**密级： 保密期限：**



**硕士学位论文**



**题目： 基于Openstack的视频监控云**

**资源调度技术研究与应用**

**学 号： 2013110568**

**姓 名： 杨贤达**

**专 业： 计算机科学与技术**

**导 师： 李文生**

**学 院： 计算机学院**

**2016 年 1 月 21 日**

独创性（或创新性）声明

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

申请学位论文与资料若有不实之处，本人承担一切相关责任。

本人签名： 日期：

关于论文使用授权的说明

学位论文作者完全了解北京邮电大学有关保留和使用学位论文的规定，即：研究生在校攻读学位期间论文工作的知识产权单位属北京邮电大学。学校有权保留并向国家有关部门或机构送交论文的复印件和磁盘，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，可以允许采用影印、缩印或其它复制手段保存、汇编学位论文。（保密的学位论文在解密后遵守此规定）

保密论文注释：本学位论文属于保密在年解密后适用本授权书。非保密论文注释：本学位论文不属于保密范围，适用本授权书。

本人签名： 日期：

导师签名： 日期：

基于Openstack的视频监控云资源调度技术研究与应用

摘 要

视频监控云是一种新兴的云计算应用模式，为大规模视频监控处理任务提供弹性的云资源管理能力。然而不同的视频处理任务会有不同的资源需求，从而导致云平台虚拟机配置的异构性，如果采用不合理的资源分配方式将会造成集群资源极大的浪费。

目前的视频监控云资源管理系统并没有充分利用物理机与虚拟机资源的异构性，而常用的一维资源调度准则并不能适应多维资源分配场景。因此我们研究面向视频监控云异构环境的多维资源调度方法。首先，我们针对视频监控云的应用需求，设计了基于Openstack的支持多区域部署的视频监控云资源管理系统。该系统在实现Openstack Java SDK服务的基础上，提供计算资源、网络资源以及存储资源的WEB管理服务，并通过调用Libvirt库的方式提供虚拟机的动态监控服务。其次，我们在分析了视频监控云资源特性的基础上，提出了面向视频监控云异构环境的主导资源优先分配（DRFA）算法，该算法在集群全局的角度进行资源分配，能够充分利用物理机资源与虚拟机资源的异构特性，并根据多维资源匹配标准选择合适的物理机。最后，我们实现了基于Openstack的视频监控云资源管理系统，并开源了Openstack的Java SDK。对DRFA算法的测试结果表明，DRFA能够充分利用虚拟机与物理机资源的异构特性，显著地提高了集群的资源利用率。

关键字：资源调度，视频监控云，异构性，云计算

RESEARCH AND IMPLEMENTATION OF RESOURCE ALLOCATION TECHNOLOGY FOR VIDEO SURVEILLANCE CLOUD BASED ON OPENSTACK

ABSTRACT

Video surveillance cloud is an emerging cloud computing paradigm which can provide the elastic resource management ability for large-scale surveillance video processing tasks. The video processing tasks usually require extensive computing resources, and different tasks have different resource configuration requirements, thereby resulting in the heterogeneous feature. It is challenging to find the optimal fine-grained resource configuration for various video processing tasks.

The present video surveillance cloud resource management system can not take full advantage of the features of physical server and virtual machines, and the single resource scheduling can not adapt to the multi-resource allocation scene. In this paper, we study how to map the heterogeneous virtual machine requests to the heterogeneous physical machines. First, we design a video surveillance cloud resource management system. The system is based on Openstack and designed for multi-region architecture to support the video surveillance application. On the basis of implementation of Openstack Java SDK, the system provide WEB management service for compute resource, network resource and storage resource. In addition, the system provide dynamic resource monitor service by calling Libvirt. Second, we propose a multi-resource virtual machine allocation algorithm named Dominant Resource First Allocation (DRFA). Our aim is to maximize the resource utilization in heterogeneous cloud computing environment. By computing the dominant resource under multiple resource dimensions, our proposed algorithm DRFA can make full advantage of the heterogeneous physical resources. Finally, we implement the cloud resource management system and conduct function testing. The experimental results show that our resource allocation approach outperforms other widely used approaches.

KEY WORDS: resource schedule, video surveillance cloud, heterogeneous, cloud computing

目录

[第一章 绪论 1](#_Toc444855181)

[1.1 研究背景和意义 1](#_Toc444855182)

[1.2 国内外研究现状 2](#_Toc444855183)

[1.3 论文的主要研究内容 3](#_Toc444855184)

[1.4 论文的组织结构 4](#_Toc444855185)

[第二章 相关技术介绍 5](#_Toc444855186)

[2.1 云计算技术 5](#_Toc444855187)

[2.1.1 云计算概念 5](#_Toc444855188)

[2.1.2 虚拟化技术 5](#_Toc444855189)

[2.2 Openstack技术 6](#_Toc444855190)

[2.2.1 Openstack概述 6](#_Toc444855191)

[2.2.2 Openstack关键组件 8](#_Toc444855192)

[2.3 资源调度相关技术 14](#_Toc444855193)

[2.4 本章小结 14](#_Toc444855194)

[第三章 视频监控云资源管理系统设计 15](#_Toc444855195)

[3.1 系统总体架构设计 15](#_Toc444855196)

[3.1.1 跨地域物理架构设计 15](#_Toc444855197)

[3.1.2 系统逻辑架构设计 16](#_Toc444855198)

[3.2 基于Java的OpenStack SDK服务设计 17](#_Toc444855199)

[3.2.1 API与SDK区别 17](#_Toc444855200)

[3.2.2 SDK服务设计 19](#_Toc444855201)

[3.3 云资源管理服务设计 20](#_Toc444855202)

[3.3.1 WEB管理服务设计 21](#_Toc444855203)

[3.3.2 虚拟机监控服务设计 22](#_Toc444855204)

[3.4 本章小结 24](#_Toc444855205)

[第四章 视频监控云多维资源调度方法 25](#_Toc444855206)

[4.1 Openstack资源调度问题研究 25](#_Toc444855207)

[4.1.1 Openstack资源调度工作流程 25](#_Toc444855208)

[4.1.2 Openstack资源调度策略 26](#_Toc444855209)

[4.2 视频监控云资源特性分析 27](#_Toc444855210)

[4.3 多维资源调度方法 28](#_Toc444855211)

[4.3.1 问题建模 28](#_Toc444855212)

[4.3.2 主导资源优先分配算法 30](#_Toc444855213)

[4.4 本章小结 33](#_Toc444855214)

[第五章 系统实现与测试分析 35](#_Toc444855215)

[5.1 基于Openstack的跨域物理架构实现 35](#_Toc444855216)

[5.2 视频云资源管理系统实现 37](#_Toc444855217)

[5.2.1 Openstack Java SDK实现 37](#_Toc444855218)

[5.2.2 WEB管理服务实现 40](#_Toc444855219)

[5.2.3 虚拟机监控服务实现 44](#_Toc444855220)

[5.3 功能测试与结果验证 49](#_Toc444855221)

[5.3.1系统环境 49](#_Toc444855222)

[5.3.2 功能测试 50](#_Toc444855223)

[5.3.3 结果验证 53](#_Toc444855224)

[5.4 本章小结 57](#_Toc444855225)

[第六章 总结与展望 59](#_Toc444855226)

[6.1 总结 59](#_Toc444855227)

[6.2 展望 60](#_Toc444855228)

[参考文献 61](#_Toc444855229)

[攻读硕士学位期间的主要研究成果 64](#_Toc444855230)

[致谢 65](#_Toc444855231)

# 第一章 绪论

1.1 研究背景和意义

近年来，视频监控系统已经广泛部署在我们生活的各个角落，逐渐成为我们日常安全保障不可或缺的一部分。利用IP摄像头，人们可以在互联网的任一端对监控视频进行点播[1]。在“数据即价值”的时代，人们已经不再满足于传统视频监控系统的视频存储、点播、回放等功能，而更希望能够从大量的视频监控数据中获得更多有价值的信息，并且获取各种有意义的增值服务，比如交管部门更关心如何利用视频监控系统对交通状况进行自动化智能监控，实现目标识别、越界检测等功能，商家厂家的直播服务可能更关心视频流的转播问题，这些问题都可以归结为视频监控数据的智能处理问题。但是由于视频数据特有的非结构化特征[2]，在处理方面无法依赖于常见的结构化数据处理方式，处理一份视频数据通常需要耗费大量的计算资源。传统的集中式的视频数据处理方式过多的依靠单服务器的个体处理能力，无法满足视频数据的规模不断扩大以及视频处理任务灵活多变的需求，同时，当视频数据规模变小且视频处理任务减少时，这种方式又会造成大量的资源浪费。面对视频规模以及用户弹性变化的需求，人们开始使用云计算技术来处理智能监控视频分析中的问题[3]。

视频监控云就是利用云计算技术的弹性资源分配能力来解决视频监控大数据处理问题的新范例。利用云计算技术，可以将大规模集群物理资源抽象成广阔的资源池，然后再实现统一管理、按需分配。在视频监控云中，一台物理机能够通过虚拟化技术提供多台相互独立而且并发执行的虚拟机，而且每一台虚拟机又可以独立的执行智能视频分析任务，实现视频处理任务与虚拟机之间的对应关系。当有新的视频处理任务到来时，应用会发出资源请求，视频监控云可以立即在合适的物理机上创建新的虚拟机来执行任务，当任务运行结束，销毁闲置的虚拟机又能实现物理资源的回收再利用，实现资源的动态管理[4]。对于云资源管理中心而言，资源请求的规模大小跟虚拟机数量的多少成对应关系，由于不同的视频处理任务有不同的资源需求，比如视频转码属于CPU密集型任务、视频转播属于I/O密集型任务等，视频处理任务的这种需求直接导致虚拟机的异构性，因此，在视频监控云的异构环境下进行资源分配非常具有挑战性，应该尽力避免不合理的资源分配而导致的资源浪费以及性能衰减问题。

1.2 国内外研究现状

自亚马逊（Amazon）推出弹性计算云（EC2）服务[5]开始，云计算技术开始得到大力推广。微软、谷歌、IBM均有自己的云计算产品，国内的阿里巴巴也推出了自己的“阿里云” [6]。而在开源云领域，常见的IaaS开源平台有OpenStack[7]、CloudStack[8]、Eucalyptus[9]和OpenNebula[10]等，国内的许多云平台，如华为云、盛大云等，均是在Openstack的基础上进行深层次的二次开发。伴随着云计算技术的发展，利用云平台来解决视频处理问题的案例逐渐增多。Encoding.com[11]提供了转码即服务（Transcoding as a Service），利用亚马逊的AWS平台来实现监控视频不同格式、分辨率、帧率以及带宽的转码工作。Yang等[12]基于开源云管理平台OpenNebula[10]，配合底层的KVM技术，实现了视频图像的云管理平台，并融合HDFS提供视频的存储检索与点播功能。Hossain 等[13]在Amazon平台实现了用于组合视频监控服务的原型系统，可以进行视频处理，并将结果传回云平台，服务有转码服务、编辑服务以及共享服务。

虽然基于云平台的应用有很多，但是并没有一个面向视频监控云的云资源管理系统。而且云计算平台的一个主要工作是进行服务器集群的资源管理，因此资源调度算法是其重中之重。当前的调度方案有两种，分别是基于单资源的调度和基于多维资源的调度。

（1）单资源调度。Openstack[7]采用的调度算法有两种，即随机调度和过滤调度。过滤调度的实现仅仅是依据内存量为指标，优先分配给内存量大的物理主机[14]。Eucalyptus[9]使用基于CPU的First fit贪心算法，采用Round-robin调度策略。Farahnakian等[15]的目标是使能源消耗以及SLA震荡最小化，因此提出了一个虚拟机管理框架，然而这个方案仅仅是基于单维的CPU资源作为指标。鉴于服务器多维资源的客观存在性，单资源调度不能够解决实际调度中存在的问题。

（2）多维资源调度。为了解决多维资源匹配约束的问题，Singh等人[16]设计实现了虚拟化数据中心，提出了用于负载均衡的VectorDot算法，采用向量积的形式进行匹配，这种方式基于多维装箱的Toyoda方法。Nguyen等人[17]深入探讨了多维资源匹配问题，提出了多维资源选择方案（MRS），提出了一些新概念，比如热门资源、资源温度等，方案的目标是根据资源类型来分散负载，以期达到各资源的综合利用。

为了解决资源异构性的问题，Kang Z等[18]采用拍卖模型算法解决不同性能特征的虚拟机分配问题。Wang, J.等人[19]提出了包括负载预测、虚拟机放置以及虚拟机移植等一系列步骤在内的贪心算法，以CPU与内存的剩余资源比值为指标来分散CPU密集型的服务与内存密集型的服务，但是该方法只能对二维的资源进行匹配，无法推广到多维资源。Ghodsi A等人[20]针对任务分配问题，提出了主导资源的概念，并实现了基于主导资源概念的最大最小公平算法。Wang,W等人[21]对该算法进行了改进，并且对服务器的异构性进行了系统的研究，将任务处理的公平概念从单服务器推广到了多服务器。

1.3 论文的主要研究内容

视频监控云的主要功能是为上层的应用提供弹性的云计算能力。本课题主要围绕视频监控云资源管理的问题，开展了以下研究工作：

（1）设计了面向视频监控云的云资源管理系统。为了满足视频监控云应用的需求，能够为上层的云应用提供弹性云计算能力，我们基于开源的Openstack系统设计了视频监控云资源管理系统。该系统在物理架构上采用多区域模式，能够满足视频监控系统的跨地域管理需求。在功能设计上，首先为了能够对Openstack进行深层次的开发，我们基于Openstack的原生的REST API，设计了Openstack的Java SDK服务，其次在Openstack Java SDK的基础上，使用SpringMVC框架，设计了云资源WEB管理系统，然后为了能够实时监控虚拟机的状态，基于 Libvirt库设计了虚拟机的动态监控服务，能够在WEB端对虚拟机的CPU、内存、硬盘、网络等使用情况进行实时的展示分析。

（2）提出并实现了基于主导资源概念的多维资源调度算法。为了适应视频监控云的应用需求，我们首先分析了Openstack资源调度的流程及策略，并讨论了其中存在的问题，其次深入研究了视频监控云资源分配中虚拟机请求与物理资源的固有特性，包括多维性、动态性以及异构性，然后以集群资源利用率最大化为目标，建立数学模型，提出了一种主导资源优先分配（DRFA）算法，该算法基于主导资源的概念，在集群整体的角度进行决策，优先选择具有相同主导资源类型的虚拟机与物理机进行匹配，并且采用多维资源匹配标准，选择最优的物理机进行资源分配。

（3）编码实现了视频监控云资源管理系统。我们针对视频监控云资源管理系统涉及的多个服务进行了技术实现。首先，通过对Openstack的keystone组件进行合理的配置实现其跨区域管理功能。然后，使用Java语言对Openstack原生的API进行二次封装，实现并开源了Openstack Java SDK。其次，在Openstack Java SDK的基础上实现了云资源的WEB管理系统，并基于Libvirt库实现了虚拟机的动态监控服务。最后，我们对系统的功能进行了测试，并对提出的DRFA方法进行测试，测试结果表明系统整体功能运行良好，而且调度方法能显著提高集群资源的利用率。

1.4 论文的组织结构

本文由六个章节组成，各章节的主要内容有：

第一章：绪论。本章主要介绍了视频监控云的研究背景及意义，对国内外主要的云资源管理平台以及资源调度算法进行总结，然后简述了本课题的主要工作内容。

第二章：相关技术简介。本章对视频监控云中用到的相关技术进行详细阐述，包括虚拟化技术以及云计算的基本概念，并着重介绍了开源云计算平台Openstack，包括Openstack核心架构、各种组件工作原理，最后主要介绍了资源调度中的相关技术。

第三章：视频监控云资源管理系统设计。本章从多个角度对系统进行设计阐述，首先介绍了云资源管理系统的跨域的架构设计，其次介绍了Openstack的Java SDK的组成模块以及其工作流程，最后描述了云资源的WEB管理服务以及虚拟机的动态监控服务的设计流程。

第四章：视频监控云多维资源调度方法。为了解决多维资源调度问题，本章对云资源的特性进行了研究，并且用数学方法建立了问题模型，提出了主导资源优先分配算法（DRFA），并主要分析了主导资源的概念、多维资源匹配标准，最后详细介绍了算法的工作流程。

第五章：系统实现与测试分析。本章从实现的角度详细说明了资源管理平台的实现过程，对于关键部分进行了详细描述，包括SDK服务、WEB管理服务以及动监控服务的实现，最后对系统进行功能测试，并对提出的算法进行测试，通过与其它算法进行比较，充分证明了DRFA的优势。

第六章：总结与展望。对本课题的研究内容进行回顾总结，在实事求是的基础上，指出资源分配算法的可优化空间，并对于视频监控云的未来方向做出展望。

# 第二章 相关技术介绍

2.1 云计算技术

2.1.1 云计算概念

云计算是一种在远端环境下为用户提供按需分配计算能力的付费服务，一般认为由功用计算、分布式计算、网格计算以及虚拟化技术等前沿技术融合而成。因此云计算作为一种服务模式，可以根据服务类型的不同进行层次的划分。一般可以划分为IAAS（基础设施即服务）、PAAS（平台及服务）和SAAS（软件即服务）。但是狭义的云计算概念仍然是指基础设施即服务。

云计算技术自其诞生以来，一直处于火热的状态，与其与生俱来的自身特性是分不开的[22]。首先是云计算的服务特性，云计算是一种按需分配的服务，这种服务模式的目的是让用户从硬件资源中解脱出来，从而把更多的精力放在应用需求上[23]，另外云计算可以完成支付即交付的速度，大大的节省了用户的时间。其次是云计算的技术特性，云计算是多种前沿技术的融合后产生的，因此必然也有很多先进的技术品质，这主要体现在云计算的弹性资源管理能力与容错能力上，弹性资源管理对互联网界的架构设计产生了深远的影响，借助弹性资源管理能力，企业和用户可以处理以往无法想象的大数据量问题，而且较强的容错能力对于提高系统架构的可靠性有重要意义。第三是云计算的商业特性，这也是很多公司参与进来的主要原因，新技术的市场化需求就是提高效率的同时能够降低成本，而这恰恰是云计算技术与生具有的能力。最后是云计算的开放特性，现在大多数的云平台都提供了开放式的使用工具，包括API接口等，用户可以根据需求来定制自己的云应用。

2.1.2 虚拟化技术

虚拟化技术要比云计算技术更早进入科研应用领域，也可以说虚拟化技术是云计算技术的基石。虚拟化技术是一个广义的概念，而与云计算相关的虚拟化技术一般是指虚拟机技术，即通过一层软件在服务器之上对硬件资源再整合，使得一个物理机可以并发运行多个虚拟机，而资源整合的任务由虚拟机监控器（Virtual Machine Monitor，VMM）负责 [24]，所以虚拟化技术仍然是软件技术，现在的虚拟机监控程序VMM一般是指工作在操作系统之上的hypervisor。在虚拟化技术中，有宿主机和客户机之分，宿主机即物理机，而客户机就是指虚拟机。

虚拟化技术有很多种分类，根据虚拟机对硬件资源访问方式的不同，就可以把虚拟化技术分为全虚拟化与半虚拟化（见图2-1）。在全虚拟机技术中，虚拟机与硬件资源的交互完全由hypervisor进行代理，因此虚拟机与硬件之间体现了较好的隔离性，有效的屏蔽了硬件不兼容的问题，但是会存在一定的性能损耗，而半虚拟化技术与此不同，半虚拟化技术可以让虚拟机与宿主机操作系统通过hypervisor来一起共享硬件资源，因为会直接操作硬件资源，所以虚拟机的性能相对较好，但是客户操作系统需要根据硬件平台做出代码的调整，并重新编译。这两种虚拟化技术也对应两种开源hypervisor技术，一种是Xen，一种是KVM.。Xen虽然对两种虚拟机化都可以支持，但是半虚拟化的场景下会表现出更高的性能。而kvm是一种基于内核的全虚拟化技术，在全虚拟化领域，KVM因为其对linux的完美支持而更受欢迎，主流的linux系统都已经在内核中整合了了KVM。

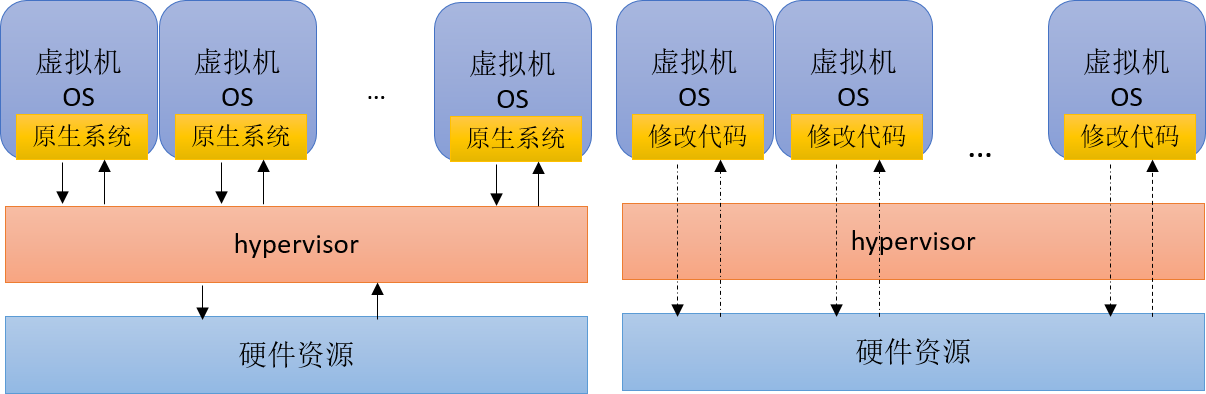


图2-1 全虚拟化技术（左）与半虚拟化技术（右）

通过以上对虚拟化技术与云计算技术的了解，可以对比二者的不同。首先，云计算技术很多种技术融合的结果，所以云计算技术与虚拟化技术是一种包含关系，虚拟化技术是云计算技术的子集。其次，虚拟化技术更强调的是数据中心本地资源的管理，而云计算技术更加强调是远端服务。最后，虚拟化技术的最终目的是摆脱操作系统对硬件资源的依赖，而云计算技术的关注点是集群中所有资源的整合，包括计算资源存储资源以及网络资源。

