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：_____

姓名：_____

专业：计算机科学与技术

导师：_____

学院：计算机学院

年 月 日

BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

Thesis for Master Degree

Title: RESEARCH AND APPLICATION ON SERVICE PARTITION AND SELECTION STRATEGY IN MICROSERVICE PLATFORM

Student ID： _____

Candidate： _____

Major： _____

Supervisor： _____

Institute： _____

February 27th, 2019

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，传统的单体服务框架的复杂性越来越高，可维护性越来越差，因此微服务架构成为研究的热点。当前的云平台中服务复用率低，代码冗余率高成为平台的主要问题，另一方面，目前的服务选择策略未能综合考虑平台性能特征以及任务特征，导致服务的执行效率降低。

为了提高平台中服务的复用率以及平台的执行效率，首先本文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中应用功能关联性大的特点以及微服务的划分原则，实现了高效的服务划分；其次本文提出一种细粒度的性能预测模型，该模型可以准确的预测每一个微服务的执行时间；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，该策略通过初始化服务选择路径和动态的自适应更新得到最优的服务路径，提高了应用的执行效率；最后，通过实验验证了本文所提出的服务划分和服务选择方法，实验结果表明，本文设计的性能预测模型能够准确的预测服务执行时间，提出的方法能够有效提高平台服务的复用率，降低服务的执行时间，提高应用的执行效率。

关键词：微服务架构 服务划分 语义耦合 服务选择

RESEARCH AND APPLICATION ON SERVICE PARTITION AND SELECTION STRATEGY IN MICROSERVICE PLATFORM

ABSTRACT

With the expansion of the cloud platform business scale, the complexity of the traditional monolithic framework is getting higher and higher, and the maintainability is getting worse and worse. Therefore, the microservice architecture has become a hot research topic. In the current cloud platform, the low service reuse rate and the high code redundancy rate have become a main problem. On the other hand, the current service selection strategies do not comprehensively consider the platform performance characteristics and task characteristics, so these strategies lead to the reduced service execution efficiency.

In order to improve the reuse rate of services in the platform and the efficiency of platform execution, Firstly, this paper proposes a semantic coupling service partitioning strategy based on domain-driven design ideas. The strategy considers the characteristics of application functions in the micro-service platform and the principle of partitioning of microservices. Secondly, this paper proposes a fine-grained performance prediction model, which can accurately predict the execution time of each microservice. Then, based on the performance prediction model, this paper proposes a performance-aware service path selection strategy, which obtains the optimal service path by initializing the service selection path and dynamic adaptive update, this strategy improves the execution efficiency of the application. Finally, we conduct extensive performance experiment for verifying the service partitioning and selection strategy. The experimental results show that the performance prediction model designed in this paper can accurately predict the service execution time. The proposed method can effectively improve the reuse rate of platform

services, reduce the execution time of services, and improve the execution efficiency of applications.

KEY WORDS: microservice architecture service extraction semantic coupling service selection

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，虽然传统分层的单体架构有易部署，易测试的优点，但是其不足之处越来越明显，随着需求的不断增加，越来越多的人加入开发团队，代码库也在飞速扩展，最终的单体应用程序变得越来越臃肿，产生一系列问题，如可维护性、灵活性逐渐降低，维护成本越来越高等，这将给整个云平台的开发、维护、部署以及后期的升级带来巨大的困难，因此单体架构很难满足快速变化的互联网时代的需要。

上述问题的解决方案是将微服务架构应用到云平台上以拆分和重组现有业务系统，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]获得广泛的关注。微服务是一种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。

相对于单体应用架构来说，微服务架构有着易于开发和维护、单个微服务启动较快、局部修改容易部署、技术栈不受限、按需伸缩等优点，但是，微服务并非是完美的，使用微服务也为我们的工作带来了一定的挑战。本文中微服务平台主要有两个问题。

问题一，如何实现高效的服务划分[3]，提高微服务平台的服务复用率，降低代码的冗余。微服务的粒度是一个难题并且经常成为辩论的焦点。我们应当使用合理的粒度划分微服务，而不是盲目地把服务做小。代码的规模最好不要作为微服务划分的依据，因为对于不同的微服务，他们的业务复杂性可能不同，代码规模也会不同。在微服务的设计阶段，我们应该确定它的边界。微服务之间应相对独立并保持松散耦合[4][5]。目前围绕微服务的划分工作是有限的，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[6]，因此服务划分技术在微服务领域应用需要进一步的研究。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。

问题二，如何进行服务路径选择，得到最优的服务路径，降低服务的执行时间，提高应用的执行效率。对于基于微服务架构的分布式处理云平台，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。目前大多数的服务选择策略只考虑服务静态特征[9]，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。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。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。

综上，在微服务平台中，设计合理的服务划分策略，实现高效的应用执行效率，还有许多工作要做。近年来，微服务架构成为了云计算领域的热点话题，微服务已经成为现代大型工程组织中的新兴趋势，然而目前还没有在微服务平台设计合理的微服务方法以及综合考虑平台动态特征和任务特征的服务选择方法的公开案例，本文研究了微服务架构，提出了一种高效的服务划分方法和服务选择策略，有效提高了微服务平台中服务的复用率，减少了服务的执行时间，提高了应用的执行效率。

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最近几年来引起了国内外专家的广泛关注。如何划分微服务以及确定微服务的粒度[13]是研究的难点，也是服务计算领域争论的焦点。在服务划分阶段，确定边界[4]是首要问题。Eberhard Wolf 在文献[14]中提出微服务的规模应该足够小，并专注于实现一个功能； Mohsen Ahmadvand 等人[15]指出微服务分解的时候应该考虑系统要求、安全性和可扩展性，然而这些并不是分解微服务时应该考虑的唯一因素；Tugrul Asik 等人[16]提出通过计算用于其他微服务或者外部服务交互的资源和客户的和来衡量一个微服务的大小。由于衡量微服务大小的因素是模糊的，因此将一个应用分解成合适的微服务是一个具有挑战性的工作。目前，在实践中已经提出了许多策略来确定微服务的大小以及如何微服务。Gerald Schermann 等人[17]将微服务的大小与代码行数（LOC）联系起来，并且建议微服务的规模应该在 10 到 100 个 LOC，文中指出计算 LOC 是微服务不应该超过的代码的行数，因此微服务的代码行数越少就增加了微服务扩展的灵活性并且简化了更改或者移除微服务的过程。但是 LOC 策略是不合适的，因为微服务有可能使用不同的技术栈构建的，而这些技

术栈在 LOC 上是不同的。此外，不同的服务类型最小的 LOC 可能是不同的。Mario Villamizar 等人[18]提出将微服务定义为独立开发和部署的单元，这些微服务的集合可以部署成一个大的应用，这些服务的划分通过注册的方式划分，被其他服务发现并且可以在部署和更新过程中编排。Ulrich Kalex 在文献[19]中通过基于业务能力和业务功能来构建微服务，但是开发人员在使用业务功能作为微服务划分边界时面临着很大挑战，如何定义一个业务能力的的粒度才能使其不会太大或者太小。可见国内外为了得到高效的微服务划分方法，或者通过代码行数来确定微服务边界，或者通过业务能力来确定微服务边界，目前还很少使用语义耦合的方式来划分微服务。本文实现的基于领域驱动设计思想的微服务划分策略，将有效提高服务的复用率。

为了提高应用的执行效率，服务选择的方法至关重要，目前国内外已有很多关于服务选择的方法涌现。M. Alrifai 等人[7]提出通过考虑服务之间的 QoS 相关性来选择合适的候选服务，该方法可以管理服务之间相关性，并显著提高生成的组合服务的 QoS 值；Shuiguang Deng 等人[8]提出了提出一种基于 skyline 的服务选择方法来进行服务组合，该方法通过减少要考虑的候选服务的数量来有效的组合服务，尽管这些方法是为了获得最优的服务组合，但是这些方法并没有考虑服务中资源的运行时状态，不能保证在线任务的处理效率；Tian Huat Tan 等人[10]提出一种基于遗传算法的方法来优化服务组合整体的 QoS，并进行服务组合的自适应更新；Wang 等人[11]提出了一种利用强化学习的方法来保证服务组合的时候的自适应性，该方法通过博弈论来决定服务优化的方向；Peng 等人[20]提出一种自适应的方法 rEDA 来支持动态的 QoS 感知的服务组合的优化；文献中[17][18]提出云计算平台中性能感知的自适应的服务选择和组合方案，该自适应服务路径选择方法满足了用户的 QoS 的同时又保证了整个系统中的负载均衡。尽管这些方法解决了仅仅依据静态 QoS 进行服务选择的问题，能够支持在线服务的自适应更新，但是这些方法并没有考虑任务的特征，因此不能将这些方法直接应用到微服务平台上进行服务选择。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。

