主要问题。而且，目前对于微服务划分方法没有明确的划分标准和方法，针对这个问题，我们根据微服务高内聚低耦合的原则，以及软件工程中领域驱动设计模型[4][5][6]的建模思想，提出一种基于领域驱动设计的微服务划分策略，提高了系统中服务的复用率，降低了代码的冗余率。

另外，对于基于微服务架构的分布式处理平台，微服务划分完成，如何组合微服务[7][8]，制定相应的微服务路径选择策略，对于提高应用的执行效率尤其重要，也是目前研究的热点问题。另一方面，如何综合考虑微服务实例细粒度的在线处理能力、任务的特征和微服务实例间的数据传输条件，制定高效的服务路径选择策略具有很大挑战性，主要面临以下两个问题：

（1）如何制定初始化服务选择策略，使其综合考虑微服务实例细粒度的在线处理能力、任务的特征以及微服务实例间的数据传输条件。一方面如果在初始化服务策略中只考虑微服务的静态特征[9]，将会导致微服务路径选择不准确，将会延长整个应用的处理时间；由于任务会有不同的特点，如输入数据量可能千差

万别，这可能会进一步导致微服务选择的不同，另一方面，由于数据在传输过程中会受到数据量以及网络带宽的影响，因此，选择合适的初始化服务选择策略对于提升平台应用的执行效率至关重要。

（2）在平台应用处理过程中，如何根据微服务的实时处理能力自适应的更新微服务路径[10][11]，以实现高效的应用执行效率。在应用执行过程中，待执行的子任务所对应的微服务实例的处理能力是随时间动态变化的，因此选择的最优服务路径会发生改变。如果只是考虑初始化的服务路径，那么初始选择的服务路径可能不是当前最优。因此，设计一个自适应的动态的服务选择策略，对于选择最优的服务路径，提高应用的执行效率至关重要。

目前大多数的服务选择策略只考虑服务静态特征，例如服务响应程度，服务利用率，吞吐量，并没有考虑服务实例运行时特征。对于由多个子任务组成的大型应用来说，当前一个子任务被执行完成后，后面的子任务对应的微服务实例的资源状态会时刻动态改变，所以初始化最优的服务路径可能是无效的。另外，当今一些动态的自适应[12]的算法被提出来，这些方法选择合适的候选服务来创建最优的服务路径，并且根据服务状态的改变，动态的更新服务路径，然而他们并没有考虑任务细粒度的特征，即使是相同的任务，不同的数据，也会导致应用处理效率不同。

1.2 国内外研究现状

微服务架构的概念作为一种新型的软件架构在今年来引起了国内外专家的广泛关注。然而微服务并不是被发明出来的，而是随着云计算技术、SOA、DepOps技术的发展，从世界各地技术发展的趋势与实践中总结出来的。如何制定合理的服务划分策略以及服务选择策略（包括初始化服务选择策略和路径自动更新策略），优化微服务平台中的服务组合，对于提升微服务平台中应用的执行效率至关重要，国内外有很多针对微服务平台中服务划分策略和服务选择策略的研究。

（1）微服务平台中服务划分策略研究

近年来，微服务架构是当今软件架构模式新的热门话题，而在微服务架构中如何划分微服务以及确定微服务的大小[13]，也就是如何确定组件边界[4]是一个主要挑战。根据 Newman[4]的观点，微服务的大小很重要，因为微服务的粒度会影响服务质量（Qos）。服务的粒度高度依赖于服务划分是否合适，虽然微服务架构鼓励小型的服务设计但是太细粒度的服务会导致很多无效的大量的交互。

Gerald Schermann[14]等人通过研究 42 家不同规模的公司的服务计算实践来研究服务计算领域中的工业实践，并特别关注微服务的趋势。重点研究了服务的

大小和复杂性，结果显示不同的服务代码行数是不同的，对于使用代码行数（LOC）来作为划分微服务的标准，该策略并不合适。因为微服务是使用不同的技术堆栈构建的，这些技术堆栈在实现功能时，对应的 LOC 可能有所不同。此外，根据服务的类型，服务也有不同最低 LOC。例如，协调多个任务之间的调用的过程服务可以具有 100 到 1000 个 LOC，并且数据服务可以实现 10 到大约 100 个 LOC。

在文献[13]中，提供了一种结构化的方式--ServiceCutter 在单体架构代码库中划分微服务。作者提出了一种通过图割的方式支持结构化的服务分解工具，该工具依据 16 种不同的耦合标准，这些标准是从专业文献中总结出来的。软件支持和文档例如领域模型和用户用例作为输入来产生耦合值，并且构建图。但是，ServiceCutter 无法从单体架构本身获取必要的信息来构建图，必须依赖用户提供在特定的模型中提供相应的软件支持。

Liu 和 Yang[15]提出基于文本的服务聚类方法，在服务文件中，使用向量空间模型来代表服务文件，通过使用多层次的聚类算法来执行服务聚类。这些方法很少将服务功能的语义考虑在内，大部分方法使用向量空间模型来减少服务文档的维度。

然而当前的服务划分方法中并没有根据平台需求以及平台中应用之间功能的相关性来综合考虑，这将导致服务划分后的结果是服务之间的耦合性比较差，代码复用率比较低，平台中代码的冗余度比较高。根据平台的应用的特征，我们的平台中应用功能相关性比较强，每个应用中代码的重用率比较高，为了提高服务的复用率，降低代码的冗余，我们提出了语义耦合的服务划分策略，该策略通过分析代码之间的语义耦合程度，来表示两个代码的耦合程度，将耦合度作为权重构建无向有权图，再通过社区发现算法 GN 算法[16]得到最好的划分结果。通过实验验证了该方法可以将应用有效划分成微服务，并且能够提高服务的复用率，减少平台的代码量。

（2）微服务平台中的服务选择策略研究

目前，随着网络服务数量的逐渐增加，许多服务选择方法已经兴起。最近的几篇文献[7][8][9]更侧重于支持基于 Qos 的服务选择和组合。但是它们只考虑静态的 Qos 特征，例如响应性、可用性、和吞吐量，并没有考虑选择的服务实例运行时的特征。应用中的前一个子任务执行完成后，之后的服务实例的 Qos 会随着时间动态改变。因此在执行大规模视频任务期间，一个预定义的最优的服务组合可能是无效的。

另外，还有一些动态的自适应的算法提出优化服务组合整体的 Qos[10][11]，文献[17][18]提出了云计算平台中性能感知的服务选择和组合方案。这些方法选择合适的组件服务来创建最优的服务组合，并且能够根据服务提供平台中的各种改变

动态的改变最优的服务组合。但是这些方法没有考虑视频处理任务细粒度特征，这些特征对视频任务总的执行时间和服务资源的性能具有重要的影响。

为了优化微服务路径选择，提高任务处理效率，在进行微服务选择时不仅需要考虑微服务实例的在线的处理能力，还需综合考虑任务的特征以及微服务实例之间的传输条件，这就需要制定新型的服务选择策略。另外一方面，为了保证服务路径总是最优，我们需要自适应的更新的服务选择，即能够动态更新服务路径。

本文提出的性能感知的服务路径选择策略在进行服务路径选择时充分考虑了微服务实例的实时的处理能力、以及任务的特征和微服务实例间的传输条件，得到了最优的服务路径，提高了应用的执行效率。

1.3 论文的主要研究内容

本文主要包括在微服务平台中制定合理的微服务划分策略和综合考虑微服务在线处理能力、任务的特征和微服务实例间的数据传输条件的微服务选择策略的研究与实现，主要研究内容有以下几点：

（1）基于领域驱动设计的微服务划分方法

本项目分析传统的分布式处理系统的代码之间的关系，结合软件工程中领域驱动设计模型的建模思想，研究基于微服务架构的云平台中微服务的细粒度剖分方法。主要研究内容包括：

通过分析当前一些主流的服务划分策略，提出了一种基于领域驱动设计思想的微服务划分方法，该服务划分方法主要是（1）建立微服务划分模型，该模型主要分为三个阶段，分别为单体架构阶段、图阶段、微服务架构阶段，单体架构阶段整个系统处于单体架构阶段，系统代码没有进行划分。图阶段会根据整个系统中所有文件之间的关系构建无向的加权图。微服务阶段通过微服务划分策略进行微服务划分。（2）在微服务划分方法中主要涉及到两个过程，第一个过程为单体架构阶段到图阶段（构建过程），在这个过程中本文提出了一个基于领域驱动设计模型中“低耦合，高内聚”思想的语义耦合策略，领域驱动设计中每一个子域的关系就是要求“低耦合，高内聚”，每个子域之间都有各自的通用语言。因此本文将单体架构中的每一个文件作为一个子域，通过分析两个文件中代码语义的相似性来表示两个文件的耦合程度，即图中每两个节点的权重，从而构建无向带权图。第二个过程为微服务划分过程（聚合过程），根据微服务的划分原则“低耦合，高内聚”，本文利用具有同样原则的社区划分算法----GN 算法，通过 GN 算法对图中的边进行移除，并计算移除后对应的图的模块性 Q 值，直到所有边都被移除，最后选择 Q 值最大的对应的图集合。

（2）性能感知的微服务路径选择方法

本文通过建立微服务平台综合监控指标体系，涵盖 CPU、内存、网络、磁盘等资源使用情况，以及微服务健康状态、处理能力、资源消耗情况等。利用微服务平台的基准程序，分析微服务平台中在线微服务实例的细粒度特征、待执行任务的特征以及微服务实例间的数据传输条件，建立针对大规模微服务平台的细粒度性能特征模型，提出高效的性能感知的微服务选择策略；分析动态自适应的服务选择策略，本项目提出通过缩减服务选择空间在线更新微服务选择路径的策略，进一步提高了微服务平台中应用的执行效率。主要研究内容包括：

①基于微服务性能模型的微服务路径选择策略

当前主流的服务选择策略，要么是没有考虑在线微服务处理能力，要么就是没有考虑任务的特征。本文提出了一种综合考虑微服务实例的特性、任务的特征和微服务实例间的数据传输条件的微服务选择策略，该策略首先建立微服务性能模型，包括数据处理时间模型和数据传输时间模型，数据处理时间与数据的大小，视频数据的分辨率，微服务实例的特征等都是相关的，因此综合考虑以上特征，我们通过机器学习方法执行不同因素的回归分析建立一个准确的数据处理时间模型。数据传输时间与数据的大小及网络中数据传输速率是相关的。基于微服务性能模型，提出并实现了一种性能感知的服务选择策略，该策略将有 N 个子任务的应用中的每一个子任务中的微服务实例作为一层节点，构建具有 N 层节点的带权图，每一条边上的权重即为时间性能模型中的数据处理时间和数据传输时间之和，构建图完成，根据最短路径算法，得到图中初始化的最短服务路径。

②动态的自适应的服务路径更新策略

当前的服务选择策略中，主要考虑了静态的微服务实例特征，并没有考虑在线的微服务特征。本文基于初始化的微服务路径，提出了动态的自适应微服务路径更新策略。该策略应用路径搜索空间缩小原则，（1）我们为每个子任务定义了时间率=数据处理时间/数据传输时间，我们将子任务分为计算主导型子任务和传输主导型子任务，每一类有固定的时间率阈值，如果当前子任务的时间率超过了计算主导型子任务的时间率阈值，则当前子任务为计算主导型子任务，如果当前子任务的时间率小于传输型子任务的时间率阈值，则当前子任务为传输主导型子任务；（2）如果子任务属于计算主导型子任务，我们就选择微服务实例中数据处理时间最短的前 m 个作为该子任务的微服务实例选择空间；如果子任务属于传输主导型子任务，我们就选择微服务实例中数据传输时间最短的前 m 个作为该子任务的微服务实例选择空间；（3）对之后的子任务应用(2)中的原则，重新构建分层的无向有权图。重复以上步骤，直到应用执行完成。

（3）平台实现和性能分析

本文搭建基于微服务架构的云平台，提供微服务划分、服务选择等关键平台支撑模块，保证微服务高效的运行和服务质量；提出基于领域驱动设计模型思想的微服务划分方法，支持原有平台功能的微服务化，提高平台服务的复用率，保证平台持续部署；提出性能感知的微服务路径选择策略，实现平台应用的高效执行。本项目通过实验，对微服务化后的系统中代码规模减少率以及服务复用率来对微服务划分方法进行性能分析，从而证明了本项目中提出的微服务划分方法能够有效划分微服务；本文利用微服务平台的应用，对文中提出的性能感知的服务路径选择策略的性能进行验证，通过对比实验以及对结果分析，从而证明本文提出的服务路径选择策略能够提高平台应用的执行效率。

1.4 论文组织结构

本论文将按照以下六个章节展开：

第一章：绪论。首先介绍了本文的研究背景，然后介绍了目前微服务平台中服务划分策略和服务选择策略的研究现状，最后介绍了本文的主要研究内容。

第二章：相关技术。主要介绍了微服务划分策略相关的一些背景知识，包括微服务架构以及软件工程相关技术，另外还有服务选择策略相关的背景知识，包括基于静态的 Qos 的服务选择路径以及基于动态的自适应更新的服务选择路径。本章为后续研究和平台开发奠定基础。

