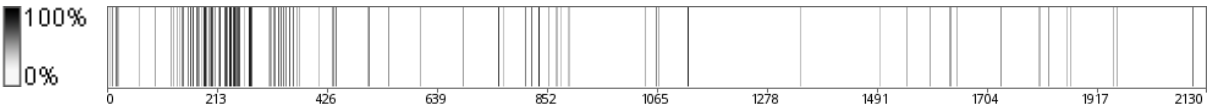
PaperPass旗舰版检测报告
简明打印版

比对结果(相似度):

总　体：11% (总体相似度是指本地库、互联网的综合对比结果)
本地库： 6% (本地库相似度是指论文与学术期刊、学位论文、会议论文、图书数据库的对比结果)
期刊库： 4% (期刊库相似度是指论文与学术期刊库的比对结果)
学位库： 4% (学位库相似度是指论文与学位论文库的比对结果)
会议库： 1% (会议库相似度是指论文与会议论文库的比对结果)
图书库： 3% (图书库相似度是指论文与图书库的比对结果)
互联网： 5% (互联网相似度是指论文与互联网资源的比对结果)

编　号：5C7B9A1CC907D6S0Q
版　本：旗舰版
标　题：[降重]微服务平台中服务划分和选择策略研究与应用v2
作　者：杨宁
长　度：43244字符(不计空格)
句子数：2130 句
时　间：2019-3-3 17:10:52
比对库：学术期刊、学位论文、会议论文、书籍数据、互联网资源
查真伪：http://www.paperpass.com/check

句子相似度分布图:



本地库相似资源列表(学术期刊、学位论文、会议论文、书籍数据):
暂无本地库相似资源

互联网相似资源列表：
1.相似度： 1% 标题：《一、微服务架构概述 - Vip灬cnblog -...》
https://www.cnblogs.com/aGboke/p/9051376.html
2.相似度： 1% 标题：《单体架构与微服务架构 - Dream it Po...》
https://blog.csdn.net/hu_zhiting/article/details/77036914
3.相似度： 1% 标题：《加权GN算法简介 - u012483487的专栏...》
https://blog.csdn.net/u012483487/article/details/50761900/
4.相似度： 1% 标题：《加权GN算法简介 - u012483487的专栏...》
https://blog.csdn.net/u012483487/article/details/50761900
5.相似度： 1% 标题：《机器学习5种回归方法及其属性-探索世界，...》
https://blog.csdn.net/ChenVast/article/details/81660844?utm_source=blogxgwz6
6.相似度： 1% 标题：《机器学习5种回归方法及其属性-探索世界，...》
https://blog.csdn.net/ChenVast/article/details/81660844
7.相似度： 1% 标题：《微服务架构 vs. SOA架构 - chszs的...》
https://blog.csdn.net/chszs/article/details/78515231

8.相似度： 1% 标题：《微服务架构 vs. SOA架构 - 江召伟 - ...》
https://www.cnblogs.com/jiangzhaowei/p/9168837.html
9.相似度： 1% 标题：《微服务理论之五：微服务架构 vs. SOA架构 ...》
https://www.cnblogs.com/duanxz/p/3710688.html
10.相似度： 1% 标题：《五种回归方法的比较 - Jin_liang - ...》
https://www.cnblogs.com/jin-liang/p/9551759.html
11.相似度： 1% 标题：《加权GN算法的Java实现 - u0124834...》
https://blog.csdn.net/u012483487/article/details/50761938
12.相似度： 1% 标题：《单体应用架构和微服务架构的区别 - qinaye...》
https://blog.csdn.net/qinaye/article/details/82840625
13.相似度： 1% 标题：《weixin_36431280的博客》
https://blog.csdn.net/weixin_36431280/rss/list

全文简明报告:

第一章绪论

1.1 研究背景和意义

随着 SOA[21]、 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， 代码库也在迅速扩展，最终的单个应用程序变得越来越臃肿。 一系列问题，如可维护性，灵活性降低和维护成本较高，这将给整个云平台的开发，维护，部署和后续升级带来很大困难。 {40%：因此，单一架构很难满足快速变化的互联网时代的需求。}

{47%：上述问题的解决方案是将微服务架构应用于云平台以拆分和重组现有业务系统。}将原始系统拆分为单独的模块，以降低系统的整体复杂性，代码的冗余以及各子系统和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。 {58%：微服务是一种非常流行的系统架构解决方案。} 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， 然而，微服务并不完美，微服务的使用给我们的工作带来了一些挑战。 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。{57%：微服务的粒度很难并且经常成为辩论的焦点。} {68%：您应该使用合理的粒度来划分微服务，而不是盲目地最小化服务。} {81%：代码量不能作为微服务划分的基础，因为不同微服务的业务复杂性不同，代码量也不同。} {82%：在微服务的设计阶段，应确定边界。} {57%：微服务应该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]， {42%：因此，服务分区技术在微服务领域的应用需要进一步研究。} 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，即使具有相同的任务，不同的数据也可以导致不同的应用程序处理效率。

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，{65%：减少服务的执行时间，提高应用程序的执行效率。}