综上所述，国内外为了解决服务划分问题，或者将代码行数作为边界，或者通过分析平台的业务能力作为边界，目前服务计算领域很少将语义耦合的概念使用到服务划分中；为了解决服务选择问题，国内外的研究很多只考虑服务静态 QoS，目前很少研究将机器学习方法与服务选择技术来提高应用执行效率。本文结合平台应用的特点，设计了语义耦合的服务划分方法和性能感知的服务选择方法，减少了服务执行时间，提高了平台的执行效率。

1.3 论文的主要研究内容

上文提到的微服务平台中遇到的两个难点问题：（1）在进行微服务划分时，如何确定服务的边界，来确定微服务的粒度，制定合理的服务划分方法，提高平台中服务的复用率。（2）如何进行服务路径选择，得到最优的服务路径，降低服务的执行时间，提高应用的执行效率。本文针对以上问题提出了基于领域驱动设计的服务划分方法和性能感知的服务选择策略，主要研究内容有以下几点：

（1）基于领域驱动设计的服务划分方法

为了解决难点 1，本文分析了当前主流的服务划分方法，当前的服务划分要么从代码行数来考虑服务的边界，要么从业务能力来考虑服务的边界，不能很好的来确定服务的粒度。本文提出了一种基于领域驱动设计思想的服务划分方法，该方法首先建立服务划分模型，包括单体架构阶段、图阶段、微服务阶段，通过构建过程和聚合过程两个过程得到最优的微服务集合，并通过实验证明，该方法可以有效的提高平台中服务的复用率，减少应用代码的冗余。

（2）性能感知的服务路径选择方法

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。为了解决难点 2，本文利用我们微服务平台中原有的应用程序，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，建立针对微服务平台的细粒度的性能预测模型，基于建立的性能预测模型，提出了性能感知的服务选择策略，通过缩减服务选择空间在线更新微服务选择路径，进一步提高了微服务平台中应用的执行效率。

（3）实验验证和性能分析

本文提出一种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。论文首先介绍了研究背景，之后介绍了当前微服务平台中服务划分和服务选择方法的研究现状，最后介绍了本篇论文的主要研究内容。

第二章：相关技术。本章介绍了微服务划分策略相关的一些背景知识，包括单体架构、SOA 架构以及微服务架构，另外我们介绍了机器学习相关技术，保证回归技术以及常用的线性回归技术和非线性回归技术的对比，最后介绍了最短路径选择算法，包括迪杰斯特拉算法和维特比算法。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预测模型，然后基于性能预测模型，详细介绍本文提出的性能感知的服务路径选择方法，最后本篇论文详细介绍算法的实现过程。

第五章：系统实验与测试分析。首先描述了实验环境包括软件配置、硬件配置、数据说明以及子任务实现，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，以及服务路径选择方法的性能进行实验验证，并对得到的实验结果进行了详细的分析。

第六章：结束语。对本篇论文的所有研究工作做出详细总结，并且结合当前行业热点展望微服务平台中服务划分和服务选择方法研究趋势，认真分析本篇论文中提出的服务划分和服务选择方法进一步的优化方向。

第二章 相关技术介绍

本章主要介绍了后面的章节所涉及的相关技术方面的知识，首先详细介绍了单体式软件架构，重点指出该架构的一些严重的问题，从而引出今年来兴起的微服务软件架构，并且叙述了该架构的优势。然后介绍了本文中使用的回归分析技术，主要详细的对比了线性回归和非线性回归技术的差异，以及线性回归技术的优点。

2.1 微服务相关技术

2.1.1 单体式软件架构

众多的项目都是以单体应用作为开始的，单体应有本身的比较容易部署、测试的优点，使得单体应用在项目的初期可以很好地运行。然而随着互联网需求的不断扩张，越来越多的人加入开发团队，代码库也在飞速地扩展。慢慢地，单体应用规模变得越来越庞大，可维护性、灵活性逐渐降低并且维护成本越来越高。单体应用存在的主要问题如下：

（1）复杂性高：在规模较大的单体应用中，项目包含相当多的模块、模块的边界比较模糊、模块与模块之间的依赖关系也很模糊、代码质量不均匀并且混乱地堆砌在一起，整个项目非常复杂。每次工作人员修改代码都会很棘手，甚至添加一个很小的功能，或者修改一个很小的 Bug 都会带来隐含的缺陷。

（2）技术债务：随着时间推移、需求变化和人员流动，应用程序的技术债务会逐渐形成并且积累。“不坏不修（Not broken,don’t fix）”，这在软件开发中非常普遍，而这种想法在单体应用中更是如此。已经使用的系统设计或者代码很难被修改，因为应用程序中的其他模块可能会以想不到的方式使用它。

（3）部署频率低：随着代码的增多，构建和部署的时间也会增加。而在单体应用中，每次功能的变化或 BUG 的修复都会带来整个应用程序重新部署的结果。全量部署的方式耗时长、影响范围大、风险高，这使得单体应用项目上线部署的频率较低。而部署频率低又导致两次发布之间会有大量的功能变更和缺陷修复，出错率比较高。

（4）可靠性差：某个应用 Bug，例如死循环，可能会导致整个应用程序的崩溃。

（5）扩展能力受限：单体应用只能作为一个整体进行扩展，无法根据业务模块的需要进行伸缩。例如，应用程序中有的模块属于计算密集型的，它需要的

CPU 资源比较多；有的模块则属于 IO 密集型的，需要更大的内存。由于这些模块部署在一起，不得不在硬件的选择上做出妥协。

（6）阻碍技术创新：单体应用通常使用统一的技术平台或技术方案解决所有的问题，团队中的每个工作人员都必须使用相同的开发语言和框架，要想引入新框架或新技术平台会非常困难。例如，一个使用 Struts 2 构建的、有 100 万行代码的单体应用，如果想要换用 Spring MVC，毫无疑问切换的成本是非常高的。综上，随着业务需求的发展，功能的不断增加，单体架构很难满足互联网时代业务快速变化的需要。

2.1.2 SOA 架构

上小节介绍了传统的单体式软件架构的不足，在上一世纪 90 年代，面向服务的体系架构 SOA 软件架构[21]得到了广泛的关注。面向服务的体系架构（SOA）是一种软件体系结构，其中应用程序的不同组件通过网络上的通信协议向其他组件提供服务。通信可以像数据传递一样简单，也可以是两个或多个服务彼此协调连接，这些独特的服务执行一些小功能。面向服务的架构不是关于如何对应用程序进行模块化构建，而是关于如何通过分布式、单独维护和部署的软件组件的集成来组成应用程序。这些通过技术和标准来实现，而这些技术和标准使得组件能够更容易地通过网络（尤其是 IP 网络）进行通信和协作。SOA 架构中有两个主要角色：服务提供者（Provider）和服务消费者（Consumer）,而软件代理则可以扮演这两个角色，该 Consumer 层是用户与 SOA 交互的点，和 Provider 层则由 SOA 架构内的所有服务所构成。SOA 在 90 年代中期得名，使得大家认识了这个软件架构的新趋势。该架构相比于单体架构的优点主要有：

（1）能够拆分模块并且模块与模块之间的通信方式是接口，这样减少模块与模块之间的耦合度。

（2）把项目分成几个子项目，不同的团队负责不同的子项目。

（3）添加功能时，只需要添加一个额外的子项目，接口调用可以使用其他系统中的相关接口。

（4）在分布式部署时，有很高的灵活性。

但是 SOA 架构也有一些问题：

（1）模糊的系统和服务的界限，使得开发及维护不利于进行。

（2）架构中虽然使用 ESB 来通信，但是服务的接口协议会变化，种类众多，非常不利于系统维护。

（3）提取的服务的粒度太大，导致系统和服务之间的高耦合性。

综上所述，SOA 可能比较适合需要与许多其他应用程序集成的大型复杂企业应用程序环境。换句话说，小型应用程序不适合使用 SOA 架构，因为它们不需要消息中间件组件，而微服务架构，在某些方面，对于基于 Web 的系统是更适合于较小和良好的分割。在下一小节中，本文将详细介绍微服务架构。

2.1.3 微服务架构

上小节介绍了单体应用架构和 SOA 软件架构存在的问题，微服务架构模式有助于解决这些问题，当前微服务的概念已经成为软件架构的关注热点之一，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。微服务架构由单体架构到面向服务架构 SOA 慢慢发展而来，它的理论基础来自于康威定律[34]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]: 微服务架构主要思想是将一个单体应用程序转化为一组小型服务，这些服务的构建过程是基于业务能力的，每个服务都可以在自己的进程中运行，服务间通信通常用 HTTP 等轻量级通信机制，而且这些服务可以自动化独立部署，它们之间共享最小型的集中式的管理模式，而且可以通过使用不同的开发语言来开发并使用不同的数据存储技术。它和传统的单体式架构、SOA 服务架构的不同如图 2-1。