第三章：基于领域驱动设计思想的语义耦合的服务划分策略。首先对当前的服务划分策略进行了分析，然后结合我们微服务平台的特点设计出适合我们微服务平台的服务划分策略，最后通过算法对比，给出了适合我们微服务平台的服务划分策略。

第四章：性能感知的服务路径选择策略研究。这部分是本文的核心。首先对当前的服务选择策略进行分析指出不足之处，然后结合我们微服务平台的特点以及微服务应用的特征设计出性能感知的时间预测模型，然后介绍了基于时间预测模型的初始化路径选择策略，并详细的介绍了该策略的实现过程。最后介绍了微服务路径的动态的更新策略，并详细的介绍了该策略的实现过程。

第五章：系统实现与测试分析。首先描述了微服务系统中主要组成部分的详细实现，接着论述了服务划分策略和服务路径选择策略的详细实现，之后介绍了测试环境配置，最后基于视频浓缩算法功能镜像对本文提出的服务路径选择算法的性能、以及提出的性能感知的时间预测模型，以及服务路径选择策略的性能进行实验测试，并对实验结果进行了分析。

第六章：结束语。对本文的所有研究工作做出总结，结合目前行业热点展望微服务平台中服务划分和服务选择策略为了的研究趋势，分析本文提出的服务划分和服务选择策略进一步的优化方向。

第二章 相关技术介绍

本章主要介绍了后续章节涉及的一些技术知识，主要包括与本文所优化的视频监控云平台相关的云环境相关技术，混合存储体系结构相关技术以及当前云平台中数据分布相关技术。

2.1 微服务相关技术

2.1.1 微服务及其框架简介

微服务架构是一种互联网应用服务的软件架构，主要应用于互联网应用服务的服务端软件开发。微服务架构由面向服务架构 SOA 发展而来，其核心理论基础来自于康威定律[19]中关于组织结构与其设计的系统结构之间关系的描述，即任何组织设计的系统，其结构都是组织本身沟通结构的复制。2014 年学者 Martin Fowler 正式提出微服务架构的概念[1]：微服务架构以一套微小的服务的方式来开发和部署一个单独的应用，这些微小的服务根据业务功能来划分，通过自动化部署机制独立部署运行在自己的进程中，微服务之间使用轻量级通信机制来进行通信。一个典型的微服务架构应该包括客户端、微服务网关、服务发现、微服务原子层、数据库、部署平台等模块，根据不同应用类型及服务规模，可以增加负载均衡、权限认证、服务熔断、日志监控等模块，来满足服务的非功能性需求。

虽然 Martin Fowler 给出了微服务的一种定义，但是他也同时指出，微服务并不局限于该定义。Martin 尝试归纳和描述微服务架构风格所具有的共同特点，这些特点并不是所有微服务架构风格都要拥有的，也不是用来定义微服务架构本身的，而是微服务架构风格被希望要拥有的特点。也就是说，微服务架构风格不是微服务化的终点，而是微服务化的方向。下面简单介绍一下微服务架构风格的特点。

i 服务组件化 微服务中，服务可以被当作进程外组件，独立进行部署，服务之间利用网络服务请求或者远程过程调用来进行通信。一个好的微服务架构的目标是通过服务合同中的解耦服务边界和进化机制来帮助各个微服务独立部署运行。微服务架构的设计者希望对任何一个组件或者服务的改动和升级都只需要重新部署该服务而不需要重新部署整个应用程序，并且在升级过程中尽可能少地改变服务间通信的接口。

ii 围绕业务功能组织服务 当把一个大的应用拆分成小的部分的时候，通常的方法都是根据技术层面分为 UI 团队、服务端逻辑团队和数据库团队。但是这

种拆分团队的方式会使得即使一个简单的变动都会导致整个团队需要耗费时间和预算来适应和协调。康威在文献[19]中提出了康威定律，其中有一条提到: 任何组织设计的系统，其结构都是组织本身沟通结构的复制。根据康威定律的这条描述，微服务架构采用围绕业务功能来拆分应用和组织服务的方法。在微服务架构中，设计组织被分为小的开发团队，与之相对应的是，应用被拆分为小的服务，应用之间的沟通过程就是团队之间的沟通过程。为保证应用之间沟通过程清晰明确，团队之间需要划分清晰的服务边界。这样的划分方法也要求每个小团队本身提供开发过程中所需要的所有技能。

iii 基础设施自动化 许多开发团队都是使用持续交付和持续集成技术来构建微服务架构的应用和系统的，这使得基础设施自动化技术得到了广泛的应用。而随着容器技术、云计算技术等技术在过去几年的快速发展，基础设施自动化技术取得了长足的进步，这也间接降低了微服务构建、部署、运行的复杂性。

2.2 服务划分相关技术介绍

2.2.1 软件工程相关技术

2004 年，建模专家 Eric Evans 出版了《领域驱动设计——软件核心复杂性应对之道》，该书叙述了当前企业复杂业务开发的弊端，提出了领域驱动设计的基本思想并系统地讲述了应用领域驱动设计的方法，介绍了常用的技术经验，并强调了在处理复杂领域的业务逻辑时应该坚持的基本原则[20]。

2006 年，Jimmy Nilsson 全面详细地阐述了领域驱动设计的理论，结合具体的实例生动地展示了领域驱动设计在具体项目中的实际应用，首次为领域驱动设计开发人员揭示了完整的开发路[21]。随着开发人员对领域驱动设计研究的深入，逐渐出现了支持领域驱动设计的开发框架。

Java 平台有 Axon Framework，它是一个基于领域驱动设计而搭建的命令查询职责分离（Command Query Responsibity Segeration，CQRS）架构，特点是可以整合 Spring，简化了开发人员建立基于领域驱动设计架构的 CQRS 应用，它通过一些 Annotation 使得代码和测试分离，它支持丰富的领域事件，开发人员只需要把精力于集中业务逻辑的设计上。

领域驱动设计的主要思想是将业务领域中的概念与软件元素对应于软件设计中，组成领域驱动的理论和技术其实早已存在，它实现的基础是面向对象的编程方法(Object Oriented Programming，OOP)，领域驱动中的实体跟 OOP 中的对象相对应，OOP 的继承、封装和多态等特征都适用于领域驱动的开发，领域对象应可以设计成简单的类或接口。

领域对象需要与服务、仓储以及工厂协同工作。与此同时，系统需要密切跟踪领域状态的变化、缓存以及事务管理等元素，虽然它们是非领域相关且可重用，但是把它们散布在领域层的所有代码中会导致领域层的繁杂与混乱。为了实现领域对象与上述关注点的解耦，并妥善管理代码间的依赖关系，单纯使用 OOP 技术不能为领域驱动提供完美可行的解决方案。因此依赖注入(Dependency Injection，DI)[1]和面向方面编程(Aspect Oriented Programming ，AOP)[22]等概念可以作为 OOP 的补充，辅助实现领域驱动，以达到软件模块间的“高内聚、低耦合”的目标。

领域驱动的具体实现需要依附于成熟的开发工具，目前主流的 Microsoft.NET 和 J2EE 平台都已研发出足够优秀的集成开发工具支持领域驱动的项目实践，应用前人的开发经验，可以避免 “重复制造轮子”，最终降低项目的开发周期。基于这些项目经验分享，开发人员将领域驱动应用到了实际系统的设计开发中，在带来业务架构改进的同时，也发现一些问题[23]：

(1)领域驱动方法需要对业务领域进行全局考虑并进行整体设计，要求分析人员具备比较高的建模能力；(2)没有按照领域驱动进行抽象和封装公共的业务，在设计开发时容易出现多种风格[24][25]。现有的领域驱动设计思想的在软件开发的实际应用中，“领域驱动+ES”开发方式是比较流行的方式，但是它仍然沿用了传统的基于“请求/响应”模型的体系结构[26]，这种架构存在通信耦合程度高、并发能力不足的问题[27]，而事件模型将事件作为对象之间交流的媒介，系统以“事件”为驱动，通过解耦的方式让系统组件进行通信，适用于实时高并发的 Web 应用环境[28][29]。

2.2.2 语义相似度相关技术

本文中提出的服务划分方法使用了语义耦合的划分方法，主要使用语义相似度来表明两个文件之间的关系。相似性是存在于任意两个对象之间的一种普遍关系，而相似度是对相似性的定量表示。相似度计算是信息检索、数据挖掘、知识管理、人工智能等领域的基本问题。随着本体的广泛应用，基于本体语义的相似度计算及应用成为心理学和计算机科学交叉研究的一个重要课题。

目前关于信息对象之间的相似度计算主要是依据对象特征描述的外在的表现形式，如传统的文档相似度计算主要依据特征词是否均在两个文档中出现，特征词必须完全相同。在同一个概念以不同的表达方式出现或同一词汇在不同的上下文有不同的语义的情况下严重影响相似度计算的准确性。

基于本体的概念相似度计算是非常重要的基础工作，一个好的概念相似度计算方法对于语义 web、信息检索、数据挖掘、信息集成、知识管理等研究工作具

有重要的意义。同时，它还是文本文档之间、控制文档之间以及服务之间的语义相似度计算的基础。

Dekang 提出了一组具有广泛意义的相似度定义告诉我们，对象和对象之间的相似度与它们之间共性和差别相关，两个对象所拥有的共性越多，则相似度越大，而两个对象之间的差异越多，则相似度越小。当两个对象是同一个对象时，相似度是最大的。当两个对象不相关时，相似度最小。

相似度模型总的概括分为几何模型和特征对比模型[30][31]。用空间中的点表示对象，用点之间的距离反映对象间的相似度，将这一类模型统称为几何模型。最初对相似度的研究可追溯到于 Osgood 于 1952 年提出的语义微分方法[32][33]。Osgood 用一组意思相反的形容词构成一个特征空间，分别用这些形容词度量单词的语义，从而形成单词在特征空间的坐标。Osgood 定义单词在特征空间中的几何距离为单词之间的距离。在几何模型中可进行多维尺度分析和潜在语义分析[34][35]。几何模型中用几何空间中的距离来描述关于相似度的认识，在几何空间中，距离是对称的，然而许多心理学实验对几何模型提出了反例， Tversky 认为对象之间的相似度是非对称的。从二十世纪七十年代起，心理学家提出了许多非几何相似模型，其中最著名的是 Tversky 模型。Tversky 认为对象的一些属性无法用数字量化，这些属性更适合用定性的方式描述。Tversky 模型通过特征属性集合描述对象，相似度被定义为关于特征共同性和差异性的函数。Tversky 认为对象的相似度应满足匹配性、单调性和独立性[36][37]。在 Tversky 模型中，匹配性表明对象的相似程度由对象间的相同性和差异性共同决定单调性则表示相似程度随着共同性的增加和差异性的减少而增加独立性表明决定相似程度的三个因素之间互不影响。

通过以上两种相似度模型的分析，可以将相似度的计算概括分为以下三个基本步骤：1. 特征提取与特征选择。一般情况下，对象是以自然语言进行描述的，不能直接被计算机处理。对象的特征提取指抽取出对象的主要特征来描述对象。现有特征描述主要依靠统计方法构建相应的数学模型来表征对象目的是从对象中抽取出最有价值的特征信息。但是还会遇到特征项的维数问题，因此需要通过特征选择进行特征项的筛选。特征选取需要根据某个准则选择出最能反映对象特性的相关特征，从而达到降低特征维数、简化计算、提高相似度计算准确度的目的。2. 对象的表示。一般来说，用多个特征来描述一个对象，用间隔尺度来量化特征，间隔尺度是使用实数来表示的数量信息。假如选择了个特征，那么这个对象就可以表示成一个的矩阵。3.相似度计算方法。设 T 和 T`为矩阵中的两个对象的特征向量，常用相似度公式如下：内积、Dice 系数、余弦函数、Jaccard 系数等。

2.3 服务路径选择相关技术介绍

2.3.1 基于 Qos 的 web 服务路径选择技术

针对 web 服务和 web 服务组合的成果相当丰富，但是基于 Qos 的服务选择研究仍然方兴未艾。基于 Qos 的服务选择研究是近年来 SOA 计算和 web 服务组合研究的一个热点，也涌现了不少研究成果。意大利 Sannio 大学的 GerardoCanfora 利用遗传算法解决基于服务质量驱动的海量候选 web 服务的选取以及非线性的 web 服务质量聚合算法化简问题[38][39]。Zeng 等人利用了整数规划方法 自动得到满足用户需求的服务序列，满足全局和局部质量约束和用户喜好的同时，大大改善了算法的时间效率[40][41]。文献[42]介绍了一种基于知识的规划算法。文献[43][44]将服务的动态选择问题看作一个 Markov 决策过程，寻求能够使服务质量最优的决策方案。