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。{70%：在微服务的设计阶段，应该确定边界[4]。} 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。 {42%：该方法可以管理服务之间的相关性，并显着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； 田华坦等人。 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。 {40%：虽然这些方法仅解决了基于静态QoS的服务选择问题，但它们可以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，{62%：获得最佳服务路径，提高了应用程序的执行效率。}

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，{43%：无论从服务能力的角度考虑服务的边界，确定服务的粒度都是不好的。}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。{51%：为了解决难点2，本文采用微服务平台的应用，}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

本文提出了一种基于领域驱动设计模型思想的微服务平台微服务划分方法。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。

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， {42%：它包括基于静态 Qos的服务选择路径和基于动态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，通过算法的比较，给出了适合我们的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， {49%：然后介绍了基于时间预测模型的初始化路径选择策略，并详细介绍了该策略的实现过程。} {50%：最后，介绍了微服务路径的动态更新策略，并详细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， {46%：并通过实验测试了服务路径选择策略的性能，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，首先详细介绍了单体式软件架构，重点指出该架构的一些重要问题， {47%：这导致了今年出现的微服务架构的出现，说明了架构的优势。} 然后介绍了

2.1微服务相关技术

2.1.1单体式软件架构

{41%：许多项目都以单个应用程序开始，单个应用程序更易于部署和测试。} 但是，随着需求的不断增加，越来越多的人加入了开发团队，代码库也在迅速扩展。 慢慢地，单体的应用变得越来越膨胀，可维护性，柔韧性逐渐降低，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： {58%：在较大的单细胞应用程序中，整个项目包含许多模块，模块的边界模糊，依赖性不清晰，} 代码质量不均匀，并且混乱堆积在一起……… 整个项目非常复杂。 {65%：每次更改代码时，您都会感到害怕，甚至添加一个简单的功能，或者修改错误会带来隐藏的缺陷。}

（2）技术债务： 随着时间的推移，需求的变化和人员的变化，应用的技术债务将逐渐形成和积累。 “不坏不修（Not broken，don’t fix）” 这在软件开发中非常普遍，而这种想法在单个应用程序中更是如此。 {85%：已使用的系统设计或代码难以修改，

因为应用程序中的其他模块可能以意外的方式使用它。}

（3）部署频率低： 随着代码的增长，构建和部署的时间也在增长。 {70%：在单个应用程序中，每个功能的更改或缺陷的修复都可能导致需要重新部署整个应用程序。} {64%：完全部署的方法需要很长时间，影响范围大，风险高，这使得单个应用项目的部署频率降低。} 较低的部署频率导致两个版本之间的大量功能更改和错误修复，并且错误率相对较高。

（4）可靠性差： {40%：应用程序错误（例如无限循环）可能导致整个应用程序崩溃。}

（5）扩展能力受限： {78%：单个应用程序只能作为一个整体进行扩展，不能根据业务模块的需要进行扩展。} {45%：例如，应用程序中的某些模块是计算密集型的，需要强大的CPU;} 有的模块则是IO密集型的，需要更大的内存。 由于这些模块一起部署，因此必须在硬件选择上妥协。

（6）阻碍技术创新： {81%：单个应用程序通常使用统一的技术平台或解决方案来解决所有问题，团队的每个成员都必须使用相同的开发语言和框架。} {94%：引入新框架或新技术平台将非常困难。} 例如，使用Struts 2构建了100万行代码的单个应用程序，如果要切换到Spring MVC，毫无疑问切换成本非常高。 综上所述，随着业务需求的发展和功能的不断增加，单一架构难以满足互联网时代快速变化的需求。

2.1.2SOA架构

上小节介绍了传统的单体式软件架构的不足，早在二十世纪90年代，基于接口/组件的SOA软件架构[21]获得了很不错的发展势头。 {82%：面向服务的体系结构（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。} {68%：通信可以像数据传输一样简单，也可以是两个或多个相互协调的服务。} {60%：面向服务的体系结构不是关于如何模块化应用程序，而是关于如何通过集成分布式，单独维护和部署的软件组件来形成应用程序。} {61%：这些是通过技术和标准实现的，这些技术和标准使组件能够通过技术和标准在网络（尤其是IP网络）上更轻松地进行通信和协作。} SOA架构中有两个主要角色： {51%：服务提供商和服务消费者以及软件代理可以扮演这两个角色。} {70%：Consumer层是用户与SOA交互的点，Provider层由SOA框架内的所有服务组成。} 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）拆分模块并使用接口通信来减少模块之间的耦合。