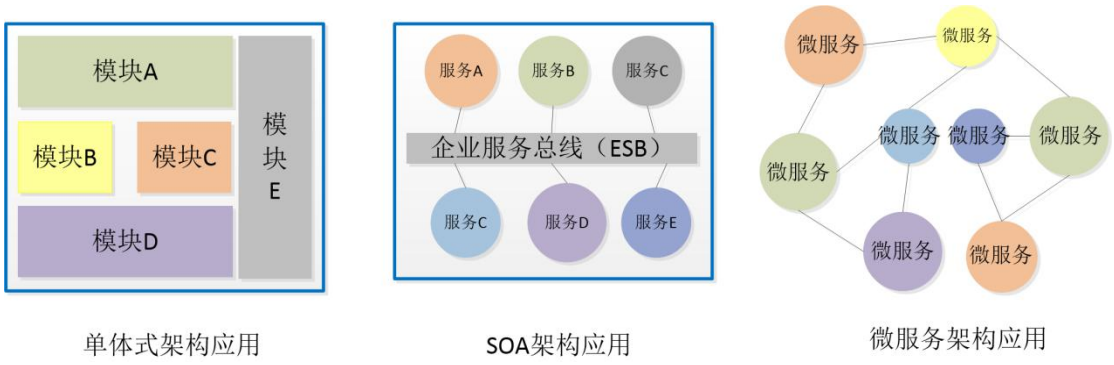

图 2-1 单体架构、SOA 架构和微服务架构

微服务架构具备以下特性：

（1）易于开发和维护：一个微服务仅会关注一个特定的业务功能，因此它具有明确的业务、少量的代码。开发和维护单个微服务是比较简单的。整个应用程序是由几个微服务构建而成的，因此整个应用也会保持在一个可控状态。

（2）单个微服务启动较快：单个微服务代码规模比较小，因此启动速度会比较快。

（3）局部修改容易部署：只要修改了单体应用，就必须重新部署整个应用，微服务就解决了这样的问题。通常，对某个微服务进行修改，只需要重新部署相应的服务。一个微服务只有一个具体的功能，可以由单独的开发团队来开发。

（4）技术栈不受限：在微服务架构中，可以结合项目业务及团队的特征并且合理地选择技术栈。例如默写服务可使用关系型数据库 MySQL；某些微服务有图形计算的需求，可以使用 Neo4j；甚至可根据需要，部分微服务使用 Java 开发，部分微服务使用 Node.js 开发。

（5）按需伸缩：可根据需求，实现细粒度的扩展。例如，系统中的某个微服务遇到了瓶颈，可以结合这个微服务的业务特点，增加内存、升级 CPU 或者是增加节点。

综上，单体架构和 SOA 架构的缺点，恰恰是微服务的优点。本篇论文更青睐微服务架构在构建云服务时的优势，这是因为每个微服务可独立运行在自己的进程里，并且一系列独立运行的微服务一起构建起整个系统，一个微服务只有一个具体的功能，可以由单独的开发团队来开发，且微服务之间通过一些轻量级的通信机制进行通信。另外，微服务的轻量级的优点，使微服务平台更容易实现负载均衡、减少资源碎片、更易于管理，并且提高资源利用率。

2.2 回归分析相关技术

机器学习（Machine Learning）是一门涉及到概率论、统计学、计算机科学以及软件工程的多领域交叉学科。机器学习是指一组工具或方法，通过这些工具或方法，使用历史数据将机器“训练”进而“学习”成某种模式或规律，并建立预测未来结果的模型。

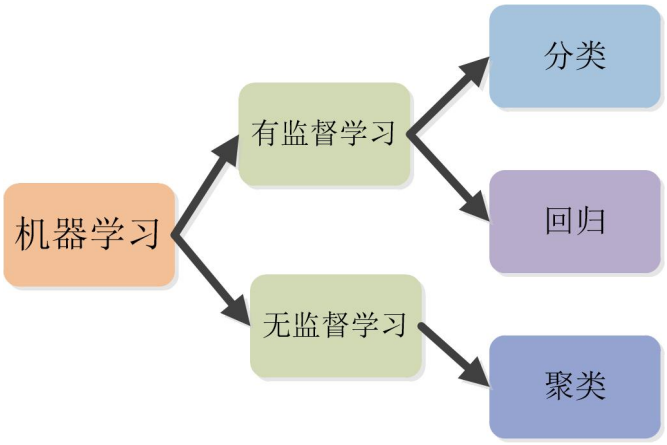

图 2-2 机器学习

回归分析是属于机器学习方法中的一类，机器学习主要涉及两类学习方法（如图 2-2）：有监督学习，该方法主要包括分类和回归，主要用于决策支持；无监督学习，该方法主要包括聚类，主要用于知识发现。

2.2.1 回归分析技术简介

本小节我们主要讲解以下回归分析技术。在统计学中，回归分析（regression analysis)是一种用于确定两种或两种以上变量间相互依赖的定量关系的统计分析方法。且回归分析应用的很广泛，主要可以根据变量的数目、自变量因变量的关系等进行分类。本文中应该的线性回归分析就是通过分析自变量和因变量之间的关系来分类的，另外，本文模型中涉及的特征由多个，所以自变量有多个，而且自变量和因变量之间的关系是线性相关，所以本篇论文中使用的是多元线性回归方法。在大数据分析中，回归分析是一种预测建模技术，研究的是因变量（目标）和自变量（预测变量）之间的关系。回归分析技术在机器学习中属于监督学习的范围，主要用于预测分析等，本文使用回归分析来建立时间预测模型，其目标是给定 D 维输入变量 x，并且每个输入向量 x 都有对应的值 y，需要为新数据预测其对应的连续目标值 t。

如果要预测的值是连续的，则它属于回归问题；如果要预测的值是离散的即一个个的标签，则它是分类问题。这个学习处理过程如图 2-3。

图 2-3 的学习过程中的常用术语：输入的数据集称为训练集 training set；输入变量 x 为特征 features；输出的预测值 y 为目标值 target；拟合的曲线，通常表示为 y = h(x)，称为假设模型 hypothesis；训练集的条目数称为特征的维数。

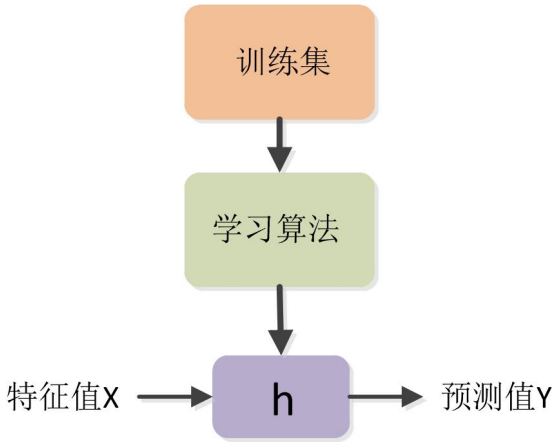

图 2-3 学习处理过程

回归分析方法的特点如下：

（1）可以简单且方便的分析多因素模型；

（2）使用回归模型，计算出的结果只和模型以及数据相关，如果模型和数据一样，那么结果就唯一。但是以图形和表格的形式，数据之间关系的解释通常因人而异，不同分析者得出的拟合曲线很可能也是不同；

（3）回归分析可以准确地测量各因素与回归拟合程度之间的相关程度，提高预测方程的效果；

（4） 在回归分析的情况下， 一元回归分析更适用于单个变量影响明显高于其他因素的变量的情况，多元回归分析法更适用于多个因素影响比较均匀时使用。

综上，本文结合回归分析的特点，使用回归分析的方法来训练第四章提到的性能预测模型，通过实验证明，该训练方法可以准确简单的学习得到预测模型。

2.2.2 线性回归与非线性回归对比

回归是一种用于建模和分析变量之间关系的技术，通常是它们如何贡献并且与一起产生特定结果相关。线性回归指的是完全由线性变量组成的回归模型。主要有单变量线性回归和多变量线性回归，前者是一种用于使用线性模型（即线）来模拟单个输入自变量（特征变量）和输出变量之间的关系的技术。更一般的情况是后者，其中为多个独立输入变量（特征变量）和输出因变量之间的关系创建模型。模型保持线性，输出是输入变量的线性组合。一般根据自变量与因变量之间的函数表达式是线性还是非线性，分为线性回归(Linear Regression)和非线性回归(Non-linear Regression)。我们可以建模多变量线性回归，如公式 2-1。

$$Y = a_1 * X_1 + a_2 * X_2 + \cdots + a_n * X_n + b \tag{2-1}$$

其中a_n是系数，X_n是变量，b 是偏差。可以看到此函数不包含任何非线性，因此仅适用于对线性可分离数据进行建模。线性回归的特点主要有：

（1） 建模速度快且简便，当要建模的关系不是特别复杂且没有大量数据时，建模尤其有用。

（2）非常直观地理解和解释

（3）对异常值，线性回归非常敏感。

自变量与因变量之间的函数表达式的非线性体现在至少有一个变量的指数不是 1。当使用非线性回归分析问题时，一般将非线性回归方程式变换为线性回归方程式，获得参数后，通过逆变换将线性回归方程转化为非线性回归方程。当将非线性回归转化为线性回归时，主要是确定变量之间存在的非线性关系的类型。该方法可基于专业知识的理论推导及直接生产或试验数据，并从点分布的特征中来选取合适的曲线类型。根据线性回归和非线性回归的特点，本文选择线性回归