2.2 Openstack技术

2.2.1 Openstack概述

Openstack是IAAS的范例，因此，利用Openstack，用户可以很方便的管理自己的基础设施，并提供虚拟机实例。自2010年，NASA和Rackspace联合推出Openstack开源项目以来，Openstack从最初的第一个发行版Austin版本，以每年两个或两个以上版本的速度成长，到2015年10月，推出了发行版Liberty版本，在项目组成上，最初的Austin版本仅集成了NASA的Nebula平台与Rackspace的Cloud Files平台，也就是我们熟知的Nova模块与Swift模块，而现在的Openstack的代码已经超过了两千万行，发展为了涵盖Nova、Swift、Neutron、Cinder、Keystone、Glance等核心模块的完整的云平台。Openstack是开源云领域发展最快的云平台之一，由非盈利组织Openstack基金会进行管理，且项目遵循Apache 2.0协议[25]，目前已经超过500家公司且超过三万开发者向其贡献代码，这些支持公司包括IBM、VMware、Red hat等业内主流的组织和厂商，在得到广泛支持的背后，上百家公司基于Openstack进行开发，并发展适合自己IAAS领域的公有云或私有云业务，这包括国内的华为、UCloud等公司。

作为一个开源的云资源管理平台，Openstack由很多独立的组件组成并协调工作，因此Openstack可以看做是一个生态系统，而不是一个软件[26]。Openstack也可以说是一个云操作系统，采用虚拟化技术，将原本独立在单服务器内的资源，抽象成广阔的资源池（计算资源、网络资源、存储资源），然后统一进行资源调度与控制管理，使得整个数据中心能够对外提供按需付费服务。Openstack致力于实现公有云和私有云领域的云计算平台开放标准，旨在提供一个实现简单，具有高可伸缩性以及丰富特性的云解决方案。

Openstack作为一套优秀的云计算解决方案，之所以备受关注，与其优秀的架构设计是分不开的。Openstack在设计之初，就有自己的设计原则。

**可扩展性和弹性**：这是Openstack的第一设计目标，任何限制该准则的组件都需要设计成可选的。Openstack采用基于组件松耦合的设计方式，除了核心组件外，许多组件都是“按需安装”，除此之外，很多资源管理组件都可以无缝的水平扩展，比如计算节点、存储节点等，为云的弹性扩充提供了极大的优势。

**无共享性**：Openstack的架构采用无共享（SN-share nothing）的分布式计算架构。无共享架构是指架构中的每一个组件都是独立的（independent）且自给的（self-sufficient），避免出现共享内存和共享硬盘的情况，组件之间采用其它的通信方式，比如网络通信等，避免组件之间的互相影响，单点故障。

**灵活性**：与其说Openstack是一款开源产品，不如说Openstack一个技术框架，因为Openstack预留了充足的空间供第三方技术进行集成，用来满足不同用户的自身业务需要。可以说Openstack的每一个模块，更像是一个中间件，可以通过驱动的编写来适配其它第三方成熟的技术。比如nova模块背后就支持很多hypervisor技术，比如KVM、Xen、VMware、LXC等。如表2-1所示。

**高效性**：为了提高系统效率，Openstack避免阻塞操作，组件之间采用异步的RPC（remote process call）调用机制，而且使用RabbitMQ进行组件之间的通信。由于Openstack采用分布式的设计，为了保证高可用性，基于CAP原则，接受最终一致性。另外Openstack对代码有较高的规范，要尽量保持代码的测试覆盖率来提高代码质量。

表2-1 Openstack 驱动程序列表

|  |  |
| --- | --- |
| 组件名称 | 通过驱动兼容的技术 |
| Nova | KVM, Xen, LXC, VMware |
| Cinder | Iscsi, NetAPP |
| Glance | Swift, Amazon S3, NFS, 本地存储 |
| Keystone | MySQL, LDAP |

2.2.2 Openstack关键组件

Openstack的组件有很多，可以按照功能的不同分为基础服务组件、资源管理组件和扩展服务组件。其中基础服务组件负责Openstack的基础管理，比如认证服务与访问服务等。资源管理组件负责集群资源的管理，包括计算资源、存储资源和网络资源等，扩展服务组件主要是指为了完善服务而扩展的组件，比如资源统计等。

（1）认证授权组件

安全认证是每一个系统的基础功能，Openstack的安全认证与授权服务由keystone组件负责。Keystonte通过三种实体概念通过互相关联来实现复杂的认证和授权服务，三种实体概念是指组织结构、服务目录以及凭证信息（见图2-2），其中，组织结构包括租户和用户，服务目录包括service以及endpoint，另外凭证信息包括密码以及token信息。

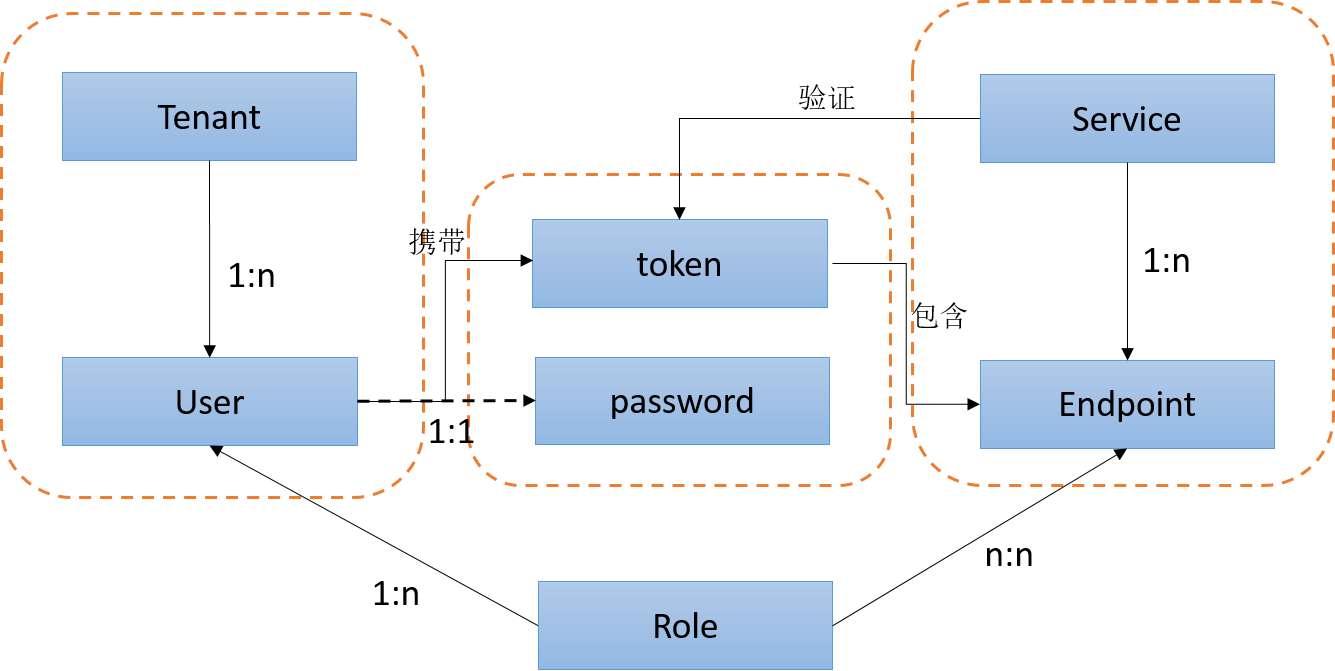


图2-2 keystone安全认证实体图

在Openstack的组织结构中，租户就是集群中有限资源的集合，资源包括真实资源（cpu、内存、带宽等）以及用户信息等。租户中真实资源的数量属于配额信息，是保证用户可以使用的最大数量，而配额信息的设置由云提供商决策。租户中的用户通常分为管理员和普通用户，可以通过角色分配来实现。而对Openstack的访问需要服务目录的配合，由于Openstack是由很多组件共同组成的分布式结构，为了保证每个组件的访问安全性，Openstack会在keystone中注册所有的组件（service），同时存储访问以及管理组件需要的网关URL（endpoint），用户在访问前，并不知道组件的真实URL，只有在keystone中通过安全检查后，才可以得知组件的访问地址。

Keystone使用两种凭证信息来保证访问的安全性，一是通过密码来保证用户的可靠性，用户名和密码是唯一对应的，二是通过token机制来保证访问的可靠性，token指的是一串比特值或者字符串，用来作为访问资源的记号，通常会包括可访问资源的区间以及有效时间[27]。Openstack安全访问的步骤是用户需要首先通过keystone的用户名和密码验证，通过之后，系统会分配给用户一个token，此时token就能代表用户的身份信息，用户每次访问都需要携带token，组件接收到访问请求时，都会向keystone查询token的真伪性，多层次的保障访问的可靠性。

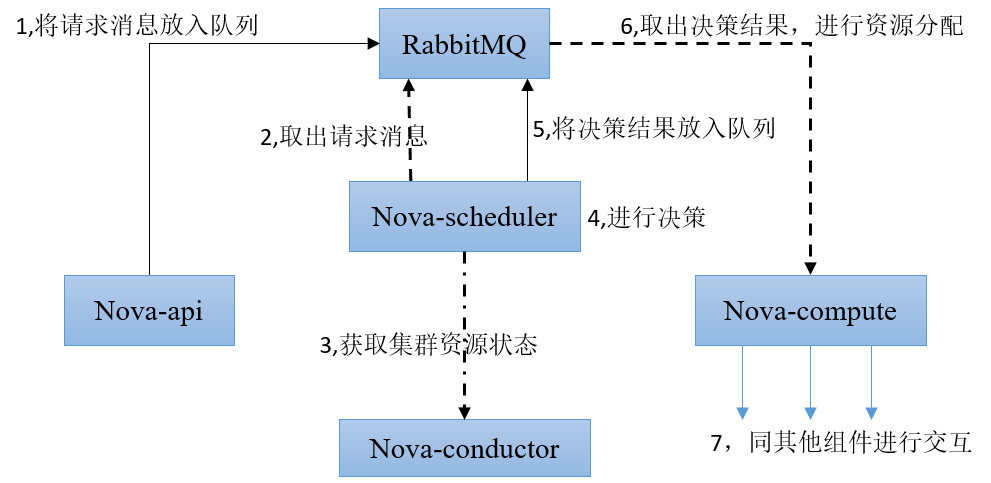
表2-2 nova模块与职责列表

|  |  |
| --- | --- |
| 模块名称 | 职责 |
| nova-API | 安装在controller节点，属于nova模块的入口，负责接收来自keystone以及CLI的命令，并转发到相应的组件。 |
| nova-compute | 安装在计算节点，与底层Libvirt库交互，直接负责虚拟机的管理，并同与虚拟机资源相关的其它组件进行交互，比如cinder、neutron等。 |
| nova-cert | 提供X509证书服务，并兼容EC2 API。 |
| nova-conductor | 负责数据库访问服务，避免将nova-compete与nova-conductor安装到同一节点上。 |
| nova-novncproxy，nova-consoleauth | 负责虚拟机的网络终端服务，consoleauth负责验证与IP端口映射。 |
| nova-scheduler | 负责资源调度决策，决定虚拟机的最终安装位置。 |

（2）计算资源管理组件

Openstack作为云管理平台，其中主要的部分就是Nova组件，Nova是一个分布式的集群虚拟化管理软件，负责服务器集群各种资源的整合以及虚拟机的生命周期管理。Openstack的架构判别标准也多以nova为主，比如Openstack中的controller节点与compute节点就是直接对应nova的nova-API与nova-compute。为了完善分布式的虚拟化管理，Nova同样由很多模块组成，包括nova-API、nova-cert、nova-conductor、nova-consoleauth、nova-novncproxy 、nova-scheduler等每个模块职责不同，都可以单独部署。

Nova工作的流程（图2-3）是，用户向nova-API发出创建虚拟机的请求，nova-API得到虚拟机的配置信息，并调用rpc.cast将消息放入队列中，监视消息队列的nova-scheduler取出虚拟机创建请求，并从nova-conductor中取出集群资源的统计信息，此时nova-scheduler根据请求与统计信息进行决策，将决策结果重新放回消息队列中，决策结果通常是物理主机的ID，nova-compte会判断目标主机是不是自己，如果是，那么nova-compute进入虚拟机创建阶段。

图2-3 nova创建虚拟机流程图

（3）网络资源管理组件

Openstack对于网络资源的管理有两种方式，一种是通过nova-network，一种是通过neutron。Nova-network 可以提供虚拟机之间以及虚拟机跟外网之间的通信，通常是在计算节点上建立网桥，然后虚拟机通过网桥进行通信，nova-network也有广泛的用户，已经被验证为可信赖的且具有扩展性的网络通信方式。但是Nova-network只能由管理员进行组网的管理，用户只有使用的权利。目前nova-network支持三种网络管理形式：FlatDHCP模式、Flat模式以及Vlan模式。其中FlatDHCP模式使用最广泛，在这种模式下，Openstack会启用一个DHCP服务器来进行虚拟机IP地址的分发，所有的虚拟机处在同一个子网中，而且只有管理员才能管理。其它两种网络模式都有其特殊的限制，Flat模式因为需要手动配置网桥导致操作比较繁琐，Vlan模式虽然可以提供子网的划分，但是需要特定硬件配置（vlan-capable switch）的支持。

因为Nova-network的诸多问题，Openstack已宣布会逐渐废弃对Nova-network的支持，由其一直致力于研究软件定义网路neutron组件代替。Neutron中使用了许多先进的技术，让用户更灵活的管理分配物理网络，能够满足多租户环境下的子网的独立性。Neutron中采用了Open vSwitch技术，Open vSwitch 是开源的软件定义虚拟交换机，也就是常说的软交换。Neutron通过vSwitch与 hypervisor的配合工作，实现虚拟机之间的路由交换功能。Neutron 的openvswitch 插件由两个组件构成：一个由Neutron服务加载的运行期插件，用于处理API接口调用以及把逻辑网络数据的后台数据库存储工作。另外一个是Agent，运行在计算节点上。Agent的工作有：收集配置信息，与本地的Open vSwitch实例进行通信，基于逻辑数据模型来实现网络。

Neutron在实现上，会依赖keystone组件进行用户认证和API请求的授权。网络作为实例创建的一部分，因此Nova-compute组件会直接调用Neutron的API来执行网络分配工作。在前端，Horizon组件整合了Neutron的API，可以使用WEB界面来保证租户对网络和子网的可视化操作。

在Neutron中，一共可以创建两种网络，一种是供应商网络（provider networks），一种是租户网络（Tenant networks）。两种类型的网络都可以实现虚拟机之间的网络连接。但是，在Openstack中，供应商网络是默认存在的，由管理员负责管理，工作在租户与租户之间共享的flat模式下，用于保证虚拟机实例跟集群底层的物理网络进行通信。而租户网络是由用户创建，且只在租户内或者一组用户内工作，租户网络不会跟其它租户共享，因此租户网络的工作在是vlan以及GRE[28]模式下。租户网络需要使用L3 agent以及Neutron 路由器，在基于GRE的租户网络中完全可以开启路由功能进行通信，如果没有Neutron路由器，那么这些网络就会处于完全隔离的状态。

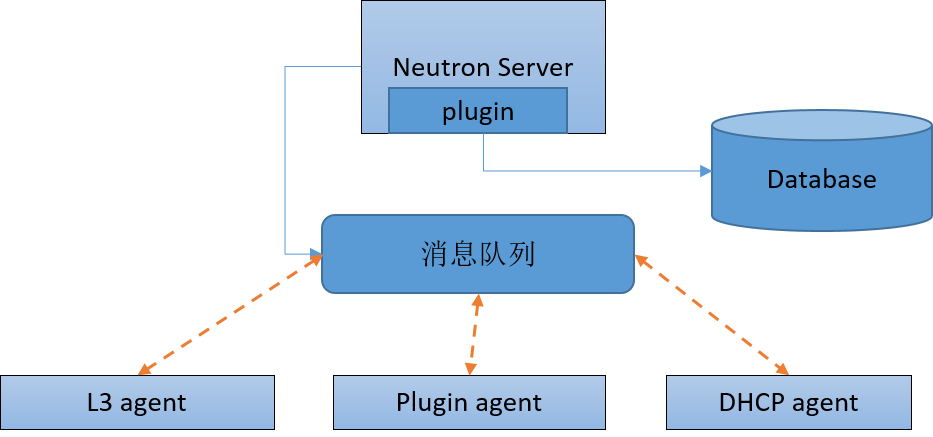


图2-4 Openstack 网络工作图

（4）存储资源组件

Openstack中的存储有三种类型，分别是临时存储（Ehemeral Storage）、块存储（Block Storage）以及对象存储（Object Storage），每一种存储都发挥着不可替代的作用。

临时存储：临时存储的生命周期跟虚拟机的生命周期相同，在虚拟机实例创建时由nova负责分配，在虚拟机销毁时自动释放，仅用于提供虚拟机操作系统镜像运行时必要的存储空间。临时存储的大小，可以在flavor模块中设置，flavor是用户启动虚拟机的一系列参数的集合，包括CPU、RAM、name、swap区的大小，其中Disk参数的大小属于系统盘的大小，而Ephemeral属于第二块临时磁盘，最初是空闲且未格式化。临时存储跟虚拟机的运行息息相关，一旦出现操作系统故障等情况，临时存储中的数据都面临不可访问的困难。

块存储：通常又被称为卷（volume）存储，由cinder模块进行管理，其生命周期不受虚拟机生命周期的限制，而是由用户去支配。用户可以通过在虚拟机挂载（attach）逻辑卷设备的形式进行访问，用于向虚拟机提供额外的块存储服务。块存储设备是持久性的，而且跟虚拟机保持状态独立性。块存储设备可以在一个虚拟机实例上解除挂载后，挂载到其它实例，数据不变，但是一次只能挂载到一个虚拟机实例中，虚拟机销毁后，块存储设备不受影响。Cinder目前支持快照（snapshot）功能，即对存储卷在某个时间点的数据创建只读副本，快照可以在创建新存储卷的时候作为数据源。支持本地存储和网络存储两种方式，其中本地存储是指使用集群内的硬盘，需要LVM的支持，网络存储是指NFS等其它存储设备。

Cinder的工作流程跟nova类似[29]，如图2-5 所示，Cinder-API在接受到创建Cinder 盘的Rest请求之后，将消息封装并放入消息队列。Cinder-Scheduler会监听消息队列中创建硬盘的消息，经过一定的调度策略后选择合适的Cinder-Volume 来执行资源分配。创建好的硬盘可以挂载到虚拟机上进行使用。

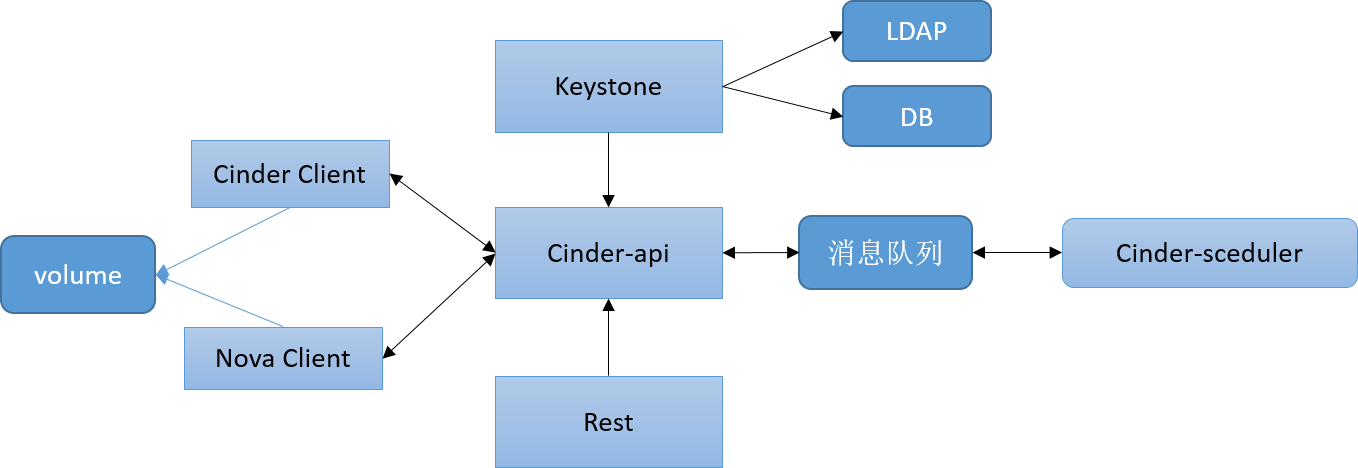


图2-5 Cinder工作流程图

对象存储：相对于Cinder而言，Swift更为成熟，Swift是可以独立的子项目，既能整合到云平台中，可以用单独部署使用，而Cinder是Nova的一个附属功能。对象存储跟块存储有两个区别，一是存储对象不同，块存储是把数据进行平均切分之后再进行存储，对象存储不将数据进行切分，而是将数据整体一起存储起来。二是存储的组织形式不同，对象存储在存储文件时采用扁平的组织形式，直接使用唯一的ID去检索，而不是常用文件系统的目录层级索引，这也是对象存储比传统的文件系统具有更大扩展性的原因。因为对象存储在实现上更简单，使用对象存储就会涉及到更少的元数据，减少了管理文件元数据的负担，因此，理论上来讲，对象存储可以通过扩展节点的形式使得规模达到无限大。但是对象存储不具有增量修改能力，而且每一次的操作都需要把对象当做完整的单元进行操作。Openstack swift 提供冗余、弹性的分布式对象存储服务。分布式存储是指每一份数据都会在集群里保留额外的备份，而且可以通过配置改变备份的数目，在生产环境里，每一份数据都至少存在三个副本。