{42%：（2）将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，可以调用其他系统的接口。

（4）可以灵活的进行分布式部署。

但是SOA架构也有一些问题：

{72%：（1）系统和服务之间的界限模糊，不利于开发和维护。}

{85%：（2）虽然使用了ESB，但服务的接口协议并不固定，种类繁多，不利于系统维护。}

{70%：（3）提取的服务的粒度太大，系统与服务之间的耦合度很高。}

{92%：因此，SOA更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。}{80%：也就是说，小型应用程序不适合 SOA架构，因为它们不需要消息中间件组件，而是需要微服务架构，} {80%：另一方面，它更适合较小和良好的分割，基于网络的系统。}{55%：在下一节中，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热门话题之一，微服务架构是一种互联网应用服务的软件架构， {52%：主要用于服务器软件开发的 Internet应用服务。} 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。 {51%：2014年，学者Martin Fowler正式提出了微服务架构的概念[1]：}{72%：微服务架构风格是将单个应用程序开发到一小组服务中的一种方式，每个服务都在自己的进程中运行。} {81%：服务间通信使用轻量级通信机制（通常使用HTTP资源API）。}{74%：这些服务围绕业务功能构建，可以通过完全自动化的部署机制独立部署。} {71%：这些服务共享最小的集中管理，可以使用不同的语言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。

{48%：图2-1单一体系结构，SOA体系结构和微服务体系结构}

微服务架构具备以下特性：

（1）易于开发和维护： {56%：微服务仅关注特定的业务功能，因此它具有明确的业务和少量代码。} {100%：开发和维护单个微服务相对简单。} 整个应用程序由几个微服务构建，因此整个应用程序保持在受控状态。

（2）单个微服务启动较快： 单个微服务代码更便宜，因此启动速度更快。

（3）局部修改容易部署： {52%：只要修改了单个应用程序，就必须重新部署整个应用程序。} {48%：通常，要修改微服务，只需重新部署服务即可。} {64%：每项服务都是为独立业务而开发的，而微服务只关注特定功能。}

（4）技术栈不受限： {85%：在微服务架构中，可以结合项目业务和团队的特征合理地选择技术堆栈。} 例如，听写服务可以使用关系数据库MySQL； 某些微服务有图形计算的需求，可以使用Neo4j； 可以根据需要使用Java开发一些微服务，并使用Node.js开发一些微服务。

（5）按需伸缩： 可根据需求，实现细粒度的扩展。 {58%：例如，系统中的微服务遇到瓶颈，可以结合微服务的业务特性来增加内存，升级CPU或添加节点。}

总之，单一体系结构和SOA体系结构的缺点正是微服务的优势所在。 本文认为微服务架构在构建云服务中具有更明显的优势，这是因为每个微服务可独立运行在自己的进程里，{68%：并且一系列独立运行的微服务一起构建整个系统，每个服务都是一个独立的开发业务，} 而微服务只关注特定的功能。 {85%：微服务通过一些轻量级的通信机制进行通信。}除此之外，由于微服务的轻量级特性使云平台更易于管理、更容易实现负载均衡、减少资源碎片、提高资源利用率。

2.2机器学习相关技术

{58%：机器学习是一门涉及概率论，统计学，计算机科学和软件工程的多学科学科。}{64%：机器学习是指一组工具或方法，通过这些工具或方法，历史数据用于将机器“训练”和“学习”成模式或模式。} 并建立预测未来结果的模型。

{69%：机器学习涉及两种学习方法（图2-2）：} 主导用于决策支持的监督学习使用经过训练的历史数据来执行对新数据的识别的预测。 {94%：监督学习方法主要包括分类和回归;} {91%：无监督学习，主要用于知识发现，它在历史数据中发现隐藏的模式或内部结构。} {100%：无监督学习方法主要包括聚类。}

图 2-2 机器学习

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。 {52%：在统计学中，回归分析是一种统计分析方法，用于确定两个或多个变量之间的定量关系。} {75%：应用非常广泛，回归分析根据所涉及的变量分为单向回归和多元回归分析;} {81%：根据自变量的数量，可分为简单回归分析和多元回归分析;} {91%：根据自变量与因变量之间的关系，可以分为线性回归分析和非线性回归分析。} {79%：如果在回归分析中，只包括一个自变量和一个因变量，并且两者之间的关系可以用直线近似，这种回归分析称为线性回归分析。} {63%：如果回归分析中包含两个或多个自变量，并且自变量之间存在线性相关，则称为多元线性回归分析。} {76%：在大数据分析中，回归分析是一种预测建模技术，研究因变量（目标）和自变量（预测变量）之间的关系。} {85%：该技术通常用于预测分析，时间序列模型和变量之间的因果关系。}{67%：例如，研究驾驶员鲁莽驾驶与道路交通事故数量之间关系的最佳方法是返回。} {90%：机器学习中的回归问题属于监督学习的范畴。} {75%：回归问题的目标是为给定的D维输入变量x，并且每个输入向量x具有对应的值y，需要为新数据预测其对应的连续目标值t。}

{61%：如果要预测的值是连续的，则它是回归问题;} {73%：如果要预测的值是离散的，即一个标签，则它是分类问题。} 这个学习处理过程如图2-2。

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； {70%：拟合曲线，通常表示为y = h（x），称为假设模型假设;} {65%：训练集中的条目数称为要素的维度。}

图2-3 学习处理过程

回归分析方法的特点如下：

{41%：（1）在分析多因素模型时，回归分析更简单方便;}

（2）使用回归模型，只要模型和数据相同，就可以通过标准统计方法计算独特的结果，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，不同分析师得出的拟合曲线可能不同;

（3）回归分析可以准确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;