目前对于 web 服务和 web 服务组合中的 Qos 驱动服务选择问题的研究，主要分为两大途径: 基于语义和基于 Qos 属性计算而在基于 Qos 属性计算的研究领域，根据研究背景的不同，可以大致分为工作流构件组装和形式化三种思路。基于 QOS 的服务选择实质上是一个优化问题，即从一组功能相同或相似的中选择性能最优的服务基于工作流的 web 服务组合过程可以分为两个部分，在每个流程节点，对所有具有相似功能的同类候选服务需要进行一个优劣排序；分别从每个流程节点挑选服务，组合成一个整体，要求这个整体服务的 Qos 最优。

2.4 本章小结

本章首先介绍了微服务相关技术，包括微服务以及微服务框架相关技术，然后介绍了服务划分相关技术，包括软件工程中领域驱动设计思想相关技术以及语义相似度相关技术，最后介绍了服务选择相关技术主要是基于 Qos 的服务选择相关技术。本章节主要为后续章节的研究提供基础。

第三章　基于领域驱动设计思想的语义耦合的微服务划分策略

目前对于微服务架构，业界尚无明确定义，而且围绕微服务的工作是有限的。在现有技术水平上，微服务缺乏工具支持，很大一部分工作只是概念性的。如何划分微服务，制定划分策略，满足微服务内部的高内聚性和微服务之间的低耦合性，是我们面临的主要问题。最初的一些服务划分思想或者只根据服务的粒度来划分或者根据特点的业务逻辑来划分，都不能满足我们平台应用的需求，本章根据平台中应用的特点，微服务高内聚低耦合的原则，以及软件工程中领域驱动设计模型的建模思想，提出一种基于领域驱动设计的微服务划分策略，保证了代码的复用率。

3.1 基于领域驱动设计思想的语义耦合的微服务划分策略

3.1.1 传统的服务划分策略及问题

目前许多研究人员已经使用了许多策略来确定微服务的大小以及如何划分微服务[13]。Gerald Schermann[14]等人通过研究 42 家不同规模的公司的服务计算实践来研究服务计算领域中的工业实践，并特别关注微服务的趋势。重点研究了服务的大小和复杂性，结果显示不同的服务代码行数是不同的，对于使用代码行数（LOC）来作为划分微服务的标准，该策略并不合适。因为微服务是使用不同的技术堆栈构建的，这些技术堆栈在实现功能时，对应的 LOC 可能有所不同。此外，根据服务的类型，服务也有不同最低 LOC。例如，协调多个任务之间的调用的过程服务可以具有 100 到 1000 个 LOC，并且数据服务可以实现 10 到大约 100 个 LOC。

Liu 和 Yang[15]提出基于文本的服务聚类方法，在服务文件中，使用向量空间模型来代表服务文件，通过使用多层次的聚类算法来执行服务聚类。这些方法很少将服务功能的语义考虑在内，大部分方法使用向量空间模型来减少服务文档的维度。

另外，当前的服务划分方法中并没有根据平台需求以及平台中应用之间功能的相关性来综合考虑，这将导致服务划分后的结果是服务之间的耦合性比较差，代码复用率比较低，平台中代码的冗余度比较高。根据平台的应用的特征，我们的平台中微服务应用功能相关性比较强，每个应用中代码的重用率比较高，为了

提高代码的复用率，我们提出了语义耦合的服务划分策略（DSCS），该策略通过分析文件中代码之间的语义耦合程度，来表示两个文件的耦合程度，将耦合度作为权重构建无向加权图，再通过社区发现算法 GN 算法来找到最优的划分结果。通过实验验证了该方法可以将应用有效划分成微服务，并且能够提高服务的复用率，减少平台的代码量。

3.1.2 解决方案

为了实现低耦合高内聚的微服务划分，提高应用中代码的复用率，降低平台中代码的冗余，在进行微服务划分时，我们遵循微服务低耦合高内聚的划分原则，并且考虑到平台中应用之间的功能相关性比较强，两个应用中服务的复用率比较高，因此，我们基于软件设计中领域驱动设计思想，提出了语义耦合的服务划分策略，源自软件工程中领域驱动设计的有界上下文的概念被提出作为微服务及其边界的一种设计思想，根据该思想，每个微服务应该对应于问题域中唯一的一个有界上下文。这将保证了集中于一个职责的微服务的可扩展性和可维护性。因此从软件设计领域提出划分策略，通过信息检索技术检查源代码文件的内容和语义成为一种划分方法，其中我们利用 tf-idf 方法分析原有系统中文件代码之间语义之间的相关性，得到系统中每个文件的之间的耦合度，构建无向加权网络图，之后利用具有“低耦合高内聚规则”的社区划分算法--GN 算法，对构建的无向加权图进行划分，得到符合服务划分原则的最优的划分结果。通过实验验证了该方法可以将应用有效划分成微服务，并且能够提高服务的复用率，减少平台的代码冗余。

3.2 基于语义耦合策略的服务划分模型

为了实现微服务的划分，使平台中应用中的服务复用率有效提高且降低平台中代码的冗余，我们设计了基于语义耦合策略的服务划分模型，该模型主要有三个阶段组成：第一个阶段单体阶段，第二个阶段图阶段，第三个阶段微服务阶段。在每两个阶段之间包括一次转换，共涉及两次转换，第一次转换是从单体阶段到图阶段的转换，称为构建图过程，第二次转换是从图阶段到微服务阶段的转换，称为聚类过程。下面是我们构建图过程的详细步骤：

原有平台中的应用处于单体阶段，从单体阶段到图阶段我们称为构建图阶段，此过程如图 3-1：

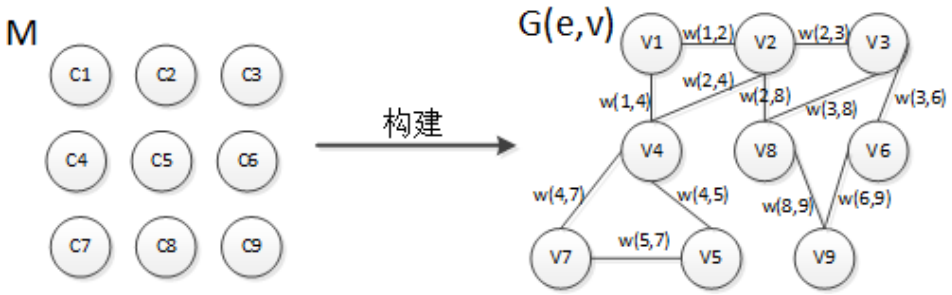

图 3-1 构建图阶段

构建图过程中图$G(E,V)$为加权无向图，图中每个顶点$v_i \in V$，对应于单体阶段中的每一个代码文件$c_i \in C$。每一个边$e_k \in E$有一个权重 w。

因为每个代码文件中都有标识符（变量名，方法名）来表示该文件具有的功能。因此我们可以从每个代码文件中抽取方法名来表示该文件。语义耦合策略就是使用表示该文件的标识符作为 tf-idf 的输入，计算出一个表示该文件的向量 X，之后再通过计算两个向量的余弦相似性来表示两个文件之间的耦合程度。具体步骤如下：

首先每个代码文件被标记，任意一个代码文件使用一组具有功能代表的词组$W_j = \{w_1, w_2, \dots, w_n\}$来表示，这些词组中我们会过滤掉停止词（的，在，也等）以及所有特殊词组。在划分过程中，两个代码文件c_i，c_j的词组列表$T = \{t_1, t_2, \dots, t_k\}$将会被建立，T 是由代表代码文件$c_i$的词组$W_i$以及代表代码文件$c_j$的词组$W_j$的并集组成。接下来是计算代表每个代码文件$c_i$的向量 X，向量 X 的维度是词组列表 T 的长度，向量的第 k 个元素的是词组列表 T 中第 k 个元素在代码文件词组中的 tf-idf 值表示。向量 X 中的元素x_k计算公式如下：

$$x_k = tf(t_k, W_i) * idf(t_k, W_{all}) \tag{3-1}$$

用同样的方法得到向量 V 中的元素，W_{all}表示两个词组的并集即：

$$W_{all} = \{W_i, W_j\} \tag{3-2}$$

词频$\mathrm{tf}(t_k, W_i)$表示t_k在W_i中出现的频率的计算公式如下：

$$\mathrm{tf}(t_k, W_i) = n_{k,i} / \sum_j n_{j,i} \tag{3-3}$$

其中$n_{k,i}$表示t_k在W_i中出现的次数，$\sum_j n_{j,i}$ 表示所有元素在W_i中出现的次数之和。

逆向文件频率$\mathrm{idf}(t_k, W_i)$表示一个词语普遍重要性的度量。某一特定词语的 idf，可以由文件总数目除以包含该词语文件的数目，再将得到的商取以 10 为底的对数得到：

$$\mathrm{idf}(t_k, W_{all}) = \log(|W_{all}| / n(t_k)) \tag{3-4}$$

$W_{all} = \{W_i, W_j\}$表示两个词组集合的集合，$\mathrm{n}(t_k)$表示包含词组t_k的文件数目。

根据以上方法可以计算出向量 X，V 中的每一个元素，通过求的两个向量的余弦值来表示两个代码文件之间语义耦合的程度。根据以上方法计算出所有代码文件之间的耦合程度作为建立的图中边的权重，如果耦合度值为 0 则表示两个定点之间没有连线，那么我们可以建立起相应的无向加权图，我们使用邻接矩阵 A 来表示建立的无向加权图，邻接矩阵中的元素表示对应两个节点之间的权重，矩阵如公式 3-5：

$$A = \begin{bmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,n} \end{bmatrix} \tag{3-5}$$

其中$a_{i,j}$表示节点v_i和节点v_j之间的边的权重。

建立无向加权图后，现在处于图阶段，由图阶段到微服务阶段，我们称为聚类阶段如图 3-2：

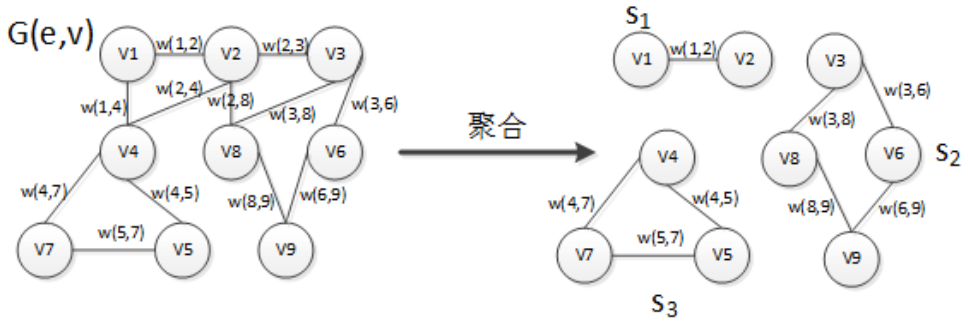

图 3-2 聚类阶段

在聚类过程中，我们使用经典的社区发现算法 GN 算法来实现微服务的划分，GN 算法基本思想是不断的删除图中具有相对于所有源节点的最大的边介数的边，然后，再重新计算图中剩余的边的相对于所有源节点的边介数，重复这个过程，直到图中所有边都被删除，使用模块性 Q 来衡量划分质量，当模块性 Q 函数最大时表示图中服务划分的最好，每个服务之间都保证了“高内聚，低耦合”原则。

在本文中，我们使用如下公式定义 Q 函数：

$$Q = (1/2M)\sum_j\left[\left(a_{ij}-k_i k_j/2M\right)\delta(\sigma_i,\sigma_j)\right] \tag{3-6}$$

公式中，a_{ij}为图的邻接矩阵的元素，如果 i 和 j 两节点相连，则a_{ij}为边的权重，否则等于 0，δ 为隶属函数，当节点 i 和 j 属于同一微服务时，隶属函数为 1，否则为 0；M=0.5，$\sum a_{ij}$为加权无向图中边的权重之和。k_i为节点的点权，对联通矩阵的第 i 行求和。

在该算法中最关键的是计算图中的边介数，本文中使用最短路径边介数方法来度量边介数，具体是指从某源节点 S 出发通过该边的最短路径的数目，对所有可能的源节点，重复做同样的计算，并将得到的相对于各个不同的源节点的边介数相加，所得的累加和为该边相对于所有源节点的边介数。