Swift采用层次的数据模型，每一个层次都可以通过扩展来扩充规模，每一个对象的访问路径由三个元素组成：/account/container/object。其中account、contrinaer、object分别对应不同的服务器，而且真正存储对象的是Ojbect服务器，Object 服务器是一个可以存储、检索、删除对象的简单的BLOB存储服务器。对象是以二进制文件的形式存储在服务器上，元数据存储为文件的扩展属性（xattrs）[3]。为了解决对象索引问题，Swift采用Ring的数据结构以及一致性hash算法来解决数据在分布式集群中的寻址问题。

（5）镜像存储glance

Glance的模块图如图六所示。

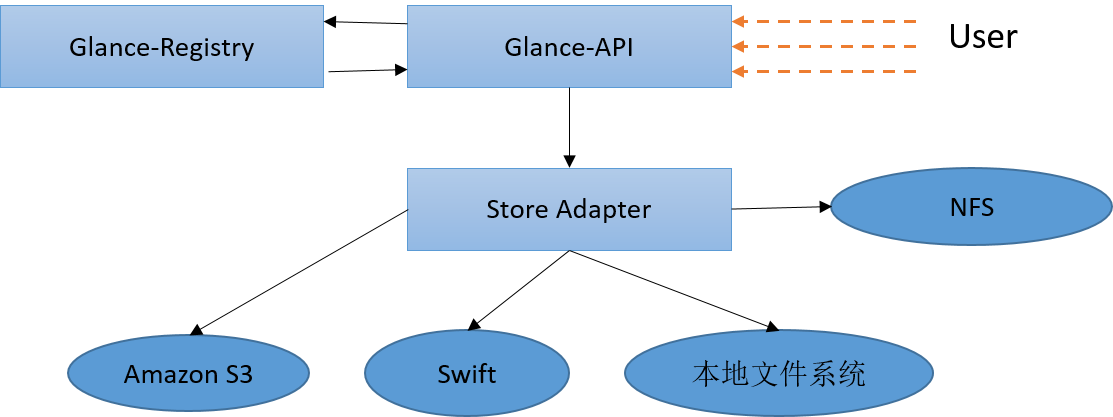


图2-6 Glance模块图

在云计算中，资源是虚拟机运行的必要条件，而资源的真正执行者是操作系统，也就是镜像，Openstack中的Glance提供了强大的镜像管理能力，包括镜像的上传、下载以及模板功能。不同于ISO镜像，云镜像属于硬盘镜像，是根据需求定制的可以直接启动的镜像，因此要比普通镜像大很多，尤其是快照功能镜像，通常会包含很多虚拟机已经安装的程序文件。Glance组件由Glance-API、Glance-registry以及store三个模块组成，其中Glance-API仍然是充当glance的入口，负责http请求的接入，而Glance-registry用于管理镜像基本信息，并提供CRUD访问控制，相当于元数据的管理，而镜像的真正存储位置可以自由指定，为此，glance提供了相应的驱动程序，因此镜像文件既可以存储在块存储设备上（本地文件系统或NFS），也可以存储在对象存储设备上 ，包括Amazon S3存储以及Openstack的Swift。

2.3 资源调度相关技术

常见的资源调度算法一般退化为一维装箱问题，传统的分配方法有首次适应算法（First Fit）、循环首次适应算法（Next Fit）和最佳适应算法（Best Fit）。

首次适应算法（First Fit）是将集群中的服务器按照一定规则组成链表，当有虚拟机请求到来时，会从链表头结点开始，判断服务器是否满足虚拟机的资源请求，如果满足则进行资源分配，一次分配过程结束，如果不满足，则依次遍历服务器链表，直到找到第一个满足资源请求的服务器为止，链表遍历结束时仍未发现满足资源的服务器，则抛出错误。首次适应算法会导致资源分配的不均匀，永远是排列在前端的服务器会优先满载，而链表后端的服务器因为得不到利用而闲置，资源负载不均衡。在资源分配的过程中，常常会因为分配的原因，导致服务器内部有一部分资源，因为容量太小而永远得不到分配的情况，这种现象称为资源碎片化，碎片化的资源会导致资源的浪费。

为了解决首次适应算法中资源分配不均的问题，可以在一次分配结束后，下一次分配不再从头部节点进行查找，而是从上次分配结束之后的下一个节点进行查找分配，这样就会使得资源分配得更加均匀，这种算法叫做循环首次适应算法（Next Fit）。

为了解决资源碎片化的问题，可以将服务器链表按照资源量的数值由到大排序，然后采用首次适应算法，这样能保证每一次都能使得当资源得到分配后，服务器中剩余的资源量最少的，有效的减少的了资源碎片化，提高了资源利用率，这种改进叫做最佳适应算法（Best Fit），最佳适应算法需要一次分配结束后，对服务器节点进行重新排序。而如果按照资源量进行从大到小排序，则分配算法叫做最坏适应算法（Worst Fit）。

然而多维资源的匹配问题，又可以称为Vector packing问题，是一维装箱问题的升级版，但是多维装箱问题被称为NP-hard问题[31]，没有最优解，只能找到近似最优解。解决多维装箱问题的方法有很多，包括蚁群算法、基于启发式策略等，但是都是基于多目标优化的。而最简单的多为资源匹配仍然是FFD（First Fit decreasing），也有很多从向量匹配的角度进行多维资源调度的方法。

2.4 本章小结

本章首先介绍了云计算相关技术，通过与虚拟化技术进行比较，加深对云计算是一种服务的理解，然后主要介绍了Openstack以及其各大组件的基本模块组成与流程，包括授权认证组件、镜像组件以及存储组件最后介绍了常见的资源调度基本知识，最后介绍了资源调度的常见的方法。

# 第三章 视频监控云资源管理系统设计

本章首先介绍了多区域的视频监控云资源管理系统的架构设计方法，然后对系统的主要服务，包括SDK服务、WEB管理服务以及资源监控服务等提出了详细的模块设计以及分层设计方案。

3.1 系统总体架构设计

3.1.1 跨地域物理架构设计

视频监控云的上层应用，主要通过对大量监控视频进行有效的数据分析来构建有价值的智能服务，包括车流量检测、交通违章事件检测以及视频直播等，而视频数据的分析处理就需要首先解决视频源的传输问题，但是由于视频采集设备以及存储设备的地域性，视频数据的传输难免会因为网络带宽的因素，产生很大的时延，甚至影响视频数据的质量，从而影响最终的任务处理时间及处理结果。用户与数据中心之间的距离会导致QoS(Quality of Service)的衰减，而且安全认证问题也是需要解决重要问题。[32]。因此本系统在物理架构设计上需要满足两个需求：

1. 实现云资源管理系统的跨区域部署，而且每一个区域应该有独立的计算能力、网络能力与存储能力。
2. 跨区域的云资源系统部署需要满足统一的授权认证服务，否则就失去了跨域部署的意义。

本系统为了满足视频监控云应用的地域性需求，采用了跨域（Region）的设计模式，如图3-1 所示，这种模式既能满足监控视频任务处理的地域性，又能在宏观上对系统施行统一管理。从图3-1中可知，每一个地域都对应一个Region，Region相当于Openstack的一个子集，承担云资源系统最重要的计算服务、网络服务与存储服务，而不同域之间采用统一的授权认证服务（keystone）以及WEB管理服务，达到统一管理的效果。

对于域内系统架构的选择，Openstack提供的架构类型共三种，第一种是把所有的组件安装在同一台服务器中，即All-in-One的安装模式，但是这种架构仅适用于测试，无法在实践环境中得到运用。第二种是在考虑网络需求时，采用nova-network来管理网络，这是Openstack的两节点架构，但是网络模式都是扁平化的，无法划分子网，这种模式已经逐渐被Openstack 废弃。所以本系统在区域内采用第三种架构，即使用Neutron网络节点的三节点架构，能够把网络管理能力下放到租户，用户可以根据需求设置自己的子网及路由。

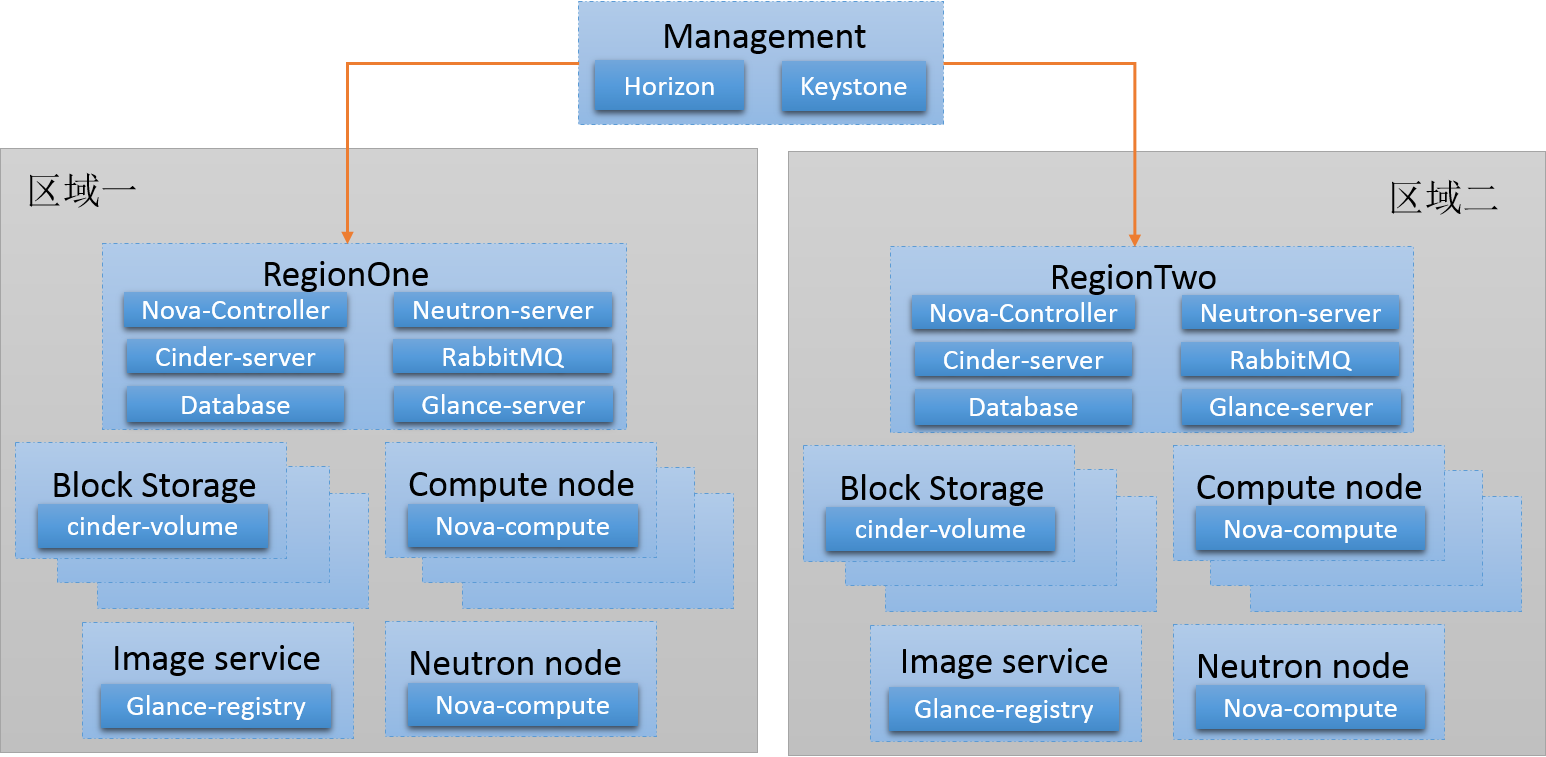


图3-1 视频监控云跨域的物理架构设计

基于分域概念的云资源管理系统中，由于不同的域拥有各自独立的计算能力，因此不同域的资源调度是隔离的，而且不同的域可以使用不同的调度策略。如果想对每一个域中的服务器进行更细致的拆分，Openstack提出了cell的概念，nova-cell用于满足对同一域内的服务器再次分组的需求，比如同一机架的服务器或者具有同一特性的服务器可以放在一个cell内。在划分cell的云资源管理系统中，资源调度时会先选择cell，然后再对cell内的服务器进行调度。

基于分域概念的物理架构设计具有良好的扩展性，一是域内使用的Openstack原生的计算架构，计算节点与存储节点都具有良好的扩展性，二是支持域的可插拔性，新的独立区域可以灵活的加入或退出，而且不会影响现有的架构。用户在使用时，为了提高响应速度，减少网络传输时间，应该尽量依据就近原则来部署自己的任务。

3.1.2 系统逻辑架构设计

视频监控云资源管理系统是在Openstack系统基础上进行的二次封装开发，所以本系统的主要功能都建立在Openstack的集群管理能力上，但是部分功能需要针对视频监控云的应用场景进行调整。本系统的软件逻辑架构如图3-2所示，本系统基于采用B/S网络架构模式，设计上采用分层分模块的的软件设计风格，系统从底层Openstack开始，共分为三个主要服务，首先是Openstack的Java SDK服务，这是整个系统开发的基石，其次是虚拟机动态资源监控服务，在两个服务至上，构建了用于系统逻辑的业务层，最后使用SpringMVC框架提供用户访问系统的WEB管理与API服务。

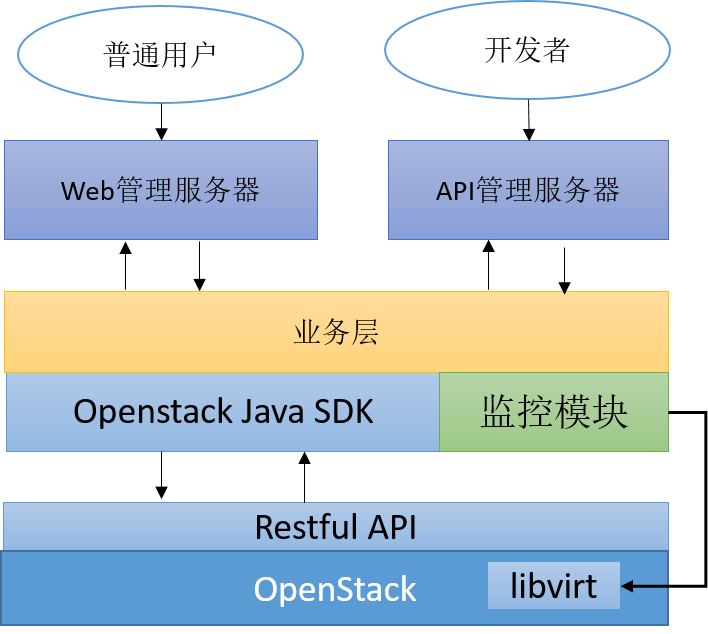


图3-2 软件架构设计

SDK服务：本服务基于Openstack的原生API，使用Java语言重新设计并实现了基于Java语言的Opesntack SDK，SDK主要面向开发人员；

监控服务：监控服务的主要目的是对虚拟机的实时状态进行监测，通过对底层的Libvirt库进行直接的调用，实现虚拟机监控的可视化工作；

访问服务：访问服务分为API服务和WEB管理服务，普通用户通过WEB管理服务就可以实现与后台系统的交互，而对于上层应用开发者，可以通过访问API服务的形式使用视频监控云的处理能力，并定制实现自己的平台应用。

本系统要遵循以下设计原则：

（1）对扩展开放、对修改封闭的原则，由于Openstack版本发展较快，API也已经推出了v2版，因此需要考虑到SDK的后续扩展性。

（2）单一职责原则，一个类应该只对某一模块的功能负责，避免同一个类有交叉职责的情况出现，这样才能保证开发过程的独立性。

（3）依赖反转原则，采用针对接口编程的规则，使定义和实现分离，避免过高的耦合性，实现各个模块的异步升级。

3.2 基于Java的OpenStack SDK服务设计

3.2.1 API与SDK区别

Openstack的访问方式有三种，一种是通过cURL命令或其它Rest Client工具对Openstack的原生API进行访问，另外一种常用的方式是在终端通过Openstack的命令行接口（CLI）进行访问，最后一种是面向开发人员的SDK。

原生API是对平台系统的基本功能的抽象，提供对系统平台最基本的访问能力，通常这些API都是一些原子操作，偶尔的调用可以使用cURL命令，然后观察状态码并自己解析复杂的数据格式，但是如果需要完成复杂的功能，就会出现大量的冗余操作，而SDK（Software Development Kit ）是为使用特定编程语言的开发人员准备的，提供一些简化API访问的工具集合，通过封装设计而屏蔽掉一些不必要HTTP的参数配置，通过一些API的组合而完成一些必要的工作，最后以函数调用的形式进行使用。

Openstack提供了设计优秀的基于HTTP协议的restful API[33]。因为是基于组件式的设计架构，而且采用基于token授权认证策略，因此任何一个组件都需要得到keystone组件中注册自己的endpoint。但对于普通用户而言，如果通过访问原生API进行Openstack的使用，则至少需要两步操作，第一步向keystone请求token，keystone会返回token以及endpoint等信息，第二步再根据endpoint发送真实请求，比较复杂，而SDK就为了使API的访问更简易化。

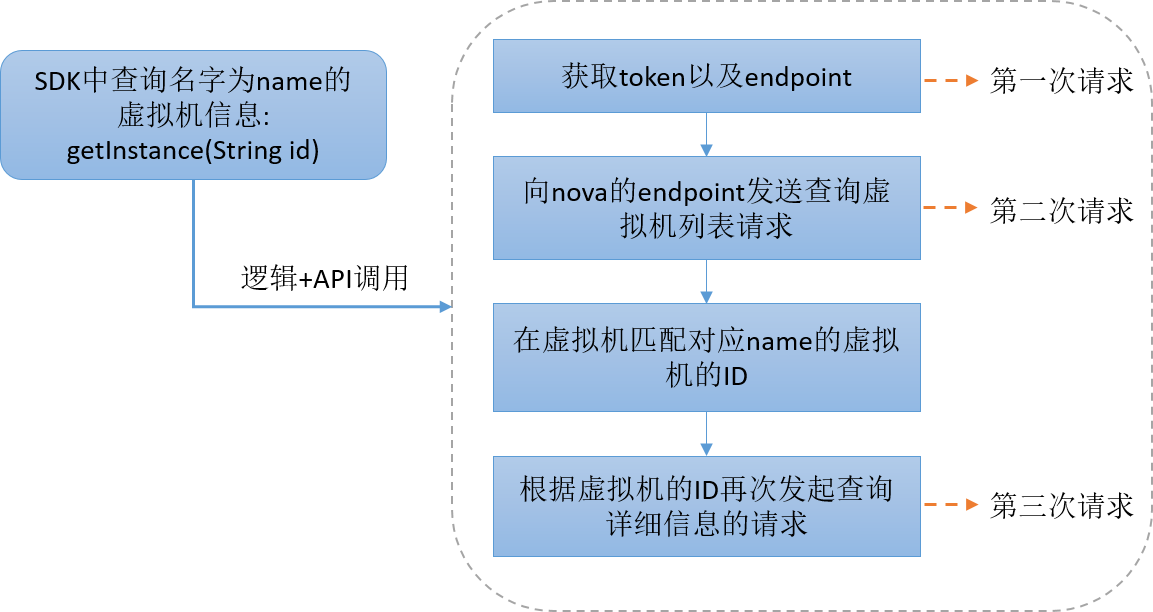


图3-3 查询虚拟机信息流程图

|  |
| --- |
| $ nova –debug show <name> |

调用上述命令查看名字为name的虚拟机的详细信息，可以查看命令执行的详细过程，流程图如3-3 所示。*nova show*命令会先获取到token以及各组件的endpoint，然后再向nova组件查询所有的虚拟机信息，然后找到匹配该名称的虚拟机的id，然后再根据id向Openstack查询该虚拟机的详细信息，因为虚拟机信息里保存的flavor的id以及image的id，于是需要根据Imageid 向glance查询image的详细信息以及向flavor查询flavor的详细信息，最后把信息整合后显示出来。因此，一个简单的展示虚拟机详细信息的操作需要5次API的调用，而SDK就是把这些API调用整合起来，只提供一个展示虚拟机新的接口。从上述可以看出在开发过程中使用SDK的重要性，SDK会根据功能需求而对一系列API进行合理的编排并提供更适合应用的API接口。接下来会介绍Openstack的SDK服务设计。

3.2.2 SDK服务设计

综上所述，Openstack的原生API虽然优秀，但是如果要满足更复杂的调用需求，则需要多种API的合理编排，因此需要在SDK中对原生API进行重新组装，并提供适应实际需求的易于调用的API。

Openstack Java SDK的整体模块设计如图3-4所示，共由3个主模块以及若干公共模块组成，主模块分别是 Connector模块、 Service模块以及Session模块，主模块负责SDK的主要工作流程，而公共模块包括实体操作、日志管理以及数据库管理，负责所有模块共同需要的服务。Openstack Java SDK中各模块之间的调用关系以及从输入到输出的工作流如图3-5所示。SDK中主要模块的功能介绍如下：

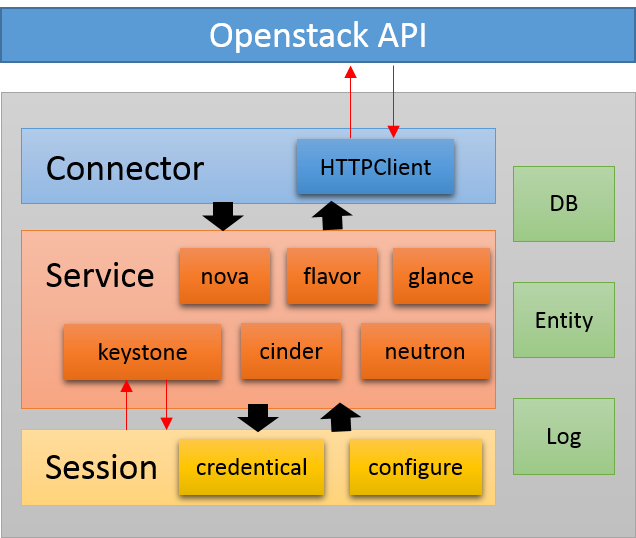


图3-4 Openstack Java SDK 模块图

1. Connector模块

Connector模块的设计目标是把API调用中有关HTTP请求与数据处理的部分完全抽象出来。Connector模块使用Java开源HTTP访问工具HTTPClient 去调用Openstack的Restful API，整个工作流程是：发送HTTP请求并接收HTTP返回的数据，HTTP返回的数据包括状态码以及JSON格式的真实数据，然后对状态码进行合理的解析（异常处理等），并对JSON数据进行必要的解码工作（JSON数据与实体数据之间的转化）。除此之外，在请求发送之前，Connector模块还会对HTTP请求header部分的参数进行填充，比如X-Auth-Project-Id、X-Auth-Token参数等。

