教育行业垂直云平台架构与设计研究

（申请清华大学工学硕士学位论文）

培 养 单 位：自动化系
学　　科：控制科学与工程
研 究 生：陈 伟
指 导 教 师：曹 军 威 副 研 究 员

二〇一三年五月

Architecture and Design Research of Education Industry Cloud Platform

Thesis Submitted to

Tsinghua University

in partial fulfillment of the requirement

for the degree of

Master of Science

in

Control Science and Engineering

by

Chen Wei

Thesis Supervisor : Associate Professor Cao Junwei

May, 2013

关于学位论文使用授权的说明

本人完全了解清华大学有关保留、使用学位论文的规定，即：

清华大学拥有在著作权法规定范围内学位论文的使用权，其中包括：（1）已获学位的研究生必须按学校规定提交学位论文，学校可以采用影印、缩印或其他复制手段保存研究生上交的学位论文；（2）为教学和科研目的，学校可以将公开的学位论文作为资料在图书馆、资料室等场所供校内师生阅读，或在校园网上供校内师生浏览部分内容。

本人保证遵守上述规定。

（保密的论文在解密后应遵守此规定）

作者签名：_____　　导师签名：_____

日　　期：_____　　日　　期：_____

摘 要

云计算经过多年的积淀，进入了快速发展的时期。云计算在近几年的发展中，行业垂直云平台逐渐显示出其重要性。在医疗，金融等领域都有着重要的应用。教育领域是一个前景非常广阔的领域，云计算在其中可以发挥非常重要的作用。教育行业垂直云平台有自己独特的特性，网络课程、网上教学、网上问答等都是教育行业云平台应用的可行方向，其中在线视频教学是一个很明显的需求，因此作者设计和实现了一个教育云平台的示例产品，其中实现了用移动设备，在云平台上播放教学视频。

作者还对多云环境下的视频服务虚拟机的部署方案进行了研究。越来越多的云计算服务商出现，使得选择多个数据中心同时提供服务是一个很好的选择。本文提出了一种在多个数据中心配置虚拟机个数的方法，通过分析现有用户和新增用户的位置，以及每个数据中心内虚拟机资源占用情况，动态地调整每个数据中心的虚拟机个数，以达到提高总体服务质量和降低每用户服务成本的目标。最后的仿真实验结果表明，这种优化算法很好的控制了服务质量，同时降低了每用户的服务成本。在不同数据中心的价格不同，以及存储成本和虚拟机成本具有不同比例的情况下，该算法都行之有效，且实验数据的分析结论提供了在配置服务时的一些参考依据。作者的主要工作是：

1. 开发云计算中间件 elop 软件包，包括元素层，逻辑层和流程层的工作。elop 中间件可以提供资源管理，流程访问，权限控制等一系列云计算基础功能。
2. 在 elop 的基础上，完成流媒体播放云平台，能够通过流媒体服务软件提供 rtsp 协议的流媒体服务，供客户端使用。
3. 设计用于提供课程信息的 Java 服务器，作为教育云平台的数据服务器。该服务采用无状态设计，能够很容易的部署多台共同提供服务。
4. 开发 Android 系统下的应用，具有连接到教育云平台，观看云平台中的课程课件，视频等功能。
5. 研究在多个云服务提供商并存，各个数据中心分布在不同的地点，有不同的价格的情况下，如何动态的配置不同的数据中心的虚拟机个数以及所服务的用户，才能够保证服务质量，并且降低服务成本。

关键词：云计算；视频流媒体；多云；移动互联网；服务质量

Abstract

Cloud computing has been developed for several years and is increasingly significant for the Internet now. Industry cloud is showing the importance now. Education industry has a very bright future and has huge potential when it meets cloud computing.A cloud platform for education has special features and strong demands. Network courses, online teaching, questions and answers are all feasible direction of education industry cloud. A demo application of education cloud platform is designed and implemented , in which playing education videos on mobile devices is an essential function.

The algorithm and implementation of video streaming service on multi-cloud are studied. More and more cloud providers appear so there are increasing cloud platforms to choose. A better choice is to use more than one data center, which is called multi-cloud. In this paper an approach for optimizing QoS and cost is proposed. Modules of monitoring and controlling data centers are required as well as the application feedback such as video streaming services. Experimental results show that the algorithm works to achieve a good QoS and low cost level. When the prices of data centers are different with each other, and the ratio between storage cost and virtual machine cost changes, the algorithm works well, and the results are good references of configuring services. The main work of this dissertation is as follows:

1. Development of a cloud computing software middleware.
2. Design and implementation of a video streaming cloud platform and using Darwin Streaming Server to provide a video streaming service.
3. Design of a Java server application to support the education cloud platform.
4. Development of an application on Android, in which users can play education video and scaning coursewares.
5. Research of the deployment of video streaming service on multi-cloud environment. An algorithm is proposed to help choose cloud providers and data centers in a multi-cloud environment as a video service manager. Performance evaluation of the algorithm is included with different video service workload.

Key words: cloud computing; video streaming; multi-cloud; mobile internet; QoS

目 录

第 1 章 引言 1
1.1 云计算 1
1.2 行业垂直云 2
1.3 论文工作与安排 3
第 2 章 相关工作和动态 5
2.1 教育云的发展 5
2.2 视频流媒体技术 6
2.3 云计算背景下的视频服务 7
2.4 多云环境和成本控制 8
2.5 云计算的服务质量保证与成本控制 9
第 3 章 云计算中间件 elop 软件包 11
3.1 总体架构 11
3.2 元素层 12
3.3 逻辑层 14
3.4 组织层 18
3.5 流程层 18
3.6 管理界面 19
3.7 远程访问虚拟机资源 22
3.8 自动安装软件 24
3.8.1 软件安装类型 26
3.8.2 自动安装软件 26
第 4 章 系统实现 29
4.1 整体架构 29
4.2 流媒体服务器安装 30
4.2.1 Darwin Streaming Server(DSS) 的安装 30
4.2.2 DSS 测试 32
4.3 客户端实现 34
4.3.1 设计思路 34
4.3.2 界面设计 35
4.3.3 网络通信 36
4.4 JAVA 服务端 37
4.4.1 平台 37
4.4.2 服务架构对比 38

4.4.3 数据库设计 40
4.4.4 与 elop 接口 41
4.5 elop 云平台端 41
4.5.1 元素层设计 41
4.5.2 逻辑层设计 43
4.5.3 流程层设计 43
第 5 章 多云环境下的虚拟机配置算法 45
5.1 背景条件 45
5.2 系统模型 46
5.3 算法设计 47
5.4 效果分析 48
5.4.1 数据设定 48
5.4.2 在多云中动态配置和使用单一云平台的对比 50
5.4.3 引入价格系数之后的实验结果 53
5.4.4 变化存储费用的实验结果 53
5.5 综合分析与结论 60
第 6 章 总结与展望 62
6.1 论文工作总结 62
6.2 工作展望 63
参考文献 64
致 谢 67
声 明 68
个人简历、在学期间发表的学术论文与研究成果 69

第 1 章 引言

1.1 云计算

云计算近年来取得了突飞猛进的发展，这不仅得益于技术的不断更新换代，更重要的是云计算的低成本，高可扩展性的优势。众多的中小企业借助云计算的服务，能够更快更好的提供服务。如图1.1，云计算主要有三种模式。

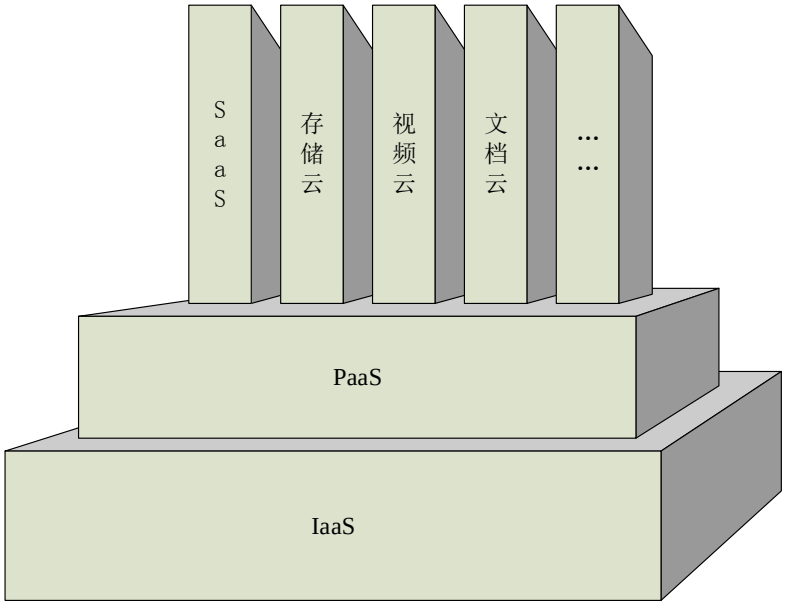

图 1.1 三种云计算模式示意图

IaaS[1]（Infrastructure as a Service）是设施即服务，可配置性最强。是指将云端计算资源以最基本的设施作为服务开放出来给用户用。代表产品有 Amazon 的 EC2。

PaaS[2]（Platform as a service）是平台即服务，是指将提供一个可编程的平台给用户作为云服务。代表产品有 Google 的 GAE。

SaaS[3,4]（Software as a Service）是软件即服务，是指直接提供给用户可以使用的软件作为云服务，对用户最友好，但可配置性也很低。代表产品有 Google Docs，Dropbox 等。还有各种特色的专业云平台，比如语音识别云平台[5] 等。

从服务人群划分，云计算还可以分为公有云，私有云和混合云三种模式。公有云主要指对所有人开放，通过互联网访问，采用免费或用户付费的方式提供服务的云计算模式。私有云则与之对应，不对外界开放，为某个或某几个机构服务，

仅对机构内部人员开放的云计算模式。混合云则是同时利用公有云和私有云的服务，协调工作的一种方式。在本课题中，主要以公有云为背景，研究其在教育领域的应用。

1.2 行业垂直云

行业垂直云是指针对行业特征和需求，定制行业所需要的云产品，这样的垂直云不面向所有用户，因此在产品的适用范围上不如普通公有云产品，但是针对行业内人士，垂直云平台能够更好的帮助用户完成所需要的任务。本课题立足于教育行业垂直云平台的架构与设计研究，结合最新的移动平台，利用 elop[6] 软件包进行了教育垂直云平台的架构与设计研究。该系统包含了客户端，服务端，后台视频云平台等一整套服务。

本课题的教育行业垂直云平台基于云计算中间件 elop。云计算目前包含的技术跨度非常广，为适应各种需求而产生的很多技术都难以在一个产品中体现。elop 软件包以需求为导向，首先确定好整个系统基础的四层框架，然后实现了云计算中最为核心的虚拟化，用户管理，存储等需求，是一个轻量级的云计算平台。在这个架构之上，可以通过扩展开发的方式实现所需要的功能，比如本课题针对教育行业的需求所添加的视频流媒体播放等功能。通过这种方式构建的教育行业垂直云平台，既能具有普通云平台中 IaaS 的可配置型和可定制性，又能够提供 SaaS 一样的便利工具。

在教育领域，云计算具有极大的潜力。借助云平台，学校，政府，教育机构，教师，学生，读者都可以很方便的创造和消费内容。而且利用云计算的架构，可以节省大量的重复的基础设施建设。云平台搭建完成之后，学生，学校，教育机构，独立教师，教育机构等，都可以通过连接到教育云平台来参与教育活动。对于学生，在云平台之上可以查看课程列表，观看课程视频，查看课程信息和课件，对课程作出评价和意见等。对于教育机构和学校，教师等，可以制作课程并上传云平台，可以回答学生的提问等。

教育云平台包括不仅仅包括基础设施服务，还有平台和软件服务系统，如图1.2。云平台通过虚拟化技术，将计算，网络系统虚拟为虚拟机，网络服务，存储服务提供给用户；平台服务则以数据库为主要实现形式，提供认证，授权，数据管理等功能；软件服务则是云平台的对外接口，用户通过门户或客户端软件登陆，使用云平台的服务，软件服务的水平直接影响用户体验。

要做到弹性的分配计算资源和数据资源，虚拟化[7] 的支持必不可少。通过虚拟化，我们可以将本来固定数目的硬件，模拟为数目不定的虚拟硬件，从而应付

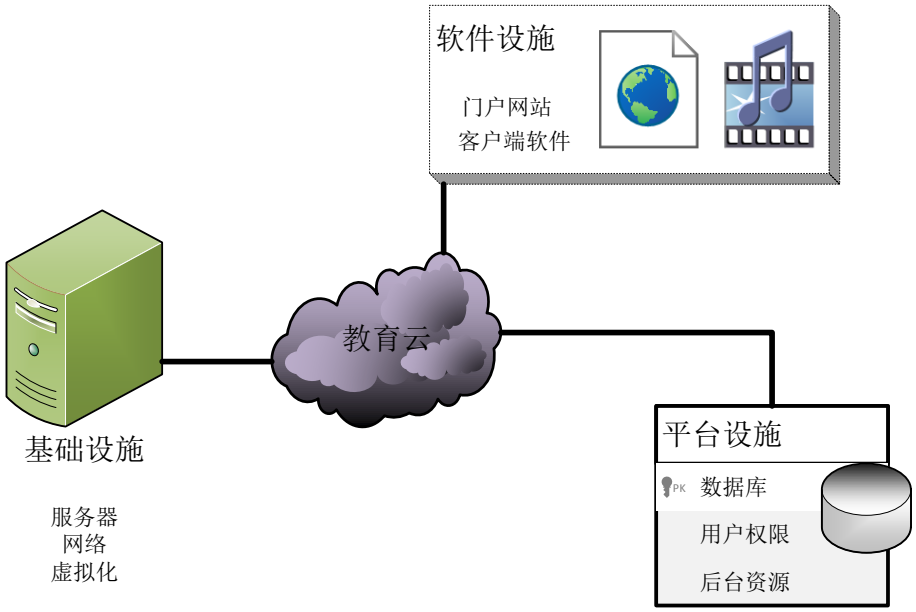

图 1.2　教育云平台示意图

各种情况的需求。在用户数量增加的时候，虚拟化可以在脱离人员管理的情况下自动的调整资源数目，来满足用户的需求。

在基础设施层面，虚拟化主要的方式就是建立虚拟机。目前虚拟机软件多种多样，常见的有 VmWare[8,9]，VirtualBox[10]，Xen[11,12]，KVM[13] 等。虚拟化是云计算的核心支持技术，随着这些年云计算的突飞猛进，虚拟化技术也随着快速发展。VmWare 作为商业软件，在技术上更加成熟，在商业应用中也具有相当大的规模，支持更加完善。而另外三款虚拟化软件，都是有商业公司支持的开源软件，有着开源软件的灵活的特点，也有不如商业软件的稳定性的问题。VirtualBox 相对起步较晚，在个人用户方面具有较好的友好度和易用性，但是效率较低。而 Xen 和 KVM 则各有优势，都比较注重效率[14–16]，适合作为云服务的基础。在本系统中，使用了 elop 作为云计算的中间件，因此虚拟化支持也由 elop 中的相应组件完成。Elop 中的虚拟化组件使用 libvirt 作为虚拟机接口，支持上述所有的虚拟机软件，在测试平台中，我们使用了 VirtualBox 和 Xen 作为实验平台。