{61%：（4）在回归分析的情况下，由于实际的一个变量仅受单个因子的影响，因此需要注意模型的合适范围。} 因此，确实存在单向回归分析的应用。 {42%：多元回归分析更适合实际经济问题，并在与多种因素结合使用时使用。}

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，实验表明，该训练方法可以准确，简便地学习预测模型。

2.2.2线性回归与非线性回归对比

{93%：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产生特定结果相关。} {91%：线性回归是指完全由线性变量组成的回归模型。} {88%：从简单的情况开始，单变量线性回归是一种使用线性模型（即线）模拟单个输入自变量（特征变量）和输出变量之间关系的技术。} {94%：更一般的情况是多元线性回归，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。} {100%：模型保持线性，输出是输入变量的线性组合。} {69%：根据自变量和因变量之间的函数表达式是线性还是非线性，它分为线性回归和非线性回归。} 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

+…+

a

n

*

X

n

+b (2-1)

其中

a

n

是系数，

X

n

是变量，b是偏差。 {94%：您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：

{73%：（1）建模快速简便，当要建模的关系不是很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

{86%：（3）线性回归对异常值非常敏感。}

{41%：自变量和因变量之间的函数表达式的非线性反映在至少一个变量的索引不是1（幂函数，指数函数，对数函数，S函数等）的事实中。} {63%：在分析非线性回归问题时，非线性回归方程通常转化为线性回归方程。} {56%：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。} {57%：当非线性回归是线性回归时，主要是确定变量之间存在的非线性关系的类型。} 该方法可以基于专业知识的理论推导和直接生产或测试数据，并从点分布的特征中选择合适的曲线类型。 {46%：根据线性回归和非线性回归的特点，本文选择线性回归建立第4章的绩效预测模型。} 实验证明，我们的绩效预测模型能够准确预测服务的执行时间。

2.3最短路径算法

2.3.1迪杰斯特算法

{86%：Dijkstra算法是从一个顶点到其余顶点的最短路径算法，解决了有向图中的最短路径问题。} {54%：Dijkstra算法的主要特征是它从起始点延伸到外层，直到到达终点。} 该算法的主要思想如下：

{65%：该算法按路径长度的递增顺序生成：}

把顶点集合V分成两组：

（1）S： 找到的顶点集（最初只是源点V0）

（2）V-S=T： 尚未确定的顶点集合

按升序添加T到S中的顶点，确保：

{74%：（1）从源点V0到S中其他顶点的长度不大于从V0到T的任何顶点的最短路径长度}

（2）每个顶点对应一个距离值

S中顶点： 从V0到此顶点的长度

T中顶点： {70%：从V0到此顶点的最短路径长度，仅包括S中的顶点作为中间顶点}

依据： 可以证明V0到T中顶点Vk的，或是从V0到Vk的直接路径的权值； 或是从V0经S中顶点到Vk的路径权值之和。

2.3.2维特比算法

{65%：维特比算法是一种特殊但广泛使用的动态规划算法，用于栅栏网络（Lattice）有向图的最短路径问题。} 篱笆网络如图2-4。 围栅网络有向图的特征在于同一列中的多个节点并与前一列节点交织。

维特比算法的思想如下：

{63%：（1）从点 S开始，对于第一状态 X1的每个节点，假设存在 n1，并且计算从 S到它们的距离（ S，} X1 i），其中 X1 i代表任意状态1的节点。 因为只有一步，所以这些距离都是S到它们各自的最短距离。

（2）对于第二状态X2的所有节点，计算从S到它们的最短距离。 对于特征节点 X2 i，从 S到它的路径可以通过 n1的状态1中的任何节点 X1 i， 对应的路径长度就是 d(S， X2 i)= d(S， X1 i)+ d(X1 i， X2 i)。 {73%：由于j有n1种可能性，我们必须逐个计算它们才能找到最小值。} 即：

{81%：因此，对于第二状态的每个节点，需要n1次乘法计算。} 假定这个状态有n2个节

点，这些节点的距离计算一次，有O（n1 n2）计算。

（3）接下来，类似地，如上所述从第二状态到第三状态，并且到达最后状态，获得从整个网格的开始到结束的最短路径。 {78%：每个步骤的复杂度与两个相邻状态Si和Si + 1的节点数ni和ni + 1的乘积成正比，即O（ni ni + 1）}

{84%：（4）假设该隐马尔可夫链中节点数最多的状态有 D个节点，} {70%：即整个网格的宽度为 D，那么任何步骤的复杂性都不会超过 O（ D2）。}

图 2-4 篱笆网络

分析以上的算法的特点，维特比的主要思想就是知道到第i列所有节点

X

i

{48%：最短路径，然后到第 i+1列的最短路径等于第 j列节点的第 i列最} {49%：短路径+第 i列 j节点和第 i+1列之间的最小距离。}

{48%：Dijkstra算法是一种贪心算法，只能获得局部最优结果。} {40%：维特比算法是一种动态编程算法，每一步都是全局最优解，最终结果是全局最优。} 由于本文第四章中的到的有向加权图即为篱笆图，要求的结果为篱笆图中全局的最优路径，因此采用维特比算法。