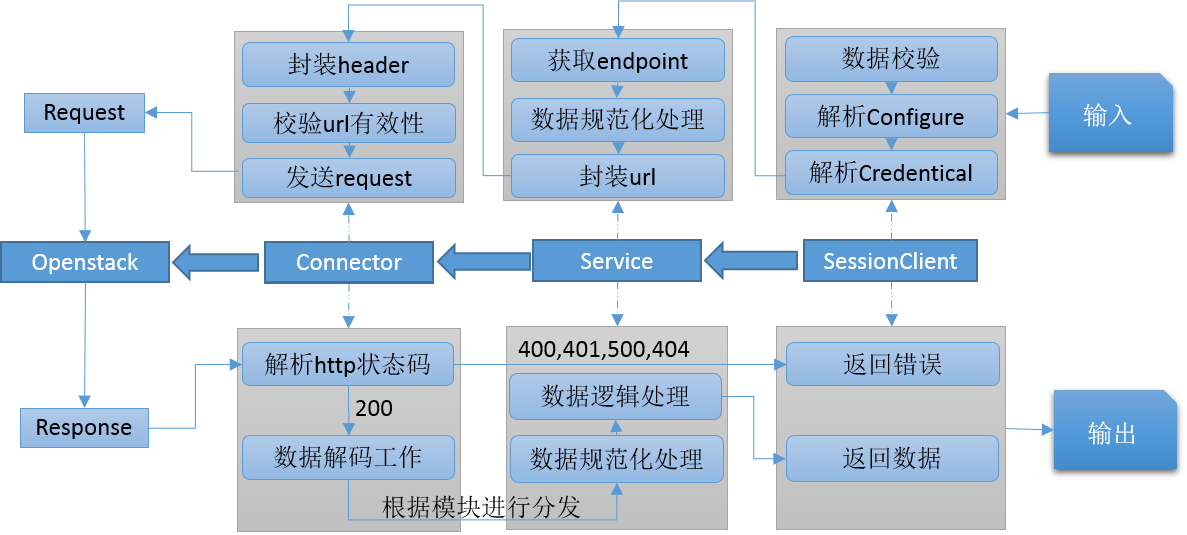


图3-5 Openstack Java SDK工作流

(2) Service模块

Service模块不需要考虑HTTP请求与响应的处理，只需要关注具体组件的逻辑实现以及数据格式化操作。Service模块的具体职责是：定义与Openstack的组件相对应的接口并实现；针对不同组件的接口，实现其必要的url的组装工作，这包括对endpoint、component以及operation的封装。

(3) Session模块

Session模块是整个SDK的入口，需要维持SDK与Openstack之间的Session连接，并使用静态工厂的方式，生成每一个服务模块的实例，提供Openstack的登陆接口。除此之外，Session模块在初始化时需要对配置文件进行加载，并从中获取必要的信息。

(4) 其它模块

其它三个公共模块是日志管理、实体管理以及数据库管理，日志模块负责整个SDK的日志收集工作，使用log4j来控制日志的输出级别，实现日志的定期打包等功能。实体模块中定义了与Openstack的数据表相对应的POJO实体，在Connector模块中，对HTTP返回的JSON数据，使用注解的形式将其转化为POJO实体，另外在发出HTTP请求时把实体数据转化JSON格式，作为参数组装在http url中发送出去。数据库模块提供使用JDBC的数据库访问功能，用于对Openstack数据库的直接操作，比如为了满足业务需求而创建新的数据表等。

3.3 云资源管理服务设计

云资源管理服务分为两部分，第一是是系统的用户访问服务，第二是虚拟机监控服务。其中用户访问服务是让用户通过对接口的访问实现对后端服务的操作，并从中获取有效的结果信息，用户访问服务一般有三种形式，API服务、命令行接口以及WEB管理服务，本系统提供API服务以及WEB管理服务。

3.3.1 WEB管理服务设计

WEB管理服务，又称为仪表盘服务，是用户对后台服务进行操作的可视化接口，并且用于操作结果的实时呈现。Openstack的dashboard由 Hoziron组件管理，仅提供简洁而又基本的私有云集群WEB管理服务。本服务的的模块逻辑层次图如3-6所示，本服务基于MVC的架构设计模式将WEB管理服务进行纵向逻辑层次的划分，具体可分为业务层、控制层和表现层。MVC架构的目的是通过添加控制层（Controller）来实现数据处理工作（Model）与数据展现工作（View）的分离，控制层担当了类似路由转发的工作，当服务器接收到用户的http请求时，控制层将请求定位到特定的业务模块进行处理，对于业务模块的处理结果再发送到指定的前端视图。

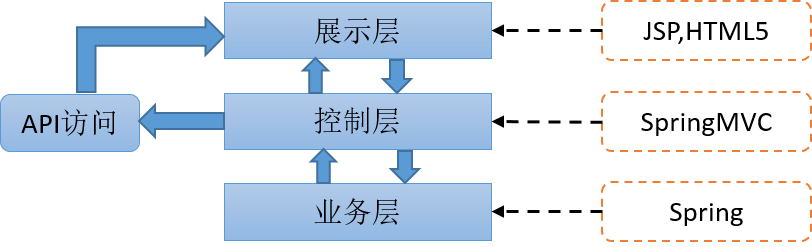


图3-6 用户访问服务逻辑层次划分

1. 业务层与控制层

为了既能提供API服务，又能提供展示层的可视化服务，本系统在控制层采用SpringMVC[34]框架来实现。SpringMVC框架基于请求驱动（Request-Driven）的方式进行设计，由继承 Java servlet的 DispatchServlet负责请求的接收与分发。SpringMVC架构通过业务层的主要工作是通过SDK的调用来完成复杂的业务逻辑，控制层与业务层的管理由Spring负责，Spring是一款轻量的企业级软件开发工具包，实现了控制反转、依赖注入、切面编程等优秀的设计模式，可以跟SpringMVC进行无缝的集成，完全实现基于接口的设计。控制层调用业务层的服务，业务层调用SDK的接口。

（2） API 服务

API服务是面向上层应用开发者的服务，体现了WEB2.0时代互联网软件的开放性与包容性。视频监控云资源管理系统对应于云计算架构的IAAS层，属于基础设施即服务，需要为上层的视频监控应用平台提供弹性计算能力。在3.2.1节有提到Openstack的Restful API的访问过程。API设计风格分为SOAP（Simple Object Access protocol）与Rest（Representational State Transfer）两种，本系统采用Rest[Restful]风格的API，Rest API是基于HTTP协议实现的一种面向资源的设计风格，通常采用名词来表示资源，用HTTP的请求类型（表3-1）来表示操作。

表3-1 http方法与操作对应表

|  |  |  |
| --- | --- | --- |
| http方法 | 对应操作 | 描述 |
| GET | READ | 读取一项或多项资源 |
| POST | CREATE | 新建一个资源 |
| PUT | UPDATE | 修改资源 |
| DELETE | DELETE | 删除 |

API的数据传输格式有两种，一种是XML格式，一种是JSON格式。虽然XML也是常用的用于数据交换的数据格式，但是相对于JSON格式而言，XML的格式复杂且解析困难，而JSON格式就有轻量级且读写方便的特点。因此本系统中使用JSON作为数据传输的格式。

WEB管理服务的模块划分依据Openstack原有的组件划分，包括虚拟机管理模块、镜像管理模块、块存储管理模块以及控制面板模块。模块与对应的代号如表3-2所示，模块基于Openstack的Java SDK进行开发，控制层的实现除了提供WEB管理服务，还提供API访问服务。

表3-2模块与对应代号

|  |  |
| --- | --- |
| 模块名称 | 代号 |
| 虚拟机管理 | server |
| 物理机 | host |
| 镜像管理 | image |
| 监控管理 | monitor |
| 存储管理 | cinder |
| 密钥管理 | key |

（2）展示层

数据可视化工作通常会用到Html、Javascript等静态页面技术，本系统为了实现复杂的逻辑需求，配合SpringMVC框架，使用JSP动态页面技术来实现，但是因为JSP技术实现机制的原因，会有响应慢的缺点，所以为了保证响应速度，页面的展示采用JSP技术，而之后的数据操作更多的使用AJAX技术作异步请求来实现。

3.3.2 虚拟机监控服务设计

虚拟机云监控是云资源管理必要的模块。监控服务的目标是实现对虚拟机运行期的状态进行有效的可视化管理，运行期的资源消耗统计分析有助于我们了解虚拟机任务的运行情况，并针对虚拟机的负载情况做出必要的策略性处理。本系统的的虚拟机监控服务由三个模块组成（图3-7）：

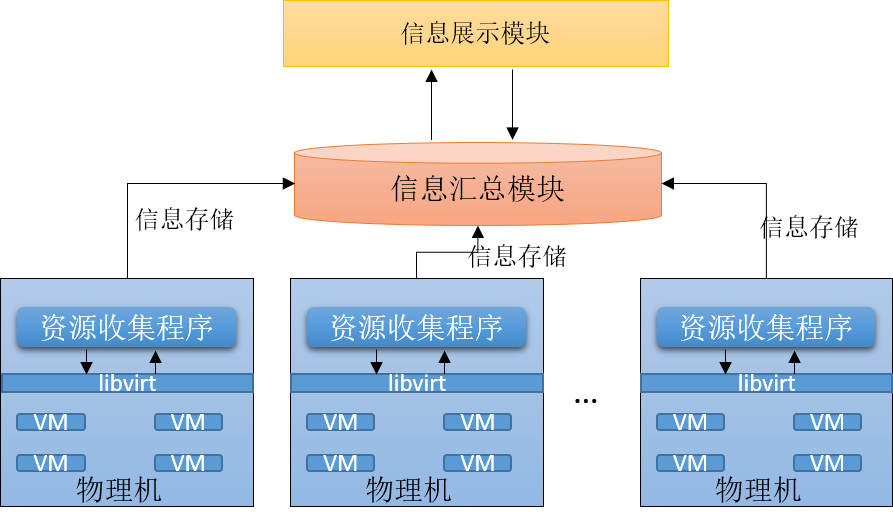


图3-7 虚拟机监控服务模块图

（1）资源收集模块

此模块用于收集各个虚拟机的资源使用情况，并上报到信息汇总模块。对虚拟机进行监控有两种常见的方式，一种是在虚拟机内部安装代理的方式[35]，另外一种是直接通过Libvirt库直接从Hypervisor层读取。安装代理的方式会在每个虚拟机内部安装用于资源收集的代理客户端，这种带有侵入性的方式不是我们所提倡的，因此本系统采用基于Libvirt库的方式。从用户的角度，虚拟机是一种提供计算服务的操作系统，但是对于宿主机而言，虚拟机是宿主机操作系统内的一个进程，Libvirt库[36]是开发者与Hypervisor之间的驱动程序，直接通过监控线程的方式来获取到虚拟机的信息。

（2）信息汇总模块

此模块负责对资源收集模块收集来的信息进行持久化处理。本系统采用Libvirt库读取虚拟机状态的方式，读取信息可以分为两种方式，一种是用户请求后，直接从Libvirt库获取信息。这种方式有两个缺点，一是网络延时大，无法满足页面实时展示的需求，二是无法存储历史信息，用户看不到历史负载状态。因此我们在监控服务中加入了信息汇总模块，用于虚拟机监控信息的存储。在实践环境中，因为资源收集是一个持续的过程，所以资源收集模块会不断的把数据上报给信息汇总模块，因此信息汇总模块需要具有存储大量数据的能力。本模块使用开源的MySQL数据库作为存储设备，而且为了提高可用性，需要对数据表进行合理的分区处理。

（3）信息展示模块

此模块通过前端技术把虚拟机的资源使用情况展示给用户，包括虚拟机的CPU、内存、网络以及磁盘的实时负载情况。一般的图表用于展示静态数据，但是因为资源监控的实时性，所以对图表也需要满足动态变化的要求，为此本系统采用highcharts图表[37]，highcharts是一款功能强大开源图表展示工具。因为视频监控云的应用环境是面向大量视频的处理，因此通常会由一组虚拟机去执行分配的任务，本系统提供了监控组的概念，用户可以把感兴趣的虚拟机放在监控组，然后再进行统一的监控查看。

3.4 本章小结

本章主要从设计的角度对视频监控云资源管理系统进行了详细的介绍。首先针对视频监控云的应用场景设计了跨地域的物理架构。然后，本章基于Openstack的原生API以及Java语言对Openstack SDK进行设计，并且在Openstack Java SDK基础上，对云资源管理系统的WEB服务以及虚拟机监控服务进行了详细设计。

# 第四章 视频监控云多维资源调度方法

上文已经对云资源管理系统的服务进行了详细设计，但是在云资源管理系统中，并没有可以直接运用在视频监控云的场景中的高效的算法，为了能够将视频监控云环境下集群的资源利用率最大化，我们提出了一种基于主导资源的DRFA算法。本章首先分析了Openstack资源调度中的问题，然后讨论了视频监控云中的云资源的三大特性以及它们对调度问题的影响，最后在建模的基础上详细介绍了算法的流程。

4.1 Openstack资源调度问题研究

数据中心就是资源的中心，数据中心的管理归根结底是资源的管理，一种好的资源管理策略对提高数据中心的服务水平与提高数据中心的资源利用率有很大帮助，否则适得其反。云平台的资源调度主要是指在虚拟机的创建请求与服务器集群的某台服务器之间建立一种合适的映射关系，这种映射关系确立的前提是服务器能够满足虚拟机创建请求且需要遵循某一种规则或者策略，通过这种策略，服务器集群能够整体对外展现一种优化的状态，这种策略称为调度策略。因此资源调度问题需要满足两点，第一是服务器能够满足虚拟机的请求，第二是服务器集群能够满足一种优化的目标。

4.1.1 Openstack资源调度工作流程

Openstack有独立的资源调度模块nova-scheduler。nova-scheduler在设计上秉承一贯的扩展性原则，用户可以根据需要定制，配置适合自己的资源调度策略，以满足自己对于资源调度的需求。

nova-scheduler工作的流程大致分为两步，第一步是资源搜集，第二步是资源调度。云计算的提供的资源依然是硬件资源，只不过是更细粒度的根据需求进行分配，所以，虚拟机运行所必须的资源仍然是CPU、RAM 和 Disk [38]。通过阅读Openstack相关源码和数据库相关表可以得出，Openstack只统计四类资源：CPU、RAM、Disk以及网络。nova- compute会搜集所在节点的资源信息，然后通过nova-conductor写入nova-DB数据库。

nova-scheduler工作的过程如图4-1所示。nova-API接收用户发出的虚拟机创建请求，并通过路由机制转到nova-scheduler模块，nova-scheduler首先从nova-DB中获取整个集群的完整的资源信息，在拥有整个集群的资源信息后，nova-scheduler通过用户配置的策略对目标主机进行选择，确定目标主机的hostID，然后将此hostID经过封装后发布到消息队列中，时刻监听消息队列的nova-compute跟自己的ID进行匹配后，开始进行资源分配

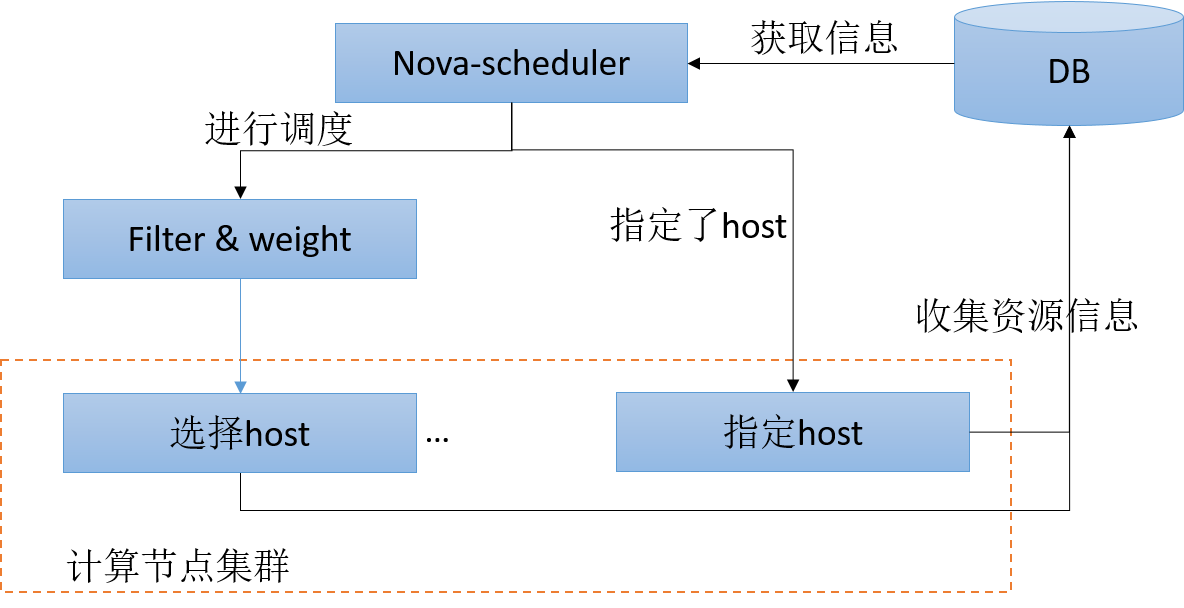


图4-1 nova-scheduler工作流程图

4.1.2 Openstack资源调度策略

Openstack提供的调度策略有ChanceFilter和FilterScheduler两种，ChanceFilter是随机调度，在服务器集群中随机一个服务器进行与虚拟机请求的映射，而Openstack默认的调度策略是FilterScheduler（见图4-2）。FilterScheduler由两阶段操作组成，当一个计算请求到来，第一阶段进行Filter，将不满足基本虚拟机请求的服务器主机过滤掉，比如会将资源不足、架构特性不符等主机过滤掉，留下符合资源虚拟机请求的主机，组成一个集合，然后第二阶段是进行Weight，对filter之后的所有服务器主机进行一种权重计算，最终根据计算出来的权重值选择合适的服务器主机进行虚拟机的资源分配。

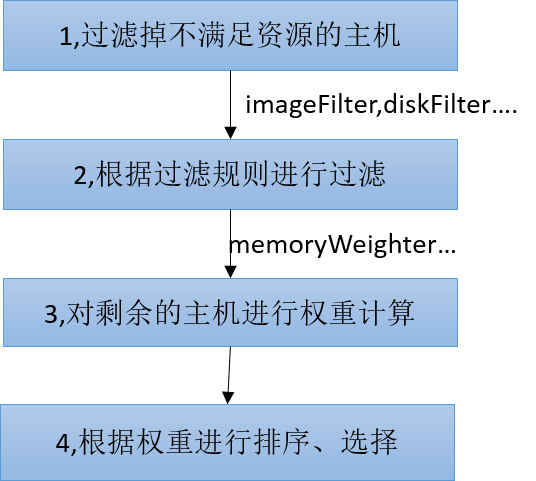


图4-2 FilterScheduler调度流程

Openstack提供的filter有20种之多，这些filter经过合理的组合之后，可以基本满足用户的过滤需求，Filter可以总结为以下几类[39]：

（1）Resource-based Filters：基于资源的过滤器。主要依据虚拟机请求的资源情况以及物理机的可用资源进行过滤，通常会过滤掉不能满足资源请求的物理机，比如coreFilter、RamFilter等。

（2）Image-based Filters：基于镜像的过滤器。在Openstack中，可以依据镜像的属性来选择合适的主机，比如某种镜像对CPU架构以及hypervisor都会与特殊的要求。

（3）Host-baresed Filters：基于主机的过滤器。该过滤器主要考虑到主机的分组或者分域情况，对于大规模数据中心，通常会有机房、机架甚至地域的区别，比如对于有固态硬盘（SSD）请求的虚拟机，肯定会优先选择那些安装有固态硬盘的服务器组，而过滤掉其它的。对地域（region）的选择通常是就近原则，尽量减少响应时间。比如SameHostFilter会选择与指定虚拟机相同位置的主机。

（4）Net-based Filters：基于网络的过滤器，会依据网段进行选择，CIDAFilter会选择在同一个网段的虚拟机。

（5）其它过滤器：包括用户自定义的过滤器等。

假设最初的服务器集合是A，在Filter结束之后，会得到一个服务器主机子集B，此时对主机集合进入权重计算，然后会对到服务器主机一个依据权重值的排序，最后在得分最高的N个主机组成的子集C中再随机选择一个主机进行虚拟机资源分配。

在Openstack中，目前实现的weighter种类仅有RamWeigher，依据内存进行调度，如果主机拥有的可用内存越多，那么其权重越大，因此，Openstack在最终的调度策略上，是采用的单一类型指标——内存。

通过研究Openstack的调度策略，我们发现Openstack的调度存在严重的问题，一个是单资源调度问题，忽视了计算机资源的多维性，单资源调度的问题会导致“短板效应”，即当某种资源分配结束后，则该服务器不再具有分配资源的能力，这就造成了大规模的资源浪费情况，导致服务器资源利用率降低。另一个问题是随机调度问题，采用过多的随机调度[40]。合适的随机调度会使得资源分配达到有效的均衡处理，但是大量的随机调度对提高服务器集群的资源利用率并无益处。

4.2 视频监控云资源特性分析

OpenStack作为一个开源的云平台，只能满足基本的云计算功能，而针对不同场景的优化问题，需要用户自己进行需求分析并提出适合自己环境的方案。本课题通过对视频监控云资源的特性进行深入研究，总结出云资源的三大特性，