1.3　论文工作与安排

本论文首先展示了教育垂直云平台的系统构建，该系统基于 elop 云计算中间件，使用虚拟化平台进行视频流媒体服务，在移动平台上播放教学视频等。同时，

作者研究了在多云环境下虚拟机数目的动态配置，以提高服务质量并降低成本。

本论文共分八章，各章内容如下：

第一章介绍主要问题背景和课题意义。

第二章阐述国内外对本文所研究问题的研究进展和最新动态，包括教育云的发展，流媒体技术，基于云平台的视频播放，多云环境，云计算的服务质量保证。

第三章展示 elop 云计算中间件，包括元素层，逻辑层，组织成，流程层，以及基于 elop 之上的可视化系统，桌面云，自动安装软件等。

第四章系统实现。包括客户端，服务端，云平台三大部分。

第五章提出在多云环境下的虚拟机数目配置算法，并进行仿真验证。对实验结果进行总结归纳，提出了在多云平台上构建不同类型的服务时的针对性策略。

第六章对所做的工作进行总结，并提出可以改进的方面，对今后的研究方向做出展望。

第 2 章　相关工作和动态

2.1　教育云的发展

云计算[17,18] 的发展越来越引人注目，其低成本，高效率，可扩展性强的特点对目前的大数据量的环境来说越来越重要。像 Amazon，Google 这样的公司都在努力推动云计算的发展，对于中小企业来讲，云计算已经成为降低 IT 成本的首选[19,20]。在教育行业，在线教育在某些方面也已经凸显出其不可替代的优势，而云计算天生的低成本和可扩展性的特点，对在线教育行业有着强烈的吸引力。大量的教育机构无力支付建立大规模在线教育平台的成本，但他们却有着第一线的教学资源，通过我们的云平台，可以以最低的成本将他们的教育产品提供给各地的消费者。

在文献 [21] 中，作者从云计算的定义出发，找出了云计算应用与教育领域的主要问题。学院和大学一直在寻求升级他们的软件和硬件，以吸引学生和跟上数字技术的迅速发展。云计算提供了这样的可能。在软硬件不断升级的环境下，可以通过按需租用的方式，不断的使用最新的设备和软件，提供足够的运算能力，并且将价格保持在可以接受的范围内。此外，将维护服务器，机房设施的任务交给云服务的提供商，将可以节约大量的成本和重复劳动，将需要比以前更少的 IT 服务人员，且能提供更优质的教学环境。

图2.1表示的是传统的教育机构的 IT 服务模式，这种模式正在逐步被图2.2的方式所替代。

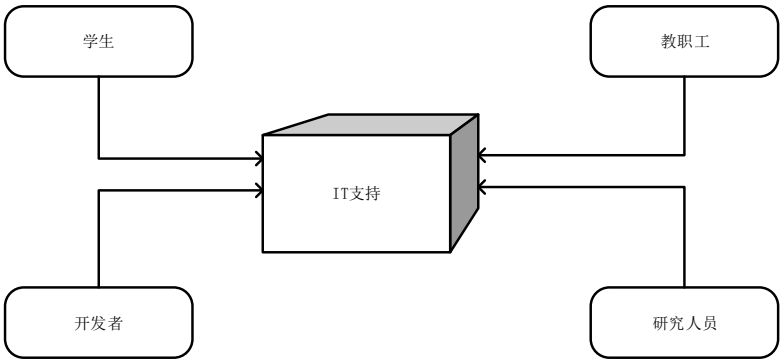

图 2.1　传统的教育机构 IT 组织[21]

同时，作者也研究了这个过程中可能出现的问题：法律和规定方面的限制，知识产权保护，安全和隐私问题[22]，服务可靠性。

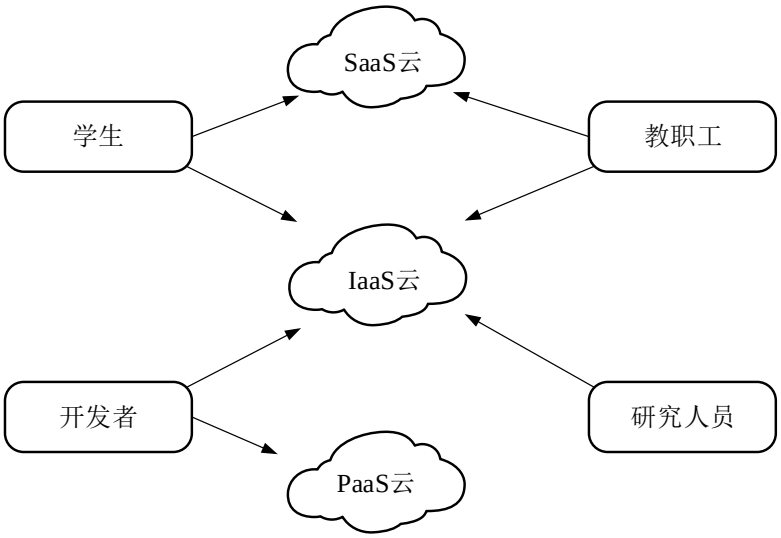

图 2.2　云计算环境下教育机构 IT 组织[21]

教育的最基础模式就教师——学生的讲授，因此视频教学是目前在线教育中效果比较好的方式，而大规模的视频教学也是对基础设施和云计算构架最具有挑战的一部分。因为视频是数据量相对较大的一种数据类型，大规模的进行视频传输，对服务器的负载能力和整个网络的架构都提出了一定的挑战。

2.2　视频流媒体技术

流媒体技术经过多年的发展，目前技术上已经趋于成熟，有很多开源的流媒体服务软件可以提供简单的流媒体服务，比如苹果的 Darwin Streaming Server[23] 等，然而对于大规模的流媒体需求，仍需要针对实际运行情况进行进一步的开发和优化。

CDN(content delivery network) 通常是视频流媒体服务上采用的一种降低服务器压力，提高服务质量的方式。但是在目前规模巨大的需求下，单纯的 CDN 也无法很好的满足用户的需求，因此在 CDN 的基础上衍生出很多提高性能的方式。文献 [24] 是香港科技大学的一篇论文。在该文章中，作者使用 CDN 和 P2P 混合的方式进行实时流媒体传输。CDN 和 P2P 都是流媒体传输中常用的技术。内容最初被放置在媒体源中，CDN 可以将内容缓存在大量的节点中，从而让更多的用户可以通过缓存的数据来访问内容。而 P2P 可以在用户之间传输数据，从而减轻源服务器的压力。这项工作取得了不错的效果，但在目前的网络环境和与日俱增的用户需求下，这些技术仍然不足以使源服务器摆脱过载的可能。同时，这两项

技术均存在一个问题，就是针对热门内容有效，对于非热门视频，更多的还是需要从源服务器抓取。与一般的视频网站以及直播需求不同，网络课程，在线教育等，对内容的需求更加均衡，更多的非热门内容会被访问到。因此，提高源服务器的承载能力是本文关注的重点。本文将不会讲述到新的 CDN 和 P2P 的使用和配置，而是单纯的从原服务器的能力来考虑。但这并不妨碍在本文的系统给之上附加 CDN 和 P2P 的服务来进一步提高服务质量。

对于云计算的环境是否能够满足视频流传输这样的高 IO，低计算量的需求，已经有实际的成功案例供我们参考。Netflix 是一个基于 Amazon 云服务之上的视频流媒体服务商，该服务全部使用 Amazon 的云计算资源提供服务，所需要的资源大小可以按需配置，能够很容易的满足快速增长的用户数，以及在用户数目不多时降低成本。

文献 [25,26] 中，作者研究了网络上进行视频和声音流媒体传输的服务质量问题。服务质量收传输时延和丢包率影响，时延越大，丢包率越高，服务质量越差。作者从视频流本身的传输状况的角度测试了视频传输的服务质量，而不是传统上的根据网络整体表现来判断。测试结果被用来做路径选择时的决策依据。在本课题研究中，服务质量的仿真数据也是通过服务器的负载情况和服务器与客户端的距离来估计丢包率和传输时延，作为动态部署的决策的依据。

文献 [27] 中的这项工作系统的研究了在网络上传递视频时，多个服务器共同承担服务时的优化问题。根据作者的结论，不够流行的视频，应该从源端直接传递给需求端，而流行的视频，本地服务器应该接受内容并保存完整的数据备份。如果各端之间能够进行交流，可以显著的缩小整个系统的开销。本文的背景环境从多服务器提供视频服务，转换到了多云平台，并且需求端从本地服务器变换到了移动设备。针对这些变化，作者在本文中借鉴并改进了该参考工作的结论。流行的视频将会被复制到更多的云上，而本地服务器之间进行的交流，也变为了移动设备服务端可以集中处理的数据，所进行的决策和优化将更有效率。

2.3 云计算背景下的视频服务

香港大学的一项研究 [28] 针对视频点播的应用需求，使用云平台提供的动态资源调整特性，更好的支持视频流并降低成本。在他们的这项研究中，作者引入一个排队网络的基础模型来描述用户的浏览行为在一个多通道视频点播应用, 推导出服务器能力需求，进而调整云平台的资源供给，实现动态的自适应平台，并进行了大规模的实验验证。他们的工作介绍了利用云服务模式支持大规模的网络应用程序。使用视频点播应用的例子演示了按需分配云资源如何满足动态和集约

资源需求的视频点播需求。

作者的主要贡献是:

1. 提出了一个新颖的排队网络模型描述用户的浏览行为, 推导出平滑播放对于客户机——服务器的需求和 P2P VoD 实现。
2. 将云平台参数考虑在内, 制定两个与 VM 配置和存储租赁相关的优化问题, 提出了一些有效的解决方案。
3. 提出了一个实用的动态云配置算法的设计和实现，视频点播提供商可以据此有效地配置云服务, 以满足其要求。在本文中，作者将问题的背景放在了未来更有市场的多云环境下，从而在云服务器的配置上有了更多的选择。

他们的工作在研究云计算环境下的大规模视频流媒体需求提供了很好的思路和实用的指导。我们的工作也参考了他们对于用户需求的分析模型和解决思路。然而，他们的工作集中于普通的云计算环境，也就是租赁虚拟机的模式，而且局限于单个数据服务中心的模型，随着云计算的快速发展，多云模型越来越引人关注，单个数据中心已经很难满足各种情况下的需求。本文的工作将背景放在了分散的云计算环境中，多个云平台独立提供服务，资源并非完全共享，各个云之间的数据传递需要成本。尤其是在在线视频教学这样的需求中，这种模型将最大程度的降低整个网络的带宽成本，提高用户体验。

2.4　多云环境和成本控制

多云（multi-cloud[29,30]) 架构，是指利用多个数据中心提供的云服务，搭建一个混合的云平台供某一个垂直应用使用。在可以预见的一段时间内，云服务提供商的数量将会大大增加，给用户更多的选择。选择不同的云服务提供商的服务，可以在价格和服务上获取更多的利益。多个数据中心将可以分布在不同的地理位置，这样可以充分利用某些应用的地域性特征，降低整个网络开销，从而降低成本，提高服务质量。本文重点设计了一套在多云架构下选择云服务提供商提供网络视频教学服务的算法，来提高服务质量，降低成本。

在 [30] 中，作者研究了，在有多个云平台的环境下，如何动态的配置虚拟机以降低成本。文中使用了一个云平台价格的预测模型，这个模型使用历史数据来推断未来一段时间的云服务价格。然后通过动态的调度用户的虚拟机资源，来更多的使用低成本的云服务。

作者的主要贡献是:

1. 设计了一个云计算服务代理架构，使得用户能够在多云环境下无差别的进行控制，也便于算法自动进行控制。

2. 设计了一个基于动态价格机制的云资源调度器。
3. 对于一个由一系列虚拟机组成的服务集群，调度器能够提供一个优化的部署方案，以将总费用最小化。
4. 在一个实际的场景下进行了效用分析。

这是一个多云环境下成本控制的有意义的尝试，尤其是在云平台服务商的数目正在快速增加的时候。在本课题中，作者也将不同云平台的价格因素作为决策的依据之一，同时还有服务质量 QoS。

在文献 [31] 中，作者研究了多服务器的动态控制来降低云计算的成本。本文从云计算设施的运营者角度出发，将云计算的很多因素考虑在内，比如服务质量保证，工作负载，租赁费用，能源消耗，运营利润等，通过这样一个模型将问题归纳为一个优化问题并加以解决。在云计算中，成本是一个非常重要的参考因素，因为云计算最能够吸引人的特性之一就是低成本。关于如何降低运营和使用云服务的成本的问题，已经有了大量的研究。本课题在从云计算的使用者角度出发，研究了在多个云平台中动态选择，以降低服务成本的配置算法。

2.5　云计算的服务质量保证与成本控制

在 [32] 中，作者提出了一种基于闭环的云计算服务保证和优化的方法。通过云计算的监控和应用自身的反馈，传递给优化控制器之后，规划资源的分配方式，然后传递到执行机构执行。

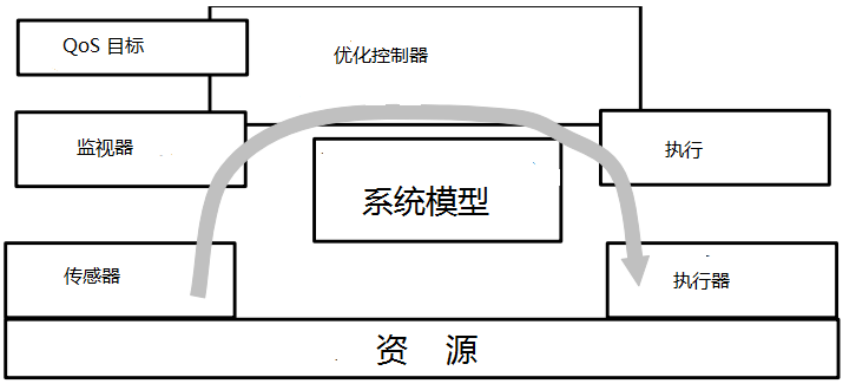

图 2.3　系统模型

这项工作总结了一个新的优化思路，并显示出通过云服务，利益最大化的控制比以往都更容易实现。文中提到了如何实现这样的优化，以及所需要的管理工具。

这种优化途径在面对大规模的云计算能力时非常有效而且易于扩展。基于模型实现的闭环控制方案也已经被实现并应用于一个运行中的实验室级别的服务中。云计算给出了一个全新的商业模型，并为自动化的优化提供了可能。云平台的规模非常巨大，这种性能的优化可以被拆分为多个小目标，从而适应各种规模的系统。