2.4 本章小结

{45%：本章首先介绍与微服务相关的技术，包括单片架构，SOA架构和微服务框架相关技

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，最后介绍了本章用到的最短路径技术，包括Dijkstra算法和Viterbi算法。 本章节主要为后续章节的研究提供基础。

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独特业务时执行的操作。 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，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。 {42%：这将确保专注于一项责任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，　构造了无向加权图，并根据服务划分原则得到了最优划分结果。　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，有一个建模领域驱动设计（DDD）[22]。　构成领域驱动理论的理论和技术早已存在。　域驱动程序中的实体对应于OOP中的对象，OOP继承，封装和多态都适用于域驱动的开发。　域对象应设计为简单的类或接口。　该思想中主要有三个主要的概念。

（1）域，域与特定的开发技术无关，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。　核心领域是业务成功的关键。

（3）限界上下文。　边界上下文由系统，应用程序，业务服务和一组实现业务的复杂组件组成。　通用语言是上下文中每个字段的术语或短语。　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。　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。　{46%：相似性是任何两个对象之间存在的一般关系，相似性是相似性的定量表示。}　{52%：相似度计算是信息检索，数据挖掘，知识管理和人工智能领域的基本问题。}　{41%：随着本体的广泛应用，基于本体语义的相似度计算和应用已成为心理学与计算机科学交叉的重要课题。}

Dekang提出了一套广泛定义的相似度，告诉我们物体和物体之间的相似性与它们的共性和差异有关。　{57%：两个对象越常见，相似性越大，两个对象之间的差异越大，相似性越小。}　{51%：当两个对象是同一个对象时，相似性最大。}　当两个对象不相关时，相似度最小。

相似性模型的推广分为几何模型和特征比较模型[30]　[31]。　{67%：空间中的对象由点表示，点之间的距离用于反映对象之间的相似性。}　{43%：特征比较模型通过一组特征属性描述对象，并　且相似性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。　{49%：对象的特征提取是指提取对象的主要特征以表示对象。}　现有的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最有价值的特征信息。　但是，也可能遇到要素项的尺寸问题，因此要素选择需要进行特征选择。　{69%：特征选择需要根据一定的标准选择最能反映对象特征的相关特征，以减小特征维数，简化计算，提高相似度计算的准确性。}　2.对象的表示。　通常，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定量信息。 如果选择了某个要素，则该对象可以表示为向量。 3.相似度具体计算方法。 设T和T’为矩阵中两个对象的特征向量。 内积、Dice系数、余弦函数、Jaccard系数等。 {56%：本文使用余弦函数来计算两个对象的相似性。}

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

，

为加权无向图，图中每个顶点

∈ ，对应于单体阶段中的每一个代码文件

∈ 。 每一个边

∈ 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

与顶点

之间的权重

，

值越大，表示文件

与文件

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。 语义耦合策略是使用表示文件的标识符作为 tf- idf的输入来计算表示文件的 {49%：向量 X.然后通过计算两个矢量的余弦相似度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。 {43%：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算法的基本思想是使用相对于所有源节点的最大边缘媒体连续删除图的边缘。 {65%：然后，重新计算图形的剩余边缘相对于所有源节点的边缘，重复该过程，直到删除图形中的所有边缘。} 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

2

–

2

,

(3-6)

公式中,

{60%：对于图的邻接矩阵的元素，如果i和j由两个节点连接，那么}

{85%：是边的权重，否则等于0，δ是隶属函数，当节点i和j属于同一个微服务时，隶属函数为1，否则为0;}　M=0.5，

为加权无向图中边的权重之和。

{52%：对于节点的点权重，求和连接矩阵的第i行。}

该算法最关键的部分是计算图中的边缘媒体。　在本文中，我们使用最短路径边缘中值

方法来测量边缘中值。 {56%：具体而言，它指的是从源节点S通过边缘的最短路径的数量。} 对所有可能的源节点重复相同的计算，并为每个不同的节点获得相同的计算对源节点的边缘介体求和， {60%：得到的求和是边缘相对于所有源节点的边缘数。}

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。

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

=

1

,

2

, … ,

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

的词组

以及代表代码文件

的词组

的并集列表T，

=

∪

j

=

1

，

2

，… ，

(3-1)

接下来是计算代表每个代码文件

的向量 X，向量 X的维度是词组列表 的长度，向量的第 k个元素的是代码文件短语中短语列表中第k个元素的tf-idf值。 {45%：tf-idf值表示整个联合列表T中第k个元素（第k个短语）的重要性。} 向量 中的元素

计算公式如下：

=

，

*

，

(3-2)

用同样的方法得到向量V中的元素，

表示两个词组的并集即：

=

,

(3-3)

词频

,

表示

在

中出现的频率的计算公式如下：

,

=

，

，

(3-4)

其中

，

表示

在

中出现的次数，

，

表示所有元素在

中出现的次数之和。

逆向文件频率

，

{58%：衡量一个词的普遍重要性的尺度。} {53%：可以通过将文件总数除以包含该单词的文件数，然后从基数10对数中得到结果商来获得特定单词的idf：}

，

=

log

(3-5)

=

，

表示两个词组集合的集合，n

表示包含词组

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：

A=

1，1

…

1，

⋮

⋱

⋮

，1

…

，

(3-6)

其中

，

表示节点

和节点

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，适当粒度的服务构件能使服务在各角度进行服务装配； {81%：通过松散耦合技术减少服务使用者和提供者之间的技术依赖性。}

（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， {41%：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（} {41%：加权无图聚类算法）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

1

根据公式得到每条边的权重eij

2.