**（1）多维性**：多维性是云资源最明显的特征。虚拟机的运行是多种资源共同协作的结果，每一种类型的资源都发挥着不可替代的作用，因此在分配资源时，需要考虑CPU、内存、磁盘、网络带宽等资源的合理配合。而如果忽视云资源的多维特性，往往会导致资源分配的不合理，比如对于一台服务器而言，当一种资源的分配率达到100%时，而其它资源的使用率可能比较低，但是此时服务器已经进入了不可分配资源的状态，于是造成了资源极大的浪费。这种短板问题，即我们常说的“木桶定律”。

**（2）动态性**：资源分配是一个动态的过程。在数据中心中，会有不断的资源分配请求以及资源释放请求，因此服务器的资源利用率是不断动态变化的。而且，任何一次资源分配都会直接影响下一次资源分配的决策，而且同一个资源请求在不同的时刻发出，分配结果往往也是不同的。对数据中心而言，一般都会有能力保证足够充裕的资源来处理特定时刻资源请求峰值的压力，但是一旦压力下降，就会出现大量的资源闲置情况，这也是资源动态性导致的问题。

**（3）异构性**：异构性是指同一物理机或虚拟机中不同资源之间所呈现的不平衡关系，云资源因为多维的原因，必然呈现异构性。物理机出现异构特性的原因有很多，一是因为数据中心中物理机的更新换代以及不同物理机的异步升级，二是因为虚拟机的不断分配。而虚拟机的异构性通常是由运行其上的任务特征决定的，不同的任务对于不同资源的需求不同。在视频监控云环境中，我们启用一个虚拟机来执行视频处理任务，虚拟机资源的分配依赖于任务的特性，比如车牌识别任务，需要对视频数据一帧一帧的处理，因此会消耗大量的CPU资源，而监控视频转发任务，则需要对某一路摄像视频向多个目的地点转发，就会对网络资源有大的需求，因此，在视频监控云处理环境中，面对如此复杂的资源请求环境以及云平台的特性，如何进行虚拟机与物理机的映射，是一大挑战。

4.3 多维资源调度方法

4.3.1 问题建模

云资源调度问题，是数据中心里VM资源请求集合 与服务器集群集合 之间的多对一的映射问题。现假设服务器集群中的资源共有n种，可以描述为，比如（cpu, memory, bandwidth）。资源请求集合*V*中的任意元素代表一个虚拟机创建请求，可以表示为一个多维向量 ，服务器集群*P*中的一个元素可以描述为多维向量 ，其中的元素代表了pi拥有的r1资源的数量，如果把vi的请求分配到pi上，此时pi的资源则变为：

|  |  |  |
| --- | --- | --- |
|  |  | （4-1） |

当服务器Pi的资源完全分配出去之后，pi就变成了零向量（0,0,…,0），但是这属于理想状态，因为资源多维性与异构性的存在，服务器中难免会出现资源碎片化。引起多维资源碎片化的原因有两个：（a） 某资源数量太少以致于无法进行分配；（b） 某资源数量仍然很充裕，但是因为其它资源已经分配完毕或者出现碎片化，导致同服务器的该资源无法进行分配。在多维资源分配过程中，a与b描述的现象通常同时存在，而且b现象是造成资源浪费的主要原因。

以上是任何云资源调度算法都会涉及到的最基本的建模。对于服务器集群中的某一种资源类型的剩余总容量，可以通过加和的方式计算，计算公式为，

|  |  |  |
| --- | --- | --- |
|  |  | （4-2） |

因此，所有资源的剩余总量又可以组成剩余资源向量：。剩余资源向量是对服务器集群资源宏观的表达。

**定义 1：资源分配的极限状态：**由于资源调度是一个动态的过程，因此，剩余资源向量也会随着分配过程而变化，当服务器资源被完全分配后，*T*最后变成零向量，但是这是理想状态，因此有可能会出现剩余资源向量中的每一个元素都不是零，但是却无法继续分配资源的情况出现，例如，如果剩余资源向量为（1,1），但是集群中两台服务器的剩余资源向量分别是（0,1）和（1,0），即便是集群中有再多的剩余资源，但是集群中每一台机器都已经不具有继续分配资源的能力了，我们称集群的这种状态为资源分配的极限状态。当计算集群的总资源时，则需要排除掉处在极限状态的服务器。

在数据中心中，资源分配应该是一个全局的决策过程，在以往的资源分配策略中，对服务器之间的比较大多是以服务器的剩余资源向量或者服务器的资源使用率作为度量，虽然这种比较仍然可以进行策略的选择，但是这样的比较就会导致服务器整体资源之间没有任何相互制约关系，如果想把握某服务器的资源相对于集群的资源特征，就不能仅局限于其在服务器内部的资源剩余比例，而应该考虑其在集群剩余资源总量中占的权重，因此，虚拟机请求可以表示为VM资源比例向量，计算方式为

|  |  |  |
| --- | --- | --- |
|  |  | （4-3） |

同理，服务器也有PM资源比例向量，

|  |  |  |
| --- | --- | --- |
|  |  | （4-4） |

面对异构的服务器与异构的虚拟机创建请求，充分认识异构特征才能更好的进行资源调度。在对于任务的资源分配时，加州大学伯克利分校的AMPLab在其Apache Mesos中，使用了主导资源公平（DRF）算法，目的是实现基于max-min公平实现资源分配的公平性。DRF算法提出了主导资源的概念，主导资源能够在整个集群的宏观概念中，服务器的主导资源能够代表服务器相对于其它资源的优势性，而虚拟机请求的主导资源能够代表虚拟机请求的热门资源。主导资源能够在集群总资源的视角更好描述个体的资源优势性，能够很好的把握服务器内资源的异构特征。

**定义 2：主导资源**[20]资源比例向量中数值最大的资源，代表了该资源在集群中相对其它资源更充裕，称为主导资源。因此，虚拟机的主导资源可以描述为，

|  |  |  |
| --- | --- | --- |
|  |  | （4-5） |

而服务器的主导资源可以描述为

|  |  |  |
| --- | --- | --- |
|  |  | （4-6） |

因为资源调度的动态性，剩余资源向量会随着虚拟机资源请求的分配而变化，在不同时间，服务器的主导资源是变化的，即便是同一个虚拟机请求，其主导资源类型也会依据剩余资源向量的变化而变化。

关于主导资源的概念，可以通过例子来描述，比如现在有两台服务器，资源类型R={CPU,memory}，服务器A的配置为4 cores，8GB，可以表示为 p1=（4,8）；服务器B的配置为2 cores, 2 GB，可以表示为p2=（2,2）,剩余资源向量为T=（8,10），p1比例向量为（2/3,4/5），p2资源比例向量为（1/3,1/5）。因此p1的主导资源是memory，p2的主导资源是CPU。此时有虚拟机请求（2,2），其虚拟机资源比例向量为（1/4,1/5），该虚拟机的主导资源是CPU。可以理解为该虚拟机请求的资源中CPU占的比重更大一些。而在服务器中，p2的主导资源是CPU，因为其各种资源中，CPU占有较大的比重。

本节用数学方法对对资源分配问题进行了描述，并提出了资源分配的极限状态以及主导资源的概念，下一节将集中介绍基于主导资源的优先分配算法。

4.3.2 主导资源优先分配算法

在考虑云资源的多维性、动态性和异构性的前提下，我们利用了视频监控云中任务（虚拟机）与服务器（物理机）的异构性特征，提出了适用于视频监控云的主导资源优先分配的调度算法（Domain Resource First Allocation）。

在4.3.1节中我们提到了主导资源的概念，服务器的主导资源通常是指从数据中心全局的角度来看，主导资源相对于其它资源更为丰富。举个例子，有两台服务器，p1=<2，4>，p2=<1，4>，可以计算此时的总资源向量为t=<3,8>，所以分别的资源比例向量是<2/3,1/2>和<1/3,1/4>，两台服务器的主导资源都是CPU，如果只看单服务器，那么内存数量更多，但是如果从整体而言，CPU却更有优势。假如此时有虚拟机请求v=<1,2>，会优先分配到p1上。

主导资源优先分配算法由三个部分组成，分别是主导资源优先分配算法，寻找候选物理机集合算法，以及寻找最佳匹配物理机算法。其中主导资源优先分配是整个算法的框架部分，另外两个算法属于子算法。

|  |
| --- |
| **算法1 ：**Dominant Resource First Allocation |
| **输入**：pmList，vmQueue  **输出**：allocation\_of\_VM  1 while vmQueue不空 do  2 在vmQueue 中取出vm；  3 r ← vm.RequestRatioVector；  4 candidateSet ← SeclectCandiateSet(r,pmList)；  5 if candidateSet 不是空集 then  6 candidate ← BestMath(vm,candidateSet)；  7 else  8 candidate ← Random(pmList)；  9 end if  10 allocation = (vm, candidate)；  11 return allocation；  11 end while |

主导资源优先分配算法在一维资源的情况下，会退化为Best Fit最佳匹配算法，而在多维资源的情况下，会根据主导资源的概念以及多维资源匹配标准进行最佳服务器的选择。其流程可以总结如下：

算法每次会在VM请求队列中根据FIFO原则取出VM，然后根据VM资源请求量找出所有能满足VM的PM集合，剔除那些不符合的物理机，如果此时的PM集合为空，则说明没有能够满足VM的物理机，则抛出异常；其次会根据计算出的VM的资源请求向量，求出主导资源r；在得知虚拟机主导资源的情况下，进入算法2，求出PM的候选集合；如果PM的候选集合为空，则在集群中随机选择一个可用的虚拟机进行资源分配，否则会对于候选集中每一个PM，计算其与VM的匹配距离，并选择拥有最小距离的PM，进行PM与VM的映射。

|  |
| --- |
| **算法2 :** Seclect Candidate Set |
| **输入**：r，pmList  **输出**：candidateSet  1 candidateSet ← ∅；  2 for ∀pm ∈ pmList do  3 if pm meets vm then  4 r2 ← argmax(pm.resourceRatioVector)；  5 if r2 equals to r then  6 candidateSet = candidateSet ∪ pm；  7 end if  8 end if  9 end for  10 return candidateSet |

算法2用于寻找满足虚拟机的候选物理机集合，其流程是，首先对candidateSet进行初始化为空集，然后迭代pmList，如果PM的资源能够满足vm的请求，那么计算PM的主导资源，此时已经有了虚拟机的主导资源以及物理机的主导资源，根据我们主导资源的概念，拥有同样主导资源的虚拟机与物理机进行匹配，因此判断虚拟机与物理机的主导资源是否相等，如果相等，则加入候选物理机集合。最后算法结束后，返回候选物理机集合。因此我们可以知道候选集合中的物理机都拥有根虚拟机一样的主导资源。如果此时候选集合中物理机的数量大于1，则需要进行下一步的判断。

**定义 3：多维资源匹配标准** 对于虚拟机的资源请求部署到哪一台服务器上，我们在满足主导资源的前提下，采用欧式距离（7）作为虚拟机与服务器匹配的标准，距离越小，我们认为越匹配，服务器选为目标节点的可能性越大，如果距离为零，则虚拟机的资源请求跟服务器的资源情况完全符合。

|  |  |  |
| --- | --- | --- |
|  |  | (4-7) |

计算向量之间距离的方法有很多，常用的是欧氏距离和余弦距离。我们采用欧式距离的原因是，余弦距离常用来区分相似度，而用量来区分远近，而不是相似度。因为第一步已经确定了相似度，不需要再次确定了。

算法3是进行资源分配的最后一步，依据多维资源匹配标准来选择最佳匹配的物理机。算法会迭代候选物理机集合，并计算候选物理机与虚拟机之间的欧式距离，根据多维资源匹配标准，会在迭代的过程中确定距离最小的物理机，最后直接将确定的物理机返回即可。

|  |
| --- |
| **算法 3 ：**Best Match |
| 输入: candidateSet, vm  输出: candidate  1 minDistance ← ∞  2 for ∀pm ∈ candidateSet do  3 distance ← DIST ANCE(pm, vm)；  4 if minDistance > distance then  5 minDistance ← distance；  6 candidate ← pm；  7 end if  8 end for  9 return candidate |

4.4 本章小结

本章首先分析了Openstack的默认调度算法FilterScheduler，并总结出它的不足之处，然后研究了视频监控云中的云资源的三种特性：多维性、动态性、异构性，基于这些特性，本章在对问题进行建模的基础上提出了适用于视频监控云环境的的资源调度算法DRFA，该算法基于主导资源的概念，从集群全局的角度优先把具有相同主导资源的物理机和虚拟机进行匹配，有效地利用了物理机与虚拟机的异构性。

# 第五章 系统实现与测试分析

在第三章的系统设计以及第四章的算法分析的基础上，本章详细介绍了系统的具体实现过程，并针对系统的功能以及算法进行了测试分析。本章首先介绍通过Openstack的keystone组件实现跨域物理架构，其次详细讨论使用Java语言进行Openstack SDK的实现，并在Openstack Java SDK 的基础上，使用SpringMVC框架对云资源WEB管理系统进行实现，然后对如何调用Libvirt库进行虚拟机动态的资源管理进行了详细阐述。最后本章描述了视频监控云资源管理系统的功能实现展示，并且针对多维资源调度方法进行了测试。

5.1 基于Openstack的跨域物理架构实现

多区域的云资源管理系统需要满足多域分布式部署，但是统一管理的需求。本系统基于Openstack系统来实现，将集群环境划分为两个Region，每个Region承担各自的计算服务、镜像服务、网络服务以及存储服务，而认证服务由额外的节点来提供。每个Region需要安装的具体组件如表5-1所示。

表5-1 节点与组件对应图

|  |  |
| --- | --- |
| 节点 | 组件 |
| 认证节点 | keystone |
| 控制节点 | nova-API, python-novaclient, python-neutronclient, nova-cert,  nova-conductor,nova-consoleauth,nova-novncproxy,  nova-scheduler, neutron-server，neutron-plugin-ml2, |
| 计算节点 | nova-compute, neutron-plugin-ml2, neutron-plugin-openvswitch-agent, ipset |
| 镜像节点 | glance-registry, glance-API, python-glanceclient |
| 网络节点 | neutron-plugin-ml2, neutron-plugin-openvswitch-agent,  neutron-l3-agent, neutron-dhcp-agent ipset |

Openstack的基于组件式的设计架构为分域设计提供了方便，因为所有组件的认证服务都由keystone统一负责，安装组件的时候需要在keystone注册相应的endpoint。而且为了保证安全，用户的每一次访问都必须向keystone进行身份认证，如图5-1 所示，用户需要在登录系统时验证用户名和密码，如果成功则可以获取相关的token和endpoint，然后用户再携带token向确定的endpoint发送数据请求。因此keystone可以实现region name与endpoint的路由映射功能，用户在访问keystone时指定region的名字，keystone就会返回指定域的相关endpoint。在命令行中使用keystone模块的命令时，可以使用—os-region-name <region-name>指定region的名称，如果不指定，keystone永远只访问默认的第一个region。

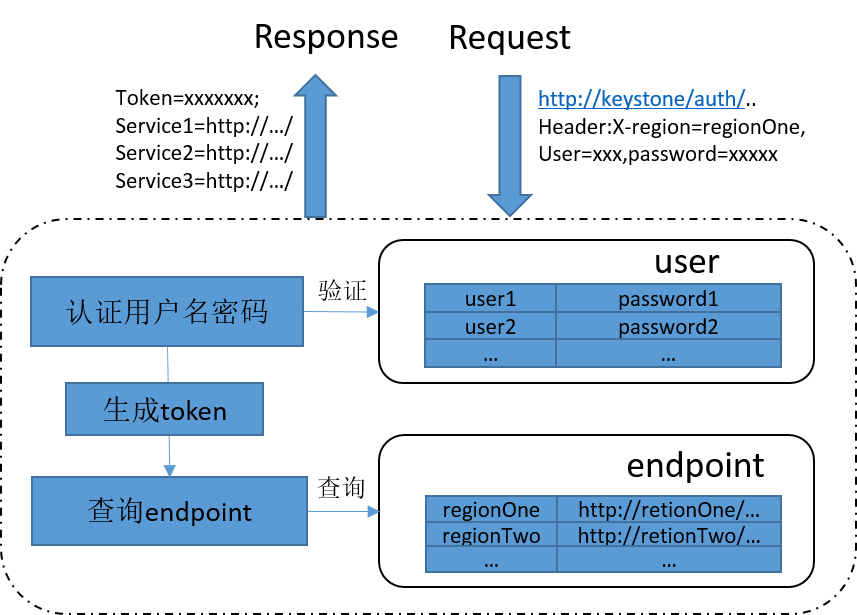


图5-1 keystone多region工作流程图

Openstack提供了region的概念，默认是只有一个region，对于多region的情况，需要在keystone中手动的添加endpoint信息，命令如下所示。Openstack针对不同的应用场景，提供不同的访问endpoint，其中pulibcurl是用户可以访问的url，而internalurl用于内部用户的访问，adminurl仅用于管理员对组件进行操作使用。

|  |
| --- |
| keystone endpoint-create \  --service-id $(keystone service-list | awk '/ identity / {print $2}') \  --publicurl *http://**regionOne: 5000/v2.0* \  --internalurl *http:// regionOne:5000/v2.0 \*  --adminurl *http:// regionOne:35357/v2.0 \*  --region regionOne |

基于分域概念的的物理架构设计具有良好的扩展性，一是域内使用的Openstack原生的计算架构，计算节点与存储节点具有良好的扩展性，二是支持域的可插拔性，当有新地域加入或退出时，在不影响到现有架构前提下，即可以实现灵活的扩展性。用户在使用时，为了提高响应速度，减少网络传输时间，应该尽量依据就近原则来部署自己的任务。

5.2 视频云资源管理系统实现

5.2.1 Openstack Java SDK实现

Openstack JAVA SDK是为使用Openstack的Java语言使用者设计的。我们的目标是帮助Java开发人员访问Openstack的底层服务，利用Openstack提供的强大的云管理能力来构建自己的云应用。Openstack Java SDK中实现了nova、glance、keystone、cinder、neutron等服务。其中nova服务的类图如图5-2所示，虽然是一个服务，但已经涵盖了整个SDK中各模块之间的继承调用关系。

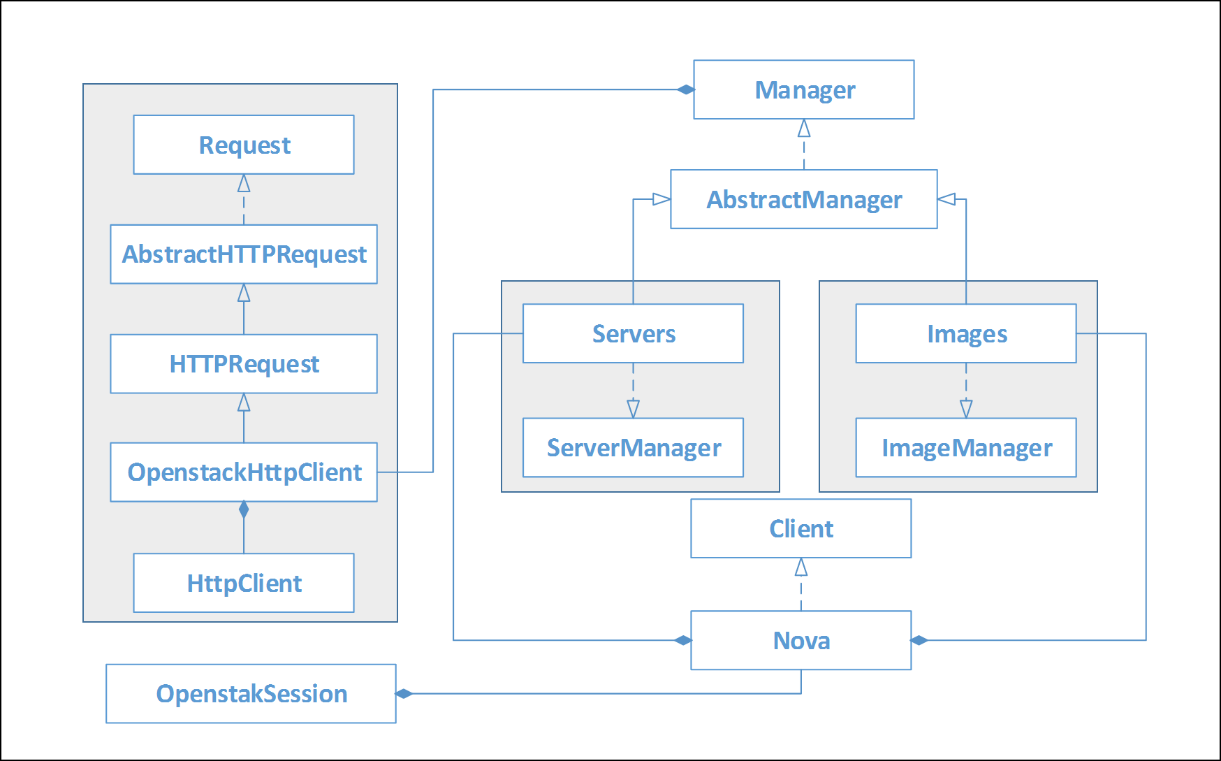


图5-2 Openstack Java SDK中 nova服务类图

（1）Connector模块

Connector模块是HTTP请求的实际发出者与接收者，共由四个主要的类构成：Request、AbstractHTTPRequest、HTTPRequest与OpenstackHTTPClient。其中，Request是一个接口，包含了所有HTTP协议定义的操作——GET、POST、PUT、DELETE还有HEAD方法，而AbstractHTTPRequest是抽象类，实现了Request接口并规范了请求的逻辑，HTTPRequest集成了AbstractHTTPRequest，并包含了CloseableHttpCLient，是请求的真正执行者，而OpenstackHTTPClient在类中添加了对于Openstack访问所必须的依赖方法。