这项工作非常具有启发性，在本课题研究中，在视频服务这一特定的背景下进一步扩展了这种服务保证的优化方式。

第 3 章　云计算中间件 elop 软件包

3.1　总体架构

elop 是一套轻量级的云计算中间件。本软件包提供了资源管理，用户管理，安全认证，流程管理等一系列云计算中需要用到的功能，并且具有极强的可扩展性，便于进行二次开发。本文中搭建的教育云平台就是构建于 elop 软件包的基础之上。

我们将云计算环境分为四层，元素层，逻辑层，组织层，流程层。

P (process) 层：实现直接和业务逻辑相关的操作与处理，即和终端用户的交互接口。

O (organization) 层：组织层实现系统的合理布局，存储与安全访问机制，提供认证、授权、信任机制等安全策略，完成体系的动态构建、非授权访问控制等关键问题。

L (logic) 层：逻辑层实现平台的应用逻辑，将底层大规模异构的元素 (element) 屏蔽起来，并提供统一的访问操作机制，使得上层应用能够不用关心具体元素的类型、位置等详细信息，只用按照逻辑层规定的接口采用与之适应的方式进行访问和处理。

E (element) 层：元素层是中间件架构的最底层，其基本功能是定义整个应用架构最基本的元素 (element) 作为处理和调度的基本单位，完成元素间无缝的互操作、共享、访问控制与透明的管理功能。

elop 的实现依赖于很多开源组件：基于 CA 和数字签名的身份认证，基于虚拟组织（Virtual Organization，简称 VO）的权限管理和访问控制，基于 Socket 与 X.509 证书的加密通信，基于 VirtualBox 的虚拟机配置和部署，基于嵌入式系统和 RDP 协议的虚拟桌面访问，基于 ODBC 的数据库管理和访问。

elop 将各种资源都抽象为元素，比如虚拟机，存储资源，软件资源等。然后通过提取各种资源的共性，将公用部分放在基类的实现中，有效的减少了二次开发的工作量。同时，元素可以通过虚拟组织控制权限，可以针对不同的用户提供不同的服务。

3.2　元素层

元素层是 elop 对各种资源的抽象。资源包括文件，存储服务，虚拟机，软件等。元素层的共同特征是，能够通过网络连接对外提供服务，具有唯一的身份认证标识，运行之后会启动一个监听程序来接收请求。

表 3.1　元素静态信息列表

元素静态信息属性	属性说明
MemOwner	元素属主
MemStatus	元素状态
MemAuth	元素认证方式
MemAddr	元素地址
UNIQMemCert	元素唯一的证书标识
MemPort	元素服务端口
MemType	元素类型
MemName	元素名称

元素层都包括一个静态信息列表，来记录证书信息以验证身份，记录元素类型等等，具体信息如表3.1。

目前已经实现了的元素类型有：存储（Storage），数据（Data），服务器（Host），软件（Software）。其中，服务器下的虚拟机服务器是 elop 里比较重要的一个基础元素。Host 元素表示如图3.1。

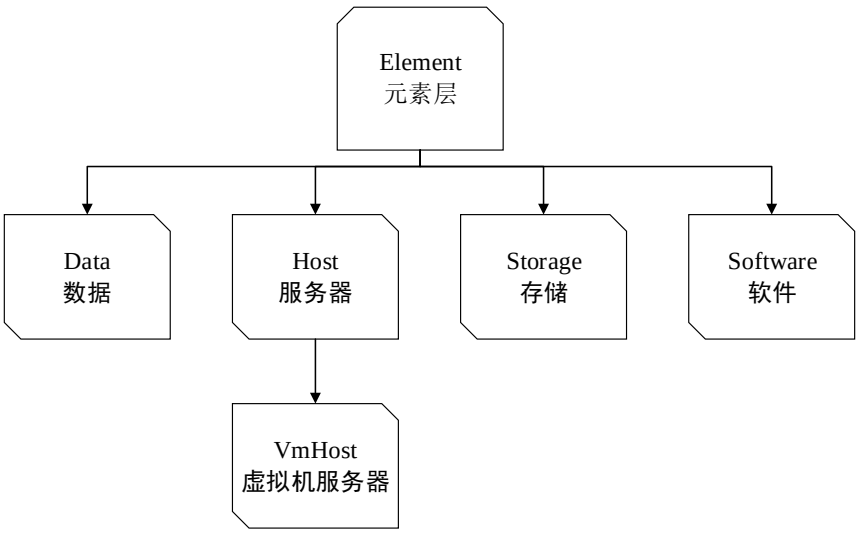

图 3.1　虚拟机服务器元素

表 3.2　Host 元素动态信息列表

元素动态信息属性	属性说明
Used_CPU	CPU 占用率
Nice_CPU	可暂缓任务 CPU
System_CPU	系统占用 CPU
Idle_CPU	空闲 CPU
Total_Memory	总内存
Free_Memory	空闲内存
Buffers	用于缓冲区的内存大小
Cached	用于缓存的内存大小
total_swap	总交换分区大小
free_swap	空闲的交换分区大小

表 3.3　Host 元素静态信息列表

Host 元素静态信息属性	属性说明
HostName	Host 名称
VirType	虚拟化类型
VirVersion	虚拟机软件版本
LibvertVersion	Libvert 管理工具版本
URI	唯一标示
MaxSupportCPUs	单个虚拟机最大支持 CPU 个数
HostFreeMem	用于缓冲区的内存大小
Model	模型
HostMem	宿主内存大小
HostCPUs	宿主 CPU 数量
CPU_Mhz	宿主 CPU 频率
Numa_nodes	共享存储节点数量

Host 元素时继承自元素层基类，因此包含了基类中静态信息的所有内容。除此之外，针对 Host 自身的特性，还有一些新的静态信息成员，如表3.2。

静态信息是指元素中不会轻易改变的信息，这些信息在元素层启动的时候，就会发送给逻辑层，在整个运行期间不会再做改动，如果有变化，则需要重启元素层服务，才能够更新这些信息。而元素层在运行的过程中，也会有一些不断变化的信息，比如内存占用率等，而这些信息对于帮助逻辑层作出决策等也是非常重要的，因此在运行时会定时或接受请求后发送给逻辑层服务器。对于 Host 类型的元素，包含的动态信息如表3.3。

虚拟机服务器是 elop 里面提供云计算环境的重要基础。elop 软件包用的是

libvirt[33] 作为中间件，可以实现对 Xen，KVM，VirtualBox 等多款虚拟机软件进行控制。

Libvirt 是一个软件集合，可以用来管理虚拟机和使用其他虚拟化功能。这个软件包包括一个 API 库，一个 daemon，一个命令行工具，libvirt 的存在可以让用户以单一的方式控制各种虚拟化工具。

elop 软件包中主要使用了 libvirt 的 API 包，能够进行丰富的虚拟机生命周期操作，比如启动，停止，暂停，保存，恢复和迁移等等。这些功能都在 elop 中得到了体现。用户可以通过 elop 发送开启，关闭，复制，迁移虚拟机的命令，从而对自己具有权限的虚拟机进行一系列的控制操作。除此之外，elop 软件包也会根据服务器的负载情况，自行进行虚拟机的迁移操作，以优化服务器集群的状态。

VmHost 这个元素层单元，通过网络接收到 elop 私有格式发送过来的信息时，会对任务进行解析，无法解析的任务将传递回基类继续解析。解析到是对虚拟机进行的操作时，首先会通过 VO 管理层验证用户的权限，如果有权限则调用 libvirt 的相应 API 进行操作，如果没有对应的权限，则会拒绝该操作。

虚拟机对应着云计算中的计算资源虚拟化，相当于 AWS（Amazon Web Service）中的 EC2 服务，AWS 中的另一个重要服务 S3，则对应着存储服务。在 elop 中，也实现了一个大规模分布式存储服务的元素，作为 Element 的一个子类：Storage。

Storage 底层使用的 MooseFS[34] 作为分布式存储的管理工具。MooseFS 是一个分布式的存储框架，相比于大多数同类型的分布式存储软件，他使用通用文件系统进行访问，能够最大限度的兼容已有程序，部署简单，高可用，提供回收站，可在极小牺牲性能的情况下设置更高的冗余级别，在中等规模的分布式文件存储中非常有效。性能方面，使用大文件测试时，写的速度大约在 20-30MB/s，读的速度为 30-50MB/s，能够满足大多数存储任务的需求。

3.3　逻辑层

逻辑层的最主要任务是收集和管理元素层数据，并提供抽象的访问元素的方式。元素的具体物理特性被抽象为逻辑层描述信息，这样用户在访问不同的元素，甚至是经过二次开发的全新的元素时，并不需要知道元素的具体接口，而只需要访问抽象的逻辑层来提交任务，获取信息等。通过这种方式，具体应用并不需要关心元素的具体实现，位置和状态等，而只需要调用统一的逻辑层接口。Logic 层与数据库紧密连接，因为元素的具体信息，状态，历史数据等都由 Logic 层负责存入和读取。在 elop 中，我们使用了 ODBC 作为数据库接口，MySQL 作为数据库

底层服务。由于 ODBC 的兼容性，可以使我们在不改变任何代码的情况下，将底层数据库替换为其他的关系型数据库。图3.2表示了逻辑层读写数据库的示意图。

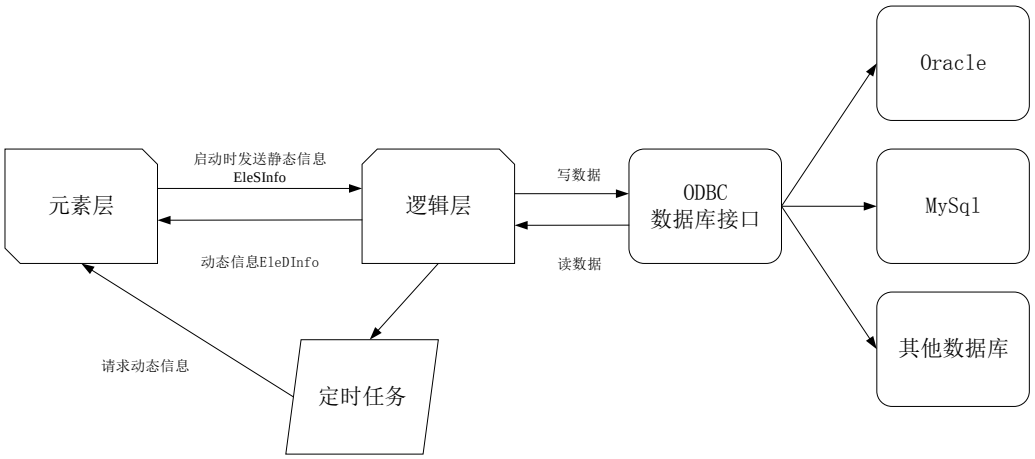

图 3.2　逻辑层读写数据库示意图

每个元素层都和一个逻辑层服务器相连，在启动时，元素层首先找到对应的逻辑层服务器，然后发送消息，报告元素层上线信息，并提交元素静态信息（EleSInfo），随后逻辑层会将元素层提交的信息写到数据库中。如果该元素是第一次上线，将创建该元素的记录，如果是以前出现过的元素，将更新元素新的静态信息，并标记为上线。

除静态信息外，每个元素在运行过程中会有大量的动态信息需要记录，比如虚拟机服务器，CPU 和内存使用率都是非常重要的动态信息，在创建新的虚拟机和迁移虚拟机时，都是重要的参考指标，因此这个信息也都需要记录在数据库中。

发送动态信息机制有两种，一种是元素自己定时发送动态信息，一种是逻辑层设置一个定时任务，向元素层要求动态信息，然后元素层进行回复。这两种方式在 elop 中都有使用，对于元素层的动态信息更新频率要求不高，或动态信息本身变化不大时，通常采用第一种方式，这样元素层可以根据自己的负载情况，选择合适的时机收集和发送动态信息，在网络较为繁忙，或动态信息一直没有较大变化时，可以暂缓发送动态信息，有利于整个系统的稳定运行。对于元素的动态信息实时性要求较高时，通常选择第二种方式，逻辑层可以根据需要来调整获取动态信息的时间间隔。比如用户请求非常多，经常需要最新的元素层信息时，可以加快获取动态信息的频率，或是在用户请求时实时发送，就可以获取最新的状态。在用户请求比较少的情况下，就可以降低发送的频率从而节省带宽消耗。

不同的元素动态信息的格式完全不同，比如虚拟机服务器，动态信息主要是服务器的负载状态，而文件服务器，动态信息则主要是文件列表信息，文件的更

新状态等。对于二次开发的新元素，则更有着无法预料的信息格式，如何将这些信息存入数据库是一个重要的问题。在 elop 中，作者使用了一个动态的描述基类来描述所需要的数据库格式。

```
enum TYPENUM    ///define database table property type
{
    TYPE_INT = 0,
    TYPE_FLOAT = 1,
    TYPE_DOUBLE = 2,
    TYPE_CHAR = 3,
    TYPE_CHARSTRING = 4,
    TYPE_STRING = 5
};
```

图 3.3　数据库项枚举类型

图3.3表示数据库字段类型的枚举，用在后面 TYPENUM 信息中，表示这一字段是什么类型的。

```
typedef union tableValue  ///define database table property type
{
    int int_value;
    char char_value;
    float float_value;
    double double_value;
    char charstring_value[MAXSTRINGLENGTH];
}tableValue;
```

图 3.4　数据库项的值

字段的实际内容，被存储在一个 union 类型的字段中。如图3.4。

图3.5所表示的这个类是存储一条数据的基类，在该类型中，真实的数据都存储在数据结构 vector 中，通过 namelist 存储字段的名称，typelist 存储每一个字段的类型，valuelist 存储内容。同时，该类还实现了 ToString 和 FromString 的方法，能够将数据格式化成字符串，并能够通过格式化的字符串转换回该类。从而弥补了 C++ 中没有很好的序列化和反序列化类的缺陷，使元素和逻辑层之间交换信息变得容易了很多。

同时，针对该类还有一个数据库的读写函数，能够直接将该类中的信息写到数据库中，或从数据库中获取信息填到类里面。有了这一个通用的方法之后，就不再需要针对每一个人物，编写数据库操作代码，而只需要将对应的字段名和字段内容填到这个类的实现中，就可以调用统一的函数完成数据的读写操作。