根据公式得到所有连接边的边介数Bij

3.

边介数除以权重得到边权比

4.

whileedges不为空do

5.

if ∈ 的边权比最高

6.

移除e并将e存储起来

7.

ife有多条

8.

将具有最高边权比的多条e移除并存储

9.

计算此时的图的模块性 值并存储

10.

endwhile

11.

取得 的最大值

以及 值对应的边

12.

return

和e

13.

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：　{62%：忽略边缘的权重，并使用非特权网络计算网络中所有连接边缘的边数。}

,

;

,

=

(3-7)

第二步：　{68%：将边数除以相应边的权重，得到边重量比}

,

;

,

=

,

,

(3-8)

第三步： {89%：找到边权比最高的边将它移除，并计算图的模块性 函数，在计算中当边权比最高的边由多条时，} {92%：同时移除这些边，并将此时移除的边和 值进行存储；}

第四步： {75%：重复步骤（1），（2），直到图中的所有边缘都被移除;}

第五步： {95%：GN算法划分结束后，取出 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， 基于所提出的性能预测模型，提出了一种新的性能感知服务路径选择算法。

4.1传统的服务选择算法概述

{49%：目前，国内外对服务选择算法的研究很多。} {43%：要考虑功能需求，还要关注服务的非功能属性，包括成本，可用性，响应时间和吞吐量。} 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

{44%：目前，在服务计算领域，QoS是一个重要的标准，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， 并使用混合证书计划模型来选择最佳候选服务。 {56%：但是，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。 由于此关联将影响QoS值，因此在生成具有最佳QoS值的组合服务时，在考虑QoS相关性的同时研究如何选择合适的候选服务是很重要的。 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依赖性和修剪非最佳

候选服务的服务。 {44%：最后，实验表明，该方法可以管理服务之间的复杂关联，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。 {43%：在执行过程中，从微服务实例池中选择并按顺序执行的微服务实例形成微服务路径。} 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，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。 在执行应用程序中的子任务之后，后续服务实例的QoS将随时间动态变化。 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]， {41%：文献[11][12]提出了云计算平台中的性能感知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。 王等人。 他们运用博弈论来决定服务优化方向。 彭等人。 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。 {42%：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。 {41%：此外，为了优化服务选择并减少服务选择所花费的时间，Skyline服务还用于缩短服务选择的计算时间。} Alrifai等。

{40%：这些方法选择适当的服务来创建最佳服务组合，并根据服务交付平台中的各种变化动态地改变最佳服务组合。} 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， 它可以自动地自动更新服务路径，确保有效的应用程序执行效率。

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， =

=1，2，… ，

。 我们定义微服务类集合 =

=1，2，… ，

。 每个微服务类Si由所有的微服务实例

（j=1，2，… ，

j

i，

j

i

代表微服务类

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

由微服务类

中的一个微服务实例

.

执行。 子任务

的执行时间

定义为：

=

+

(4-1)

其中

是微服务实例

.

数据处理的执行时间,

是上行微服务实例

-1.

到下行微服务实例

.

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

1.

，

2.

，… ，

.

，其中

.

是微服务类

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：

=

=1

(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。 {41%：为了优化服务路径选择，我们需要量化每个微服务实例。}

即子任务

的执行时间

。 根据公式(4-1)可以得出

为数据执行时间

与数据传输时间

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。 {47%：为此，我们需要建立一个精确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

{46%：第二部分是获取数据传输时间的数据传输时间模型。}

，性能预测模型为两部分之和即

。

数据执行时间

由数据处理时间模型得到。 {61%：每个微服务实例的数据执行时间}

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。 {48%：因此，我们使用线性回归模型来表示数据处理时间模型。} {41%：本文使用机器学习方法训练线性回归模型中的参数 W.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

{57%：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

上运行的视频子任务

由向量 =

1，… ，

∈

表示，

{46%：对于子任务的特征值，m表示预测模型中的特征值的数量。} 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， {56%：例如， CPU核心数， CPU时钟频率， CPU占用率，内存使用率和内存带宽。} 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。 {43%：在我们的回归预测模型中，为了获得更好的线性回归模型，我们采用每个特征值}

2

。 我们的回归函数如下：

，

=

=1

2

(4-3)

其中 =

0

,

1

…

是模型的参数向量。 {42%：在视频任务处理流水线中，每个子任务的输入（不包括第一个子任务）是前一个子任务的输出，而不是原始视频文件。} 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 ( )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。 {100%：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常规术语，如表达式（4-4），以防止出现经验风险， 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：

argmin

,

–

log

2

2

+

2

(4-4)