来建立第四章中的性能预测模型，实验证明我们的性能预测模型能够准确的预测服务的执行时间。

2.4 本章小结

本章首先具体介绍了微服务方面相关的技术，主要包括单体式架构、SOA 架构以及微服务框架相关技术，然后介绍了机器学习相关技术，包括回归分析技术以及线性回归非线性回归的对比。本章节主要为后续章节的研究提供基础。

第三章　语义耦合的微服务划分策略

本文主要介绍了基于领域驱动设计思想的语义耦合的微服务划分策略的设计和实现过程。首先总结了当前国内外一些微服务划分方法并且提出了一些不足之处，然后根据平台中应用的特点，微服务高内聚低耦合的划分原则以及软件工程中领域驱动设计的建模思想，提出一种语义耦合[47]的服务划分策略，提高了服务的复用率，减少了应用的代码冗余。

3.1 当前的服务划分方法概述

目前在服务计算领域，已经有一些研究人员对微服务划分进行了相关研究，当前微服务领域主要的挑战是确定微服务的大小以及如何划分微服务[13]。下面是本文总结的主要的服务划分策略：

（1）代码行数。一些研究人员将微服务的大小与代码行数（LOC）相关，并且推荐一个服务的代码行数应该在 10 行到 100 行代码之间。微服务少的代码行数增加了微服务扩展的灵活性，更易于更改或移除微服务。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。但是该策略并不合适，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。

（2）部署单元。该思想中微服务被定义为一种开发和部署的单元，主要是在云环境中部署大型或者中型的应用，且这些服务可以独立的开发、测试、部署、扩展、操作、升级。这些服务根据注册的方式来划分，在部署和升级的过程中，能被其他服务发现并且可以被编排。但是该方法得到的微服务的粒度有可能会很大或者很小。

（3）业务能力。业务能力定义为系统在执行唯一的业务时所执行的操作。构建微服务是根据是否一次能解决业务需求或者实现一个业务功能。然而，开发人员在使用业务功能作为微服务的边界时面临着挑战，定义业务能力应该适合的粒度级别，以便它不会太小或太大。

综上，以上服务划分策略存在许多问题或者挑战，不能得到高效的服务划分策略，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。

3.2 解决方案

为了实现低耦合高内聚的微服务划分[48]，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（DSCS），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。

3.3 基于语义耦合策略的服务划分模型

本节语义耦合的服务划分模型是结合软件工程中领域驱动设计思想以及语义相似性来构建的。

3.3.1 领域驱动设计思想

在软件工程中有一种建模思想领域驱动设计（DDD）[22]，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起来应用于软件涉及中，实现领域驱动涉及的基础是面向对象的编程方法 (Object Oriented Programming，OOP)[23][24]，OOP 中的对象相对应的是领域驱动中的实体，领域驱动的开发中，OOP 的继承、封装和多态等特征很适用，领域对象应可以设计成调用的类或接口。该思想中主要有三个主要的概念。

（1）领域，领域与特定的开发技术没有关系，它是与软件系统需要解决的实际问题相关的所有内容的集合。

（2）子域。子域是领域更细粒度的划分，根据功能和重要程度主要分为核心子域、支撑子域、通用子域。核心域是业务成功的关键，支撑子域是支持核心域的，通用子域是业务系统的公用部分。

（3）限界上下文。限界上下文是由系统、应用程序、业务服务以及一组实现业务的复杂组件组成。通用语言是限界上下文中的每种领域术语、词组。通用语言不是一时促成的，而是各方的人员在讨论中提炼出来的。领域模型是将通用语言表达为软件模型。

3.3.2 语义相似度技术

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[25-29]是任何两个对象之间存在的普遍关系，相似度是相似性的定量表示。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和应用成为心理学和计算机科学交叉的重要课题。

Dekang 定义了一组具有广泛意义的相似度，告诉我们，对象和对象之间的相似度与它们的共性和差异相关，两个对象的共性越多，相似度就会越大，当两个对象之间的差别越大，相似度就会越小。当两个对象是同一个对象时，相似度达到最大。当两个对象没有相关性时，相似度最小。

相似度模型分为几何模型和特征对比模型[30][31]。对象用空间中的点来表示，而点之间的距离来表示对象与对象之间的相似程度，将这一类模型统称为几何模型。特征对比模型通过一组特征属性集合描述对象，相似度[32]被定义为特征共性和差异的函数。

本文分析对象的相似度使用的是几何模型，能够清晰简单的表示出两个对象相似度。语义相似度的计算步骤是：1. 特征提取与特征选择。一般情况下，我们使用的对象是用自然语言来描述的，这样的话就只能间接被计算机处理。对象的特征提取是指提取出对象的主要特征以表示对象。现有特征描述是根据统计方法构造相应的数学模型来表示对象，目的是从所表示的对象中提取出最有价值的特征信息。但是也可能遇到特征项的维数问题，因此需要通过特征选择筛选出特征项。特征选取需要根据一定的标准选择出最能反映对象特性的相关特征，以减小特征维数、简化计算、提高相似度计算准确性。2. 对象的表示。通常，我们使用至少两个特征来描述对象，并且使用间隔尺度来量化特征，间隔尺度是由实数表示的数量信息。假如选择了个特征，则这个对象可以表示为向量。3.相似度具体计算方法。设 T 和 T`为矩阵中的两个对象的特征向量，通常使用的相似度

公式如下：内积、余弦函数、Dice 系数等。本文使用余弦函数来计算两个对象的相似度。

3.3.3 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，我们基于领域驱动设计思想，设计了服务划分模型，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：

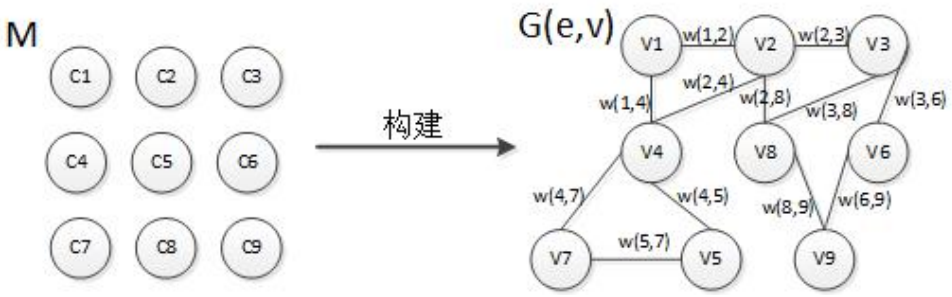

图 3-1 构建图阶段

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，并且通过定义的权重公式得到。权重的大小代表了两个文件之间的耦合程度，顶点v_i与顶点v_j之间的权重$w_{i,j}$值越大，表示文件c_i与文件c_j之间的耦合程度越高。

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。

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[58]阶段如图 3-2：

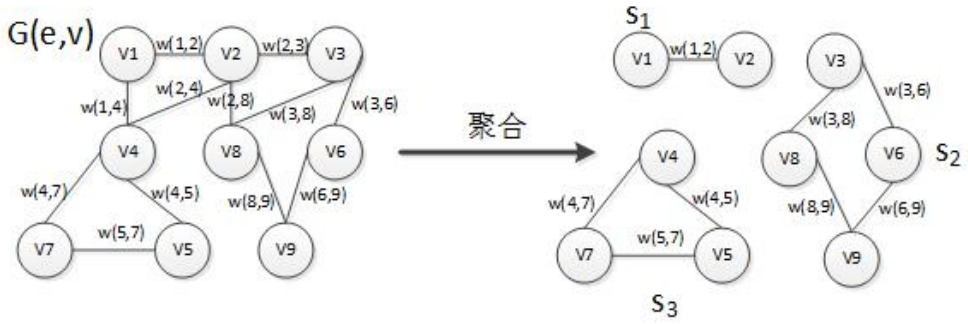

图 3-2 聚类阶段

聚类过程即将文件之间的耦合图，转换成相应的微服务。在聚类过程中，我们使用经典的社区发现算法 GN 算法[34]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：

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

公式中，a_{ij}为图的邻接矩阵的元素，如果 i 和 j 两节点有边相连，则a_{ij}为边的权重，如果a_{ij}等于 0，则 i 和 j 两节点不相连，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，对图中所有的源节点，都做同样的操作，并将得到的相对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，即我们求得的该边的边介数。

综上，可以看出该服务划分模型通过两步可以得到最优的服务划分组合，且该服务划分组合符合微服务的“高内聚低耦合”原则。

3.3 微服务划分策略

上一小节详细讲述了微服务划分模型，本小节主要讲述服务划分模型中使用的划分策略——语义耦合策略[47]。

3.3.1 语义耦合策略

基于上节的服务划分模型，如何得到代码库中每个文件之间的关系，本文使用最直接的方式——语义耦合，通过分析两个文件之间的语义相似性，来量化两个文件之间的耦合程度，从而得到整个应用中全部文件之间的关系图——无向加权图。