以一次get请求为例，Connector模块各个类的代码实现如图5-3所示。Request定义了doGet接口，在AbstractHTTPRequest实现了doGet方法，但是同时又定了doGet的请求逻辑，包括三个抽象方法before、after、以及真正的处理者get方法。所有的真正的请求由HTTPRequest负责，在HTTPRequest调用了execute方法发出请求，到目前为止，这三个主类都与Openstack的业务没有多大关系，可以适用于其它HTTP处理。而真正面向Openstack请求的参数处理，包含在OpenstackHTTPClient中。至此可以了解到整个Connector模块的工作流程。

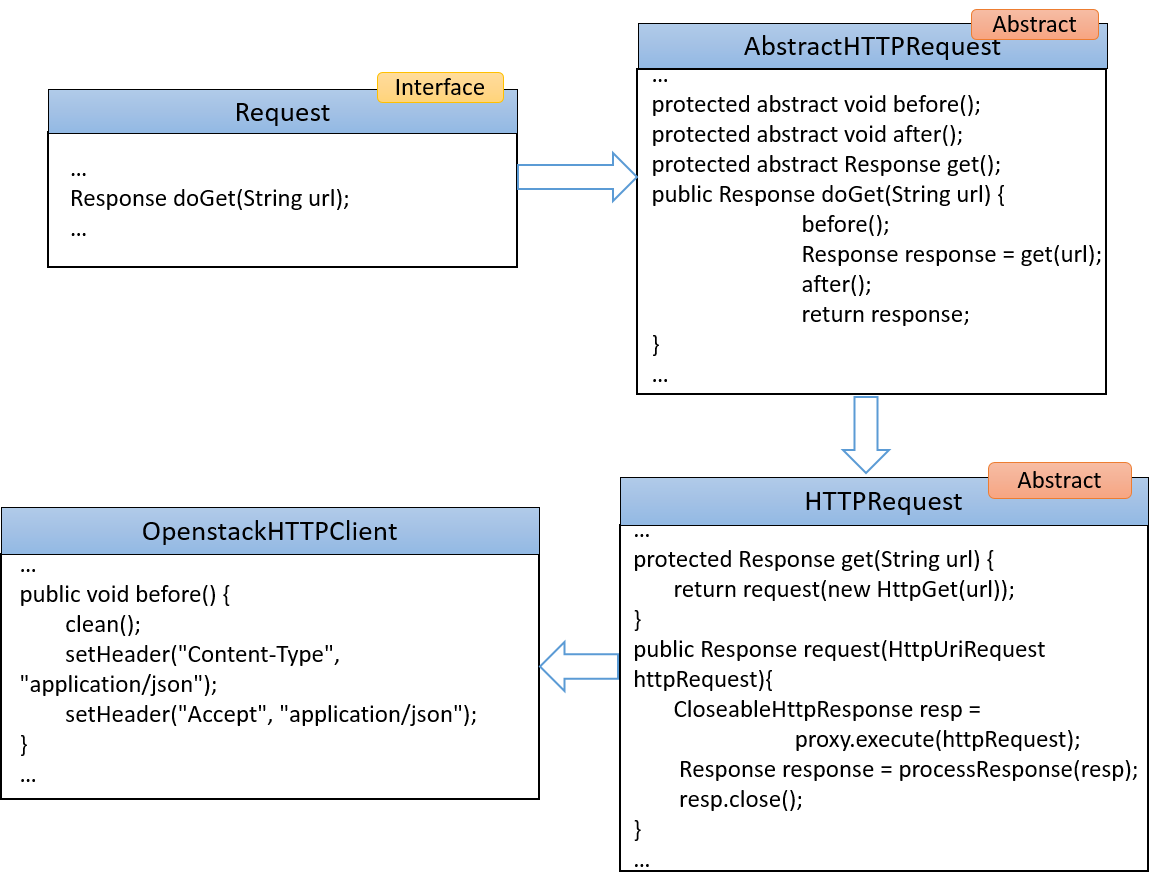


图5-3 Connector中get方法请求流程示意图

Response也是一个接口，定义了HTTP响应所需要的操作，其中getCode获取HTTP响应的状态码，其它方法是将响应流以不同的格式表现出来。

（2）服务组件模块：

|  |
| --- |
| public interface Response {  String getBody(); //获取  byte[] getBytes();  InputStream getInputStream();//以流的形式读取  int getCode(); //获取状态吗  boolean isSuccess(); //判断是否请求成功  } |

在图5-2中，由Manager、AbstractManager以及Servers、ServerManager组成的模块是服务组件模块，其中Manager与AbstractManager是公共服务类，Manager类中可以直接调用Connector模块的请求方法。Servers 继承了AbstractManager，并实现了ServerManager接口，负责虚拟机服务。ServerManager接口中定义了所有与虚拟机操作相关的方法。

服务模块的工作流程，以一次GET请求为例，当用户调用Servers的get方法时，需要提供虚拟机的id的作为参数，GET方法会将id封装为“/servers/id”的形式，然后调用Manager的\_get方法，\_get方法具体代码流程如下。

|  |
| --- |
| public Entity \_get(String url, Class cl){  Response response = null;  url = getEndpoint() + url; //调用AbstractManager的方法  response = client.doGet(url); //调用OpenstackHttpClient实例的get方法  int code = response.getCode();//获取状态码  String body = response.getBody(); //获取内容  if (*notIn(code, 200, 300)) { //根据状态码判断请求是否成功*  throw new OperationException(“ERROR”);  }  return JSONConvert.*responseToEntity(body, cl,); //数据处理*  } |

服务组件模块包括虚拟机组件、镜像组件、块存储组件以及认证服务组件，目前主要实现基础服务，在设计上，支持对其它组件的扩展，而且兼容Openstack API的升级。

（3）Session管理模块

Session模块的工作是提供用户统一的访问入口，并提供各个组件的访问实例。Session模块主要由OpenstackSession、Secret、Credentical三个类组成。OpenstackSession 类采用静态工厂的模式提供各种组件的实例。Session模块的处理流程是，先通过认证服务组件获取有效认证，然后再获取不同的组件认证。OpenstackSession的调用方式如下所示：

|  |
| --- |
| OpenstackSession session = OpenstackSesion.getSesion(user,password);  Nova nova = session.getNovaClient(); //获取组件实例  Server server = nova.get(serverID); //获取某个虚拟机的详细信息 |

OpenstackSession.getSession(user,password)方法需要提供用户名和密码信息，其处理过程如下所示：

（a）从本地读取配置文件，获取API的版本，数据库的配置信息。

（b）根据用户名，从数据库中读取其管理的租户信息。

（c）根据用户名密码以及租户信息组成Secret三元组<user, password, tenaniId>，通过Keystone.authenticate（Secret）方法进行认证。

（d）认证成功则获得认证信息Authenticated credentical。Credentical中携带了endpoint，token等必要信息。

SDK的配置文件放在/etc/openstack/openstack.proporty文件夹下，描述的具体内容示例如下：

|  |
| --- |
| #定义Openstack 入口地址  authURL = http://controller:5000/v2.0  isCached = true #是否开启缓存功能  isLazy = true #是否开启延时加载，默认为true  DBUrl = jdbc:mysql://controller:3306 #数据库访问url  DBUsername = username #数据库用户名  DBPassword = password #数据库密码  DBDriver = com.mysql.jdbc.Driver #数据库驱动程序 |

5.2.2 WEB管理服务实现

（1）Tomcat服务器配置

本系统的WEB管理服务采用Tomcat作为服务器，Tomcat是开源的Java servlet容器，对静态网页（html）和动态网页(JSP)都有很好的支持。在WEB开发中，通过动态网页技术，用户与后台的交互可以实现页面的动态刷新。而在实际应用过程中，为了让Tomcat达到更佳的工作状态，需要针对一些参数进行调整。首先需要调整Java虚拟机（JVM）的运行参数，Tomcat是Java语言编写的，也会受到Java语言的限制，为了避免JVM过多的垃圾回收对Tomcat性能的影响，因此需要调节JVM堆内存空间的大小：

|  |
| --- |
| JAVA\_OPTS="-XX:PermSize=64M -XX:MaxPermSize=128m  -Xms512m -Xmx1024m” |

Tomcat中的Connector模块负责HTTP连接的接入，但是默认采用的是阻塞的BIO线程模型，请求数量与线程数量成线性关系，当请求过多时，会导致占用过多的计算资源，且响应时间变长，所以为了提高Tomcat的响应能力，需要配置异步NIO线程模型的Connector，采用较少的线程数就可以完成大量连接请求的处理。具体的配置如下所示。

|  |
| --- |
| <Connector port=”8080”  protocol=”org.apache.coyote.http11.Http11NioProtocol”  maxThreads=”100”  connectionTimeout=”20000”  redirectPort=”8443”> |

1. API服务实现

SpringMVC主要是通过注解的形式实现API服务。为了更好的面向资源进行系统实现，SpringMVC抛弃了传统WEB设计中一个Servlet对应一个请求的模式，而是通过注解@RequestMapping的形式，实现请求与方法之间的映射，达到资源更细粒度的管理。而且因为SpringMVC采用@Responsebody注解，既能返回某个view页面，也能直接返回JSON数据。SpringMVC中处理Restful API的流程如图5-4所示。一般在实现中，一种资源对应一个Controller，方法对应于对资源的操作，具体的Controller类图如图5-5所示。

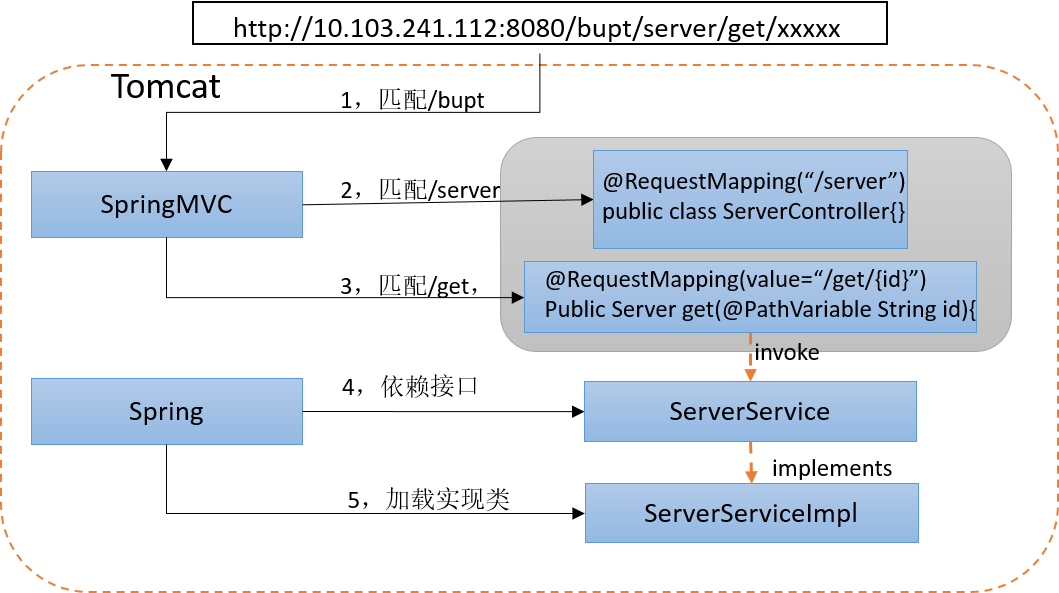


图5-4 SpringMVC与Spring处理url流程

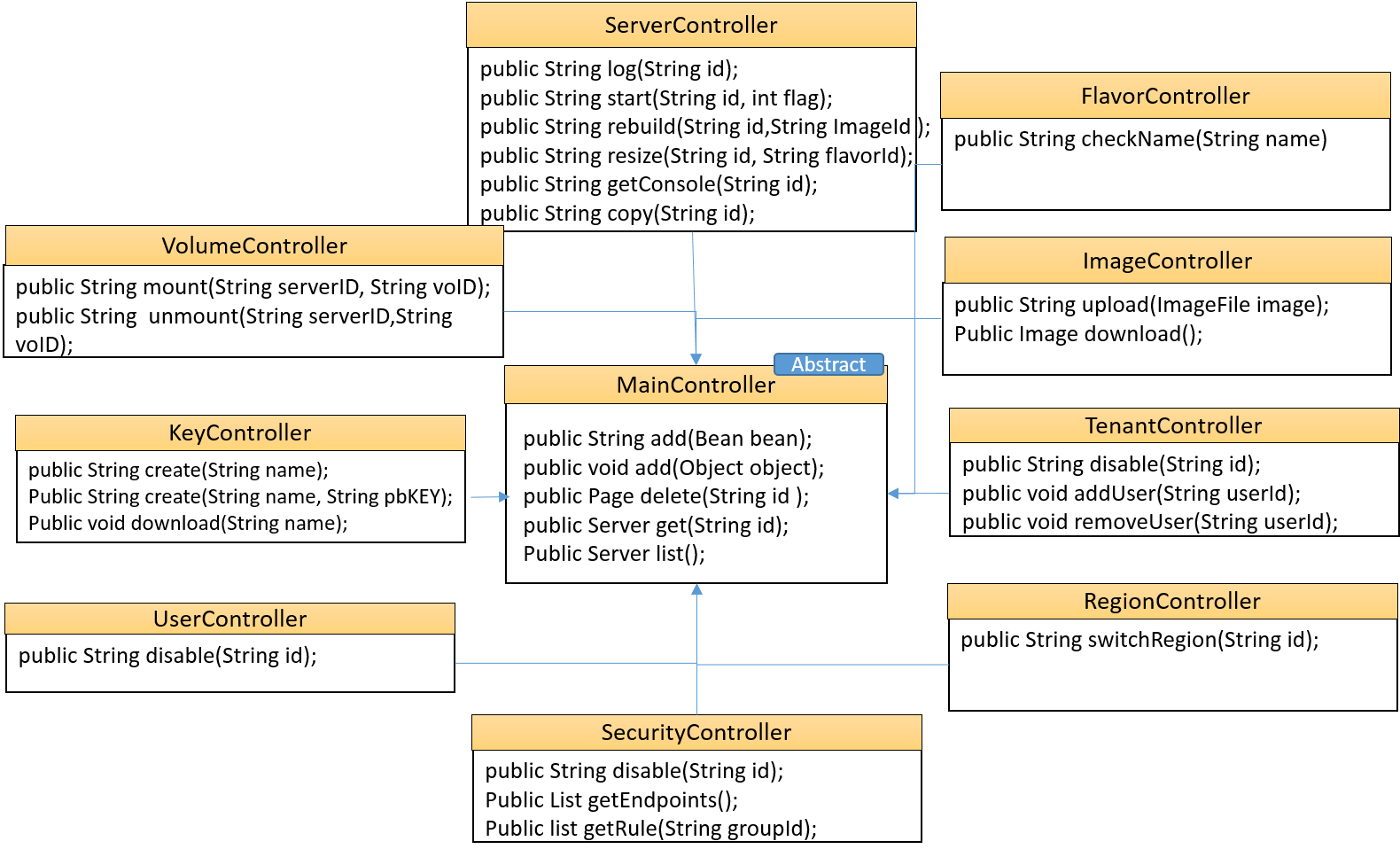


图5-5 Controller类图

API服务的功能主要通过调用Openstack Java SDK实现。虚拟机服务在控制层主要由ServerController类负责，虚拟机是资源管理的主体，同样需要依赖其它资源的支持，比如FlavorController、ImageController 、VolumeController类，其中FlavorController提供对虚拟机配置的管理，ImageController负责镜像的管理，VolumeController负责对块存储的管理。虚拟机创建提供的接口如图5-5所示。重点介绍一下虚拟机创建流程，

(a)创建虚拟机接口

创建虚拟机的具体方法如下代码所示，需要提供的参数见表5-2.

|  |
| --- |
| http://10.103.241.112:8080/bupt/server/create?name=<name>&flavorId=<flavorID> &imageId=<ImageID> &keyName=<keyName> |

表5-2 创建虚拟机API参数

|  |  |  |
| --- | --- | --- |
| 参数名 | 参数类型 | 描述 |
| name | String | 虚拟机的名称 |
| imageID | String | 镜像的ID |
| flavorID | String | 虚拟机配置的ID |
| keyName | String | 登陆密钥的名称 |
| SecurityGroup | String | 安全组的名称，默认为default |

（b）上传镜像接口

镜像服务主要由ImageController类以及ImageService类负责，主要提供镜像的基本管理功能，包括镜像列表、上传、删除等。镜像属于硬盘镜像，即不需要安装可以直接启动。镜像上传功能的API如下代码所示，具体参数如表5-3所示。

|  |
| --- |
| http://10.103.241.112:8080/bupt/image/upload?name=<name> &minDisk = < minDisk > &diskFormat=<format > &minRam=< minRam > |

表5-3 镜像服务上传接口参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数类型 | 描述 |
| name | String | 镜像名称 |
| minDisk | int | 磁盘最小量 |
| minRam | int | 内存需求最小值 |
| diskFormat | String | 磁盘格式 |
| private | Boolean | 是否公开 |
| file | File | 镜像文件 |

（c）挂载云盘接口

块存储服务由volumeController以及volumeService负责，可以提供逻辑卷的列表、创建、删除、挂载以及解除挂载功能。镜像挂载功能的API如下所示，具体的参数如表5-4所示。

|  |
| --- |
| http://10.103.241.112:8080/bupt/volume/mount?id=<volumeID>&serverID=< serverID > |

表5-4 虚拟机挂载云盘接口参数列表

|  |  |  |
| --- | --- | --- |
| 参数名 | 参数类型 | 描述 |
| id | String | 逻辑卷的id |
| serverID | String | 虚拟机的ID |

（3）WEB管理服务

WEB管理主要体现在页面展现上，而页面展现的数据主要是来自对API的调用，因此http请求的主要流程如图 所示。因为本系统是对openstack进行的封装，需要维持对openstack的session连接，所以本系统将openstackSession保存在session中，使用springMVC拦截器对于每一次http请求进行拦截，并检查用户是否登陆以及openstackSession是否已经过期，如果已经过期则跳转到登录页面，否则将openstackSession放入sessionUtils类中，提供给sevice层使用，避免了通过传递参数的繁琐。而且表现层采用JSP技术来实现页面的动态请求展现，同时为了避免JSP加载缓慢的缺点，在页面中尽可能的使用ajax异步技术提高页面加载速度。具体的请求流程如图5-6所示。

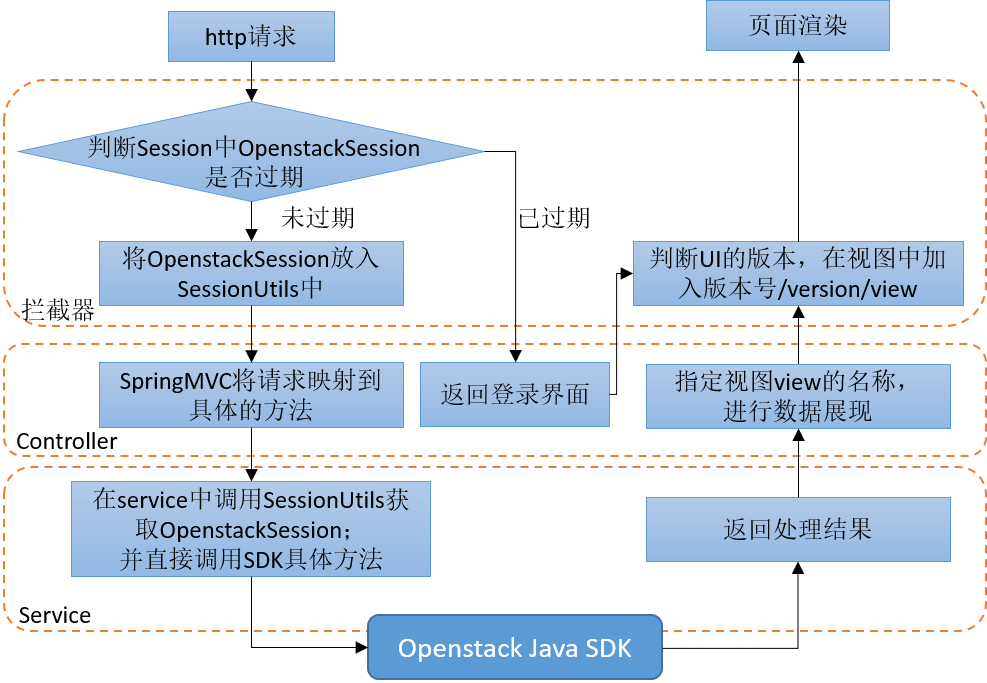


图5-6 WEB请求流程

5.2.3 虚拟机监控服务实现

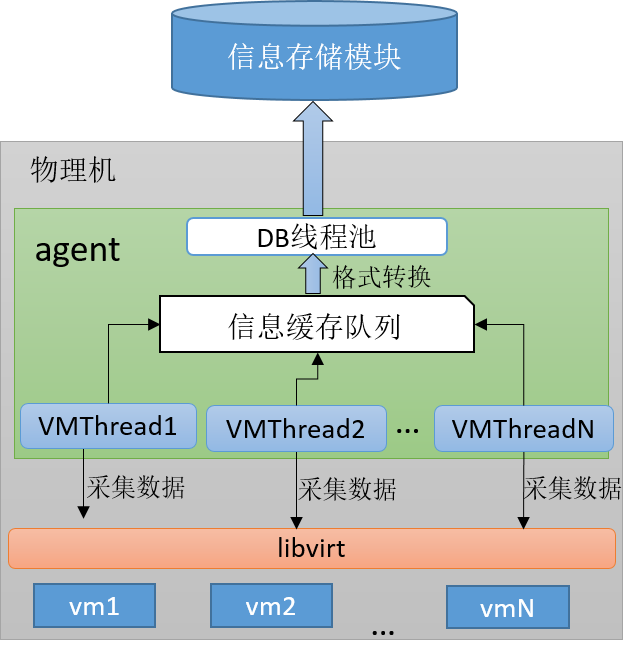


图5-7 资源收集模块

（1）资源收集模块

为了达到信息收集的目的，本模块在每一台物理机中都预先部署了资源收集代理程序Agent。Agent工作流程如图5-7所示。Agent直接通过Libvirt读取虚拟机的实时状态信息，Libvirt读取虚拟机信息的部分代码如下所示。

|  |
| --- |
| Connect con = new Connect(“10.103.241.112”);  Domain vm = con.domainLookupByUUIDString(“xxxxxxxxxx”)  vm.getMaxMemory(); //获取虚拟机内存  vm.getMaxVcpus(); //获取虚拟机CPU核数  vm.getName(); //获取虚拟机名字  … |