该类充分利用了 C++ 的多态特性，设置了几个虚函数，只需要在继承该类时实现这几个虚函数，就能够让调用者在不知道继承类的具体属性的情况下，完成

```
class RECORD
{
public:
    friend class RECORDLIST;  ///< define RECORDLIST as a friendly class
    virtual ~RECORD(){};
    std::string m_classname; ///<database table name
    char* FromString(char* inlength);
    char* ToString(int& length);
protected:
    void Set(std::string name, int& value);
    void Set(std::string name, float& value);
    void Set(std::string name, double& value);
    void Set(std::string name, char& value);
    void Set(std::string name, char* value);
private:
    bool isAdd; ///<to determin whether set/get value
    virtual int GetInfo() = 0;
    virtual std::string GetClassName() = 0;
    virtual RECORD* MakeNew() = 0;
    std::vector<std::string> m_namelist;  ///<database table column name
    std::vector<TYPENUM> m_typelist;      ///<database table column type
    std::vector<tableValue> m_valuelist;  ///<database table column value
};
```

图 3.5　数据表项基类

信息的获取。这一特性在开发者在 elop 上进行二次开发时特别有用，这样开发一个新的元素层元素时，不需要实现对应的逻辑层，就能够将动态信息传递到逻辑层，并写入数据库。

除了数据库相关的操作外，逻辑层还有一个重要的任务就是调度元素。比如在虚拟机应用中，用户请求建立一个新的虚拟机，逻辑层将会在所有在线的虚拟机服务器元素中，选择一个为其提交任务。这就是一个典型的调度问题，针对这一问题，逻辑层封装了一个 Scheduler 来实现调度的功能。用户在需要实现调度类型的逻辑时，可以直接继承 Scheduler，然后覆盖其中的调度算法，就可以自动完成调度功能了。

调度时，程序会启动数个线程来进行计算，个数由配置文件确定。以虚拟机应用为例，调度算法会从数据库中获取各个虚拟机服务器的动态信息，主要看内存占用率和 CPU 占用率，如果有超过一定阈值的，将会启动迁移算法，将占用率过高的机器上的某个虚拟机迁移到相对闲置的虚拟机服务器上，并且将会对所有虚拟机服务器进行优先级排序，占用率越低，优先级越高，这样下一次用户请求建立新的虚拟机时，将会直接找优先级最高的服务器进行交互。

在二次开发的过程中，如果所建立的元素类型也是需要进行调度的，并且需要实现新的调度算法，只需要完成一个新的 scheduler 类，继承自 BasicScheduler，并覆盖实现其中的 schedule 方法即可。

3.4　组织层

组织层的任务是管理用户以及元素之间的关系。其管理模型所有成员都在一个 CI(CyberInfrastructure) 的管理中，存在多个虚拟组织 VO，每一个 VO 有一个 VO 服务器，VO 服务器通过数据库管理着这个虚拟组织的成员信息和权限信息。通过一定格式的网络访问，成员或管理员可以查看权限内的成员信息，成员在试图访问某些资源时，管理资源的逻辑层也会自动向 VO 查询成员的权限信息，确实是否为授权操作。

根据不同用户的权限，可以执行表3.4操作中的一部分：

表 3.4　组织层任务描述

任务	说明	权限
ListAllMember	列出虚拟组织中所有用户	虚拟组织管理员或成员
SetupVO	建立 VO	普通用户，建立后自动成为管理员
AddMember	添加用户进虚拟组织	虚拟组织管理员
DelMember	将用户从虚拟组织中删除	虚拟组织管理员
DeleteVO	删除一个虚拟组织	虚拟组织管理员或最后一个用户
ListMyVO	列出用户所在的所有虚拟组织	普通用户
ListAllVO	列出所有虚拟组织	虚拟组织管理机构或所有用户
ListAllEle	列出虚拟组织中所有元素（资源）	虚拟组织成员
ListAllLogic	列出所有逻辑层信息	所有用户

在二次开发的过程中，开发者选择不使用虚拟组织的形式管理用户，而在逻辑层自行实现权限控制，但是通常使用已经完成好的虚拟组织管理资源和用户是一个更好的选择，也有利于有其他模块相结合。

组织层不仅包括这个虚拟组织，还包括管理着所有用户，元素层，逻辑层，虚拟组织服务器的证书管理分发工作的 CI 管理中心。证书采用 X.509 格式，用户可以提交信息申请个人证书，所有的元素层，逻辑层，组织层服务器也都需要一个证书才能与其他用户和服务器完成加密通讯及认证身份。

3.5　流程层

流程层是最接近用户的一层，用来读取用户请求，顺序执行一系列任务。流程层目前主要识别的格式是 XML 文件，用户可以将任务描述写在 XML 文件里，然后通过网络提交到流程层或直接放置在流程层能够访问到的位置，XML 文件里描述的任务就会被自动的执行。

比如用户请求在虚拟机服务器集群中为其分配一台虚拟机，在存储集群中为其准备 10G 空间，然后在虚拟机服务器中安装指定软件并运行，这一系列的顺序操作，就可以将任务手动写成 XML 格式或使用自动工具生成 XML 格式信息，然后提交到流程层。任务格式示意图3.6。

```
<elop>
<Job no="1" l_addr="192.168.1.101" l_port="5555"
        ele="VmHost" VO="VO1" usercert="cert1">
    <cmd>CreateVm</cmd>
    <parameter no="1">my-linux-sl6</parameter>
    <parameter no="2">SL6-full</parameter>
</Job>
<Job no="2" l_addr="192.168.1.102" l_port="5556"
        ele="Storage" VO="VO1" usercert="cert1">
    <cmd>CreateBucket</cmd>
    <parameter no="1">10G</parameter>
</Job>
<Job no="3" l_addr="192.168.1.103" l_port="5557"
        ele="Software" VO="VO1" usercert="cert1">
    <cmd>Install</cmd>
    <parameter no="1">my-linux-sl6</parameter>
    <parameter no="2">eclipse</parameter>
</Job>
</elop>
```

图 3.6　流程层任务示意

流程层在处理顺序任务时，首先会找到任务所对应的逻辑层，请求一个元素，或得到元素地址和授权之后，向元素层发送具体任务，元素层执行任务（比如创建虚拟机）后，将结果返回给流程层，流程层暂存这个结果，继续执行后面的任务，前面任务执行失败的话，会退出流程执行，将结果返回给用户。

用户获取结果的方式也有两种，一种是通过网络提交任务后保持连接，执行结束后直接将结果通过网络回复到用户，另一种是将结果写入结果文件，存放在指定目录下，由用户自行查看。针对具体的任务不同，可以根据需求选择合适的方式。

3.6　管理界面

elop 软件包除了完成了底层的一些网络通讯和逻辑外，也包含了一个管理界面，通过浏览器访问。在管理界面上可以完成一些简单的任务处理，成员管理等，与监控系统集成后，在界面上可以方面的监控资源（元素）的具体状态。

在图3.7所示的这个管理界面里，可以很直观的对用户，虚拟组织，元素进行

	ID	用户名	证书	密码	Owner	地址	端口	权限	类型	其他
1	1	vox1	vox1.pem	123456			111	2	1	
2	2	admin	admin.pem	123456			3333	1	1	
3	3		w.pem					0	0	
4	4	usertest01	usertest01.pem	123456	test	0.0.0.0	222	3	1	oooo
5	5	Test1	E3.pem	123456	public		22	4	1	
6	6	Test1112	x1.pem	123456	public		333	4	1	
7	7	AdmUser	E14.pem	123456	public		111	2	1	
8	8		E1.pem					0	0	
9	9	test	x11.pem	000000	public	0.0.0.0	0000	0	1	testuser
10	10		lin01.pem					0	0	
11	11		A.pem					0	0	

图 3.7　组织层界面

管理。用户之间的通信安全主要靠独一无二的证书实现，表格中的密码仅仅是设计管理界面时测试使用，对安全性并无影响。

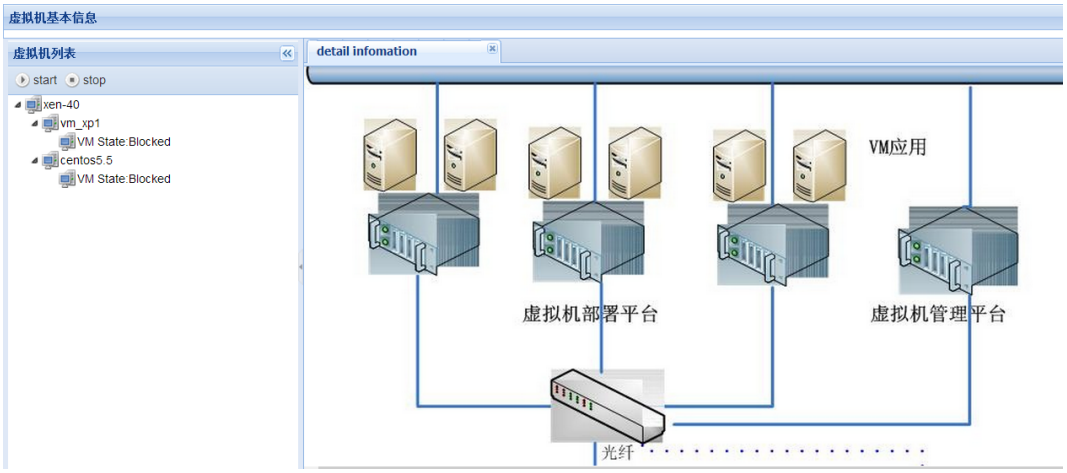

图 3.8　虚拟机组织结构

针对虚拟机的应用，在点开虚拟机列表之后显示的是虚拟机的集群拓扑结构图的示意，如图3.8。目前虚拟机集群规模较小，且全部在同一个机房，都是直接连接到路由上对外提供服务。在集群规模扩大后，将采用多级结构，这个示意图也将随之更改。

图3.9是虚拟机服务器监控图，可以随时查看列表中的虚拟机服务器的 CPU，内存使用情况，从而对集群的状态有快速直观的了解，方便做出决策。

监控系统全部就位之后，如果某台虚拟机服务器出现问题，也可以快速启动恢复措施并发出警报，由管理员来进行处理。如果某台虚拟机服务器出现负载过重，IO 异常，响应速度变慢等状况，也可以在机器彻底宕机之前及时发现，启动迁移程序，将服务器上的虚拟机快速迁移到其他服务器上，从而在不影响终端用

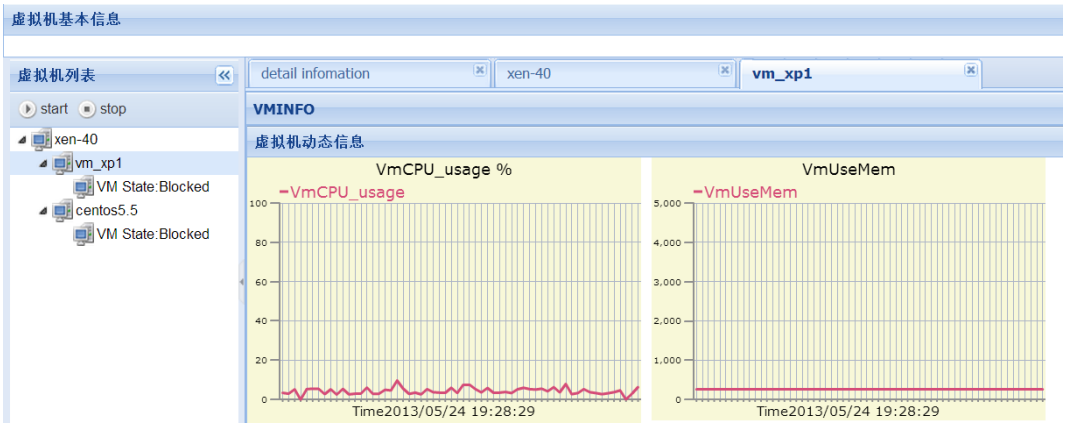

图 3.9　虚拟机状态监控

户的情况下解决问题。

存储服务的部分采用 MooseFS 这一个开源软件作为底层服务，而 MooseFS 自身已经带有监控系统，elop 软件包的界面服务将 MooseFS 的监控系统集成到一起，在列表中访问 MooseFS 服务节点时，可以直接显示出完善的监控信息，如图3.10。

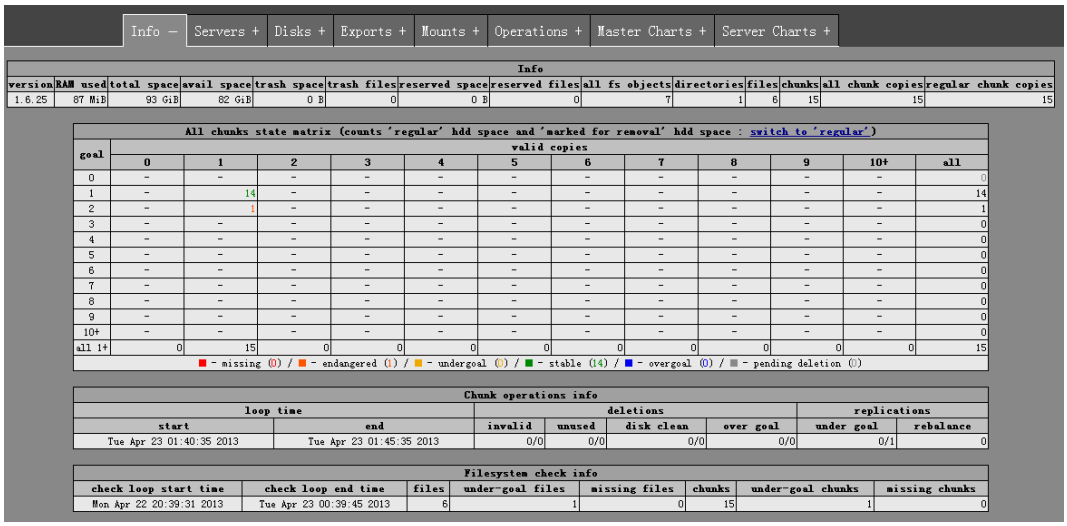

图 3.10　存储 MooseFS 监控系统

由于虚拟机服务器和存储集群部署在相邻的位置，终端用户同时使用 elop 集群的虚拟机服务和存储服务时，使用虚拟机连接存储服务的速度将非常快，为大数据量的存储，计算一体的需求提供了便利。

将 elop 部署在多个数据中心之后，不同的数据中心之间的网络连接将无法达到同一数据中心内部网络的速度和稳定性，所以在为用户分配虚拟机资源和存储资源时，调度算法可以进行扩展，如果用户需要高速稳定的连接，可以将存储服务和虚拟机服务部署在相同机房，如果用户需要分离以保证高冗余可用性，也可以为用户分配不同数据中心的服务器。在 elop 高度可扩展的模型框架下，用户的