该算法的具体实现过程为：

第一步：忽略边的权重，以无权网络计算网络中所有连接边的边介数$B_{i,j}$；

$$B_{i,j} = \sum_k n_k \tag{3-7}$$

第二步：将边介数除以对应边的权重得到边权比$w_{i,j}$；

$$w_{i,j} = B_{i,j}/e_{i,j} \tag{3-8}$$

第三步：找到边权比最高的边将它移除，并计算图的模块性 Q 函数，在计算中当边权比最高的边由多条时，同时移除这些边，并将此时移除的边和 Q 值进行存储；

第四步：重复步骤（1）、（2），直到图中所有的边均被移除；

第五步：GN 算法划分结束后，取出 Q 值最大时的序号，在原始矩阵中依次去除截止到该次划分的边，得出最终连通矩阵，矩阵的值为权值。

3.3 微服务划分策略

3.3.1 微服务划分策略

为了保证微服务划分后服务粒度的合理性、服务的高可用性以及平台代码低冗余率，我们要求服务划分策略应该满足以下要求：

表 3-1 服务划分策略要求

名称	描述	优先级
可复用性	划分后的服务的复用率比较高	高
灵活独立	服务应该具有高内聚性，适当粒度的服务构件能使服务在各角度进行服务装配;通过松散耦合技术减少服务消费者和提供者间技术依赖性。	高
耦合性	划分后的服务之间的耦合度应该尽量小	高
可执行性	算法的实现是 java 语言或者是可以被 JVM 轻松调度的语言	中
性能	算法执行的时间	中
简单性	算法的机制和参数的理解程度	低

表 3-2 是通过参考 the Computer Science Review[58] and the Physical Review[59] 的聚类算法得到的算法的对比：

表 3-2 聚类算法对比表

名称	描述	执行	性能（N：节点，E：边）	测试结果	确定性
GN 算法	一种基于边介数的加权无向图聚类算法	可以使用 JAVA 语言实现	$O(N^3)$	优	优
MCL (Markov Cluster Algorithm)	一种加权无向图聚类算法	可以使用 R 语言和 JAVA 语言实现	$O(k^2N)$	差	差
ELP (Epidemic Label Propagation)	一种加权无向图（有向图）聚类算法	可以使用 JAVA 语言实现	$O(EN)$	优	差

由表 3-2 可得出 GN 算法在测试结果和确定性中优于 MCL 算法和 ELP 算法。因此我们使用 GN 算法来实现微服务的划分，我们的算法伪代码如下：

算法 1 微服务划分算法

```
根据公式得到每条边的权重 eij
根据公式得到所有连接边的边介数 Bij
边介数除以权重得到边权比
while edges 不为空 do
    if  e ∈ edges的边权比最高
移除 e 并将 e 存储起来
        if  e 有多条
            将具有最高边权比的多条 e 移除并存储
            计算此时的图的模块性 Q 值并存储
end while
取得 Q 的最大值Q_max以及 Q 值对应的边
return Q_max和 e
end
```

3.4 本章小结

本章首先分析了传统服务划分算法在平台中的不足，然后详细叙述了微服务划分模型的建立的理论基础以及建立的过程，最后介绍了在微服务划分模型建立过程中使用的微服务划分策略的思路以及伪代码的实现。

第四章 微服务平台中性能感知的微服务选择策略

在我们的微服务平台中，微服务路径选择的基本工作流中，首先需要根据应用需求创建微服务实例，之后应用系统发送处理任务请求到平台。当平台接收到处理任务时，他首先分析任务的结构，包括每个子任务的类型，子任务之间的关系，每个子任务之间的输入输出信息，为了优化任务处理效率，平台需要提供一个最优的微服务路径。本章首先分析了当前服务选择算法的不足，例如没有考虑在线服务的细粒度的在线处理能力和任务的特征，然后综合考虑微服务实例的处理能力，处理任务的特征以及微服务实例之间的数据传输条件建立了细粒度的性能预测模型，基于提出的性能预测模型，提出一种新颖的性能感知的服务路径选择算法。

4.1 云平台中服务路径选择策略

4.1.1 传统的服务路径选择算法

基于微服务架构的云平台提供了各种各样的微服务，每种微服务能够提供一个具体的数据处理功能，例如数据收集，数据传输，数据特征提取以及数据分类，并且相同功能的微服务实例能够被创建来响应网络应用中实时的服务请求。在执行过程中，从微服务实例池中选择的并且顺序执行的微服务实例组成一条微服务路径。然而，在运行时，不同的微服务实例有不同的资源配置和运行时处理能力，能够提供不同的 Qos。因此，制定高效的服务选择策略对于微服务平台性能有重要的影响。

目前，随着网络服务数量的逐渐增加，许多服务选择方法已经兴起。最近的几篇文献[4][5][6]更侧重于支持基于 Qos 的服务选择和组合。但是它们只考虑静态的 Qos 特征，例如响应性、可用性、和吞吐量，并没有考虑选择的服务实例运行时的特征。应用中的一个子任务执行完成后，之后的服务实例的 Qos 会随着时间动态改变。因此在执行大规模视频任务期间，一个预定义的最优的服务组合可能是无效的。

另外，还有一些动态的自适应的算法提出优化服务组合整体的 Qos[7][8]，文献[11][12]提出了云计算平台中性能感知的服务选择和组合方案。这些方法选择合适的组件服务来创建最优的服务组合，并且能够根据服务提供平台中的各种改变动态的改变最优的服务组合。但是这些方法没有考虑视频处理任务细粒度特征，这些特征对视频任务总的执行时间和服务资源的性能具有重要的影响。

4.1.2 解决方案

为了优化微服务路径选择来提高任务处理效率，在进行微服务选择时不仅需要考虑微服务实例的在线的处理能力，还需综合考虑任务的特征以及微服务实例之间的传输条件，这就需要制定高效的服务选择策略。另外一方面，为了保证服务路径总是最优，我们需要自适应的更新的服务选择，即能够动态更新服务路径。

在基于微服务架构的云平台中我们提出了一种新颖的性能感知的服务路径选择方法（PSPAS）。首先为了优化服务路径，我们提出了一种微服务实例的性能模型，主要包括数据处理时间模型和时间传输时间模型。文中通过回归分析预测技术来预测每个子任务中微服务实例的执行时间，用通用的表达式来表示时间传输模型；文中性能预测模型，综合考虑微服务实例的实时的处理能力，任务的特征和微服务实例之间的数据传输条件。然后基于已经建立的性能预测模型，我们提出一种性能感知的服务路径选择策略（PSPAS）。主要包括两个阶段：初始化路径选择阶段，基于性能预测模型，构建有向加权图，使用 Viterbi 算法构建最优服务路径；自适应服务路径更新阶段，在任务执行过程中，基于路径搜索空间缩减原理实时更新服务路径。

本文提出的性能感知的服务路径选择策略在进行服务路径选择时充分考虑了微服务实例的实时的处理能力、以及任务的特征和微服务实例间的传输条件，并且可以自适应的动态更新服务路径，保证了高效的应用执行效率。

4.2 微服务实例性能预测模型

在基于微服务的云平台中，任务的核心工作流程可以描述为一个任务处理流水线，其包括多个处理子任务，例如视频预处理、目标分割、目标的形状特征和纹理特征以及基于机器学习的目标分类。每一个处理子任务可以从一组具有相同功能的微服务实例中选择一个最优的微服务实例，整个应用处理流水线可以看成由选择的微服务实例顺序执行完成的。

假设 P 是一个视频应用的处理流水线，能够被分成 n 个顺序执行的子务，$P = \{P_i\}(i = 1,2, \dots, n)$。我们定义微服务类集合$S = \{S_i\}(i = 1,2, \dots, n)$。每个微服务类 Si 由所有的微服务实例$\{s_{ij}\}(j = 1,2, \dots, j_i, j_i$代表微服务类$S_i$中微服务实例的个数$)$组成的，在每个微服务类中的微服务实例具有相同的功能，但是具有不同的执行效率。每一个子任务P_i由微服务类S_i中的一个微服务实例$\{s_{i.}\}$执行。子任务P_i的执行时间T_i定义为：

$$T_i = T_i^c + T_i^s \tag{4-1}$$

其中T_i^c是微服务实例$\{s_{i.}\}$数据处理的执行时间，T_i^s是上行微服务实例$\{s_{i-1.}\}$到下行微服务实例$\{s_{i.}\}$的数据传输时间。

我们定义微服务路径 SP，SP 是顺序的微服务集合$\{s_{1.}，s_{2.}，\ldots，s_{n.}\}$，其中$s_{i.}$是微服务类$S_i$中的微服务实例。服务路径 SP 决定了视频任务 P 的执行时间。因此，视频任务 P 总的执行时间为：

$$T = \sum_{i=1}^{n} T_i \tag{4-2}$$

对于视频处理任务 P，我们的目标是从所有合理的微服务路径中选择最优的微服务路径，能够最小化视频处理任务 P 的执行时间 T。

为了最优化服务路径选择，我们需要量化每个微服务实例s_{ij}即子任务P_i的执行时间T_{ij}。根据公式(4-1)可以得出T_{ij}为数据执行时间T_{ij}^c与数据传输时间T_{ij}^s之和。

数据执行时间T_{ij}^c由数据处理时间模型得到。每一个微服务实例的数据的执行时间T_{ij}^c受多种因素影响，例如任务的特征、微服务平台的当前资源状况。本文中我们使用在线回归方法来训练得到每个微服务实例的数据执行时间预测模型。另外，我们的预测模型不仅考虑了微服务实例历史资源状态和当前的资源状态，也综合考虑了视频任务特征对数据执行时间的影响。

假设在微服务实例s_{ij}上运行的视频子任务P_i由向量$\mathrm{X} = \left(x_{1,}\ \ldots,\ x_m\right) \in R^m$表示，$x_i$为子任务的特征值，m 表示预测模型中特征值的数量。模型的特征值包括视频任务的信息，例如视频的分辨率，视频文件大小，以及微服务资源的描述，例如 CPU 的核数、CPU 的时钟频率、CPU 的占有率、内存的使用率、内存带宽。在我们的回归预测模型中，为了获得一个更好的线性回归模型，我们对每个特征值取log_2。我们的回归函数如下：

$$f(W，X) = \sum_{l=1}^{m} w_l\ log_2 x_l \tag{4-3}$$

其中$\mathrm{W} = (w_0，w_1 \ldots w_m)$是模型的参数向量。在视频任务处理流水线中，每一个子任务（不包括第一个子任务）的输入都是上一个子任务的输出而且已经不是原始的视频文件。然而原始的视频文件对于每个子任务的执行时间有直接的影响，所以线性回归函数可以应用到任意一个子任务的微服务实例。

我们用方程作为损失函数来衡量预测值和真实值的偏差。为了防止模型的过度拟合，我们使用l_2正则化来作为损失参数，线性回归问题描述为：

$$\underset{W}{\operatorname{argmin}} \sum \left(f(W，X) - \log_2 t^{ac}\right)^2 + \lambda \|W\|_2 \tag{4-4}$$

其中t^{ac}是子任务的实际执行时间，λ是回归参数。基于从云平台中训练的数据集，回归模型通过 NAG 算法[17]训练得到。参数向量 W 训练得到后，新的执行时间的log_2值能够得到。最后我们可以根据微服务实例相应的回归模型预测出每个微服务实例的执行时间T_{ij}^c。

数据传输时间T_{ij}^{s}由数据传输时间模型得到。为了得到数据传输时间模型的表达式，我们定义了一个源子任务P_0和目的子任务P_{n+1}分别代表整个视频处理流水线的开始子任务和结束子任务。我们定义子任务集合 P←P∪$\{P_0，\ P_{n+1}\}$。另外我们定义了源微服务实例$s_{0,\ 0}$和目的微服务实例$s_{n+1,\ 0}$分别代表所有的微服务路径中的开始节点和结束节点。

假设在微服务路径中，$s_{i-1,\ k} \in S_{i-1}$是微服务实例$s_{i,\ j}$的前一个微服务实例。那么从微服务实例$s_{i-1,\ k}$到微服务实例$s_{i,\ j}$数据传输时间定义为：

$$T_{ij}^{s}(k) = Vol_i / Nr_{ij}(k) \tag{4-5}$$

其中Vol_i表示从子任务P_{i-1}到子任务P_i总的数据传输量，$Nr_{ij}(k)$表示微服务实例$s_{i-1,\ k}$与$s_{i,\ j}$之间的数据传输率。

假设Vol_0是视频处理任务中的元数据。对于在子任务P_i，我们定义$\alpha_i(i = 1，\ \ldots，\ \mathrm{n}，\ \ \alpha_0 = 1，\ \alpha_{n+1} = 0)$为输出数据总量与输入数据总量的比值。所以我们能够得到：

$$Vol_i = \alpha_{i-1} * Vol_{i-1} * \alpha_i \tag{4-6}$$

输出数据总量与输入数据总量的比值α可以由离线实验得到，数据传输率$Nr_{ij}(k)$可以由专门的测量工具测量得到。

4.3 微服务路径初始选择策略

在分布式的流水线应用处理场景中，一个应用的最终完成时间是由每一个子任务完成的时间之和决定的，如果在微服务路径初始选择时，不考虑各个微服务实例的细粒度的资源状态以及任务特征，一方面选择的微服务实例可能不是当前最优的微服务实例，造成当前的路径不是当前的最优选择，另一方面也会导致整个应用不能选择出最优的选择，降低了应用的处理效率，因此制定合理的微服务路径初始选择策略对于提升整个微服务应用的处理效率至关重要。