因为虚拟机的运行是一个动态的过程，所以需要一直不断的进行监测，本模块采用多线程的方式，一个线程负责几个虚拟机的监控。而大量的监测同样会耗费物理机的性能，因此本模块设置两次监测的间隔为5秒。虽然每一个监控线程还可以负责数据的持久化工作，但是因为集群中有大量的虚拟机，如果同时向信息存储模块写入数据，会导致信息存储模块的拥塞状态，为了避免这种现象的发生，我们采用生产者与消费者的模式，每一个监控线程收集到的资源信息都先存入缓存队列，而数据存储的工作由DB线程进行负责。因此资源收集模块的流程是：

（a）虚拟机监控线程：

InstanceThread newInstanceThread(String ip, String uuid, long time);

表5-5 虚拟机监控线程参数

|  |  |  |
| --- | --- | --- |
| 参数名 | 参数类型 | 描述 |
| ip | String | 物理机IP，默认为localhost |
| uuid | String | 虚拟机的唯一标识符 |
| time | long | 读取数据的时间间隔 |

虚拟机监控线程从Libvirt获取的数据需要保存在InstanceData的数据结构中，该数据结构与信息汇总模块的instance表对应。然后数据会放到消息队列中，消息队列采用阻塞型的队列，队列长度依据当前虚拟机监控线程的多少设定，假设监控线程数为n，根据经验，队列长度应该不小于2n。InstanceThread中提供的方法接口如表5-6所示。

表5-6 虚拟机线程提供的方法列表

|  |  |  |
| --- | --- | --- |
| 方法名称 | 参数 | 描述 |
| run() | null |  |
| monitor | String uuid | 查询虚拟机的信息，返回InstanceData |
| reConnect | String ip | 连接Libvirt库，并获取connection |
| putToQueue | InstanceData data | 将数据放入消息队列中 |

run方法中定义了对于单虚拟机监控的主要流程，如下所示。

|  |
| --- |
| while (true)  {  if connection 不可用；  conncetion 🡨reconnect(ip)；//重新建立连接  Instance data 🡨monitor(uuid); //获取监控数据  if data 不为空；//如果数据可用  putToQueue(data); //放入消息队列  } |

（b）DB线程池

DB线程池相对于监控线程，属于消费者的角色。DB线程负责取出消息队列的数据，并存储到信息汇总模块中。DB线程的数量依据监控线程数来设置，通常设置为监控线程数的一半。因为消息队列是阻塞型的，当消息队列为空的时候，DB线程会自动处于阻塞状态，等待消息的来临。创建DB线程的方法为：

|  |
| --- |
| DBThread newDBThread(); |

newDBThread的启动需要调用run方法，不需要任何参数，在DBThread类中提供的方法如表5-7所示。

表5-7 DBThread方法列表

|  |  |  |
| --- | --- | --- |
| 方法名 | 参数 | 描述 |
| run | null |  |
| getFromQueue | null | 在消息队列中获取数据 |
| insertTo | InstanceData data | 将消息存入数据库 |

（2）信息汇总模块

信息汇总模块主要是用于存储资源收集模块获取的虚拟机实时状态信息。该模块使用Mysql作为存储工具，存储引擎使用mysiam[]，相对于innodb存储引擎，myisam有更高的内存使用率，读写性能较高，而且Myisam采用的是索引和数据分离的设计方式，非常方便数据迁移。

信息汇总模块主要涉及到三个表，第一个是uuidlist表。因为在Libvirt层面，虚拟机使用uuid作为唯一标识，而Openstack在实现上又根据自己需求重新设计了ID，因此为了保证二者的匹配，我们设计了uuidlist表（见表5-8）。

表5-8 uuidlist表

|  |  |  |
| --- | --- | --- |
| 列名 | 字段 | 描述 |
| id | varchar(24) | Openstack存储的虚拟机标识符 |
| uuid | varchar(36) | 虚拟机底层的唯一标识符 |

虚拟机的实时状态信息存储在Instance表（见表5-9），主要记录虚拟机在某个时刻的信息状态。因为虚拟机的状态信息是实时获取的，以每隔5秒监控一次为例，那么一台虚拟机一天就会产生17280条记录。云平台中通常会由大量的虚拟机同时运行，因此数据量是非常大的。为了解决表中数据量大的问题，我们采用Mysql强大的partion功能实现数据表的分区，分区以后可以实现每天的数据存储在不同的表中，对不同日期的数据访问实现分流，提高并发度，而且不会因为数据量过大而导致的数据查询速度降低。下面是创建instance表的sql语句，而表中各字段的含义如表所示

|  |
| --- |
| CREATE TABLE instance (  uuid varchar(20) not null,  time datetime ,  data varchar(255)  ) ENGINE=MYISAM DEFAULT CHARSET=URF8  PARTITION BY hash(TO\_DAYS(time)) partitions 31; |

表5-9 instance表

|  |  |  |
| --- | --- | --- |
| 列名 | 字段 | 描述 |
| uuid | varchar(36) | 虚拟机唯一标识符 |
| time | Timestamp | 监控时刻 |
| data | varchar(255) | 监控信息 |

为了方便用户的监控操作，提供虚拟机的分组监控服务，所以设计了用户分组表group（表5-10）。group表中存储了用户对虚拟机进行分组的信息，包括了分组名、

表5-10 group 表字段信息

|  |  |  |
| --- | --- | --- |
| 字段名 | 字段类型 | 描述 |
| id | varchar(32) | 分组的标识符 |
| name | varchar(16) | 分组的名称 |
| userId | varchar(25) | 用户id |
| uuid | varchar(36) | 虚拟机唯一标识符 |

（3）信息展示模块

信息展示模块的目的是实现虚拟机监控的API功能，并在前端页面中以图表的形式实现。信息展示模块的页面集成在云资源管理系统中，用户的权限认证仍由资源管理系统的统一认证服务来管理。信息展示模块提供两个API，包括获取初始监控数据（见图5-8）和获取虚拟机实时数据。

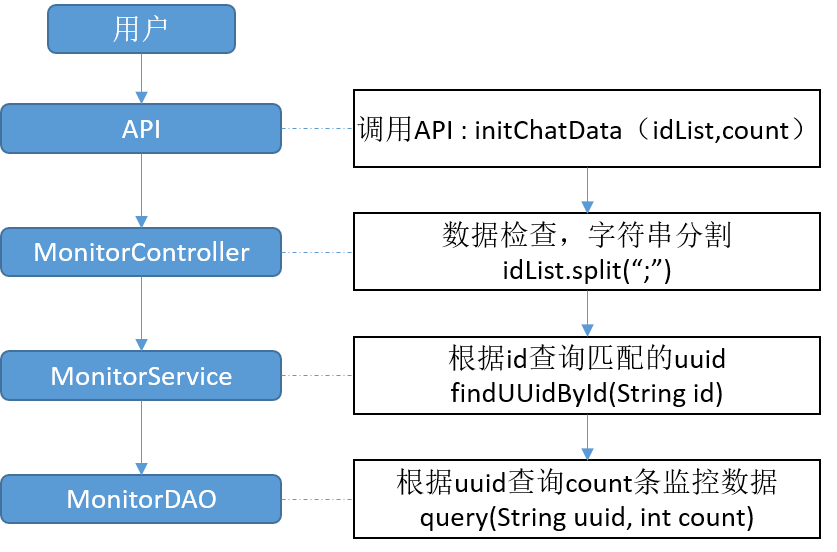


图5-8 initChatData的执行流程

1. 获取初始监控数据

String initChatData(String idList，int count);

initChartData只在图表刚开始显示时用来获取渲染数据。需要提交两个参数，一个是虚拟机的id列表，采用字符串的形式传输，不同的id用分号隔开，另外一个参数是图表初始化渲染时所需要的最少数据数目。InitChartData的处理流程如图5-8 所示，当MonitorController类收到请求时，在分割idList后，会确认每个ID的有效性，然后会在MonitorService类中调用 getUidByIds方法，把所有的id转化为uuid，然后再调用MonitorDao的getInitData方法，该方法会直接从数据库中查询出数据，最后返回到页面进行展示。

（b）获取虚拟机实时数据。

String currChartData(String idList，Long time);

当图表中已经开始展示数据的时候，页面会使用ajax技术每隔一定的时间来更新图表，实现图表的动态化。在该方法的参数中，idList仍然是虚拟机的id列表，而time是图表中显示的最后监控时间，然后按照time和当前时间的时间范围进行数据查询。具体的操作流程跟initCharData相似，但是会调用MonitorDao的getCurData方法。在页面会获取到的json数据格式如下所示：

其中开始的字符串代表虚拟机ID，t代表了时间序列，而d就是查询到的数据，包括磁盘信息（block）、CPU信息、内存信息（mem）以及网络信息（inteface）。

|  |
| --- |
| a4ce9706-280d-4f24-9a2d-4542e07e46f1: {  t: 1452219540，  d: { block: {  vda: {total: 1073741824, active\_time: 0, used: 2564096}  }  cpu: {  core\_time: "36803.2,",useAge: 0,  mhz: "1861", usedTime: "2298220.000"  }  interface: {  vnet5: {trapac: 69898, trns: 0, tb: 6834664, revs: 0, rb: 2834231673, revpac: 45048071}  }  mem: {  rss: 136960, actual: 524288, use: 104444  }  }} |

5.3 功能测试与结果验证

5.3.1系统环境

本系统采用的硬件服务器如表5-11所示。本系统是在linux环境下运行，因此所有服务器均安装Ubuntu-server 14.04发行版（Trusty），集群虚拟化管理平台采用Openstack Juno版本。在Openstack中，由于认证节点和域控制节点对性能要求不是很高，所以各安装在表5-11 序号为3的服务器中，而其它所有服务器均安装为计算节点。另外在node1和node2和node5中扩展3TB的存储，因此需要安装cinder块存储服务，具体模块安装参照表5-1。

表5-11 服务器配置列表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 类型 | 厂家 | 产品描述 | 数量 |
| 1 | 2U机架式服务器 | 浪潮NF5260M3 | CPU：32core，2.6 GHz；  内存：64GB， | 2 |
| 2 | 1U机架服务器 | Dell | CPU：32 core，2.4 GHZ  内存：32GB | 2 |
| 3 | 1U机架服务器 | Dell | CPU：4 core，2.4 GHZ;  内存：8 GB | 3 |
| 4 | 工作站 | Dell | CPU：24core，2.4 GHZ；  内存：32GB | 4 |

本系统的软件开发环境如表5-12所示。

表5-12 软件开发环境

|  |  |
| --- | --- |
| 软件 | 版本 |
| 程序开发环境 | windows 7 |
| 程序开发IDE | Eclipse Kepler |
| 运行环境 | Java JDK7 |
| WEB服务器 | Tomcat 7.0 |
| 数据库 | Mysql / mariadb |
| 第三方软件 | HTTPClient 4.3.4、Spring Framework4.0、  log4j 2.0 |

系统主要基于Java语言进行开发，并且以shell语言作为辅助。在开发结果的呈现中，Openstack Java sdk以Java的.jar为后缀名进行打包，而云资源管理系统最终以Tomcat的.war为后缀名进行打包，并部署在tomcat的WEBapp目录下。

5.3.2 功能测试

我们采用功能测试的方法进行测试。验证方式包括：一是使用测试用例的形式检验云资源管理系统中主要功能的运行情况；二是对系统创建的虚拟机的实际使用情况进行验证。

（1）用户登录

用户登录的此时用例如表5-13所示。当用户在浏览器中输入地址为http://10.103.241.112:8080/bupt/时，系统跳转到登录页面。登录系统时，会提示输入用户名和密码（图5-9），如果输入错误，则会提示“登录失败，请重新输入”，如果登录成功，则会自动跳转到资源概要信息页面（图5-10），资源概要信息中提供的当前用户的资源使用情况，包括CPU、内存、IP数量、安全组等信息，这里的资源使用量是用户的真实信息，而总量是用户能使用的最高额度信息。

表5-13 系统登录服务测试用例

|  |  |
| --- | --- |
| 用例名称 | 系统登录 |
| 测试目的 | 测试系统登录服务是否正常 |
| 测试流程 | 1. 用户输入用户名和密码 2. 查看成功跳转到系统主页 |
| 测试结果 | 登录成功后会跳转到系统首页；否则会显示登录失败 |



图5-9 系统登录页面



图5-10 资源概要信息

（2）虚拟机服务

虚拟机创建创建的用例如表5-14所示，按步骤进行填写或点击必要的信息（图5-11），在点击提交之后，虚拟机进入创建阶段，此时为了验证虚拟机的运行情况，可以通过主机管理的“打开终端功能”，对虚拟机进行操作（图5-12）。

表5-14 虚拟机创建服务测试用例

|  |  |
| --- | --- |
| 用例名称 | 虚拟机创建服务 |
| 测试目的 | 测试是否可以创建可用的虚拟机 |
| 测试流程 | 1. 进入主机申请页面 2. 输入或选择创建虚机必要的信息， 3. 点击创建 |
| 测试结果 | 通过网络终端可以直接访问虚拟机 |



图5-11 虚拟机创建功能

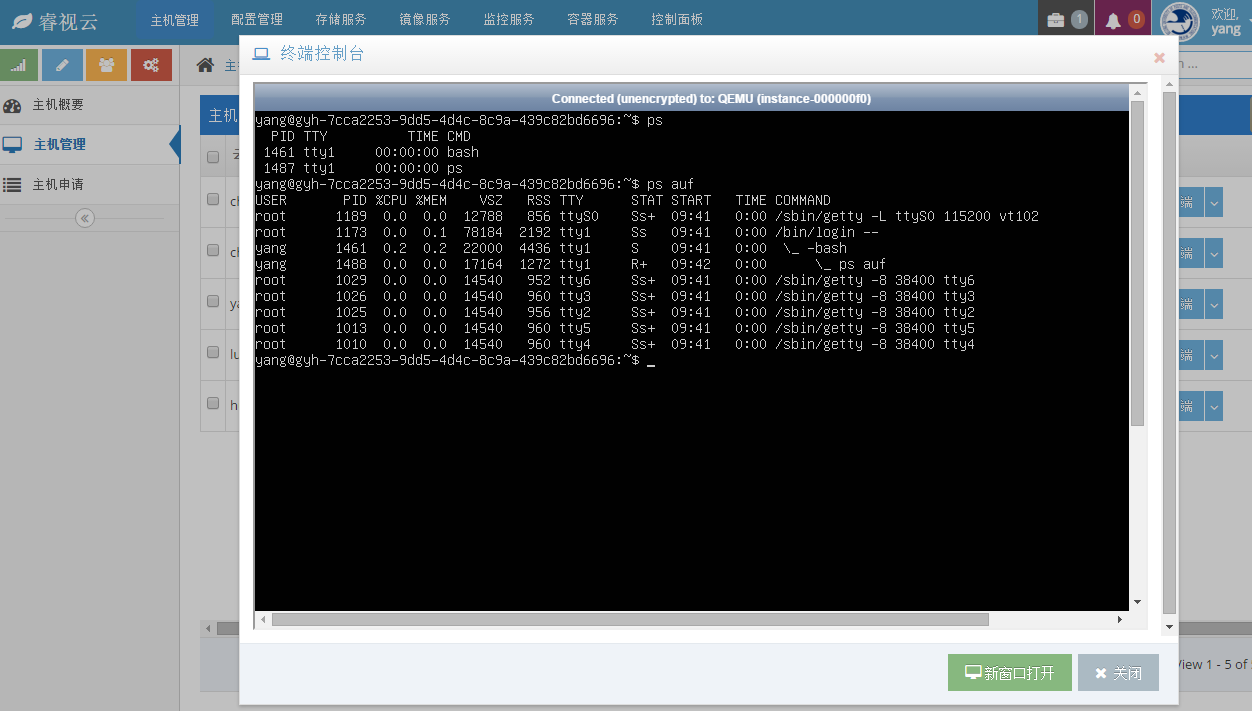


图5-12 网络终端功能

（3）虚拟机监控服务

表5-15 虚拟机监控服务测试用例

|  |  |
| --- | --- |
| 用例名称 | 虚拟机监控服务 |
| 测试目的 | 测试是否可以监控虚拟机的运行状态 |
| 测试流程 | 1. 进入监控管理页面 2. 创建监控项目并添加虚拟机 |
| 测试结果 | 显示出虚拟机的动态运行信息 |

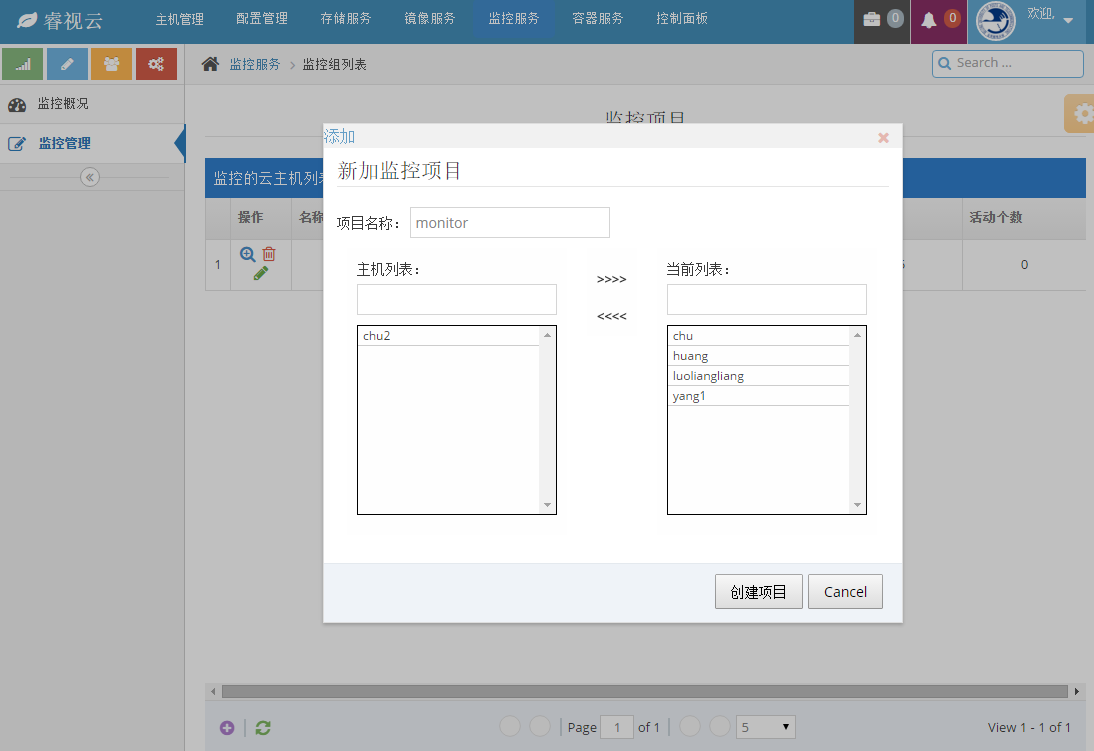


图5-13 虚拟机监控组管理功能

用户可以先选择指定的虚拟机并创建监控组，然后只对监控组中的虚拟机进行监控，如图5-13所示，监控信息如图5-14所示，每一个图表是一个虚拟机的运行状态，可以看到虚拟机的CPU、内存、磁盘以及网络的实时信息。对于每一个图表都可进行放大后详细观看，如图5-15所示。

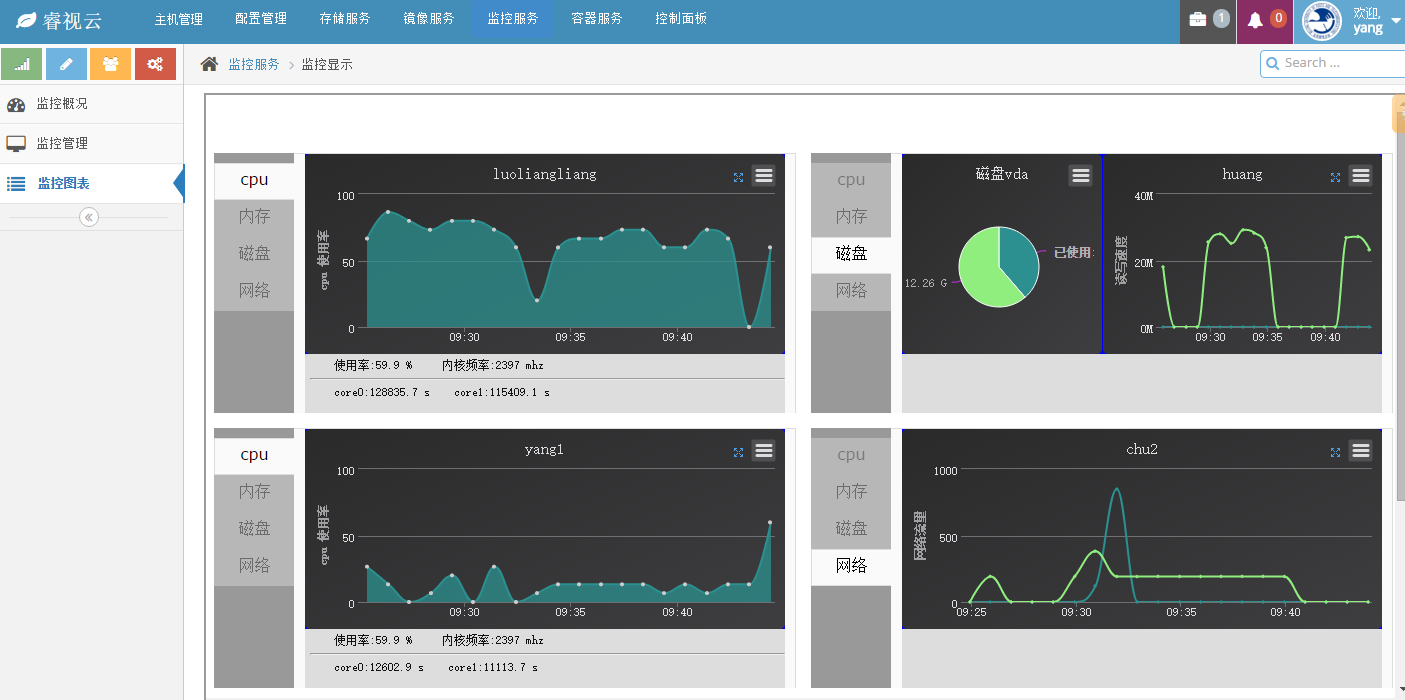


图5-14 多虚拟机监控信息图表



图5-15 单虚拟机监控信息图表

5.3.3 结果验证

在第四章中我们以最大化服务器资源利用率为目标提出了主导资源优先分配算法（DRFA），通过分析算法可以知道，DRFA主要基于服务器资源量以及虚拟机请求量进行调度，不涉及其它硬件参数，因此采用模拟的方式进行仿真是可行的。资源分配的仿真需要考虑三部分内容，包括实验环境、实验对比算法以及实验结果统计。下面就这三个部分进行介绍