其中

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

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

。

数据传输时间

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

0

和目的子任务

+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∪

0

，

+1

。 另外我们定义了源微服务实例

0，0

和目的微服务实例

+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，

−1，

∈

−1

是微服务实例

，

的前一个微服务实例。 那么从微服务实例

−1，

到微服务实例

，

数据传输时间定义为：

=

(4-5)

其中

表示从子任务

-1

到子任务

总的数据传输量，

表示微服务实例

−1，

与

，

之间的数据传输率。

假设

0

是视频处理任务中的元数据。 对于在子任务

，我们定义

=1，… ，n，

0

=1，

+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：

=

−1

*

−1

*

(4-6)

{40%：可以通过离线实验和数据传输速率获得总输出数据与输入数据总量的比α}

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

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

在上一节中，我们获得了性能预测模型。 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

，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

∈

。 在图G中第

层的节点

与第

−1

层的节点

−1，

都对应有一个边

，边

是数据处理时间

T

ij

c

和数据传输时间

之和。 {57%：微服务路径的权重是路径中所有边缘的权重之和。} 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。

=

s

00

，

s

1.

，… ，

s

n.

，

s

n+1，0

(4-7)

{42%：总之，这是一个经典的动态编程问题，因此我们可以使用Viterbi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

0

出发，对于到第一层的节点，算出

0

到他们的距离

0

，

1，1

，

0

，

1，2

，…　，

0

，

1，n

，因此只有一步，这些都是

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

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

2

时间，并逐一获得第二层的最短路径。

步骤4： 同步骤三，这一步我们也会共计算

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，上述最短路径算法可以获得最优服务路径SP。 然而，当一个子任务

由微服务实例

.

i=1，2，… ，n-1

完成，下一个微服务实例

.

i=l+1，… ，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

=

(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

，

，

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）： {45%：是指在微服务任务流水线中，该子任务的时间率 ≥}

。

传输主导型子任务（TDS）： {45%：是指在微服务任务流水线中，该子任务的时间率 ≤}

。

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，我们可以忽略子任务的输入数据的传输时间对服务路径更新的执行效率的影响。

因此，当子任务

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

是计算主导型子任务，本文在微服务集合

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

是传输主导型子任务，本文在微服务集合

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

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

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

1

。 如图4-2，在

1

{44%：在中间，源节点是第一层的节点，表示已执行的子任务。}

1

，该子任务在初始化阶段选择微服务实例

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

1

，之后我们在子图

{40%：再次使用维特比算法，获得对应于子图的当前最佳服务路径。}

2

。 从而可以选择出执行第二个子任务

2

的微服务实例

2，

，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

2，

为源节点的分层的有向带权子图

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

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

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.

3.

4.

5.

6.

7.

8.

9.

初始化 =1

while i 不等于ndo

首先计算从

0

到第一层节点的最短距离

{46%：计算从第一层的n个节点到第二层的n个节点的最短距离}

{48%：与步骤4类似，计算从第i层的第n个节点到第i + 1层的第n个节点的最短距离}

= +1

重复上述步骤以了解任务结束并获取当前最佳路径。

1

end while

初始化 =1

10.

while应用P没有结束do

11.

在最优路径

1

中调用微服务实例

.

执行子任务

12.

= +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.

{44%：基于路径搜索空间缩减操作的结果重建当前子图}

15.

使用Viterbi算法重新选择当前最佳路径

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

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，{51%：每个机架中有5台服务器，两个机架中的交换机均支持1000 Mbit/ 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，包括控制节点在内的所有物理节点都配置为工作节点，删除了两个控制节点，其他节点依次命名为 node1~ node8。 {44%：每个节点都配置为支持用于微服务的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

{40%：通过htop获取每个节点的CPU和内存使用情况，任意两个节点之间的网络传输带宽由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， {49%：本文档选择

Linux操作系统环境中的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/ Linux3.13.0-32- generic x8664的 Ubuntu14.04.1 LTS操作系统以及版本为1.11.1的 Docker Engineer。 {52%：整个平台的开发在实验室的计算机上进行。}

（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

5.1.3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，这些文件具有不同的分辨率，不同的帧速率和持续时间; {47%：其中，700个视频文件用于训练数据集，其余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，视频丰富应用主要包括以下子任务： 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，目标跟踪应用主要包括以下子任务： 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目标追踪流程图

5.2实验中任务实现

5.2.1 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，本文基于在实验室的先前工作中实现的视频会聚算法实现了视频丰富服务请求的功能图像。 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/$TASK / rm -rf /build rm -rf ~/.m2/*
VOLUME /output
CMD [“java”, “-jar”, “/videoSynopsis.jar”, “$@”]
```

写入Dockerfile后，将通过以下脚本文件构建和上载映像。