几乎所有需求都可以通过扩展开发的方式得到满足。

3.7　远程访问虚拟机资源

elop 为用户分配了虚拟机资源之后，用户如何进行访问也是一个问题。对于搭建网站，科学计算等需求，使用的是 Linux 系统的情况下，用户可以很方便的通过 ssh 等方式登录到系统中进行使用，而不需要额外的工作。但是桌面办公应用也是未来云计算的一个重要方向。因此桌面云也是 elop 系统所重点关注的一类应用。

针对使用的虚拟机软件不同，访问虚拟机资源的方式也有不同，VNC 是一个跨平台的通用的远程访问工具，因此只需要将虚拟机内预装好 VNC 服务器，用户就可以在开启虚拟机之后，通过 VNC 来访问虚拟机的桌面系统。因此这种方式适合所有的虚拟机工具所管理的虚拟机。

针对 VirtualBox 所管理的虚拟机，还有另一种方式能够提供远程桌面访问服务。VirtualBox 自带了 RDP 协议的服务端程序，因此使用 VirtualBox 创建并打开虚拟机之后，可以很容易的通过 RDP 协议进行访问。相比于 VNC 的方式，RDP 的客户端在 windows 系统中是自带的，因此客户不需要专门安装软件，而且 RDP 传输的是操作信息，而不是完全的图像信息，数据量更小，延时更小。而且即使在 Linux 系统中，也可以很方便的通过安装 rdesktop 软件来使用 RDP 协议远程访问虚拟机。

在服务端，通过相应的命令开启虚拟机，就可以让虚拟机在后台打开，并且在宿主机中的某一个端口开启基于 TCP 协议的 RDP 服务。

在客户端可以使用 rdesktop[35] 命令来远程访问该虚拟机。然而，VirtualBox 中的 RDP 协议仅有不进行认证和进行简单认证两种方式来控制终端的连接，从安全访问的角度来讲这是远远不够的。因此，我们使用了 VPN 来保证访问的安全性和唯一性。

VPN（Virtual Private Network），即虚拟专用网络。目前组建 VPN 虚拟专用网主要依靠四项技术来保证安全：隧道技术、加解密技术、密钥管理技术和身份认证技术[36–38]。在本工作中，我们用 OpenVPN 对证书的验证功能。

系统分为客户端和服务器端，效果如图3.11所示。在上图中，客户端指的是远程访问使用的显示终端，服务器端指的是虚拟机提供桌面云端。服务器在云计算中可以是提供虚拟资源的服务器集群，在虚拟组织内每位提供虚拟机共享的成员都可以作为服务器。由于 VirtualBox 的访问模式是主机的域名（或 IP）加上被访问虚拟机占用的主机端口，虚拟机访问权限的控制就可以简化为对服务器端口

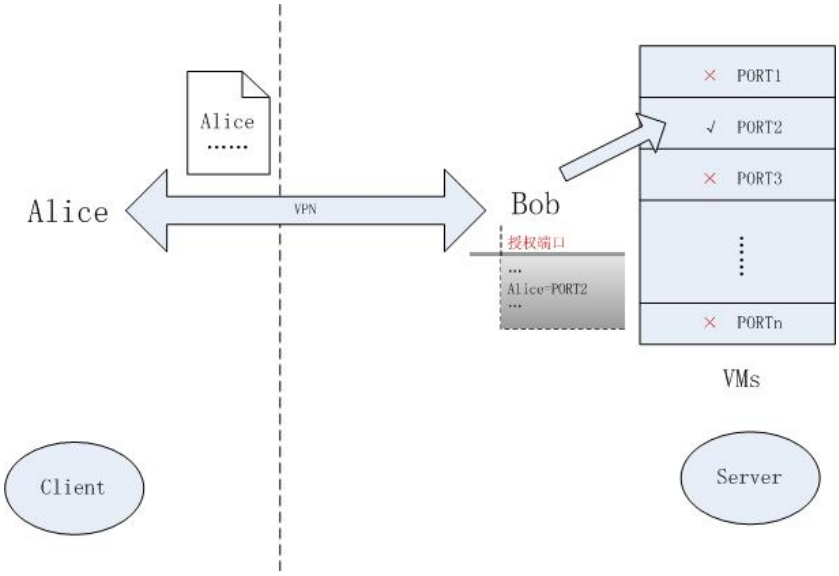

图 3.11　防火墙和授权远程访问实现

访问的控制。

防火墙开启后，所有虚拟机所占用的端口都被保护，不能被访问。如何开启某端口的访问权限，这需要 OpenVPN 的配合。

首先在服务器上安装 OpenVPN，使用 OpenVPN 进行身份验证需要 CA 的公钥和 CA 颁发给用户的证书，在配置文件中设置公钥证书的访问路径。OpenVPN 配置完毕后开启 OpenVPN 服务，客户端就可以与服务器端建立安全连接。

需要注意的是，并不是所有的证书验证通过就能获得端口访问权限，证书验证通过只能说明对方与自己来自同一个组织，这里需要增加用户访问端口的权限。读取证书后获得证书 CN(common name)，使用 CN=PORT 的格式添加使用权限，如授权成员 Alice 端口 3399 的访问权限，就可以在/etc/openvpn/clients.rc 添加 Alice=3399。

我们在服务器端开启两台虚拟机，操作系统分别为 Windows XP 和 Ubuntu，占用主机端口分别为 3389 和 3391。然后开启防火墙和 OpenVPN 服务，等待客户端的连接。

客户端使用 CA 服务器颁发的证书去连接虚拟组织服务器，如果服务器的端口访问控制策略里授权该用户使用某端口，则可以访问该端口，即可以远程控制该端口的虚拟机。

本例中服务器为证书 grrrcert.pem 授权了端口 3389 和 3391，所以可以使用 rdesktop 连接 10.8.0.1 的 3389 端口，如图3.13。客户端与服务器主机之间建立的安全通道，服务器端被定义为 10.8.0.1 这一虚拟 IP，使用服务器真实 IP 连接则被防火墙拒绝。

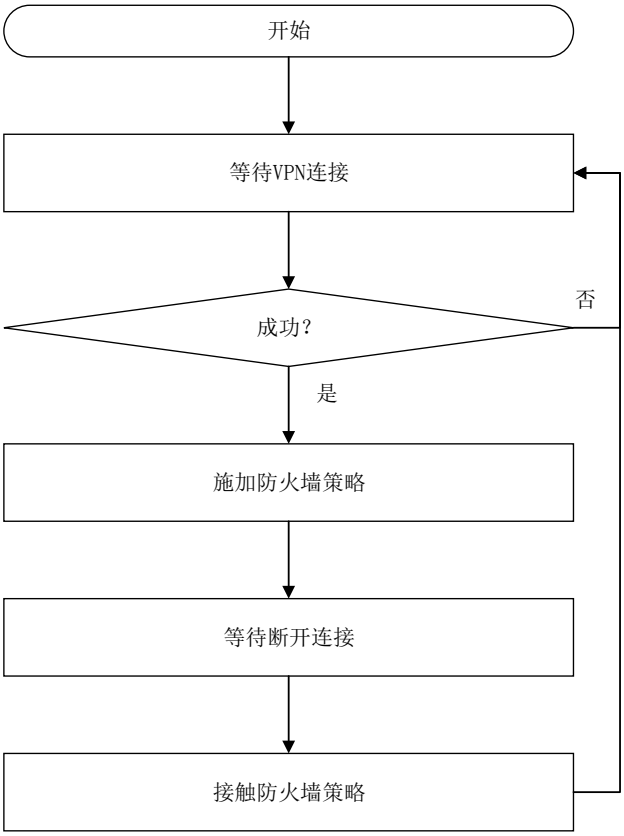

图 3.12　远程访问虚拟机流程

如图3.14所示，服务器为 alicecert.pem 授予 3389 端口的使用权限，alicecert.pem 可以正常访问该端口的虚拟机，但是对于未授权的 3391 端口则会被服务器拒绝，尽管此时 3391 端口已经为用户 grrrcert.pem 开放。

至此可以发现，使用防火墙和 OpenVPN 使虚拟机的共享安全和可靠，即控制了访问权限，又保证了数据传送的安全。

3.8　自动安装软件

云计算的一大特性是弹性计算能力，因此在用户数量突然增长时，应该能够快速提供新的计算能力给用户，在用户数量减少时，可以随时回收资源，节约成本。在为用户提供新的虚拟机时，每个用户的要求也不尽相同，而且通常会需要预装某些软件，进行一些设置，以便能够在无人管理的情况下，将新增的虚拟机资源投入使用，这也是云计算提供商所需要完成的基本任务。

在本系统中，流媒体服务软件包的自动安装是一个重要需求，因为在用户数量增加时，如果能够在新增的虚拟机中自动安装 DSS 并开启服务，将不需要有人

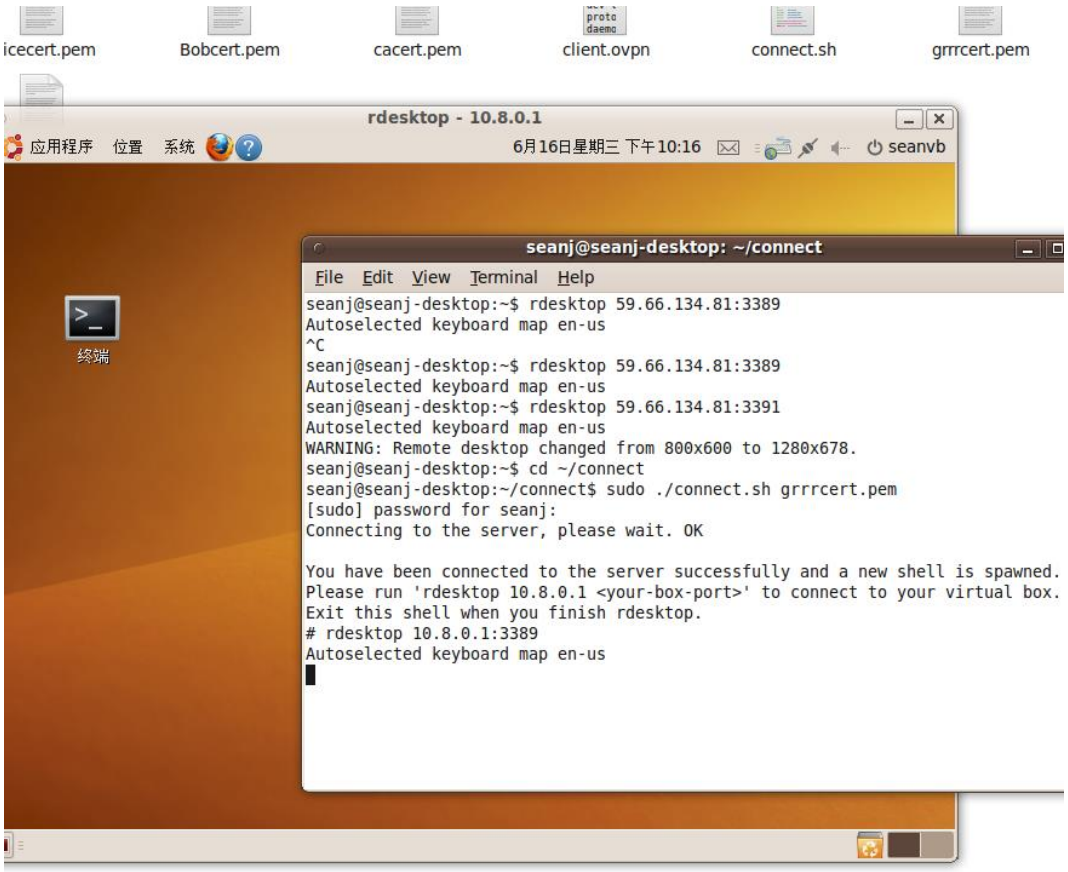

图 3.13　远程访问虚拟机效果

```
# exit
Disconnecting...  Done
seanj@seanj-desktop:~/connect$ sudo ./connect.sh alicecert.pem
Connecting to the server, please waitEnter Private Key Password:
. OK

You have been connected to the server successfully and a new shell is spawned.
Please run 'rdesktop 10.8.0.1 <your-box-port>' to connect to your virtual box.
Exit this shell when you finish rdesktop.
# rdesktop 10.8.0.1:3389
Autoselected keyboard map en-us
# rdesktop 10.8.0.1:3391
Autoselected keyboard map en-us
ERROR: 10.8.0.1: unable to connect
#
```

图 3.14　远程访问限制未授权用户

值守就可以快速提高服务能力。

类似的需求也出现在其他很多领域，比如作者之前的工作 [39]，国际科学合作组织 LIGO 中，有大量的科学分析软件，而每个用户不一定能够全面的了解或熟悉这些软件的安装，而且自己动手安装所有的软件也非常耗时，因此需要一个能够自动在虚拟机中安装软件的工具，将所需要的软件安装完成之后，再提供给用户使用。在本系统中，也使用了类似的技术来自动安装所需要的远程桌面，流

媒体服务软件等必须的工具。

3.8.1　软件安装类型

在 Linux 下安装软件，通常有三种主要的方式：源码安装，rpm 包安装和 yum/apt 管理工具安装。

源码安装：源码安装在 Linux 环境下是一个最为传统的安装方式，理论上，所有的开源程序都可以通过源码安装的方式安装到 Linux 系统上，而且通过源码安装方式安装的程序，用户可以很清楚的了解到所有的程序位置，依赖关系等等。但是随着软件的规模越来越大，依赖越来越复杂，源码安装的方式并不能满足所有的用户需求。因为在源码安装的过程中，会自动检查依赖关系，所依赖的程序不存在或版本不符合时，会自动退出安装并给出提示。用户需要自己去下载依赖程序，然后手动安装或更新依赖的程序。对于依赖关系较多的程序，这一过程会非常耗时，以至于经常无法进行下去。

RPM 包安装：RPM 包是 RedHat 系列 Linux 系统中通用的二进制安装包格式，使用这种方式安装通常会比编译安装省时很多，因为 RPM 包中的内容都是编译过的。在 Debian 系统中，也有对应的 Deb 包。两者大同小异。在 RPM 包中，有一个 xml 格式的描述符，用来描述包的所有依赖关系。如果手动的使用 RPM 包安装程序，依赖条件不满足的时候，也会给用户提示具体的信息，但是无法自动下载安装这些依赖程序。但是 RPM 包中提供的信息更加标准化，因此通过下面第三种方式安装时，就可以自动解决好所有的依赖关系，因此也是用户更加常用的安装方式。