在语义耦合策略中，我们将原有单体式应用中的文件作为输入，将生成的单体式应用中文件之间的关系图作为输出。根据第一小节中语义相似度的计算步骤，首先，本文提取单体式应用中每个文件的特征，在提取特征的过程中，我们选择每个文件中的关键字，例如变量名或者方法名来作为每个文件的特征值，因此任意一个代码文件使用一组具有功能代表的词组$W_j = \{w_1,w_2,...,w_n\}$来表示。但是考虑到代表每个文件的词组的维数可能是不同的，我们做了维数的转换过程，我们建立两个代码文件c_i的词组W_i以及代表代码文件c_j的词组W_j的并集列表 *T*，

$$T = W_i \cup W_j = \{t_1,t_2,...,t_k\} \tag{3-2}$$

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。*tf-idf* 值代表该第 k 个元素即第 k 个词组在整个并集列表 *T* 中的重要程度。向量 *X* 中的元素x_k计算公式如下：

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

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：

$$W_{all} = \{W_i,W_j\} \tag{3-4}$$

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

$$tf(t_k,W_i) = n_{k,i}/\sum_i n_{j,i} \tag{3-5}$$

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

逆向文件频率 $idf(t_k,W_i)$是一个词语普遍重要性的衡量方式。某一特定词语的 *idf*，是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：

$$idf(t_k,W_{all}) = \log(|W_{all}|/n(t_k)) \tag{3-6}$$

$W_{all} = \{W_i,W_j\}$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 3-6：

$$\mathrm{A}=\begin{bmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,n} \end{bmatrix} \tag{3-7}$$

其中$a_{i,j}$表示节点v_i和节点v_j之间的边的权重。

综上，我们可以通过语义耦合策略来得到单体式应用中代码之间的耦合关系图——无向加权图 G，并且通过使用矩阵 A 来表示无向加权图。

3.3.2 服务划分算法

上小节中我们得到了单体式应用中代码之间耦合关系图——无向加权图 G，第一阶段完成，进入第二阶段——聚合阶段，聚合阶段的服务划分算法应满足以下要求：

（1）可复用性。服务划分算法应满足得到的微服务集合具有高可复用性，这是我们算法的主要目标之一。

（2）灵活独立。服务应该具有高内聚性，恰当粒度的服务构件能使服务在多种角度组合；通过松散耦合技术缩小服务消费者和提供者间的依赖性。

（3）高耦合性。划分后的服务之间的耦合度应该尽量小。

（4）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。

（5）性能。算法执行的时间尽可能少。

（6）简单性。算法的机制和参数的理解程度尽可能简单。

在聚类过程中，为了满足以上服务划分算法的要求，我们对比了社区发现算法 GN 算法（基于边介数的加权无向图聚类算法）、MCL 算法（加权无向图聚类算法）和 ELP 算法（加权无向图（有向图）聚类算法），GN 算法具有很好的确定性，且测试结果也比较好。因此本文采取 GN 算法来实现服务划分。
实现该算法的伪代码如下：

算法 1 微服务划分算法

根据公式得到每条边的权重 e_{ij}
根据公式得到所有连接边的边介数 B_{ij}
边介数除以权重得到边权比
while *edges* 不为空 **do**
    if $e \in edges$ 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

GN 算法的逻辑如算法 1 所示，首先我们根据语义耦合策略得到单体式应用之间的耦合关系图——无向加权图，在此基础上，具体步骤如下：

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。

综上，我们的服务划分算法可以有效的将单体式应用划分为相应的微服务集合，且通过实验验证，得到的微服务具有较高的复用率，并且能够降低微服务平台中应用的代码冗余率，缩小整个的平台代码量。

3.4 本章小结

本章首先分析了当前服务划分算法以及存在的不足及挑战，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，最后介绍了在建立服务划分模型基础上，详细介绍了微服务划分策略的思路以及伪代码的实现。

第四章 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，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 传统的服务选择算法概述

当前国内外有许多关于服务选择算法的研究，为了在许多具有相同功能的 Web 服务中选择出符合用户需求的服务，处了要考虑功能需求，也要注重服务的非功能属性，其中包括花费、可用性、响应时间和吞吐量等，这些 QoS 的度量将会对服务选择的准确性产生重要影响，因此，基于 QoS 度量在 Web 服务选择中引起了许多学者的关注。

4.1.1 基于静态 QoS 的服务选择策略

目前在服务计算领域中，QoS 作为一个重要标准，在服务选择过程中受到越来越多的关注。近年来已经提出了各种 QoS 感知服务选择方法。Zeng[40]等人在以质量为驱动的服务选择方法的研究中，主要考虑多个属性 QoS 和总体约束条件，并在服务选择中的 QoS 中加入了用户权重，体现了服务选择中用户权重的重要作用，并且运用混合证书规划模型来挑选出最优的候选服务。但是，其中很少考虑服务之间的 QoS 相关性，从而导致一些性能问题。 QoS 相关可以定义为服务的某些 QoS 属性不仅依赖于服务本身，而且还与其他服务相关。由于这种相关性将影响 QoS 值，因此在生成具有最佳 QoS 值的复合服务时，研究如何选择适当的候选服务同时考虑 QoS 相关性是很重要，因此 Deng S G[8]等人提出了一种新的服务选择方法，称为相关感知的服务修剪方法（CASP）。它通过考虑可以集成到最佳组合服务中的所有服务来管理 QoS 相关性并且修剪不是最优候

选服务的服务。最后实验表明，该方法可以管理服务之间复杂的相关性，并显着提高生成的组合服务的 QoS 值。

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。但是这些基于 QoS 的服务选择算法只考虑静态的 Qos 特征[37-41]，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[42][43]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 动态的自适应的服务选择策略

由于 QoS 的不确定性，目前还有一些学者在进行服务选择时，考虑到动态的自适应更新服务选择方法。Tian Huat Tan 等人[10]等人提出了一种基于遗传算法的自动化方法来计算恢复计划，该计划可以保证恢复后的组合服务的功能特性的满足。Wang 等人[11]提出了一种利用多智能体强化学习来保证服务组合适应性的方法。 他们运用博弈论来决定服务优化方向。Peng 等人[12]提出了一种自适应方法 rEDA，以支持动态 QoS 感知服务组合的重新优化。Kwonyong Lee[20]等人提出了一种面向服务的网络虚拟化环境中的自适应服务路径选择算法，它可以保证 QoS 并同时平衡负载。所提出的算法从在虚拟机上运行的多个候选服务实例中选择适当的组件服务，并创建满足用户的 QoS 要求的最佳服务路径。该算法还可以根据各种变化动态适应最优服务路径。由于所提出的算法处理称为 NP-complete 的多约束路径选择问题的变化，文中使用蚁群优化算法来表达问题。实验结果表明，该算法在保证同时平衡负载的同时保证了用户的最大 QoS，并根据情境变化适应最优服务路径。另外为了优化服务选择，减少服务选择耗费的时间，Skyline 服务也被用于缩短服务选择的计算时间。Alrifai 等人[7]专注于一种基

于 QoS 的 Web 服务组合方法，从 Skyline 服务中选择候选服务，以此减少候选服务的个数，这样可以减少服务选择的时间。

这些方法选择合适的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。

4.1.3 传统的服务路径选择算法不足及解决方案

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

4.2.1 问题描述

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 是一个微服务应用的处理流水线，能够被分成 n 个顺序执行的子务，$P=\{P_i\}(i=1,2,...,n)$。我们定义微服务类集合 $S=\{S_i\}(i=1,2,...,n)$。每个微服务类 Si 由所有的微服务实例$\{s_{ij}\}$（j=1,2,⋯,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$\{s_{1.}, s_{2.}, ..., s_{n.}\}$，其中$s_{i.}$是微服务类$S_i$中的微服务实例。服务路径 SP 决定了任务 P 的执行时间。因此，任务 P 总的执行时间为：

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

对于处理任务 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，能够最小化处理任务 P 的执行时间 T。

4.2.2 性能预测模型

上一小节描述了主要问题，我们的主要目标是得到处理任务的执行时间，并且使 T 的值最小。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_{ii}^c与数据传输时间T_{ii}^s之和。

微服务平台应用执行的过程中，微服务实例的执行时间可能是不断发生变化的， 我们需要根据微服务实例的状态以及任务的特征来精确预测每个微服务实例的执行时间，并基于此制定合理的服务选择策略，提高应用的执行效率。为此需要建立一个精准的微服务性能预测模型。

然而，由于不用的输入文件可能具有不同的帧率、码率、以及时长，同时，微服务实例的资源状态也不同，导致每个微服务实例的数据处理时间可能会动态变化，因此，如何得到实时的服务的数据处理时间并构建精确的性能预测模型是一个关键且困难的技术点。