（1）实验环境

资源分配需要考虑物理机与虚拟机的配置情况，为了对实践环境进行较为真实的模拟，本实验中物理机的配置以表5-16所示的配置为准，而虚拟机配置类型包括两部分，一部分是采用Amazon ec2中提供的标准虚拟机实例类型，第二部分采用实验室中视频处理应用的虚拟机实例类型，如表5-17 所示，其中Standard系列是Amazon ec2提供的，而High系列，是实验室视频处理应用对应的虚拟机类型。

表5-16 物理机配置列表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 物理机 | CPU | 内存 | 网络 | 数量 |
| 1 | 64 core | 128 GB | 100 M | 2 |
| 2 | 64 core | 64 GB | 50 M | 2 |
| 3 | 32 core | 64 GB | 50 M | 2 |
| 4 | 32 core | 32 GB | 50 M | 2 |
| 5 | 16 core | 32 GB | 50 M | 1 |

表5-17 虚拟机实例配置列表

|  |  |  |  |
| --- | --- | --- | --- |
| 配置类型 | vCPU | 内存 | 网络 |
| Standard Tiny | 1 core | 1 GB | 1 M |
| Standard Small | 1 core | 2 GB | 2 M |
| Standard Medium1 | 2 core | 4 GB | 8 M |
| Standard Medium2 | 4 core | 8 GB | 4 M |
| Standard Medium3 | 4 core | 8 GB | 10 M |
| High-memory | 8 core | 16 GB | 10 M |
| High-bandwidth | 8 core | 16 GB | 20 M |
| High-CPU1 | 16 core | 16 GB | 10 M |
| High-CPU2 | 16 core | 16 GB | 20 M |

（2）实验方法

我们提出的调度方法的目的是通过分析服务器与虚拟机配置的异构性，实现服务器资源利用率的最大化，因此实验中以集群中各资源的利用率为评价指标。实验的流程是：通过随机函数产生虚拟机的请求序列，然后使用该序列进行虚拟机请求，集群根据不同的资源分配策略进行资源分配，直到集群中的服务器均不能再分配资源为止，统计所有服务器的资源利用率，并取平均值。而且为了屏蔽掉实验的客观因素，每次实验都重复20遍，取最后的平均值。

（3）实验对比算法

本实验同Openstack默认的资源调度方法FilterScheduler（简称FS）以及VectorDot[16]（简称VD）方法进行对比。其中FilterScheduler方法是单资源调度中的worst fit 方法，以内存为指标，每次选出满足请求的内存值最大的物理机进行资源分配。

VectorDot是多维资源匹配方式，以两个向量积作为标准，每次选取值最小的物理机进行资源分配。计算公式如下：

其中A=(a1,a2,a3,,,an)指物理机的资源利用率向量，B=(b1,b2,b3,,,bn)指虚拟机请求量占物理机的总资源的比率向量。比如有某物理机资源向量(8,8)，其资源利用率为A=（1/4,1/2），此时有虚拟机请求（2,2），因此会有B=(1/4,1/4)，结算为dotproduct(A,B)=3/16。

（4）实验结果

（a）实验一

为了验证算法应用场景的一般性，实验一使用表5-17 中提供的所有虚拟机类型进行资源请求，直到物理机不能再分配资源为止，统计物理机的资源利用率。通过计算得到如图5-15所示结果。

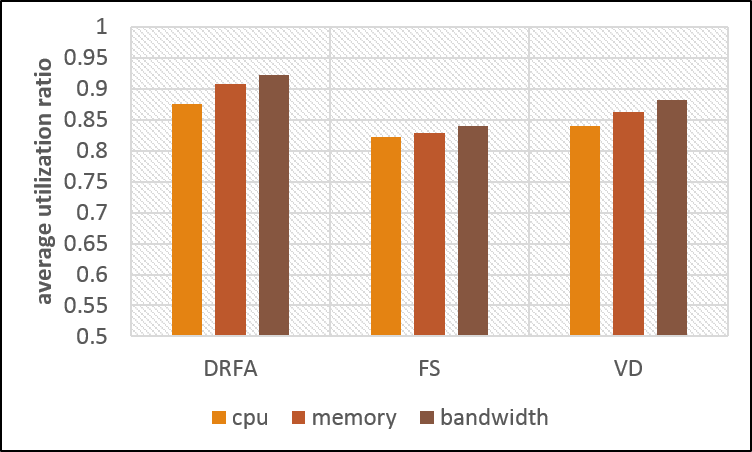


图5-15 实验一资源利用率对比图

从图5-15中可知，在一般场景下，使用FilterScheduler进行资源分配得到的资源利用率最低，而使用DRFA算法得到的三种资源利用率均高出其它两种算法，由图可知，高出FS算法大约10%，高出VectorDot算法大约5%。

（b）实验二

实验二的目的是为了验证在视频监控云的应用场景下，DRFA对于资源的分配情况，因此的虚拟机类型仅采用了表5-16 中High系列，实验环境的其它设置不变。经过统计后，实验结果如图5-16所示。在大量异构虚拟机的情况下，每种算法的资源利用率均有所下降，其中Openstack默认的FilterScheduler算法仍然是资源利用率最低的，VectorDot算法次之，而DRFA算法拥有最高的资源利用率，CPU利用率超过90%，内存和宽带的利用率均能达到85%以上，并且资源分配相对均衡。综合可知，DRFA在提高资源利用率方面效果显著，相对于其它资源分配算法，大约提高5%。

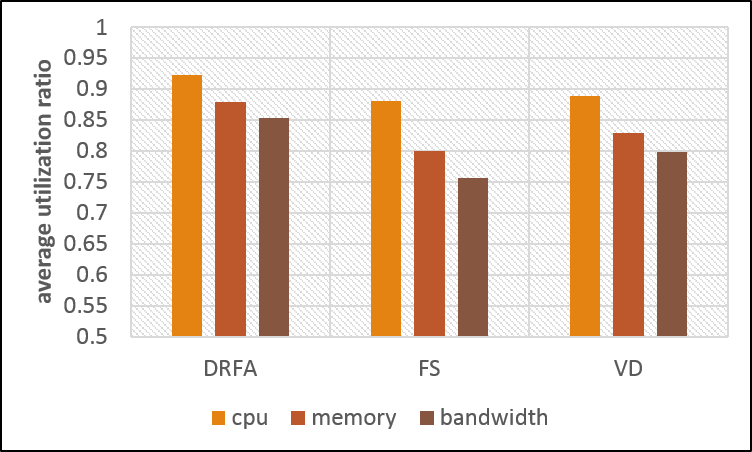


图5-16 实验二资源利用率对比图

对于实验二，我们还统计了集群中最终分配的虚拟机的数量，这也代表了集群能提供的最大服务数。如图5-17 所示，使用DRFA算法进行资源分配，能够提供更多的服务数量，对于有效提高集群的吞吐量有很大帮助。

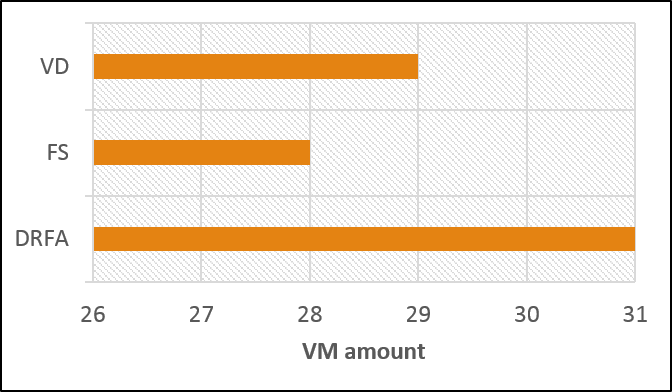


图5-17 虚拟机数量统计

（5）实验分析

DRFA不同于其它算法的有两个地方，一是采用多维资源匹配的方式，二是充分利用物理机与虚拟机的资源异构性进行分配。在两次实验中，基于内存的单资源分配方法FilterScheduler得出的资源利用率是最低的，可以说明单资源分配方法并不适用于多维资源环境。而对于VectorDot，虽然针对多维资源匹配进行了优化，但是并没有考虑资源环境的异构性。综上，在资源异构的环境下，基于主导资源的DRFA算法能有效的提高资源利用率。

5.4 本章小结

本章主要对视频监控云资源管理系统进行了编码实现。在Openstack Java SDK服务中，对Connector模块、服务组件模块以及Session管理模块的实现类图以及调用关系分别进行了描述。在WEB管理服务中使用SpringMVC框架实现了REST API的封装以及完整的WEB管理系统。在虚拟机监控服务中通过轮询的方式对虚拟机的运行状态进行查询，并且汇总到数据库中，并且以动态图表的形式展现出来。最后通过功能测试的方式验证了系统的可用性，并且通过仿真的方式对资源调度算法进行实验验证，实验结果表明算法能有效提高集群资源的利用率，优于其它算法。

# 第六章 总结与展望

6.1 总结

越来越多的行业正在利用云计算技术解决传统视频数据处理中的问题，其中视频监控云资源管理系统就是一个经典的范例。视频监控云利用云计算的弹性资源管理能力，可以实现对运行着智能视频处理任务的虚拟机进行动态的创建和销毁。然而虚拟机的资源配置通常跟视频处理任务本身对资源的需求息息相关，各式各样的任务需求会产生多种异构的虚拟机类型，如何把大量的异构虚拟机请求与异构的物理机资源有效的结合起来并实现资源的合理分配，是我们研究的重点。我们的主要工作包括几个方面：

（1）针对视频监控云的应用需求，我们对视频监控云环境下的资源管理系统进行了详细设计。首先本系统是基于Openstack进行多区域部署统一管理的设计。其次在深入研究了Openstack的原生API的基础之上，设计了基于Java语言的Openstack SDK，然后基于我们设计的SDK完成了对整个云资源管理系统的设计，本系统可以提供两种访问方式，分别是供普通用户使用的WEB管理服务以及供上层应用开发者使用的Restful API服务。最后为了实现对虚拟机的动态实时监控，设计了虚拟机的监控服务，可视化的观察虚拟机的负载情况。

（2）针对视频监控云的调度问题，我们提出了一种适用于异构环境的多维资源调度方法。我们在第四章中首先探讨了Openstack默认的资源调策略中存在的问题，其次深入研究总结出视频监控云中云资源的特性：多维性、动态性以及异构性。然后为了使异构环境下的资源分配实现资源利用率的最大化，我们提出了一种主导资源优先分配（DRFA）算法，该算法基于主导资源的概念，能充分把握资源的异构特征，并且从集群全局的角度采用多维资源匹配标准进行物理机的选择，优先把具有相同主导资源的虚拟机与物理机进行匹配。

（3）最后，我们实现了基于Openstack的视频监控云资源管理系统，并且对DRFA算法进行了测试。本系统利用Openstack组件结构特征实现了多区域管理功能，并通过使用SpringMVC框架作为控制层实现了Restful API服务以及WEB管理服务。而且我们通过定时向Libvirt读取虚拟机状态信息的方式，实现了虚拟机负载信息的实时可视化显示。最后通过功能测试，验证了系统功能运行良好，并且验证了DRFA算法能有效的提高服务器的资源利用率，优于其它传统的调度方法。

6.2 展望

云计算时代的技术革新是非常迅速的，而且真正地利用云计算技术解决实践中的问题才是主要的目标。我们的视频监控云资源管理系统已经初步完成，但是仍然有一些问题需要解决：

（1）设计应用平台。既然云资源管理系统已经初具规模，下一步应该充分利用云资源管理系统的弹性资源管理能力，实现智能视频处理服务的自动部署，这属于PAAS层的工作。

（2）优化调度算法。我们提出的算法能够较好的利用资源异构特性并提高资源利用率，下一步需要在调度过程中进行动态优化，利用云计算的动态迁移技术，在分配过程中实现虚拟机的调整，实现服务器集群更高的资源利用率。

（3）探索新技术。在虚拟机化技术中，除了hypervisor技术，最近兴起了以Docker为代表轻量级的容器技术。作为一种新兴的技术，容器技术现在还没有实现企业化应用，如何充分利用容器技术来处理视频监控云中的问题是我们下一步的工作。

# 参考文献

[1] Hossain M S, Hassan M M, Qurishi M A, et al. Resource allocation for service composition in cloud-based video surveillance platform[A]. // 2012 IEEE International Conference on Multimedia and Expo Workshops (ICMEW) [C], IEEE, 2012: 408-412.

[2] Zhao X, Ma H, Zhang H, et al. HVPI: Extending Hadoop to Support Video Analytic Applications [A]. // 2015 IEEE 8th International Conference on Cloud Computing (CLOUD) [C]. IEEE, 2015: 789-796.

[3] Zhang W, Xu L, Duan P, et al. Towards a High Speed Video Cloud Based on Batch Processing Integrated with Fast Processing [A]. // 2014 International Conference on Identification, Information and Knowledge in the Internet of Things (IIKI)[C]. IEEE, 2014: 28-33.

[4] Van H N, Tran F D, Menaud J M. SLA-aware virtual resource management for cloud infrastructures [A]. // 9th IEEE International Conference on Computer and Information Technology(CIT'09) [C]. IEEE, 2009, 1: 357-362.

[5] Amazon EC2. https://aws.amazon.com/ec2/, 2015.

[6] 阿里云. www.aliyun.com/.

[7] Openstack. https://www.openstack.org/, 2015.

[8] CloudStack. https://cloudstack.apache.org/, 2015.

[9] Eucalyptus. https://www.eucalyptus.com/,2015

[10] Opennebula. https://opennebula.org/,2015

[11] Vidwo encoding service. www.encoding.com.

[12] Yang C T, Huang K L, Chu W C, et al. Implementation of video and medical image services in cloud [A]. // 2013 IEEE 37th Annual Computer Software and Applications Conference Workshops (COMPSACW) [C]. IEEE, 2013: 451-456.

[13]Hossain M S, Hassan M M, Qurishi M A, et al. Resource allocation for service composition in cloud-based video surveillance platform [A]. // IEEE International Conference on Multimedia and Expo Workshops (ICMEW) [C]. IEEE, 2012: 408-412.

[14]Sahasrabudhe S S, Sonawani S S. Comparing openstack and VMware [A]. // 2014 International Conference on Advances in Electronics, Computers and Communications (ICAECC) [C], IEEE, 2014: 1-4.

[15] Farahnakian F, Liljeberg P, Pahikkala T, et al. Hierarchical VM Management Architecture for Cloud Data Centers [A]. // IEEE 6th International Conference on Cloud Computing Technology and Science (CloudCom) [C]. IEEE, 2014: 306-311.

[16] Singh A, Korupolu M, Mohapatra D. Server-storage virtualization: integration and load balancing in data centers [A]// Proceedings of the 2008 ACM/IEEE conference on Supercomputing[C]. IEEE, 2008: 53.

[17] Nguyen T H, Di Francesco M, Yla-Jaaski A. A Multi-resource Selection Scheme for Virtual Machine Consolidation in Cloud Data Centers [A]. // IEEE 6th International Conference on Cloud Computing Technology and Science (CloudCom) [C]. IEEE, 2014: 234-239.

[18] Kang Z, Wang H. A novel approach to allocate cloud resource with different performance traits [A]// 2013 IEEE International Conference on Services Computing (SCC) [C]. IEEE, 2013: 128-135.

[19] Wang J, Huang C, He K, et al. An Energy-Aware Resource Allocation Heuristics for VM Scheduling in Cloud [A]. // 2013 IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC\_EUC) [C]. IEEE, 2013: 587-594.

[20] Ghodsi A, Zaharia M, Hindman B, et al. Dominant Resource Fairness: Fair Allocation of Multiple Resource Types [A]. // Proceedings of the 8th USENIX conference on Networked systems design and implementation[C]. 2011, 11: 24-24.

[21] Wang W, Liang B, Li B. Multi-Resource Fair Allocation in Heterogeneous Cloud Computing Systems [J]. IEEE Transactions on Parallel and Distributed Systems. IEEE 2015, 26 (10), 2822-2835.

[22] Gong C, Liu J, Zhang Q, et al. The characteristics of cloud computing [A]. // 2010 39th International Conference on Parallel Processing Workshops (ICPPW) [C]. IEEE, 2010: 275-279.

[23] Fox A, Griffith R, Joseph A, et al. Above the clouds: A Berkeley view of cloud computing [J] // Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS, 2009, 28: 13.

[24] 林昊翔, 秦君. 虚拟化技术漫谈[EB/OL]. https://www.ibm.com/ developerworks /cn/ linux/l-cn-vt/, 2009.

[25]Wen X, Gu G, Li Q, et al. Comparison of open-source cloud management platforms: OpenStack and OpenNebula [A]. // 2012 9th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD) [C]. IEEE, 2012: 2457-2461.

[26] 吴铭. 基于OpenStack的IaaS云中动态资源分配策略研究[D]. 安徽: 中国科学技术大学, 2014.

[27] 余文治，刘凯歌，吉子敛. Openstack Keystone 简介.// http://www.ibm.com/ developerworks/cn/cloud/library/ 1506\_yuwz\_keystonev3/index.html. 2015

[28] Pfaff B, Pettit J, Amidon K, et al. Extending Networking into the Virtualization Layer[A]. // Hotnets. 2009.

[29] Rosado T, Bernardino J. An overview of openstack architecture[A]. // Proceedings of the 18th International Database Engineering & Applications Symposium[C]. ACM, 2014: 366-367.

[30] 王东晨. 网络试验平台数据存储研究与实现[D]. 北京: 北京邮电大学, 2012.

[31] Endo P T, de Almeida Palhares A V, Pereira N N, et al. Resource allocation for distributed cloud: concepts and research challenges[J]. IEEE Network, IEEE, 2011, 25(4): 42-46.

[32] Chen W, Cao J, Wan Y. QoS-aware virtual machine scheduling for video streaming services in multi-cloud [J]. Tsinghua Science and Technology, 2013, 18(3): 308-317.

[33] Fielding R T, Taylor R N. Principled design of the modern WEB architecture [J]. ACM Transactions on Internet Technology (TOIT), 2002, 2(2): 115-150.

[34] SpringMVC[EB/OL].http://docs.spring.io/spring/docs/current/spring-framework- reference/html /mvc.html.

[35] Wu S, Deng L, Jin H, et al. Virtual machine management based on agent service[A] // 2010 International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT) [C]. IEEE, 2010: 199-204.

[36] Hat R. Libvirt: The virtualization API[J]. // http://Libvirt. org, 2014.

[37] Highcharts J S. Interactive JaveScript charts for your WEB projects[J]. www.highcharts. com, 2013.

[38] Openstack如何统计资源[EB/OL]. http://blog.csdn.net/wsfdl/article/details/ 45418727.

[39] Lianhao Lu, Yingxin Chen. Utilization-based Scheduling in Openstack Compute[R]. Intel, 2015.

[40] 马志超. OpenStack调度算法研究与优化[D]. 浙江大学, 2014.

# 攻读硕士学位期间的主要研究成果

[1] Xianda Yang, Haitao Zhang, Huadong Ma, Wensheng Li, Guangping Fu, Yi Tang. “Multi-resource Allocation for Virtual Machine Placement in Video Surveillance Cloud,” 2016 International Conference on Human Centered Computing (HCC2016), Springer, 2016: 92-106. (EI检索)

[2] Yihong Gao, Huadong Ma, Haitao Zhang, Xianda Yang, Ning Cao. “Minimizing Resource Cost for Camera Stream Scheduling in Video Data Center,” 2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom), IEEE, 2015: 210-217. (EI检索)

[3] 张海涛, 付广平, 杨贤达, 张闯, 马华东等. 睿视云资源管理系统V1.0。软著登字第0968330号, 登记号: 2015SR081244.

# 致谢

逝者如斯，转眼已经在北邮度过了两年多的时间，有欢声笑语，有汗水泪水，收获最多的就是成长，能在马华东老师的团队里工作学习是非常荣幸的。在毕业论文即将完成的时候，回忆一下这两年多的经历，感慨万千。岁月不复，情谊长留。

首先我要深深感谢我的导师李文生老师和张海涛老师。感谢你们对我学业孜孜不倦的教诲和关怀，才使我在学业上有所收获，你们的治学态度和处事风范都在潜移默化的影响着我。尤其要感谢二位导师在我论文写作方面的指导，使我能够轻叩学术世界的大门，并且顺利在springer上发表论文一篇。其次要感谢陈建伟师兄、高一鸿师兄，是你们不断的帮助，让我能够在学术的道路上不至于遍体鳞伤。最后，我要感谢实验室项目组的同学付广平、张闯、唐毅和寇月，能够与你们一起共事我感到荣幸又开心，是你们，让我感受到了团队的力量。

同样的感谢需要给予我的父母和爱人，感谢你们对我生活和学业上的大力支持，谢谢你们。