上节本文结合微服务应用的特征、平台微服务实例细粒度特征以及微服务实例间网络带宽的特点构建了时间预测模型。本节将提出一种基于时间预测模型的微服务路径初始选择策略，该策略主要用于在微服务应用在初始化状态下如何选择当前最优的微服务路径。

基于上节提出的性能预测模型，我们能构建一个分层的有向加权图 G，如图 4-1 所示。

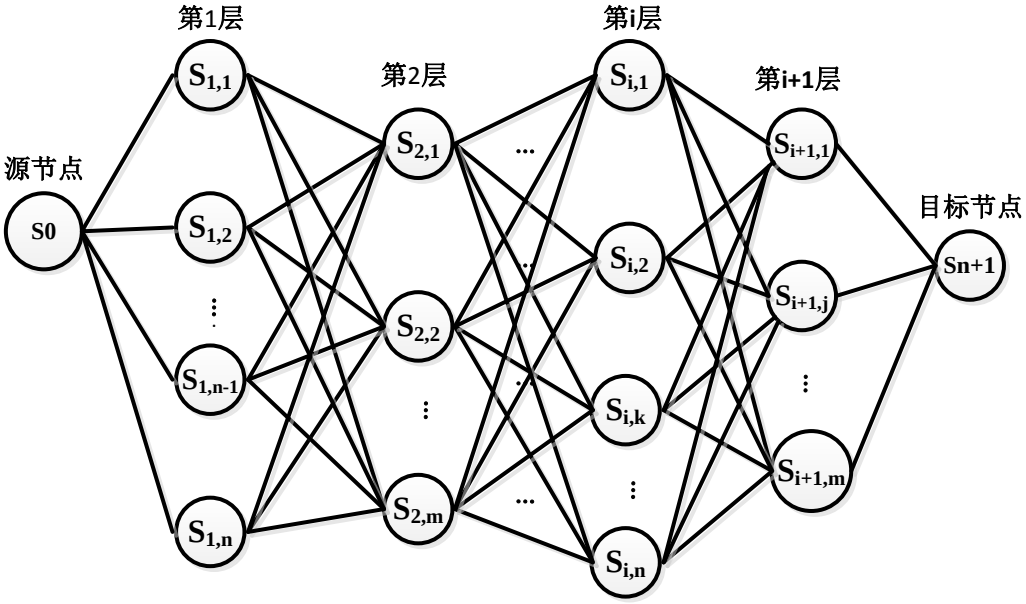

图 4-1 分层的有向加权图

在图 G 中，每一层代表一个微服务类S_i，每一个微服务类由多个节点组成，每一个节点代表一个微服务实例$s_{ij}(\in S_i)$。在图 G 中第S_i层的节点s_{ij}与第S_{i-1}层的节点$s_{i-1,\ k}$都对应有一个边$e_{ij}(k)$，边$e_{ij}(k)$是数据处理时间T_{ij}^{c}和数据传输时间$T_{ij}^{s}(k)$之和。微服务路径的权重是路径中所有边的权重之和。因此，我们的目标是在所有合理的微服务路径中选择具有最小路径权重的一组微服务组合 SP。

$$SP = \{s_{00},\ s_{1.},\ \ldots,\ s_{n.},\ s_{n+1,0}\} \tag{4-7}$$

综上所述这是一个经典的单源点最短路径问题，这里有许多经典的算法，例如迪杰斯特拉算法、贝尔曼福特算法、维特比算法，在本文中，我们选择维特比算法来获得最短路径 SP。

4.4 微服务路径动态更新策略

上一节介绍了基于性能预测模型的微服务路径选择的初始化策略，对于给定的分层加权有向图 G，上文介绍的最短路径算法就能得到最优服务路径 SP。然而，当一个子任务P_i由微服务实例$s_{i.}(i = 1,2,\ldots,n-1)$完成，下一个微服务实例$s_{i.}(i = l+1,\ldots,n)$的资源状态或者服务处理能力已经改变，这就意味着随着微服务应用的子任务的执行完成图 G 中的权重会发生改变。因此，初始化选择的最优的微服务路径 SP 就不是当前最优的路径。为了解决这个问题，最直接的找出当前的最优的服务路径的方法就是在每一个子任务执行完成后重复的调用以上

最短路径算法。但是这个解决方法有很大的计算时间代价以及对于有许多微服务实例的微服务平台的流水线任务是不合适的。

在本节中，我们提出了一个自适应的服务路径更新算法，该算法在在线执行微服务路径选择之前减小服务路径搜索空间，提高微服务选择的执行效率，得到当前最优的服务路径。

4.4.1 相关变量定义

首先我们为每一个子任务定义时间率β：

$$\beta = T_i^c / T_i^s \tag{4-7}$$

计算主导型子任务的时间率阈值β_c，传输主导型子任务的时间率阈值β_s，β_c，β_s的值通常是实验值，是在微服务平台中通过实验得到的。

计算主导型子任务（CDS）：是指在微服务任务流水线中，该子任务的时间率$\beta \geq \beta_c$。

传输主导型子任务（TDS）：是指在微服务任务流水线中，该子任务的时间率$\beta \leq \beta_s$。

计算主导型子任务（CDS）的执行时间主要是由微服务实例的计算能力所影响，因此我们能忽略子任务的输入数据的传输时间对于服务路径更新的执行效率带来的影响。

因此，当子任务P_i在微服务平台中运行的时候，我们可以根据路径搜索空间缩减原则（PSSP）来缩减路径搜索空间来提高服务路径更新效率。

路径搜索空间缩减原则（PSSP）：

i．如果P_j是计算主导型子任务，本文在微服务集合S_j选择前 m 个具有最短数据处理时间的微服务实例作为路径选择中的微服务实例集合。

ii．如果P_j是传输主导型子任务，本文在微服务集合S_j中选择前 m 个具有最短数据传输时间的微服务实例作为路径选中的微服务实例集合。

iii．如果P_j既不是计算主导型子任务又不是传输主导型子任务，本文在微服务集合S_j中选择前 m 个具有最短的平均执行时间的微服务实例作为路径选择中的微服务实例集合。

在 PSSP 中，m 被定义为微服务路径搜索空间缩减参数，通常$m \ll$该子任务对应微服务实例的个数。本文中，我们定义$m = 3$。

4.4.2 微服务路径动态更新策略

根据上述的服务路径搜索空间缩减原则，可以得到一个简化的分层的有向带权子图G_i。在G_i中，源节点代表执行子任务P_{i-1}的微服务实例，$s_{n+1,0}$是尾节点，

代表一个微服务类的每一层中有 m 个节点。注意根据微服务平台的状态，图G_i的每一条边的权重也会更新。之后，我们根据维特比算法重现选择出当前图G_i最优路径。最后我们重复以上路径搜索空间缩减原则并且重新选择当前最优路径直到微服务应用执行完成。

4.4.3 微服务路径动态更新算法实现

上一节我们介绍了微服务路径动态更新算法，下表是算法的伪代码：

算法 2 性能感知的微服务路径选择算法

1. 基于构建的性能感知模型构建分层的带权有向图 G
2. 初始化$i = 1$
3. 使用维特比算法求出当前最优路径SP_i
4. While 应用 P 没有结束 do
5. 在最优路径SP_i中调用微服务实例$s_{i.}$执行子任务P_i
6. $i = i + 1$
7. 根据路径搜索空间缩减原则对于还没有执行完的子任务执行路径搜索空间缩减操作
8. 基于路径搜索空间缩减操作的结果重构当前的子图G_i
9. 使用维特比算法重新选择出当前最优路径SP_i
10. End while

4.5 本章小结

本章首先分析了服务选择算法的不足之处，然后根据微服务平台应用的特点建立性能预测模型，最后介绍了基于性能预测模型的服务路径选择策略以及伪代码的实现。

第五章 平台实现及测试

5.1 平台环境配置

5.1.1 平台硬件环境

本文中平台硬件部分由 10 台物理服务器组成，10 台服务器分别部署在两个机架中，每个机架中有 5 台服务器，两个机架中的交换机都支持 1000Mbit/s。为了保证集群的高可用以及容错性，控制节点配置为 2 个，分别命名为 controller1 和 controller2，控制节点用于资源调度、微服务管理、服务路径选择等核心功能，同时，为了更好地利用集群物理资源，将包括控制节点在内的全部物理节点配置为工作节点，除去两个控制节点，其他节点依次命名为 node1~node8，每一个节点配置用于支持微服务的 Docker 容器引擎。整个集群物理服务器配置如下：

表 5-1 物理服务器配置列表

类型	CPU 类型	CPU 数量	内存	服务器数量
1	6 core	2	32GB	4
2	10 core	2	32GB	2
3	8 core	2	32GB	2
4	8 core	2	64GB	2

通过 htop 来获取每个节点中 CPU 和内存的使用率，任意两个节点之间的网络传输带宽通过 iperf 获得。

5.1.2 平台软件配置

目前虽然 Docker 已经支持在各种操作系统环境例如 Window、Linux、Mac 等安装部署，但是考虑到性能以及为了兼容实验室内已有的容器云环境，本文选择在 Linux 操作系统环境下进行集群的软件配置和部署。因此，在每个物理节点上都统一安装了内核版本的 GNU/Linux 3.13.0-32-generic x86 64 的 Ubuntu14.04.1 LTS 操作系统以及版本为 1.11.1 的 Docker Engineer。整个平台的开发工作实在实验室的电脑上进行，以下是具体的软件开发环境：

（1）操作系统 Ubuntu14.04.3 LTS 桌面版；

（2）程序开发环境：Vim，JetBrains IDEA，Docker；

（3）计算机视觉库：OpenCV 2.4.9；

（4）数据库：Mysql

5.2 平台功能实现

5.2.1 微服务划分功能实现

本文采用 Kubernets 作为容器调度管理的基础环境，开发环境的 Docker 容器都有 Kubernets 来管理调度。Kubernets 基于插件化的模块功能实现机制为开发者提供了便利的接口，用户可以方便地集成自定义的算法和功能对其进行扩展。

开发者可以使用 Java 语言实现自定义的功能函数或者类，然后将对应的编译完成的文件打包成 Docker 镜像，将打包好的 Docker 镜像 Push 到镜像仓库，Jenkins 执行 Shell 脚本命令，从镜像仓库拉取镜像在 Kubernets 环境中创建 Pod 和 RC 将自定义的功能文件所在的容器在 Kubernets 上执行起来。

为了实现第三章提出的基于领域驱动设计思想的语义耦合的微服务划分策略，首先基于该微服务划分策略思想实现了名为 ExtractionMicroserviceOfSCS 的类，然后将对应的文件放置到 Kubernets 安装路径的 plugin/cmd/scheduler/algorithm /algorithmprodiver 目录下即可，Kubernets 启动后会最终调用 ExtractionMicroserviceOfSCS 类中的方法进行微服务划分。本文基于我们平台的特征，将微服务划分模块基于 Dockerfile 文件进行构建，生成用于进行微服务划分的功能镜像，并命名为 extractionMicoservice。用于进行 extractionMicroservice 镜像构建的 Dockerfile 代码如下：

```
FROM java:8
MAINTAINER yangning 15032801667@163.com
VOLUME /tmp
ADD *.jar extractionMicroservice.jar
RUN sh -c ‘touch extractionMicroservice.jar’
ENV JAVA_OPTS=””
CMD exec java $JAVA_OPT -Djava.security.egd=file:/dev/./urandom -jar
/extractionMicroservice.jar
```

Dockerfile 编写完成可以通过以下命令制作镜像并上传到镜像仓库。

```
docker build -t controller1:6000/extractionMicroservice //controller:6000 为服务的仓库地址
docker run -ti -rm controller:6000/extractionMicroservice //运行测试看看制作的镜像是否可用
docker push controller:6000/extractionMicroservice //将镜像上传到镜像仓库
```

5.2.2 服务路径选择模块实现

服务路径选择模块的主要功能是通过运行性能感知的服务路径选择算法（PSPAS）并得到最优的服务路径。关于 PSPAS 算法的具体实现已经在第四章给出了详细的说明，这里不做赘述。为了便于服务路径选择模块的部署，整个服务路径选择模块采用了 Dockerfile 文件进行构建生成对应的功能镜像命名为 servicePathSelection，其对应的 Dockerfile 文件的代码如下：

```
FROM ubuntu:latest
MAINTAINER yangning 15032801667@163.com
RUN mkdir /usr/src/servicePathSelection
COPY servicePathSelection.c    /usr/src/servicePathSelection
WORKDIR /usr/src/servicePathSelection
RUN gcc servicePathSelection.c
ADD src . #将服务路径选择模块源码文件拷贝到容器中的 build 目录
RUN rm -rf /build
CMD [“./servicePathSelecion.out”]
```

5.2.3 视频浓缩服务镜像实现

为了测试本文实现的基于领域驱动设计思想的语义耦合的微服务划分算法，本文基于实验室先前工作中实现的视频浓缩算法实现了针对视频浓缩服务请求的功能镜像。在用户请求视频浓缩服务时，微服务平台中的微服务管理模块将视频浓缩服务的源代码调度到服务划分模块，将该服务划分成响应的微服务，在此过程中服务划分模块和服务注册组件通信，完成服务注册功能。