为此，本文分析微服务实例处理时间的影响因素，将性能预测模型分为两部分，第一部分为数据处理时间模型即得到数据执行时间T_{ii}^c，第二部分为数据传输时间模型即得到数据传输时间T_{ii}^s，性能预测模型为两部分之和即T_{ij}。

数据执行时间T_{ii}^{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_{ii}^{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。因此我们使用线性回归模型来表示数据处理时间模型。线性回归模型中的参数 W 在本文中我们使用机器学习[55][56]的方法来训练得到。为了得到精确的数据处理时间模型，我们的训练数据是通过在云平台上运行目标跟踪算法得到的真实数据包括1000 个具有不同分辨率和不同数据大小的视频文件，其中 700 个视频文件作为训练数据集，300 个视频文件作为测试数据集。

假设在微服务实例s_{ij}上运行的视频子任务P_i由向量 $X = \left(x_{1,\ \ldots,\ } x_m\right) \in R^m$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。由此可知，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。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：

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

其中 $W = \left(w_0,\ w_1 \ldots w_m\right)$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。

为了估量本文中回归模型得到的预测值 $f(x)$与真实值的不一致程度，我们使用平方损失函数作为本文的损失函数。损失函数是经验风险函数的核心部分也是结构风险函数的重要组成部分。为了得到训练数据集中所有数据的预测能力，我们使用模型的结构风险函数来表示。模型的结构风险函数包括经验风险和正则项，如表达式（4-4），为了防止经验风险出现过度拟合的问题，我们引入正则化项，通过降低模型复杂度来防止过度拟合的出现。因此得到模型的结构风险函数，如下：

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

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_{ii}^{c}。

数据传输时间T_{ii}^{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 $P \leftarrow P \cup \{P_0, P_{n+1}\}$。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$s_{i,i}$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$s_{i,i}$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$s_{i,i}$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$\alpha_i (i = 1, \ldots, n, \quad \alpha_0 = 1, \alpha_{n+1} = 0)$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。

综上，可以看出我们的性能预测模型T_{ij}在预测每个微服务处理能力的时候，不仅考虑了服务在线的资源特征、任务的特征，还考虑了各个文件本身的特性，可以准确的计算微服务的处理时长。

4.3 微服务路径选择策略

上一节中我们得到了性能预测模型，在此基础上，我们提出了性能感知的服务路径选择策略，主要包括两个阶段服务路径初始化阶段和服务路径动态更新阶段，本小节将详细介绍这两个阶段。

4.3.1 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。

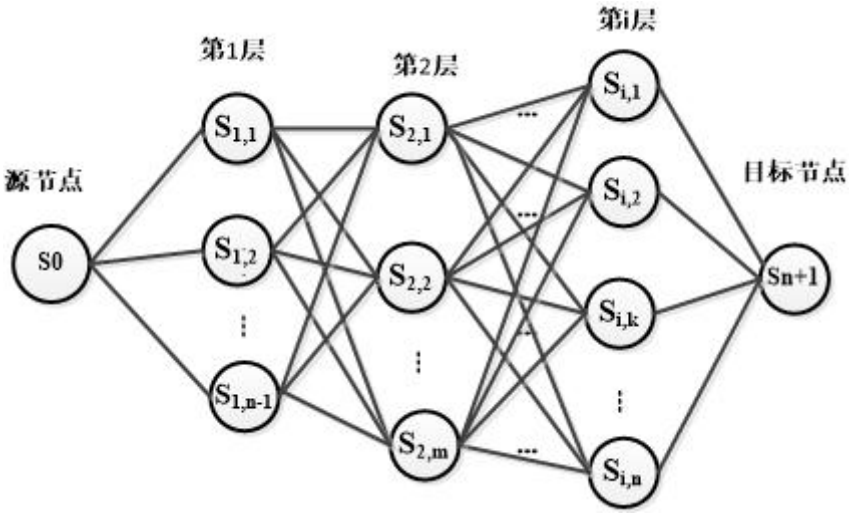

图 4-1 分层的有向加权图

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_{ii}^{c}和数据传输时间$T_{ii}^{s}(k)$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，这是一个经典的动态规划问题，因此我们可以使用维特比算法来获得最短路径 *SP*。详细的服务路径初始化策略如下：

步骤 1：基于性能预测模型可以构建出分层的加权有向图 G，图中的权重即为数据处理时间和数据传输时间之和。

步骤 2：从S_0出发，对于到第一层的节点，算出S_0到他们的距离$(S_0,S_{1,1})$，$(S_0,S_{1,2})$，…，$(S_0,S_{1,n})$，因此只有一步，这些都是S_0到他们的最短距离。

步骤 3：对于第二层节点要找出他们到S_0的最短距离，要通过第一层中的其中一个节点，所以对第二层的任意节点，我们都要计算 *n* 次，共计算n^2次，并一一比较得到到第二层的最短路径。

步骤 4：同步骤三，这一步我们也会共计算n^2次。

重复以上步骤直到任务结束，我们得到了服务路径初始化阶段的最优路径 *SP*。

4.3.2 服务搜索空间缩减原则

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$s_{i.}(\mathrm{i}=1,2,...,\mathrm{n}-1)$完成，下一个微服务实例$s_{i.}(\mathrm{i}=\mathrm{l}+1,...,\mathrm{n})$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使用 Skyline 服务思想，减少可选服务的搜索空间，提高微服务选择的执行效率，得到当前最优的服务路径。以下是详细的服务空间缩减原则步骤：

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 ≪ 该子任务对应微服务实例的个数。本文中，我们定义 m = 3。

4.3.3 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_1。如图 4-2，在G_1中，源节点即为第一层的节点代表已经执行的子任务P_1，该子任务在初始化阶段选择微服务实例$s_{1,2}$来提供服务。而后面的子任务对应的微服务实例，我们根据服务搜索空间缩减原则，只选择性能最好的前 3 个微服务实例，这样我们就得到了如图 4-2 的分层的有向带权子图G_1，之后我们在子图G_i中再次利用维特比算法，得到该子图对应的当前的最优服务路径SP_2。从而可以选择出执行第二个子任务P_2的微服务实例$s_{2,k}$，接下来我们根据更新后微服务实例的状态，依据服务搜索空间缩减原则，选择出最优的前 3 个微服务实例作为节点，就可以构造以微服务实例$s_{2,k}$为源节点的分层的有向带权子图G_2。依次类推，直到我们的任务执行完成。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。

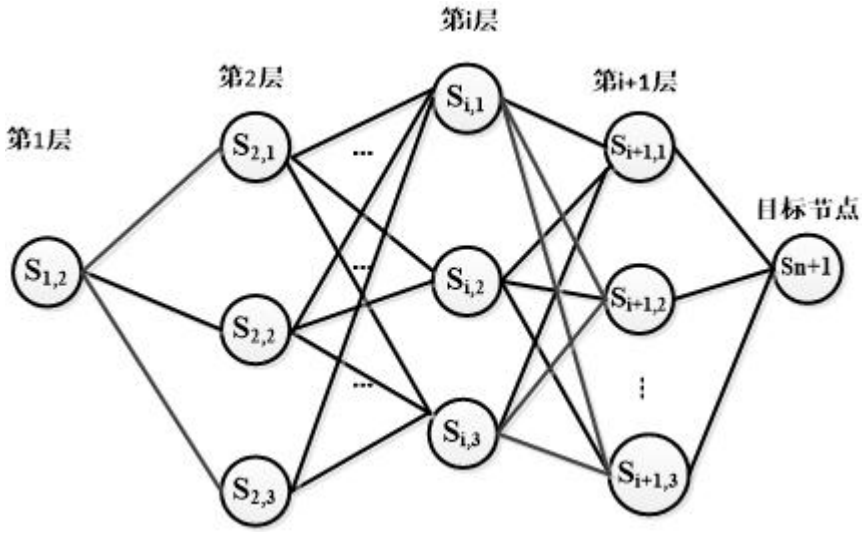

图 4-2 分层的有向加权子图 Gi

4.4 微服务路径选择算法实现

上一节我们介绍了微服务路径选择策略的详细步骤，这一节我们将介绍算法的实现。

现有的服务服务路径选择算法中，或者只考虑服务的历史的资源状况，或者只考虑没有任务的特征，所以当前服务计算领域的服务路径选择方法无法满足微服务平台中服务选择需求。因此本文利用机器学习的方法，使用线性回归模型，结合服务运行时资源特性、任务的特征以及输入数据本身的特性求得性能预测模型，在此基础上，提出性能感知的服务路径选择方法（PSPAS），PSPAS 的主要逻辑如算法 2 所示：

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. **while** i 不等于 n **do**
4. 首先计算从S_0到第一层节点的最短距离
5. 计算第一层 n 个节点到第二层 n 个节点的最短距离
6. 类似步骤 4，计算第 i 层 n 个节点到第 i+1 层 n 个节点的最短距离
7. $i = i + 1$
8. 重复以上步骤知道任务结束，得到当前最优路径SP_1
9. **end while**