管理工具安装：在 Redhat 系列中，安装包管理工具是 yum，而在 Debian 系列中，工具是 apt。他们分别对应着自己的软件包格式 rpm 和 deb。他们都是采用源来管理 Linux 中大量的软件，在源中，大量软件包的各种版本都被以规范的方式存储在一起。在系统中配置好源的位置信息之后，系统就有能力自动从源中下载和安装所需要的程序。事实上，这就是将软件安装作为一种服务提供给用户。通过 yum/apt 的方式安装软件时，软件所依赖的所有其他程序都会自动从源中下载下来，因此这是最为方便快捷的安装方法。

3.8.2　自动安装软件

以上三种方法都是手动操作时可以使用的，在安装过程中，可能随时需要一些按键配合，比如 yum 安装过程中，会需要回答 Y/N 的确认问题等等。这些交互为自动安装软件引来了麻烦。因为对于没有交互过程的安装方式，可以通过开机

运行脚本等方式实现自动安装，而对于有交互过程的安装方式，shell 脚本已经无法实现这个功能。

在这里，作者使用 Expect 工具来完成这一任务。这个工具被设计用于自动完成交互式问答的场景，比如 ssh 登陆，输入密码，回答问题等环节。能够实现 ssh 的自动登录也避免了使用开机自动脚本的麻烦。

Expect 脚本可以读取，控制，处理输入输出流，自动填写数据。自定义的脚本是通过 Tcl（Tool Command Language）的脚本语言来完成的。下面这是自动 ssh 登陆的一段示意程序，首先执行 ssh 登陆命令之后，使用 expect 语句，来等待可能的交互命令，比如收到是否继续的提问时，会自动发送 yes，在询问密码时，自动发送密码并回车。主要操作见图3.15。

```
spawn ssh -l $user -p $port $ip
set timeout 500000
    set done 1
    set timeout_case 0

    while ($done) {
            expect {
        "Are you sure you want to continue connecting (yes/no)?" {send "yes\r"}
                    "password:" { send "$password\r" }
                    "#" {
                            set done 0
                            send_user "Login Successfully...\r"
            break
                    }
            timeout {
                      switch -- $timeout_case {
                0 { send "" }
                1 {
                    send_user "Send a return...\r"
                            }
                2 {
                    puts stderr "Login time out...\r"
                                exit 1
                }
            }
        incr timeout_case
        }
            }
    }
```

图 3.15　Expect 自动登录 ssh 脚本

下图是本系统顺序安装一系列软件时的流程图，首先登陆之后，需要设置一些环境变量，然后顺序安装所需要的软件，安装过程中，yum 有提问是否继续时，自动回答是，就可以完成所需软件的安装了。流程参见图3.16。

对于不同用户，所需要的软件不一定相同，但是只要是通过 yum/apt 进行安装的软件，都可以很容易的通过同一流程完成。对于不是通过 yum/apt 安装的，则需要用户手动进行一些配置看，比如执行什么脚本，如何交互。比如本系统所用

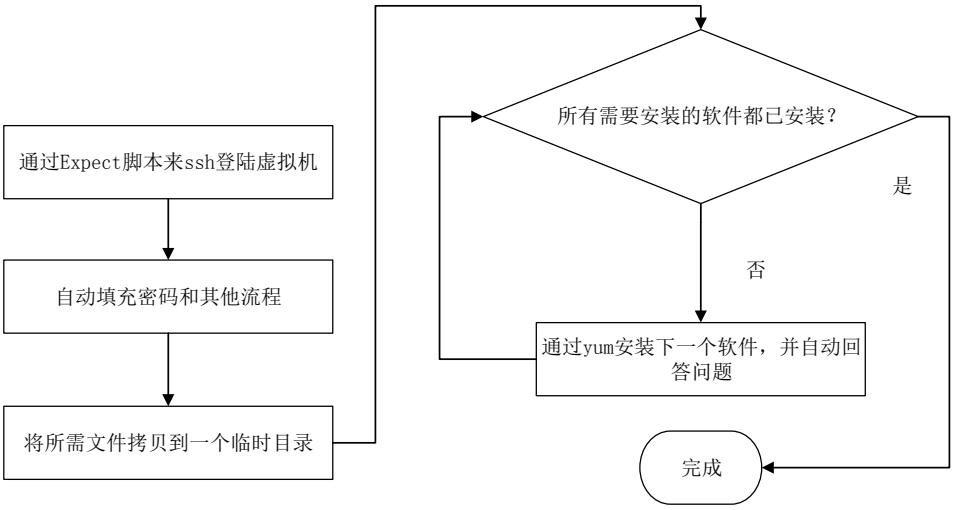

图 3.16　自动安装软件流程图

到的 Darwin Streaming Server，就需要通过自定义的脚本来完成源码安装的过程。

有了自动安装软件的工具之后，当用户数量增加，服务器性能不够时，可以手动或自动的向集群要求增加虚拟机资源，然后通过预设的程序在虚拟机中自动安装好所需要的软件，并设置好启动脚本，虚拟机开启后，就可以直接开始提供服务了。另一种实现开启虚拟机直接提供服务的方式是安装完所有软件后制作虚拟机镜像，直接从镜像中创建新的虚拟机。

作为提供服务的云平台，自动安装软件能够为云平台的客户提供大量的便利，增强了整个过程中的自动化程度，且避免了手动操作出错的可能。相比于采用安装完所有软件后制作镜像的方式，自动安装软件这种方式更加灵活，对于需要多种不同环境的虚拟机的客户来说更加实用。

第 4 章　系统实现

4.1　整体架构

本系统能够通过 Android 平板设备无线访问，观看课程视频。课程的列表和介绍信息都存储在 Java 服务器中，而流媒体内容存储在 elop 的服务器中。Android 设备首先通过 HTTP 请求联系服务端的一个 JAVA 服务器，然后由服务器提供内容列表，简介等信息，并为其联系 elop 服务器，分配一台流媒体服务器。因此整个系统主要分为客户端，Java 服务端，elop 云平台端三大部分，系统架构图4.1。

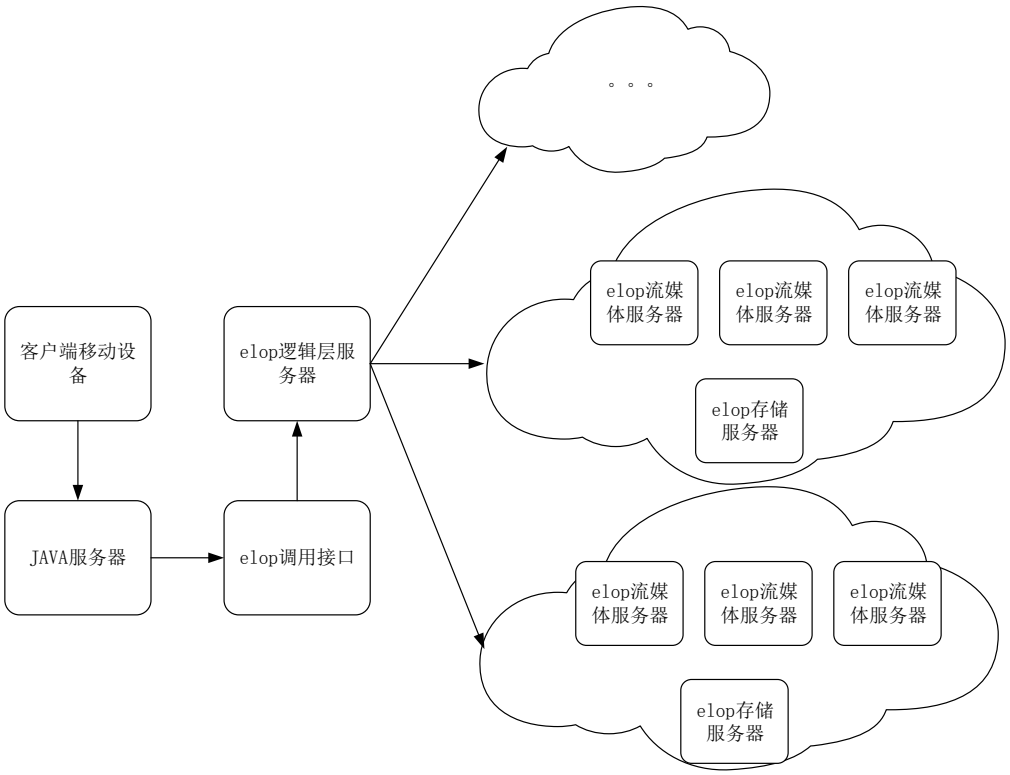

图 4.1　系统整体架构

通过上图可见，elop 集群的数目是可扩展的，因此当用户数量增加时，能够弹性的增强服务能力。而为了防止 JAVA 服务器成为服务能力扩展的瓶颈，我们

将 JAVA 服务器设计为无状态的服务模式，可以很容易的扩展到多台，通过反向代理实现负载均衡，从而提供良好的可扩展性。

客户端启动后，首先会联络服务器，获取最新的课程列表信息，并显示给用户。当用户点击某一类课程时，会显示出该类课程中具体的课程信息。此时所有的通信都是客户端与服务端之间的。如果用户选择观看某一节课程，服务端会向视频云端发起请求，选择一台合适的服务器来提供服务。客户端的播放控件就会和视频服务器建立起 rtsp 连接，并播放视频。

4.2　流媒体服务器安装

4.2.1　Darwin Streaming Server(DSS) 的安装

本教育云平台系统最主要的一个功能就是网上视频教学，因此流媒体服务器是本系统的重要组成部分。

Darwin Streaming Server 是苹果公司提供的开源实时流媒体播放服务器程序。该程序全部由 C++ 编写，运行效率高，支持 MPEG-4，H.264 压缩格式，可扩充性好。并且该程序完全开源，在 Windows，Linux，Mac OS 等操作系统下均能运行。

目前，DSS 最新的版本是 6.0.3，本系统也是基于该版本进行的测试和实现。该版本的 DSS 没有提供 CentOS 系统下的 rpm 安装方式，因此需要采用源码安装的方式。

由于官方下载版本本身并不是 Linux 版本的，因此还需要进行打补丁等操作才能正常的编译安装。为方便整个安装过程，将下载，打补丁，编译等操作集合为一个安装脚本，只需要执行该脚本就可以顺利安装。

```
groupadd qtss
adduser -s /sbin/nologin qtss -g qtss
wget http://static.macosforge.org/dss/downloads/DarwinStreamingSrvr6.0.3-Source.tar
tar -xvf DarwinStreamingSrvr6.0.3-Source.tar
mv DarwinStreamingSrvr6.0.3-Source DarwinStreamingSrvr6.0.3-Source.orig
wget http://parsa.epfl.ch/cloudsuite/software/darwin/dss-6.0.3.patch
patch -p0 < dss-6.0.3.patch
mv DarwinStreamingSrvr6.0.3-Source.orig DarwinStreamingSrvr6.0.3-Source
wget http://parsa.epfl.ch/cloudsuite/software/darwin/dss-hh-20080728-1.patch
patch -p0 < dss-hh-20080728-1.patch
cd DarwinStreamingSrvr6.0.3-Source
mv Install Install.orig
wget http://parsa.epfl.ch/cloudsuite/software/darwin/Install
chmod +x Install
./Buildit
./Install
```