由于我们先前工作中实现的视频浓缩算法的源代码是通过 Maven 进行构建的，因此需要在 Maven 项目的根目录下创建 Dockerfile 文件，然后以 Maven 的官方镜像作为基础镜像进行镜像的构建，同时为了减小镜像文件体积，提升从仓库拉取镜像的速度，在构建镜像的过程中需要将编译环境生成的中间文件通过系统命令进行删除。整个 Dockerfile 文件的代码如下：

```
FROM maven:3
MAINTAINER yangning 15032801667@163.com
RUN mkdir -p /build/input/output
WORKDIR /build
```

```
ENV TASK videoSynopsis.jar
ADD POM.xml .
ADD src src
RUN mvn package && mvn test
RUN cp target/$TASK / && rm -rf /build && rm -rf ~/.m2/*
VOLUME /output
CMD [“java”， “-jar”， “/videoSynopsis.jar”， “$@”]
```

Dockerfile 编写完成后通过以下脚本文件完成镜像构建和上传。

```
docker build -t controller1:6000/videoSynopsis
docker push controller1:6000/videoSynopsis
```

5.2.4 目标跟踪服务镜像实现

为了测试本文实现的基于领域驱动设计思想的语义耦合的微服务划分算法，需要另一个微服务应用结合视频浓缩服务来验证服务划分算法的性能，本文基于实验室之前工作中实现的目标跟踪算法实现了针对目标跟踪服务请求的功能镜像。与视频浓缩服务请求类似，当用户请求目标跟踪服务时，微服务平台中的微服务管理模块会将目标跟踪服务源代码调度到服务划分模块，并将划分好的微服务注册到服务注册组件。

与食品浓缩算法相同，目标跟踪算法的源代码也是通过 Maven 进行构建的，因此需要首先在 Maven 项目的根目录下创建 Dockerfile 文件，然后以 Maven 的官方镜像作为基础镜像进行镜像的构建，同时为了减小镜像文件体积，提升从仓库拉取镜像的速度，在构建镜像的过程中我们将编译环节生产的中间文件通过系统命令进行删除。整个 Dockerfile 文件的代码如下：

```
FROM maven:3
MAINTAINER yangning 15032801667@163.com
RUN mkdir -p /build/input/output
WORKDIR /build
ENV TASK objectTracking.jar
ADD POM.xml .
ADD src src
RUN mvn package && mvn test
RUN cp target/$TASK / && rm -rf /build && rm -rf ~/.m2/*
```

```
VOLUME /output
CMD [“java”， “-jar”， “/objectTracking.jar”， “$@”]
```

同样的 Dockerfile 文件编写完成后通过以下脚本文件完成镜像构建和上传。

```
docker build -t controller1:6000/objectTracking
docker build controller1:6000/objectTracking
```

5.3 平台功能与算法效果测试

本节主要基于 5.2.3 和 5.2.4 小节实现的视频浓缩服务功能和目标跟踪服务队平台性能以及算法效果进行验证。

5.3.1 平台功能验证

首先，为了验证平台中视频浓缩服务镜像和目标跟踪服务镜像的功能的有效性，本文准备了大约 10GB 的监控视频数据，这些视频数据均来自中国福州部署的监控视频系统，视频本身采用 H.264 进行编码，视频分辨率为 1920*1080，码率为 25fps，总时长为 10 分钟。将 1GB 的视频文件放置在 Controller 节点启动 Docker 计算引擎拉取视频浓缩服务镜像生成视频浓缩服务容器计算实例读取视频数据进行处理，最后生成的浓缩后的视频文件总大小约为 24MB，时长约 1 分钟。

如图所示，左图是我们原视频文件中的画面，可以观察到，每一个画面都存在大量的冗余信息，而右图是经过视频浓缩技术后的视频文件的画面，相比于原始视频文件的画面，经过视频浓缩技术处理的文件除去了大量的冗余信息，在很大程度上提高了视频中有效信息的密集程度。从而验证了平台中视频浓缩服务功能的有效性。

5.3.2 微服务划分算法效果验证试验

为了验证本文提出的微服务划分算法的效果，本文使用了原平台中两个服务，视频浓缩服务和目标跟踪服务，我们控制用户对两个服务的请求个数为 50~100，选取了 5 个有代表性的视频数据文件，每个视频数据文件的的数据大小以及视频的分辨率、帧率、时长都是不同的。表 5-1 是实验中数据集的详细信息：

表 5-2 视频数据集详细信息表

数据名称	视频数据大小	帧速率	时长	服务请求个数
Dataset1	56.5MB	25 帧/秒	10 分钟	50
Dataset2	512.6MB	25 帧/秒	63 分钟	50
Dataset3	1GB	35 帧/秒	92 分钟	85
Dataset4	1.5GB	35 帧/秒	101 分钟	100

我们同只考虑代码行数划分微服务的方法（LOC）进行对比。当用户发送服务请求时，微服务平台管理模块会将请求转发给微服务划分模块，微服务模块执行相应的算法将服务划分为相应的微服务并且在服务注册组件中注册。为了避免平台中其他不稳定因素对实验结果的影响，我们保证实验条件不变的情况下，将该实验在相同实验环境下执行了 10 次并将 10 次的结果的平均值作为实验结果的最终值。

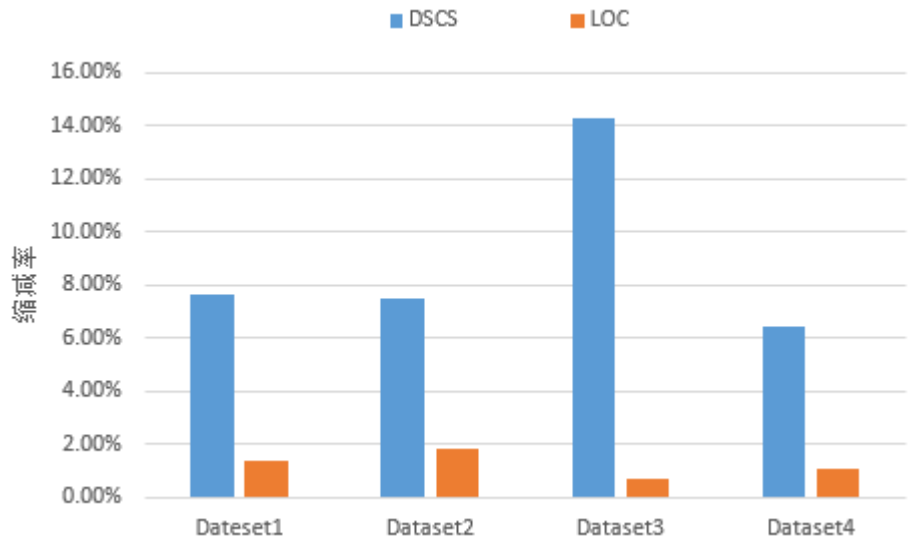

图 5-1 DSCS 和 LOC 算法的缩减率对比图

虽然在面向对象设计领域有明确已有的质量标准，但是评估微服务质量标准的研究还是很少。因此，本文使用自定义的标准来衡量算法的效果。我们使用代码缩减率（csr）和服务复用率（drr）评估算法的效果。以下是代码缩减率（csr）和服务复用率的计算公式：

$$\mathrm{csr}(M) = \big(codesize(M) - codesize(S_M)\big)/codesize(M)$$

其中M代表当前的单体架构平台中的服务，S_M代表单体架构平台中服务 M 划分之后的服务，codesize(.)表示服务的代码大小

$$\mathrm{drr}(s) = (num(app) - 1) * 100\%$$

其中$\mathrm{num}(app)$代表使用该服务 s 的应用的个数。

实验结果如图 5-1 所示。图中的横坐标代表不同的视频数据集合，纵坐标表示每一个数据集合对应的代码缩减率。可以看到，本文提出的微服务划分算法的代码缩减率是高于 LOC 算法的代码缩减率。

第二组实验中我们同样选取了相同的数据集，但是每一组数据的应用类型不同，第一组数据中同时执行两个应用，第二组数据中同时执行 3 个应用，第三组数据中同时执行 5 个应用，第四组数据中同时执行 10 个应用。我们同 LOC 算法进行了对比。同样为了避免其他不稳定因素影响，我们保证其他条件不变和实验环境不变，运行了 10 次并将 10 次计算结果的平均值作为每种服务划分算法下服务复用率的最终值。实验结果如图 5-2 所示。

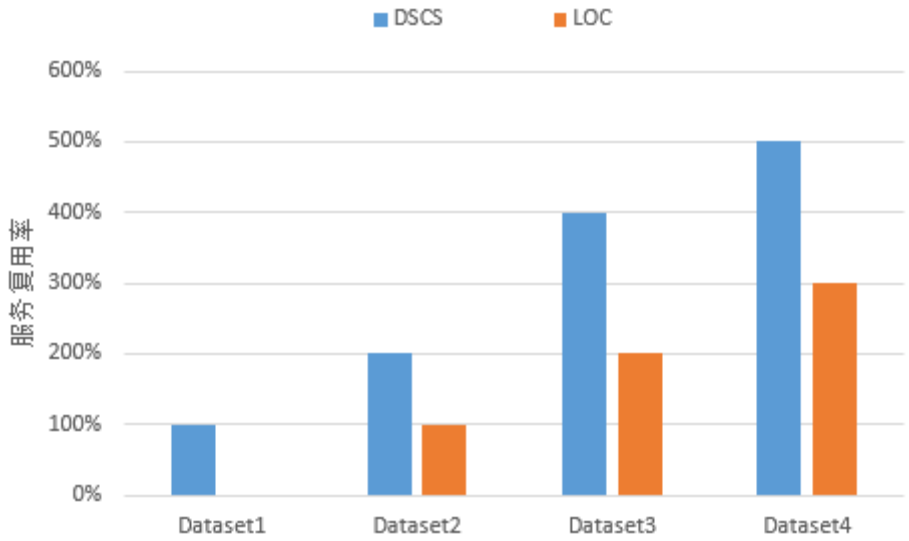

图 5-2 DSCS 和 LOC 算法的服务复用率对比图

图中横坐标表示不同的数据集，纵坐标表示每种数据集下的服务复用率。可以看出，本文提出的 DSCS 算法可以有效提高服务复用率。

5.3.3 性能预测模型准确性验证试验

为了验证性能预测模型的准确性，本文选择了的数据集包括 1000 个监控视频文件，该视频文件有不同的分辨率，不同的文件大小，这些视频数据大约是 10GB。我们将数据集分成两部分，第一部分包括 700 个视频文件，用于训练数据处理时间模型，第二部分包括 300 个视频文件，用于模型测试。目标跟踪服务包括 9 个子任务，每一个子任务对应一个微服务，每一个微服务由相应的微服务实例提供服务。子任务分别为 Data reading、Gray-scale processing、Gaussian blurring、Inter-frame difference processing、Contour extraction、SIFT feature extraction、feature matching、Tracking window drawing、Data writing。我们通过

第四章提出的数据处理时间模型来预测每个微服务的数据处理时间，并且比较预测结果和通过离线处理的实际执行的值。为了最小化我们微服务平台的不稳定性地影响，每一个子任务执行了 10 次，做准确性比较时，我们取每个子任务的平均处理时间。最后我们使用准确率和均方差（MSE）来评估我们数据处理时间模型的准确性。

均方差（MSE）的计算公式如下：

$$\mathrm{MSE}(\hat{\theta}) = \mathrm{E}\left[(\hat{\theta} - \theta)^2\right]$$

其中$\hat{\theta}$是每个子任务的预测时间，θ是每个子任务的实际执行时间，E[.]是求均值操作。结果如表 2 所示。

表 5-3 准确率和均方差比较

子任务名称	准确率	均方差（MSE）
Gray-scale processing	97.04%	0.0125
Gaussian blurring	96.43%	0.0067
Inter-frame difference processing	96.23%	0.0023
Contour extraction	96.10%	0.0196
SIFT feature extraction	93.73%	0.3211
Feature matching	97.16%	0.0460
Tracking window drawing	96.73%	0.0034

如表 2 中准确率和均方差的比较，子任务数据处理时间的平均的准确率是 96%，除了子任务 SIFT feature extraction 的准确率是 93.73%，每一个均方差（MSE）的值为都是在可接受范围内。因此可以得出我们的性能预测模型中的数据处理时间模型能够十分精确的预测微服务实例所需的数据处理时间。