初始化 $j = 1$

10. **while** 应用 P 没有结束 **do**
11. 在最优路径SP_1中调用微服务实例s_i执行子任务P_i
12. $j = j + 1$
13. 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
14. 基于路径搜索空间缩减操作的结果重构当前的子图G_i
15. 使用维特比算法重新选择出当前最优路径SP_j

16 **end while**

首先通过性能预测模型构建分层的有向加权图 G，之后再根据维特比算法得到初始化的最优服务路径SP_1（第 3 行~第 9 行），当初始化路径完成后，需要根据微服务实例的状态对服务路径动态更新（第 10 行~第 16 行），直到我们的任务执行完成。

4.5 本章小结

本章首先分析了当前的服务选择算法以及不足之处，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 系统实验及测试

5.1 实验环境配置

5.1.1 实验硬件环境

本文中实验硬件部分由 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，此次实验将全部物理节点配置为工作节点，命名方式为控制节点为 controller1 和 controller2，其他节点相应命名为 node1~node8，每一个节点配置用于支持微服务的 Docker 容器引擎。实验集群中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 实验软件配置

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 Ubuntu16.04.1 LTS 操作系统以及相应的微服务引擎。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 Ubuntu16.04.1 LTS 桌面版；

（2）程序开发环境： Webstorm，Vim，Docker；

（3）计算机视觉库：OpenCV 2.4.9；

（4）数据库：Mysql

5.1.3 实验数据说明

本文中服务划分算法中的数据我们是部署在中国福州的真实的视频监控系统中获取到的。性能预测模型中的数据集，是使用目标追踪算法实时获取到的，共获取了 1000 个视频文件，这些文件有不同的分辨率，不同的帧率以及时长；其中 700 个视频文件用于训练数据集，剩余 300 个视频文件用于测试数据集。

5.1.4 实验中子任务实现

为了测试我们算法的性能，我们实现了两个典型的视频处理任务：视频浓缩和目标检测。这两个任务包含的子任务和详细处理流程如下：每个视频子任务都是根据一个预先定义的工作流程执行的，其中有单独的输入和输出，并且子任务之间有固定的依赖关系。如图 5-1 所示，视频浓缩应用主要包括以下几个子任务：视频数据输入，GMM 背景建模，三帧差分，通道提取，通道优化，拼接、浓缩视频等。

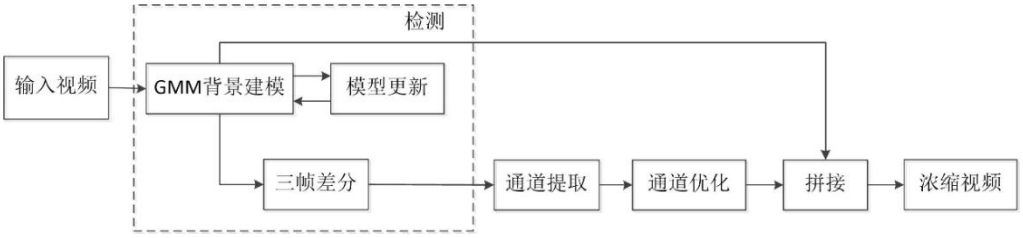

图 5-1 视频浓缩流程图

如图 5-2 所示，目标检测应用主要包括以下几个子任务：视频数据输入，GMM 背景建模，三帧差分，预处理，高斯模型，DOG 金字塔，尺度空间极值检测，提取特征向量，目标跟踪等。

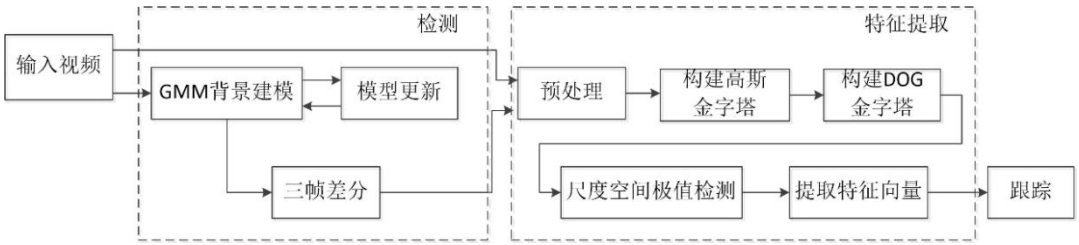

图 5-2 目标检测流程图

5.2 算法效果测试

本节主要基于实现的视频浓缩服务功能和目标跟踪服务对实验环境以及算法效果进行验证。

5.2.1 实验环境测试

为了验证平台中视频浓缩服务和目标跟踪服务的有效性，本文使用的监控视频数据均来自实验室监控视频系统，视频大小大约 10GB，视频本身采用编码格式为 H.264，码率为 25fps，视频分辨率为 1920*1080，总时长为 10 分钟。实验中将 1GB 的视频文件放置在 Controller 节点中，启动 Docker 计算引擎，拉取视频浓缩服务镜像并生成视频浓缩服务实例，之后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。

图 5-1 原视频文件中的画面

如图 5-1 是我们原视频文件中的画面，可以看到，该画面存在很多冗余信息，在我们进行目标提取时，检索的速度可能会受影响，因此我们使用视频浓缩技术将原始视频浓缩成比较短的视频文件。

如图 5-2 所示，是经过视频浓缩技术处理的视频文件的画面，与图 5-1 的视频文件的画面对比之后可以知道，视频浓缩技术处理后的视频文件，减去了大量的冗余信息，画面中的有效信息的密集程度在一定程度上有了很大提高，从而得出我们视频浓缩任务的有效性。

综上，我们使用同样的方法验证了目标检测服务的有效性。

图 5-2 视频浓缩后的画面

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。为了避免微服务平台中其他不确定因素对实验结果的潜在影响，在保证实验条件不变的情况下，我们选择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。

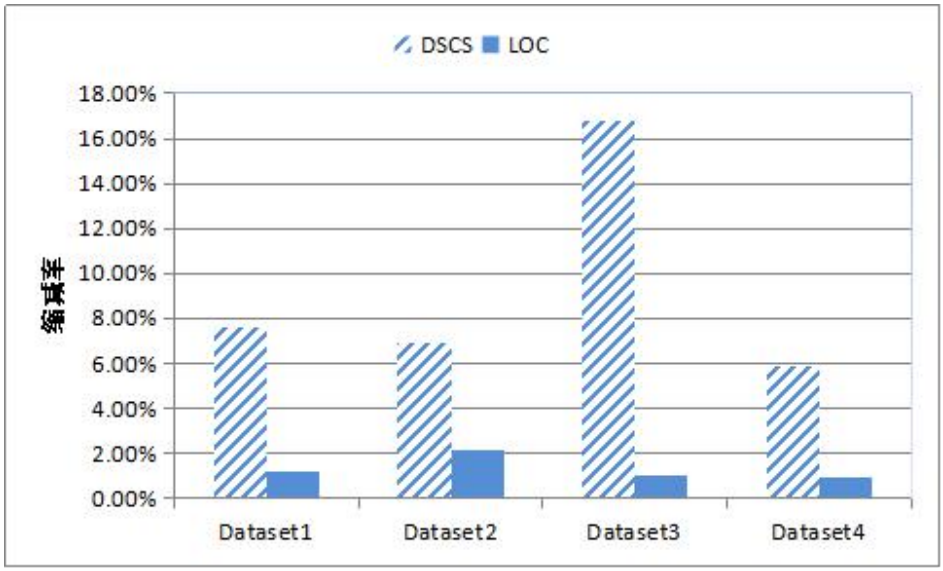

图 5-1 DSCS 和 LOC 算法的缩减率对比图

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：

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

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。图中的横坐标代表四组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，我们选择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。