图 4.2　DSS 安装脚本

安装完成之后，将工作目录配置到存放视频文件的目录下，然后就可以自动接收客户端的请求，播放流媒体文件。DSS 还自带了一个网页管理程序，可以通

过浏览器对 DSS 的配置进行修改，也可以随时监控 DSS 的工作状态，比如多少个客户端连接等。

如图4.3，在监控页面中，我们可以看到 DSS 的版本信息，目前连接数，CPU 使用率，网络流量，总数据传输量等重要信息。

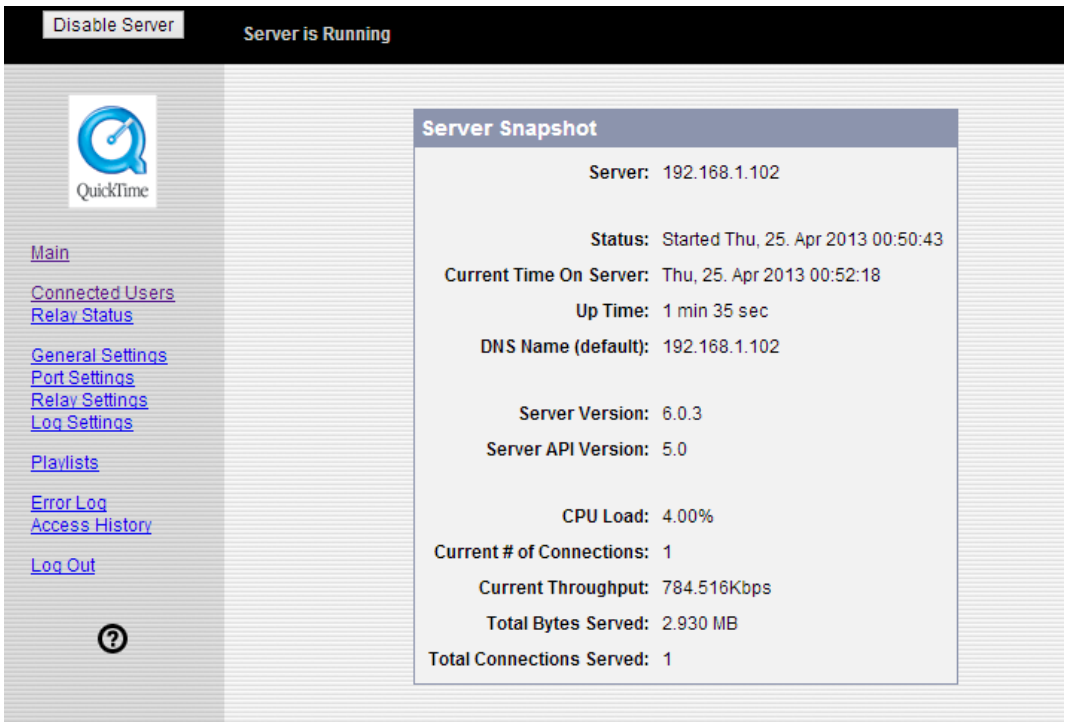

图 4.3　DSS 监控页面

如图4.4，在 Connected Users 一栏，还可以看到当前已连接用户的 ip，传输速率，丢包率，正在观看的视频文件等信息。

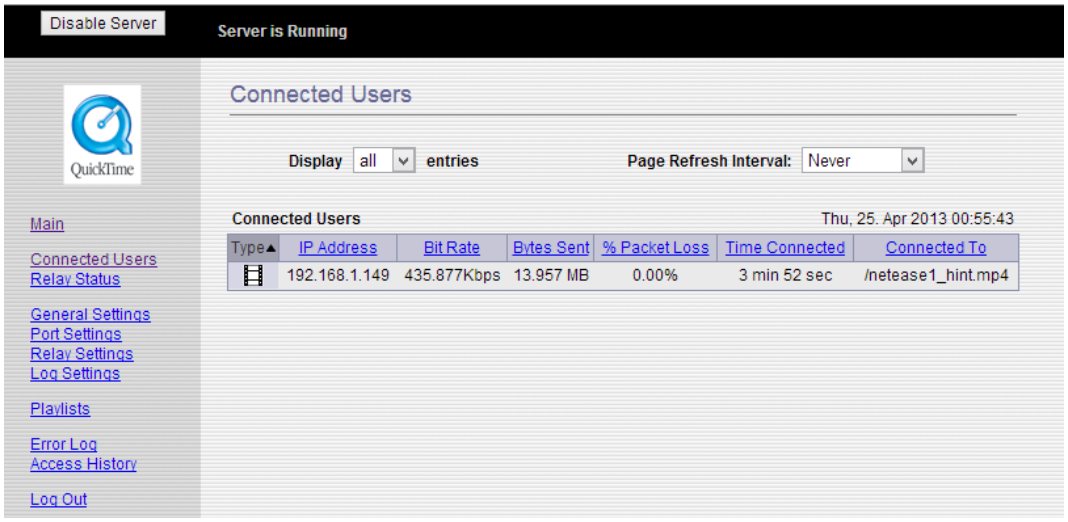

图 4.4　DSS 监控页面，当前连接用户

如图4.5，在 Access Histroy 一栏，可以看到各个文件被访问的次数，这些数据在实际的运营中都是非常有用的分析依据。

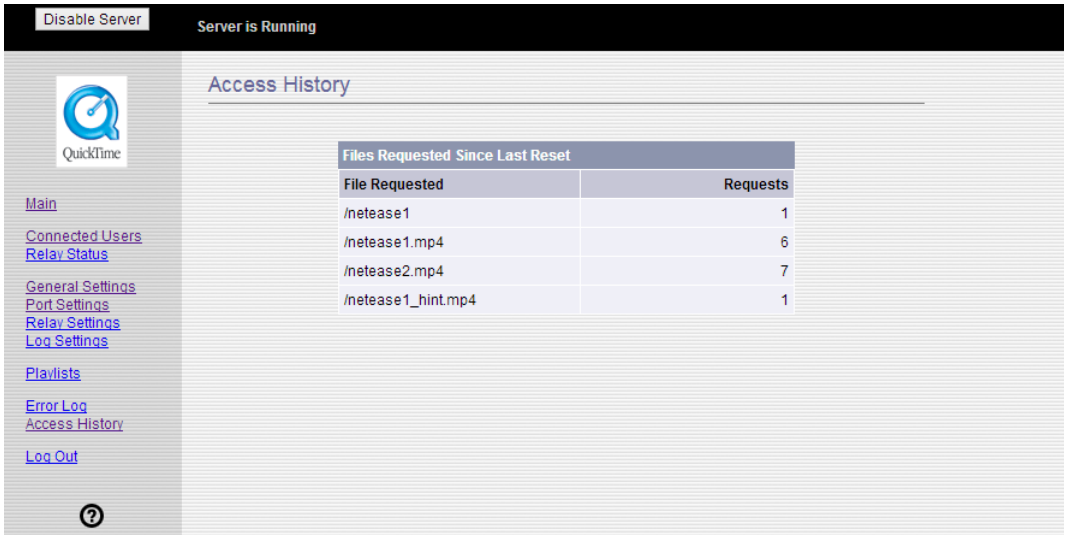

图 4.5　DSS 监控页面，播放历史

4.2.2　DSS 测试

图4.6为连接个数增长时网络吞吐量增长图，由图上可见，在客户端个数增长时，总的网络吞吐量基本保持你线性增长，每一个连接占用的网络带宽在 500kb/s 左右。

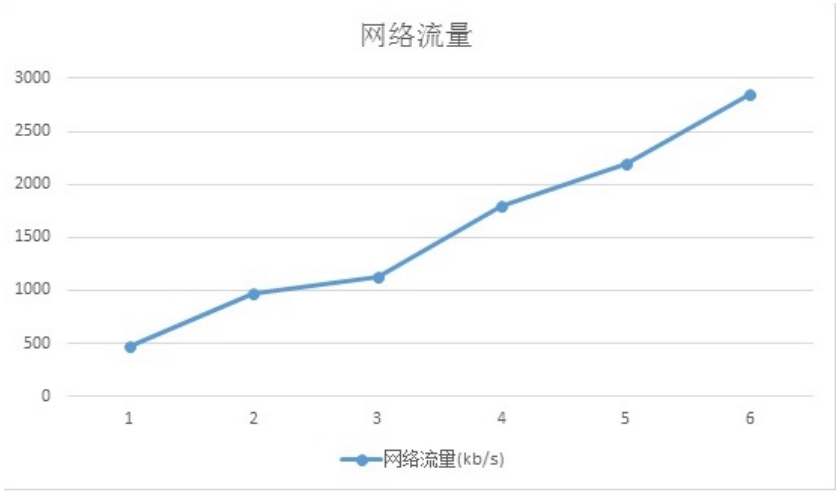

图 4.6　DSS 网络流量随连接数变化

图4.7为在连接数增长时，服务程序占服务器 CPU 的占用率增长示意图。

由图上可见，服务程序对于 CPU 的消耗并不大，服务程序初始占用 CPU 在

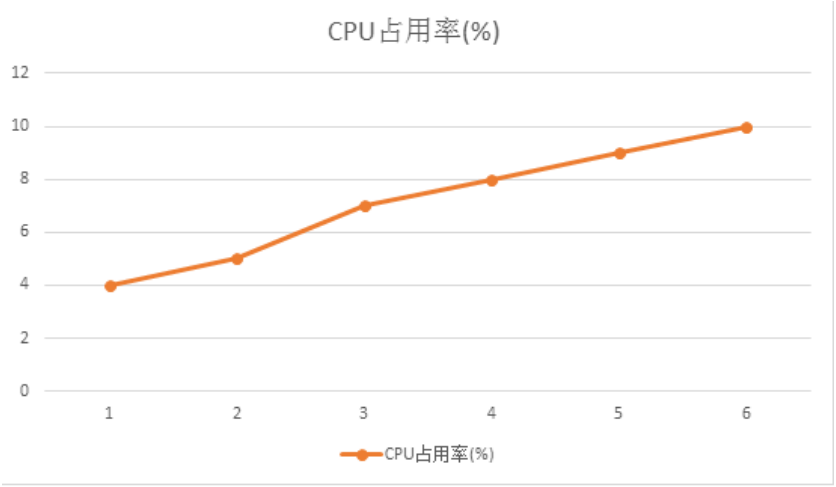

图 4.7　DSS CPU 使用率随连接数变化

4% 左右，而随着连接数增长，每个连接占用 CPU 大概在 1%。而且程序是在虚拟机中运行，只分配了一个 CPU。

图 4.8　DSS 内存使用率随连接数变化

内存使用情况可见虚拟机内的监控程序，如图4.8在采样时间内，用户数量正在增长，而内存使用几乎没有变化，因此本服务程序对内存的要求并不高。

经过上面的测试，对 DSS 的主要特性可以有简单的了解，DSS 使用 C++ 编写，效率高，优化较多，因此服务程序本身对服务器的性能要求并不高，在服务过程中占用服务器资源也并不大。占用情况最为明显的是网络带宽，这也是视频

服务不可避免的一点，因此在实际部署服务集群或在云服务商处租用计算资源时，带宽应该是最主要考虑的因素，内存和 CPU 能力可以相对较弱。

4.3　客户端实现

4.3.1　设计思路

客户端为 Android 应用，以 10 寸左右的平板为主要关注平台，因为看视频，过于小的屏幕比如手机屏幕，难以获得良好的效果，当然由于 Android 应用开发的通用性，在手机上也能够正常使用。

Android 是目前最为流行的移动计算平台，在手机和平板市场都取得了不错的成果，由于其免费性，平板市场仍有很大的潜力，Android 低价平板性能优良，价格低廉，非常适合作为教育终端来普及。因此选择 Android 平台首先支持的平台，能更有效的推广这一教育云平台。

在 Android 应用程序中开发中，主要分为本地程序和 Html5 程序两种设计模式，而这两种设计之间的竞争一直没有停止过，这也同样适用于 iOS 移动平台的状况。本地程序执行效率更高，能够展现更好的界面效果，更加快速的响应速度，而且经过良好设计的本地程序，在网络不稳定的情况下，可以给用户更好的使用体验。但是，对于本地程序而言，让用户及时更新是一个比较困难的事情，这经常会导致程序无法做大的改动，因为要考虑到旧版本的用户，而版本多了之后，就更容易出现兼容性的问题。程序的 Bug 不能及时解决，也非常影响用户体验。

Html5 模式设计的应用程序，实际上就是一个浏览器，而将实际的程序放在了服务器端。这样做的好处显而易见，就是程序不需要再本地进行更新，也就避免了上述的缺陷。当服务端有了更新或修正了 bug 的时候，只需要将服务端代码进行更新，用户甚至不会察觉。在 PC 上，网络应用已经占据了重要的位置，比如邮件，笔记等应用，大多数用户已经不再使用本地软件进行操作。然而，这种模式在移动平台上的缺陷也很明显，就是响应速度要明显弱于本地程序。无线网络连接的速度比 PC 上使用以太网或 Wifi 上网的速度要慢很多，而且并不稳定。一旦连接速度出现问题，应用就会出现无响应，假死的现象，这对于用户体现的损伤是非常大的。

通常，随着网络的进一步发展，Html5 劣势会逐渐变小，优势越来越明显，但是到目前为止，Html5 的应用在移动平台应用中，处于全面下风。Facebook 的移动应用曾经完全是使用 Html5 设计，但在几年的时间里，一直没有解决好响应速度慢等各种问题，因此也将苹果系统上的应用替换为本地应用了。因此在目前来看，使用本地应用的方式来进行开发，是更加实际更有效率的方式。

表 4.1　HTML5 程序与本地程序对比

本地程序	HTML5 程序
响应速度快	响应速度慢
视觉效果优秀	视觉效果一般
网络无连接时体验稳定	网络无连接时无法使用，或体验差
更新困难，版本乱	更新非常容易

4.3.2　界面设计

本系统的界面设计较为简洁，启动程序之后，会进入以下界面，通过分类界面选择自己感兴趣的课程，就可以点击进行播放了。

主题界面如图4.9。

图 4.9　客户端主体界面

播放课程的界面如图4.10。

这种界面简洁使用，易于理解，而且不容易出现 bug，而界面中的几乎所有内容，都是从服务器获取的，包括一级列表的信息。这样也就在一定程度上解决了更新的难题。目前是按照分类进行的课程划分，如果要修改为按照课程来源，教育机构，或者地区等等各种分类方式，都不需要在本地程序上做更新，只需要将服务器端的数据修改即可。

在播放视频中，我们使用了 Android 自带的 VideoView 播放控件。该控件很

图 4.10　客户端课程播放界面

好的支持 rtsp 协议，因此可以很方便的和服务端建立连接进行流媒体播放。

4.3.3　网络通信

服务器和客户端的通信通过 JSON 格式的信息进行交互，比如列表中课程信息表示如表4.2：

表 4.2　课程属性信息示意图

课程显示名称	REST 路径
职业	type1
人文	type2
历史	type3
科技	type4

要将这样一组信息传输到客户端，首先要进行内容打包。整个列表是一个 list，其中每一项都是多个 key-value 键值组。比如表4.2中所示的信息，就是四个元素的列表，其中每一个都包含了两个 key-value 对。key：课程显示名称，value: 职业；key：REST 路径，value：type1。

JSON 格式最易于表现的就是列表和 key-value 类型的数据。在 JSON 数据中，冒号用来表示对应关系，逗号用来间隔不同的项，中括号用来表示列表。其中的 value 也可能是一个新的 JSON 对象，因此可以实现多层的堆叠。

解析 JSON 格式通常使用各个语言的 JSON 包，比如 JAVA 中的 org.json.* 就是一个简单易用的 JSON 包。使用这个包可以将 JSON 字符串解析成 JSONObject 这样的格式，进行 key 和 value 的读取操作。

但是对于通信较为复杂的程序，对于每一类 JSON 信息都需要单独编写解析程序，这一过程也极为繁琐，且容易出错。因此在这里，作者利用 JAVA 的反射机制，设计了一个自动完成从 JSON 字符串到 JAVA 对象的转换机制。

```
JSONObject json = new JSONObject(jsonstring);
String packageClassName = prefix+className.substring(0, 1).toUpperCase()+
        className.substring(1);
Object obj = Class.forName(packageClassName).newInstance();
JSONObject content;
try{
    content = json.getJSONObject(className);
}
catch(Exception e){}
Iterator<String> it = content.keys();
while(it.hasNext())
{
    String key = it.next();
    Field field = obj.getClass().getDeclaredField(key);
    try{
        field.set(obj, content.get(key));
    }
    catch(IllegalArgumentException e){}

}
return obj;
```

图 4.11　自动转换机制实现

图4.11所示的这段程序是这个自动转换机制的核心部分。

在我们的通讯协议中，key 都是使用 JAVA 类的变量名，value 则是对应的变量值，而类名则是 JSON 对象的最外层的 key。

程序首先根据期望转换为的类的名称，确认是否该 JSON 对象传输的是这个类的信息，然后将内容取出之后进行遍历。由于反射机制的存在，可以在代码中通过运行时的变量名称字符串来访问对应的变量，而不需要预先写在程序中。C 和 C++ 语言则基本无法实现这一功能。

4.4　JAVA 服务端

4.4.1　平台

本系统的 JAVA 服务端使用 tomcat 服务作为宿主服务。编程环境为 MyEclipse，这个软件内嵌了很多服务架构包，能够比较方便编写各种类型的 JAVA 服务器程序。

使用 Java 编写的程序是跨平台的，因此在 windows 下和 Linux 下都能够正常运行。在 windows 中安装 tomcat6.0，然后将 JAVA 编译出来的 class 文件放置到合适位置，就可以对外提供服务了。在 Linux 中也类似，需要安装 tomcat，修改配置文件，然后就能够对外提供 http 的服务了。