5.3.4 服务路径选择算法效果验证试验

基于上一节验证的性能预测模型，本节我们将验证本文提出的性能感知的服务路径选择策略（PSPAS）的效果，我们和默认的最优的服务路径选择策略（OPTIMAL），最短路径选择策略（SPS），以及动态服务路径选择策略（DSPS）进行对比。最优的服务路径选择策略是通过离线任务执行而得到的最优的路径，在线执行时该方法不能用到；最短路径选择策略只是使用维特比算法得到微服务选择路径而没有考虑每一个微服务实例的处理能力动态改变；动态服务路径选择策略能够根据当微服务实例的当前状态来在线更新最优微服务路径，但是使用的时间预测模型中的特征值是历史性能记录。

为了验证本文提出的性能感知的服务路径选择策略（PSPAS）与 OPTIMAL 的性能最接近，且比 SPS，DSPS 性能更优，我们首先设置影响因子微服务实例

的个数为变量因素，当微服务实例的个数增加时，来观察四种服务路径选择策略的任务执行时间。

在验证过程中，本文使用 1GB 的视频数据集作为实验输入数据。目标跟踪服务包括 9 个子任务，每个子任务由相应的微服务类提供服务。首先，我们让每一个子任务的微服务实例的个数在[10，40]中随机选择，任务执行过程中，我们控制微服务实例的资源状态改变在 10%之内（保持轻微改变）。之后，我们通过四种服务路径选择策略执行目标跟踪任务。为了避免其他不稳定因素对实验的影响，我们保证实验在相同的实验环境下运行 10 次，并将 10 次任务执行时间的平均值作为最后结果。我们保证其他条件不变，改变每一个微服务类的个数，微服务类的区间设置在[41，70][71，100][101，130]，[131，160]和[161，190]范围内，对于每一个微服务类区间，和[10，40]区间执行相同的操作，最后记录实验数据。实验结果如图 5-1 所示：

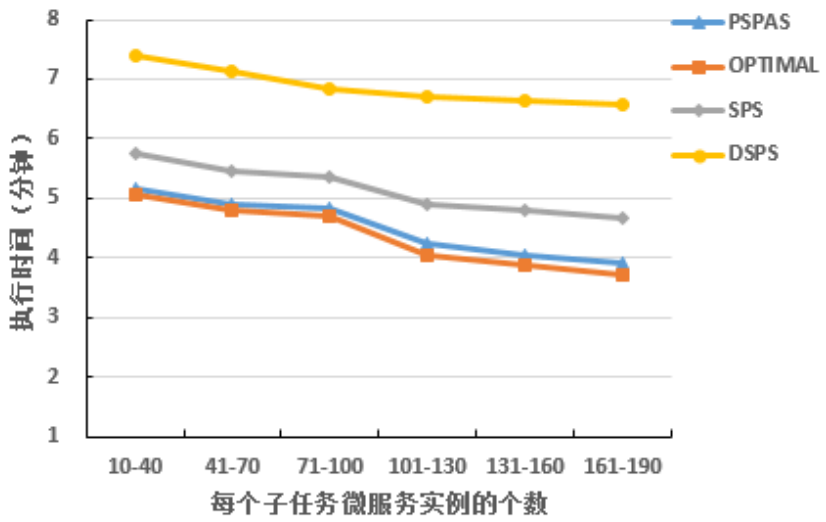

图 5-3 当微服务资源轻微抖动时执行时间变化图

从图 5-3 中我们可以看出我们的方法 PSPAS 的执行效率和 OPTIMAL 的执行效率是最接近的。然而 SPS 方法的执行时间远远大于 PSPAS，这是因为 SPS 策略只考虑了初始化的最优服务路径而没有考虑到在线任务执行过程中最优服务路径的改变。尽管 DSPS 能够冬天的更新服务路径，但是 DSPS 策略的执行效率是最差的。这说明性能预测模型的准确率对于最优微服务路径选择有很大影响。从图中我们可以得出，随着微服务实例个数的增加，每一种策略的执行时间是在减少，所以微服务路径选择策略具有良好的可扩展性。

为了和上一组实验做对比，本次验证中我们保证实验中的其他条件不变，只改变任务执行过程中，微服务实例资源状态抖动的剧烈程度，本组实验我们控制微服务实例资源抖动在 30%~60%之间。实验结果如图 5-4 所示：

图中横坐标代表微服务实例个数所在的区间，纵坐标表示整个任务的执行时间，单位为分钟。

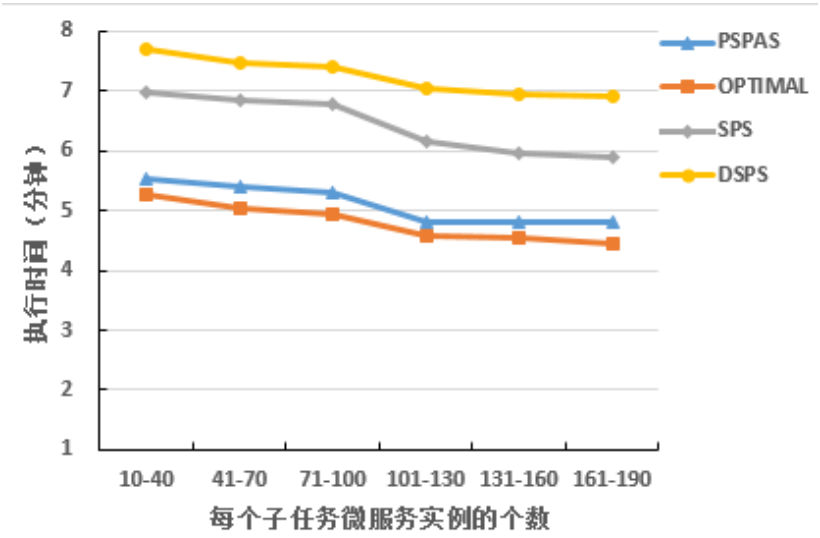

图 5-4 当微服务资源剧烈抖动时执行时间变化图

从图 5-4 中可以得出，和图 5-3 比较，可以看出 PSPAS 策略与 OPTIMAL 策略的性能曲线之间有了轻微变化，这是因为微服务平台动态的特性会降低 PSPAS 的执行效率。同样的，由于在动态环境中，静态的微服务路径选择策略会受到严重影响，所以 SPS 策略的性能也降低了。然而我们的 PSPAS 策略仍然优于 SPS 策略和 DSPS 策略。

为了验证 PSPAS 算法的有效性，我们验证了应用中子任务的个数对微服务路径选择算法性能的影响。实验中，我们使用了目标跟踪中子任务构建了不同的视频服务，每一个视频服务的子任务个数不同。我们设置每一类服务的实例个数为 90。同样的，每一个视频服务执行了 10 次并且将执行时间的平均值作为实验结果。实验结果如图 5-3 所示：

图中横坐标代表不同的视频任务，每个任务有不同的子任务数，纵坐标表示整个视频任务执行的时间，单位为分钟。

从图 5-5 可以看出，随着子任务个数的增加，每一个微服务路径选择策略的执行时间都会增加。明显的，我们的 PSPAS 策略和 OPTIMAL 策略的性能最接近并且优于其他两个服务路径选择策略，而且随着子任务个数的增加 PSPAS 策略与 SPS 策略和 DSPS 策略之间的时间差也逐渐增加，所以当执行大型微服务应用时，我们的服务路径选择策略的性能是最优的。

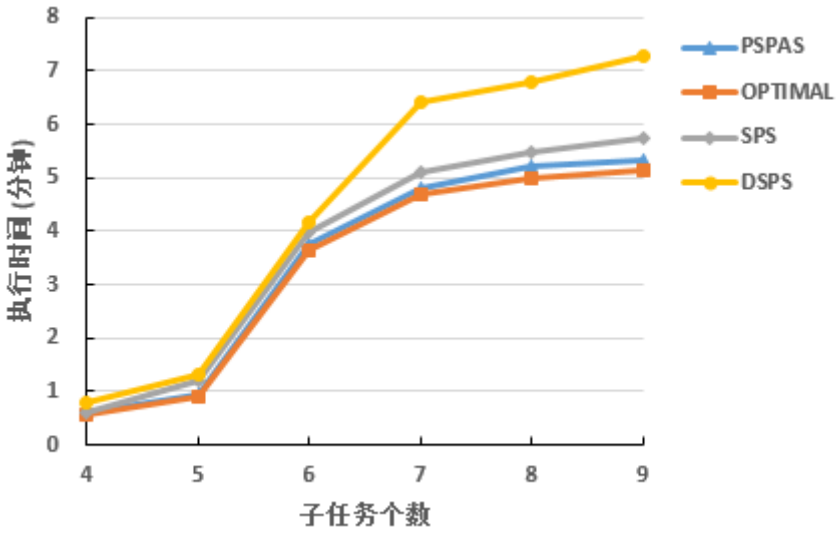

图 5-5 子任务个数改变时执行时间变化图

5.4 本章小结

本章首先详细介绍了平台的软硬件的配置环境，以及平台中的微服务划分模块和服务路径选择模块的具体实现，然后介绍了视频浓缩服务镜像和目标追踪服务镜像的具体实现，之后对平台的基本功能做了基本验证，最后通过实验验证了本文提出的微服务划分策略、服务路径选择策略的有效性以及性能预测模型的准确性，实验表明我们提出的微服务划分策略能够有效提升平台中服务的服用率减少代码的冗余，基于性能预测模型的服务路径选择策略，相比于其他服务路径选择策略，可以有效提高服务执行效率，降低服务执行时间。

第六章 总结与展望

6.1 总结

随着业务系统的扩张和业务需求的不断变更，业务系统的功能逐渐复杂，规模逐渐庞大，架构模式较为单一的应用架构已经不能满足业务的需求，因此微服务架构成为今年来解决以上需求的主流方案。然而目前对于微服务架构，业界尚无明确定义，而且围绕微服务的工作是有限的。如何划分微服务，制定划分策略，满足微服务内部的高内聚性和微服务之间的低耦合性，是我们面临的主要问题。对于基于微服务架构的分布式处理平台，微服务划分完成，如何组合微服务实例，制定相应的微服务路径选择策略，对于提高应用的执行效率尤其重要，也是目前研究的热点问题。围绕这两个目标，本文主要研究内容有以下四个部分：

（1）基于领域驱动设计思想的语义耦合的微服务划分策略

传统的服务划分方法中并没有根据平台需求以及平台中应用之间功能的相关性来综合考虑，这将导致服务划分后的结果是服务之间的耦合性比较差，代码复用率比较低，平台中代码的冗余度比较高。为了提高代码的复用率，我们提出了语义耦合的服务划分策略，该策略通过分析代码之间的语义耦合程度，来表示两个代码的耦合程度，将耦合度作为权重构建无向有权图，再通过社区发现算法 GN 算法来找到最好的划分结果。通过实验验证了该方法可以将应用有效划分成微服务，并且能够提高服务的复用率，减少平台的代码量。

（2）微服务实例性能预测模型

本文通过分析当前主流的服务选择算法，发现当前的服务选择算法只考虑静态的 Qos 特征，例如响应性、可用性、和吞吐量，并没有考虑选择的服务实例运行时的特征，而且也没有考虑任务本身的特征，因此本文提出了一种基于机器学习的时间预测模型，该模型包括数据执行时间模型和数据传输时间模型，数据执行时间模型结合微服务实例运行时特征以及数据本身的特征，例如 CPU 利用率、内存利用率、数据分辨率、帧率等，数据传输时间模型结合了微服务实例之间的传输条件，例如数据大小、网络带宽，在基于大量训练数据的基础上，通过学习得到微服务实例的时间预测模型。

（3）性能感知的微服务路径选择策略

传统的微服务路径选择策略并没有同时考虑微服务实例运行时特征以及任务的特征。本文基于所提出的微服务性能预测模型，研究并实现了一种性能感知

的服务路径选择策略 PSPAS。该策略包括两个阶段：第一个阶段是初始化微服务路径，在此阶段本文使用最短路径算法构建最佳微服务路径；第二个阶段是微服务路径动态自适应更新阶段，在应用执行过程中基于路径搜索空间缩减原则动态更新最短路径，该策略提高了微服务应用的处理效率。

（4）基于微服务技术的验证平台

传统的分布式处理云平台系统采用的是单体架构，系统规模会随着业务量的进一步增加而急剧地膨胀，进而产生架构臃肿、业务逻辑复杂、数据流向复杂等一系列问题。这些问题会给整个系统的开发、维护、部署以及后期的升级带来巨大的困难。微服务架构成为近年来云计算领域的热门技术。本文采用微服务技术构建分布式云平台，并按照本文设计的基于领域驱动设计思想的语义耦合的微服务划分策略和性能感知的服务路径选择策略（PSPAS）扩展实现了相关功能模块。最后通过视频浓缩服务验证了平台高效的任务处理效率。

6.2 展望

本文首先通过分析传统的微服务划分思想以及微服务划分策略的不足，结合微服务的设计原则以及平台的应用特性，提出了基于领域驱动设计思想的语义耦合的微服务划分算法，其次分析传统服务路径选择策略的不足，结合微服务实例的运行时特征以及微服务任务的特征，提出了性能感知的服务路径选择策略，最后通过视频浓缩服务对所述的微服务划分策略进行验证，可有效提高代码的复用率，以及减少平台代码冗余率；通过复杂的微服务应用对所述的服务选择策略进行验证，可有效提高平台的执行效率，具有一定的实用性。但是本文工作还存在一些可以进一步研究的方向：

（1）微服务实例性能预测模型。目前的微服务实例性能预测模型中的数据执行时间模型是使用机器学习中线性回归的方法来训练得到的，数据传输时间模型是直接使用公式计算得到。对于现在非线性机器学习模型，数据越多，模型改进越多，训练越准确。本文未来考虑使用更多或者更高质量的数据来优化数据执行时间模型，进一步提高模型的准确性。

（2）探索不同应用下微服务平台性能优化问题。本文主要关注基于微服务架构的分布式处理云平台的性能优化问题。本文提出的微服务划分策略结合了微服务划分原则以及我们微服务平台中应用的特征，然而我们需要研究更多类型的微服务平台的微服务划分问题，并进一步实现更通用的微服务平台的服务划分策略。

（3）更丰富的微服务应用。本文主要通过九个子任务的应用来验证微服务平台中提出的性能感知的服务路径选择算法的有效性，后期应该选择更大型的应用，远远超过 9 个子任务的微服务应用来优化算法。

参考文献

[1] Fowler M. Microservices: a definition of this new architectural term. http://martinfowler.com/articles/micro-services.html, 2014

[2] Thones J. Microservices. IEEE Software, vol. 32, no. 1, 2015, pp. 116–116.