```
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，划分的微服务被注册到服务注册组件。

与视频浓缩算法相同，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/$TASK / rm -rf /build rm -rf ~/.m2/*
VOLUME /output
CMD [“java”, “-jar”, “/objectTracking.jar”, “$@”]
```

在写入相同的Dockerfile之后，将通过以下脚本文件构建和上载映像。

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

5.3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对实验环境以及算法效果进行验证。

5.3.1实验环境测试

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10 GB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25 fps，总时长为10分钟。将1 GB的视频文件放置在 Controller节点启动 Docker计算引擎拉取视频集中服务图像生成视频丰富服务容器计算实例读取视频数据进行处理， {42%：生成的集中视频文件的总大小约为24 MB，持续时间约为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。

图5-2视频浓缩后的画面

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，与原始视频文件的画面形成鲜明对比， 经过视频浓缩技术处理的文件除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。

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。 {56%：表5-1显示了实验中数据集的详细信息：}

{58%：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5-1DSCS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：

csr

=

−

其中M代表当前的单体架构平台中的服务，

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

drr

=

−1

*100%

其中num

代表使用该服务s的应用的个数。

实验结果如图5-1所示。 {46%：图中的横坐标表示不同的视频数据集，纵坐标表示与每个数据集相对应的代码减少率。} 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。

在第二组实验中，我们还选择了相同的数据集，但每组数据都有不同的应用类型。 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5-2DSCS和LOC算法的服务复用率对比图

{50%：在该图中，横坐标表示不同的数据集，纵坐标表示每个数据集下的服务重用率。} {43%：可以看出，本文提出的DSCS算法可以有效地提高服务重用率。}

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，{44%：视频文件具有不同的分辨率，不同的文件大小，视频数据大约为10 GB。} 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： MSE

=E

−

2

其中

{54%：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，{43%：我们和默认的最优服务路径选择策略（ 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服务路径选择策略（ PSPAS）最接近 OPTIMAL的性能，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，{45%：来观察四种服务路径选择策略的任务执行时间。}

{42%：在验证过程中，本文使用1GB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：

{40%：在该图中，横坐标表示微服务实例的数量所在的间隔，纵坐标表示整个任务的执行时间（以分钟为单位）。}

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数量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：

图中的水平坐标表示不同的视频任务，每个任务具有不同数量的子任务，纵坐标表示整个视频任务的执行时间（以分钟为单位）。

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， 随着子任务数量的增加， PSPAS策略与 SPS策略和 DSPS策略之间的时差也会增加。 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子任务数更改时的执行时间更改图

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

随着业务系统的扩展和业务需求的不断变化，业务系统的功能变得越来越复杂，规模越来越大，而单一架构模型的应用架构已经不能满足业务需求。 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， {42%：提高应用程序执行效率尤为重要，也是当前研究中的热点问题。} 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，这将导致服务之间的服务耦合不良。 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单一架构，随着业务量的进一步增加，系统规模将迅速扩大。 这导致了一系列问题，例如膨胀的体系结构，复杂的业务逻辑和复杂的数据流。这些问题可能在整个系统的开发，维护，部署和后续升级中造成很大困难。 {46%：微服务架构近年来已成为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，视频丰富服务验证了该平台的高效任务处理效率。

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， {44%：将微服务实例的运行时特性与微服务任务的特征相结合，} 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。 {48%：对于当前的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。 {45%：本文重点研究基于微服务架构

的分布式处理云平台的性能优化。} 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，{41%：并进一步实现了更为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 forScalable and Secure Microservice(De)composition [C]. // IEEE 24th InternationalRequirements Engineering Conference Workshop, IEEE, 2016: 230-242.

[16]AsikT.Policy Enforcement upon Software Based on MicroserviceArchitecture [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 WebApplications in the Cloud Using AWS Lambda and Monolithic and Microservice Architectures[C].//201616 th IEEE/ ACM International Symposium on Cluster, Cloud, and Grid Computing, IEEE/ ACM, 2016: 179 - 182.

[19] KalexU.Business Capability Management: Your Key to the Business BoardRoom [M].

[20] PengS, 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 B2 C 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: 254-278.

[26] Li M, Chen X, Xin M L, et al.The Simility Metric [J].IEEE Transactions on Information Theory, 2003: 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: 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.

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

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

[33] Liu Y, Yang Y.Semantic Web Service Discovery Based on Text Clustering and Concept Similarity [J].Computer Science, 2013: 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: 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] Schae er S E.Graph clustering [J].Computer Science Review 1.1, 2007: 27-64.

[59] Kruchten P B.The 4+ 1 view model of architecture [C]. // IEEE Software, IEEE, 1995: 42 - 50.

致谢

{68%：时间过得很快，三年的毕业生生涯即将结束。} {49%：2016年9月，我很幸运地被北京邮电大学录取，并与计算机科学学院智能通信软件和多媒体北京市重点实验室建立了罕见的关系。} {57%：大学期间，他进入北京邮电大学继续攻读计算机科学与技术。} 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中的细心呵护和帮助。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， 这是我父母背后的痴迷支持，它已成为我坚强的后盾，让我前进。

最后，我要感谢所有评委和专家花时间在繁忙的日程中审阅我的文章。

{74%：作者在其学位期间发表的学术论文清单}

[1] Haitao Zhang, Ning Yang, Zhengjun Xu, Bingchang Tang, Huadong Ma. Microservice Based Video Cloud Platform with Performance- aware Service Path Selection.//2018 IEEE10 th International Conference Web Service(IEEE ICWS2018), San Francisco, USA, 2018.