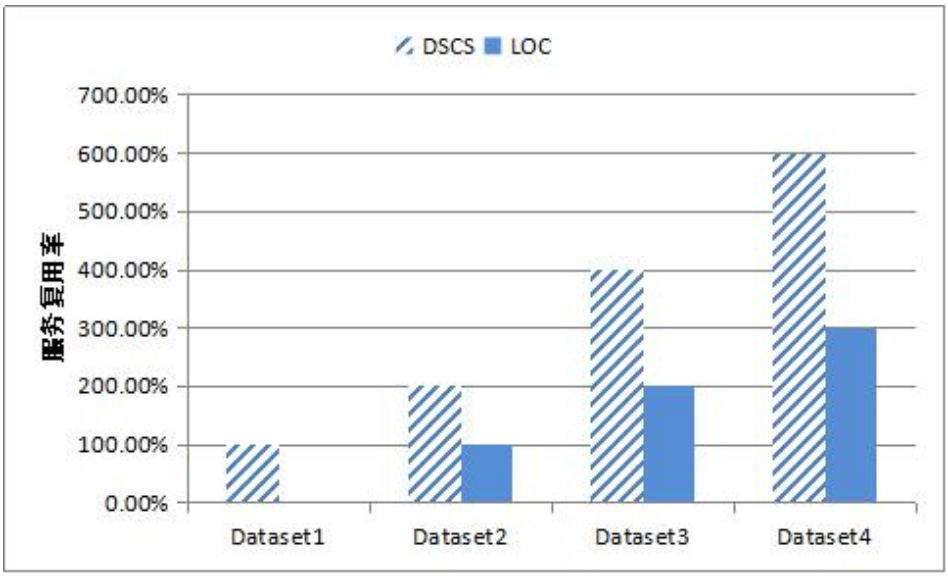

图 5-2 DSCS 和 LOC 算法的服务复用率对比图

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 5-3 所示：

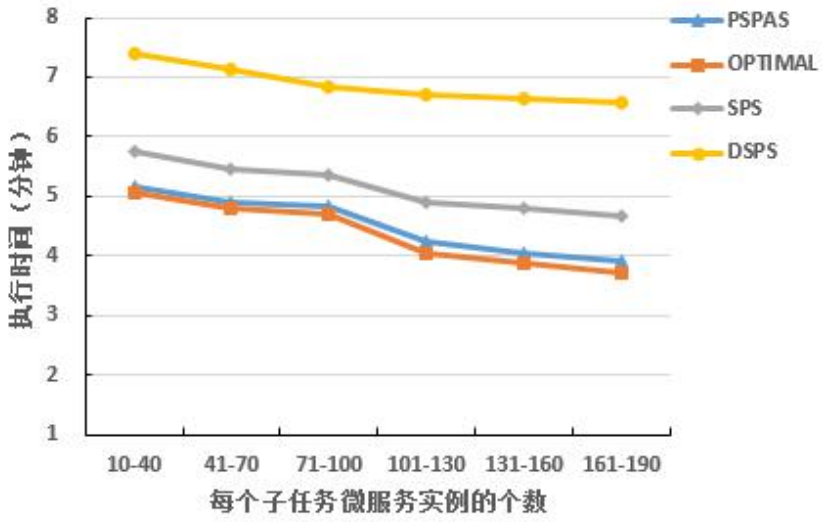

图 5-3 当微服务资源轻微抖动时执行时间变化图

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 能够动态的更新服务路径，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。

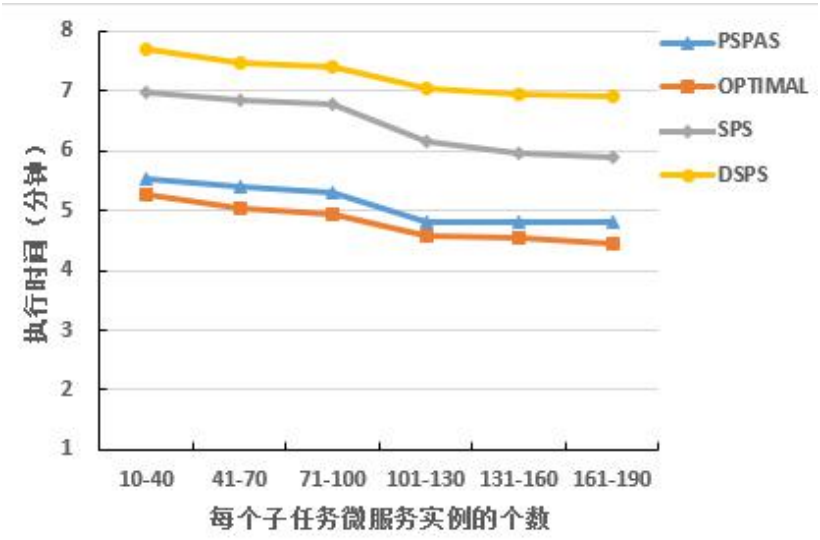

图 5-4 当微服务资源剧烈抖动时执行时间变化图

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。

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 5-5 所示：

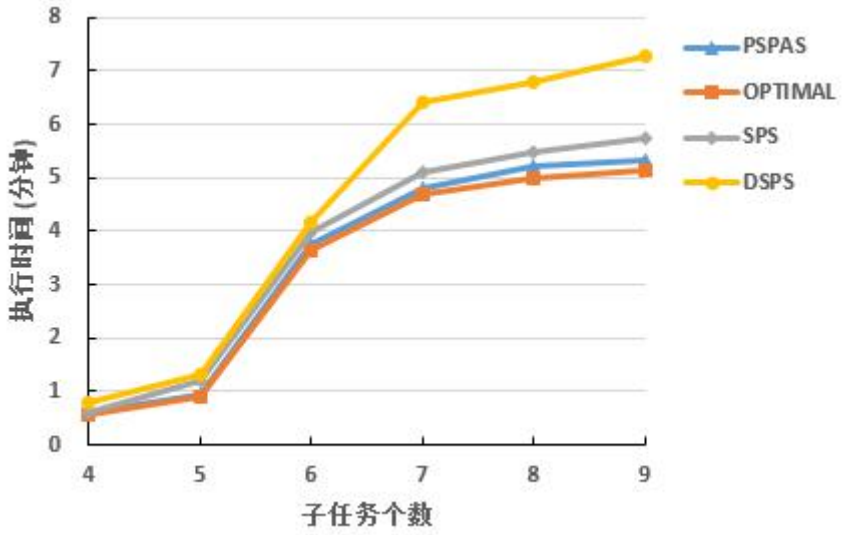

图 5-5 子任务个数改变时执行时间变化图

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。

5.3 本章小结

本章首先详细介绍了平台的软硬件的配置环境，数据来源以及实验中使用的服务的具体实现，然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，产生一系列问题如架构臃肿、业务逻辑复杂、数据流向复杂等。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中应用的特性，提出了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 [EB/OL]. http://martinfowler.com/articles/micro-services.html, 2014.

[2] Thones J. Microservices [J]. IEEE Software, 2015, 32(1): 116–116.

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

[5] Marcus A, Maletic J I. Identification of high-level concept clones in source code [C]. // Annual International Conference on Automated Software Engineering, IEEE, 2001: 107-114.

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

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

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

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

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

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

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

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

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

[14] Eberhard Wolff. What Are Microservices [M]. 2017.

[15] Mohsen A, Amjad I. Requirements Reconciliation for Scalable and Secure Microservice (De)composition [C]. // IEEE 24th International Requirements Engineering Conference Workshop, IEEE, 2016: 230-242.

[16] Asik T. Policy Enforcement upon Software Based on Microservice Architecture [C]. // IEEE computer society, IEEE, 2017, pp: 283–287.

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

[18] Villamizar M, Garces O, Ochoa L et al. Infrastructure Cost Comparison of Running Web Applications in the Cloud Using AWS Lambda and Monolithic and Microservice Architectures [C]. // 2016 16th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, IEEE/ACM, 2016: 179–182.

[19] Kalex U. Business Capability Management: Your Key to the Business Board Room [M].

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

[21] Li H, Wu Z. Research on distributed architecture based on SOA [C]. // International Conference on Communication Software and Networks, IEEE, 2009: 670-674.

[22] Zhang Y Y, Jiao J X. An associative classification-based recommendation system for personalization in B2C e-commerce applications [C]. // Expert Systems With Applications, IEEE, 2006: 33.

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

[24] Jive J. Jdon Framework [EB/OL]. http://www.jdon.com/jdonframework/.

[25] Goldston R L, Son J Y. Similarity [J]. Psychological Review, 2004, 100: 254-278.

[26] Li M, Chen X, Xin M L, et al. The Simility Metric [J]. IEEE Transactions on Information Theory, 2003, 50(12): 863-872.

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

[28] Osgood C E. The nature and measurement of meaning [J]. Psychological Bulletin, 1952, 49(3): 197-237.

[29] Lee MD. Algorithms for Representing Similarity Data [M]. 1999.

[30] Landauer T K, Dumais S T C. Solution to Plato's Problem: The Latent Semantic Analysis Theory of Acquisition, Induction and Representation of Knowledge [J]. Psychological Review, 1997, 104(2): 211-240.

[31] Tversky A. Features of Similarity [J]. Psychological Review, 1977, 84(4): 327-352.

[32] Santini S, Jain R. Similarity Measures [J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1999, 21(9): 871-883.

[33] Liu Y, Yang Y. Semantic Web Service Discovery Based on Text Clustering and Concept Similarity [J]. Computer Science, 2013, 11: 211-214.

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

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

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

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

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

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

[40] Zeng L Z, Benatallah B, Ngu A H H, et al. QoS-Aware Middleware for Web Services Composition [J]. // IEEE Transaction on Software Engineering, IEEE, 2004, 3(4): 311-327.

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

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

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

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

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

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

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

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

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

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

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

[52] Narayanan S, McIlraith S, Simulation, verification and automated composition of web services [C]. // WWW, ACM, 2002: 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 [J]. IEEE Transaction Services Computing, 2016, 9(5): 806-817.

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

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

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

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

[58] Schaeffer S E. Graph clustering [J]. Computer Science Review, 2007, 1(1): 27-64.

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.