[3] Mazlami G, Cito J, Leitner P. Extraction of Microservices from Monolithic Software Architectures. IEEE 24th International Conference on Web Services, 2017, pp. 524-531.

[4] Newman S. Building Microservices. O'Reilly Media, 2015.

[5] Marcus A, Maletic J I. Identification of high-level concept clones in source code. Proceedings 16th Annual International Conference on Automated Software Engineering (ASE 2001), 2001, pp: 107-114.

[6] Poshyvanyk D, Marcus A. The conceptual coupling metrics for object-oriented systems. ICSM, vol. 6, 2006, pp. 469–478.

[7] Alrifai M, Skoutas D, Risse T. Selecting Skyline Services for QoS-based Web Service Composition. WWW, 2010, pp: 11-20.

[8] Deng S G, Wu H Y, Hu D, et al. Service Selection for Composition with QoS Correlations. IEEE Transactions on Service Computing, 2016, pp: 291-303.

[9] Saleem M S, Ding C, Liu X M, et al. Personalized Decision Making for QoS-based Service Selection. IEE International Conference on Web Services, 2014, pp: 17-24.

[10] Tan T H, Chen M, Liu Y, et al. Automated Runtime Recovery for QoS-based Service Composition. WWW, 2014, pp: 563-574.

[11] Wang H, Wu Q, Chen X, et al. Adaptive and Dynamic Scervice Composition via Multiagent Reinforcement Learning. IEEE International Conference on Web Services, 2014, pp: 447-454.

[12] Peng S, Wang H, Yu Q. Estimation of Distribution with Restricted Boltzmann Machine for Adaptive Service Composition. IEEE International Conference on Web Services, 2017, pp: 114-121.

[13] Gysel M, Kolbener L, Giersche W. Service cutter: A systematic approach to service decomposition. European Conference on Service-Oriented and Cloud Computing. Springer, 2016, pp. 185–200.

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

[15] Liu Y, Yang Y. Semantic Web Service Discovery Based on Text Clustering and Concept Similarity. Computer Science, 2013, pp: 211-214.

[16] Cheng Z H, Huang Z. Optimization of GN algorithm based on DNA computation. IEEE International Conference on Computer and Communications, 2016, pp: 1303-1308.

[17] Matsuba H, Joshi K, Hiltunen M, et al. Airfoil: A Topology Aware Distributed Load Balancing Service. IEEE Cloud, 2015, pp: 325-332.

[18] Lee K, Yoon H, Park S. A Service Path Selection and Adaptation Algorithm in Service-Oriented Network Virtualization Architecture. IEEE ICPADS, 2013, pp: 516-521.

[19] Bailey S E, Godbole S S, Knutson C D. A Decade of Conway's Law: A Literature Review from 2003-2012. International Workshop on Replication in Empirical Software Engineering Research, 2013, pp: 1-14.

[20] Zhang Y Y, Jiao J X. An associative classification-based recommendation system for personalization in B2C e-commerce applications. Expert Systems With Applications, 2006, 33(2).

[21] Strasser T, Rooker M, Ebenhofer G, et al. Multi-domain model-driven design of industrial automation and control systems. IEEE International Conference on Emerging Technologies and Factory Automation, 2008, pp: 1067-1071.

[22] Jive J. Jdon Framework. http://www.jdon.com/jdonframework/.

[23] Fowler M. Inversion of control containers and the dependency injection pattern. 2004, pp: 3-4.

[24] 王辛. 软件调试中基于事件模型的动态信息处理方法及工具. 西安电子科技大学, 2012, pp: 2-3.

[25] 张佳强. 基于领域模型的信息管理系统的研究与应用. 江南大学, 2009, pp: 34-38.

[26] 张伟, 梅宏. 一种面向特征的领域模型及其建模过程. 软件学报, 2003, pp: 1345-1356.

[27] Van D A, Klint P. Domain-specific language design requires feature descriptions. Journal of Computing and Information Technology, 2002, pp: 1-17.

[28] Fitzgerald S. State Machine Design, Persistence and Code Generation using a Visual Workbench, Event Sourcing and CQRS, 2012, pp: 12-15.

[29] Hintjens P, Zero M Q. Messaging for Many Applications. O'Reilly, 2013, pp: 3-4.

[30] Goldston R L, Son J Y. Similarity. Psychological Review, 2004, pp: 254-278.

[31] Li M, Chen X, Xin M L, et al. The Simility Metric. IEEE Transactions on Information Theory, 2003, pp: 863-872.

[32] 邱明. 语义相似性度量及其在设计管理系统中的应用. 浙江大学, 2006.

[33] Osgood C E. The nature and measurement of meaning. Psychological Bulletin, 1952, pp: 197-237.

[34] Lee MD. Algorithms for Representing Similarity Data. 1999.

[35] Landauer T K, Dumais S T C. Solution to Plato's Problem: The Latent Semantic Analysis Theory of Acquisition, Induction and Representation of Knowledge. Psychological Review, 1997.

[36] Tversky A. Features of Similarity. Psychological Review, 1977, pp: 327-352.

[37] Santini S, Jain R. Similarity Measures. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1999, pp: 871-883.

[38] Canfora G, M.D.P, Esposito R, et al. An Approach for QoS aware Service Composition based on Genetic Algorithm. GECCO, 2005.

[39] Canfora G, et al. Service Composition (re)Binding driven by Application-specific QoS. Internation Conference on Service Oriented Computing, 2006, pp: 141-152.

[40] Zeng L Z, Benatallah B, Ngu A H H, et al. QoS-Aware Middleware for Web Services Composition. IEEE Transaction on Software Engineering, 2004, pp: 311-327.

[41] Yu T, Lin K J. Service Selection Algorithm for Web Services with End-to-end QoS Constraints. IEEE International Conference on E-Commerce Technology, 2004, pp: 129-136.

[42] Pistore M, Marconi A, Bertoli P. Automated Composition of Web Service by Planning at the Knowledge Level. IJCAI, 2005, pp: 1252-1259.

[43] Doshi P, Goodwin R, Akkiraju R, et al. Dynamic Workflow Composition Using Markov Decision Processes. International Journal of Web Services Research. 2005, pp: 1-17.

[44] Gao A Q, Yang D Q, Tang S W, et al. Web Service Composition Using Markov Decision Processes. Internation Conference on Web-Age Information Management, 2005, pp: 308-319.

[45] Gysel M, Kolbener L, Giersche W. Service Cutter: A Systematic Approach to Service Decomposition. International Federation for Information Processing, 2016, pp: 185-200.

[46] Tatsubori M, Takahashi K. Decomposition and Abstraction of Web Applications for Web Service Extraction and Composition. IEEE International Conference on Web Services, 2006, pp: 859-868.

[47] Jiang B, Ye L Y, Wang J L, et al. A Semantic-based Approach to Service Clustering from Service Documents. IEEE International Conference on Services Computing, 2017, pp: 265-272.

[48] Joselyne M I, Mukasa D, Kanagwa B, et al. Partitioning Microservices: A Domain Engineering Approach. IEEE Symposium on Software Engineering in Africa, 2018, pp: 43-49.

[49] Wang J, Zhang N, Zeng C, et al. Towards Services Discovery based on Service Goal Extraction and Recommendation. IEEE International Conference on Services Computing, 2013, pp: 65-72.

[50] Erradi A, Tosic V, Maheshwari P. MASC - .NET-Based Middleware for Adaptive Composite Web Services. IEEE International Conference on Web Services, 2007, pp. 727-734.

[51] Mohr F, Jungmann A, Bvning H K. Automated Online Service Composition. IEEE International Conference on Services Computing, 2015, pp. 57-64.

[52] Narayanan S, McIlraith S, Simulation, verification and automated composition of web services. WWW, 2002, pp. 77-88.

[53] Hossain M S, Moniruzzaman M, Muhammad G, et al. Big data-driven service composition using parallel clustered particle swarm optimization in mobile environment. IEEE Transaction Services Computing, vol. 9, no. 5, 2016, pp. 806-817.

[54] Tan T H, Chen M, Andre E, et al. Automated runtime recovery for qos-based service composition. WWW, 2014, pp. 563-574.

[55] Wang H, Wu Q, Chen X, et al. Adaptive and dynamic service composition via multiagent reinforcement learning. IEEE International Conference on Web Services, 2014, pp. 447-454.

[56] Peng S, Wang H, Yu Q. Estimation of Distribution with Restricted Boltzmann Machine for Adaptive Service Composition. IEEE International Conference on Web Services, 2017, pp. 114-121.

[57] Saleem M S, Ding C, Liu X, et al. Personalized Decision Making for QoS-based Service Selection. IEEE International Conference on Web Services, 2014, pp. 17-24.

[58] Schaeffer S E. Graph clustering. Computer Science Review 1.1, 2007, pp. 27-64.

[59] Kruchten P B. The 4+ 1 view model of architecture. IEEE Software, 1995, pp. 42–50.

致谢

时光荏苒，三年的研究生生涯即将结束。2016 年 9 月，我幸运的考上了北京邮电大学，与计算机学院智能通信软件与多媒体北京市重点实验室结下了难得的缘分，在大学期间一直憧憬进入北京邮电大学继续进修计算机科学与技术专业，为了实现自己心中美好的愿望，用自己不懈的努力，披荆斩棘，成功的得到老师的青睐，由此开始了我内心憧憬的研究生学习生活，并且在科研、学习、生活中我都成长了许多。这三年的研究生生活，使我从一个稚嫩、浮躁的大学生慢慢成长为一个踏实、努力、上进、敢于追求目标、更加严谨的硕士研究生。这一切都受到了我身边的老师、同学、朋友的影响。在此，我要真诚的感谢指导过我的每一位恩师，帮助过我的每一位学姐学长，耐心倾听我的同窗朋友，积极鼓励我的家人。

感谢我的导师张海涛老师对我的培养。感谢您在我的学习和生活给予的无微不至的关怀和帮助。您在工作中的一丝不苟、安排有序的做事风格深深的激励我并且鼓励我，让我内心又多了一份动力，您在科研上严谨的逻辑思维，更是让我受益匪浅。

感谢实验室的同窗好友。感谢高阳阳学长和朱彦沛学姐对我工作上和学习上的指导，你们对待科研的严谨态度、对每一个细节的认真处理的做事风格、努力踏实积极阳光的生活态度成为我为人处理的榜样；感谢唐炳昌同学对我在学习上的帮助，作为同学，我深深敬佩你在在学习上积极，努力，踏实的态度以及独到的思考，和你共同学习的实验室时光，将成为我受益一生的财富；感谢徐政钧学弟、耿欣学妹和实验室其他同学们，我们一起学习，一起团建，相互帮助，在最好的年华能够遇到你们，让我学习到了很多。

感谢我的家人。是爸爸妈妈对生活踏实的态度，鼓励我在每次困难面前不畏惧，勇往直前，是爸爸妈妈每一次的鼓励，让我的心态再次乐观起来，给予我无限动力，是爸爸妈妈背后默默无闻的支持，成为我坚强的后盾，让我踏实前行。

最后，我要感谢各位评审老师们和专家们在百忙之中抽出宝贵的时间审阅我的文章，谢谢你们的宝贵意见和建议。

作者攻读学位期间发表的学术论文目录

[1] Haitao Zhang, **Ning Yang**, Zhengjun Xu, Bingchang Tang, Huadong Ma. Microservice Based Video Cloud Platform with Performance-aware Service Path Selection. //2018 IEEE 10th International Conference Web Service (IEEE ICWS 2018), San Francisco, USA, 2018.