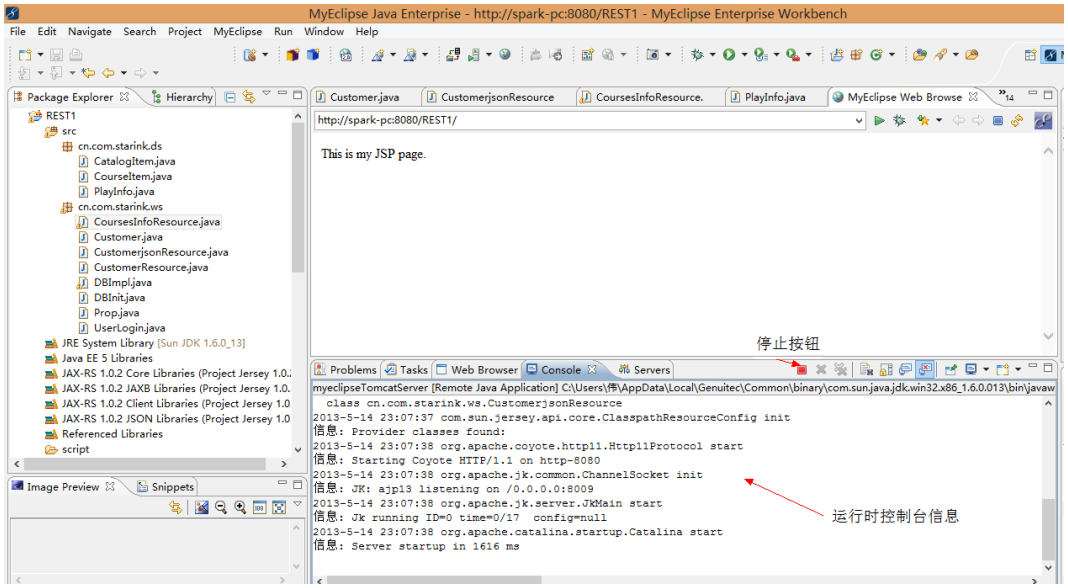

图 4.12　MyEclipse 集成开发平台

MyEclipse 能够在集成界面中直接启动服务，并显示服务器状态和控制台输出信息等，方便调试，如图4.12。而系统运行时，则将编译好的 class 文件和配置文件等，放到 Linux 系统下运行，更加方便管理。

4.4.2　服务架构对比

服务端使用基于 http 的服务，在更高一层的协议的选择上，比较常见的有 SOAP（Simple Object Access Protocol[40]）的 Web Service 和 REST（REpresentational State Transfort[41]）两种方式。

SOAP 已经在 Web Service 的领域被使用了很久，有着非常广泛的应用以及各种各样的工具。SOAP 是一个严格定义的信息交换协议，使用 XML 格式封装数据。它定义了一整套的标签，用来表示各种请求，命令，调用，参数，返回值，错误信息等等。随着功能的不断完善，SOAP 越来越庞大，对于轻量级的应用有些过于复杂。而且基于 SOAP，各个服务都可以扩展自己的 API，每个服务定义的 API 各不相同，因此有了 WSDL (Web Service Description Language)。这是一个使用 XML 编写的，用来描述有哪些服务，使用怎么样的接口，如何发现等等。这也是得 SOAP 的使用更加复杂，尤其是比较简单的程序，相对来说要耗费大量的精

力在完成这些复杂的规则和逻辑上。SOAP 本身依然基于 HTTP 协议，大多数情况下，发送请求使用 POST 方法。

而 REST，直观的描述就是客户端通过申请资源来实现状态的转换。REST 具有以下特性：

1. 客户端服务器结构（CS 架构）；
2. 无状态性，连接无状态；
3. 与 HTTP 紧密结合，可以利用 Cache 机制增进性能；
4. 层次化，系统具有树状的层次化结构；
5. 按需代码，相比于 SOAP，轻量级的应用可以根据需要仅使用非常简单的接口。

从实质上来说，REST 只是一种服务端风格，而不是具体的协议，然而由于其诸多优点，REST 已经开始被广泛应用于各类服务中。

REST 之所以能够如此的轻量级而且强大，得益于他和 HTTP 的紧密结合。在 SOAP 的实现中，它仅仅将 HTTP 作为一个载体，其所有的信息都在 HTTP 包的主体内容中，而 HTTP 本身的包头的开销等都是浪费。而在 REST 中则不同，REST 将 HTTP 的协议充分利用。使用 HTTP 头信息来表明资源的表示形式，使用 HTTP 的错误机制来返回错误信息。这样所有的接口都能够使用同一的 HTTP 本身的机制，而不需要再额外封装。并且可以通过服务端的简单配置来实现一些限制，比如禁止 GET 以外的请求方式来实现资源的只读访问。

REST 的无状态则是对本系统更加重要的一环。REST 将内容都视作资源，直接相应用户的需求，因此如果服务器出现问题，只需要使用其他服务器立即进行替代即可。而且在用户数量激增的时候，可以通过配置反向代理，实现负载均衡，而无状态的服务器在并行提供服务的时候几乎没有损失。

REST 也不是万能的，他是一种资源访问协议，并不能够取代所有的网络请求，比如事务请求等。但是在本系统中，大多数的请求都可以被看做是资源访问，因此非常适合使用 REST 来实现。

本系统的几个 REST 接口示例：

获取一级列表信息：

GET: http://example.com/info/list

直接使用 http 协议访问该地址，就可以获得 JSON 格式的分类信息。

获取二级列表信息：

GET: http://example.com/info/cataloglocation

直接使用 http 协议访问该地址，就可以获得 JSON 格式的某分类下的课程列表信息。

获取视频播放服务器信息：

GET: http://example.com/playinfo/courseid/numberid

直接使用 http 协议访问该地址，就可以获得对应课程的播放地址，包含服务器以及访问路径。

4.4.3　数据库设计

服务端的信息，比如课程列表等，都是保存在 MySQL 的数据中，因此服务端需要随时访问数据库来获取信息，提供给用户。数据库主要分为两部分，一部分为用户信息，一部分为课程信息。用户信息如图4.13。

```
CREATE table UserTable(username varchar(30) not null primary key,
                       email varchar(50), pass varchar(50), sex TINYINT ,
                       age INT, level INT not null, device varchar(30));
CREATE table UserLogin(username varchar(30) not null,
                       mycookie varchar(50) not null primary key,
                       ts TIMESTAMP, validtysecond INT);
CREATE table UserPrivilege(username varchar(30), productid varchar(20) not null,
                       status TINYINT, visittimes INT);
CREATE table Coop( coopID varchar(20) not null primary key,
                  name varchar(50) not null, location varchar(50),
                  number INT not null default 0, des TEXT);
CREATE table LogingLog(id INT not null primary key auto_increment,
                  username varchar(30) not null, success TINYINT not null,
                  time TIMESTAMP not null);
```

图 4.13　用户数据

用户数据中包含用户列表，个人信息，登陆历史，权限管理，个人设置等等信息。

课程信息如图4.14。课程信息包含了所有课程的名称，描述，类别，观看历史等。

```
CREATE table TopCourseTypes(typeid varchar(20) not null primary key,
                        name varchar(30) not null, location varchar(50),
                        number INT not null default 0, des TEXT);
CREATE table Courses(courseid varchar(20) not null primary key,
                        coursename varchar(50) not null, sourceid varchar(20) not null,
                        onlinetime DATE, availability TINYINT not null default 0 ,
                        price FLOAT not null default 0 , teacher varchar(50) , picCode TEXT,
                        classNumber INT not null default 1, des TEXT);
CREATE table CoursesOwn(courseid varchar(20) not null, typeid varchar(20) not null,
                        typelocation varchar(50) );
CREATE table CoursePlay(courseid varchar(20) not null, playid varchar(25) not null primary key,
                        numberid int not null, filepathname varchar(50) not null,
                        pptpathname varchar(50) not null, chatpathname varchar(50) not null);
```

图 4.14　课程信息数据

Java 服务程序使用 ODBC 作为数据库接口来访问数据库，这种方式提供了以后更换数据库软件的可能，在 Java 中，使用 ODBC 非常方便，仅需要调用 ODBC 包就可以完成。

4.4.4　与 elop 接口

Java 服务端需要与 elop 交互的部分主要是为用户查找一台可用的流媒体服务器。

当用户发起建立视频连接的请求时，服务端会向 elop 发起查询请求。在 demo 系统中，由于用户数量不多，采用的是临时文件的方式在服务器和 elop 中交互。Java 服务器在指定目录建立一个任务文件，elop 流程层服务器读取文件内容，并执行该流程，将返回结果写入另一个临时文件，Java 服务器获取后返回给用户。流程如图4.15。

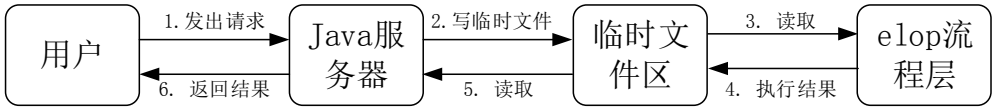

图 4.15　JAVA 服务端与 elop 交互流程

这种方式分离了 Java 服务器和 elop 服务器的逻辑关系，符合高内聚低耦合的系统组织方法，简化了系统逻辑。

4.5　elop 云平台端

4.5.1　元素层设计

elop 流媒体服务器和 elop 存储服务器，都是由 elop 的元素层派生而来，该层的主要任务是监控自身的运行状况，提交信息到逻辑层，接受流程层分配的任务。元素层派生关系如图4.16。

在同一个云中的流媒体服务器，都共享相同的存储服务，视频文件被存储在相同的这个存储服务中。每个存储服务对应着一个 elop 存储服务器，这台服务器能够访问到这个共享存储，可以和某个流媒体服务器共用一个节点，也可以是单独的一个节点。

Host 作为基础的一类元素，提供了检测机器运行状态，上报 CPU，内存使用情况等功能，而作为构建在 Host 基础上上 streaming host，仅需要提供附加的检测组件（比如对于流媒体服务比较重要的网络占用情况）以及流媒体服务监控。elop 流媒体服务打开之后，会自动监测流媒体服务程序是否开启，如未开启则开启。同时每隔一段时间就会上报一次自身的运行状态。

StreamingHost 继承了基类的 CPU，内存占用的检测及上报的方法，同时增加了网络开销的占用检测和上报。当服务器的网络占用率过高时，就不宜再将新的

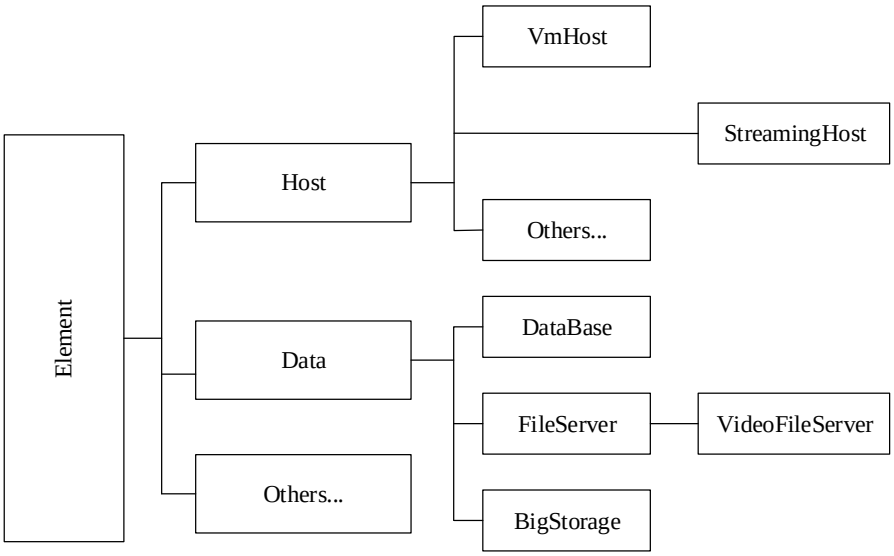

图 4.16　elop 云平台端系统元素层继承关系

任务放置在该服务器上，而应该考虑将该服务器上的任务迁移一部分到负载更低的服务器上。提供这些决策支持的数据都是由 StreamingHost 的方法实现的。

FileServer 元素的主要功能则是监控存储内容以标签，并上传到逻辑层服务器。每隔一段时间，FileServer 会扫描一遍工作目录，这个工作目录是这个云的集中存储映射到的目录的子目录。FileServer 扫描这个目录以后，会得到文件列表。该列表包含文件名，大小等信息，被组织成元素层动态信息的形式，然后使用上传动态信息的方式传输到逻辑层。

逻辑层在接收到这些信息之后，会将其存储到数据库中，而如何设计主键是一个比较重要的过程。在每一台存储单元中，文件都有一个唯一的标识，比如路径加文件名。而在不同的元素中，这些信息很可能是有重复的，因此在这里，系统使用了元素层的证书名（每台服务器都有唯一不重复的证书名）和路径名相结合的方式来存储，这样就确保了在逻辑层存储的时候，每一个文件都有唯一不重复的 key，而且可以很容易的通过元素的证书名以及所需要的文件名构造出 key，从而大大提高检索效率。

VideoFileServer 除了继承了该功能之外，还增加了对文件的标签处理的功能。因为对于视频文件，文件的说明不容易通过文件的内容获取，而文件名所能包含的信息量太小，因此需要一些额外的说明和属性标记。在 VideoFileServer 的系统中，增加了 tag 文件的信息处理方式，对于每一个需要处理的文件，都需要建立一个在文件名之后增加.tag 的后缀的新文件。比如 video1.mp4 的文件，其 tag 文件就是 video1.mp4.tag。这个文件采用文本格式，其中记录了文件的简要说明，这些信息都会一起被上传到逻辑层服务器中，以供检索或查看。

FileServer 还具有在不同服务器中传输文件的功能。当需要将一个存储服务中的文件复制到另一个的时候，只需要通过逻辑层发送一个命令，FileServer 之间会自动进行协商并传输文件。

FileServer 接收到传输文件的指令后，首先解析命令，获取需要复制的文件信息，要复制到的服务器信息等。然后通过证书认证的方式与目标服务器建立连接，然后启动传输文件的任务，目标服务器接收文件之后，将其存储在相应的目录下。此时，系统中有了多个存储节点，每一个代表一个云存储服务，而通常在每一个云中，都包含了多个流媒体服务器，流媒体服务器的数量可以通过云平台的弹性资源分配动态的添加删除。

4.5.2　逻辑层设计

在本系统中，逻辑层的主要任务是存储元素层的信息，以及分配和调度可用的元素。在系统中有 StreamingScheduler 和 FileServerLogic 两个模块。

StreamingScheduler 继承实现了 schedule 的方法，用来对所有的 StreamingHost 进行选择和分配。StreamingScheduler 接受流程层的任务，其中最主要的任务就是挑选合适的 StreamingHost 给用户提供服务。在该服务中，会循环遍历所有的元素，并且考察他们的 CPU，内存，网络占用情况，并进行排序。对于占用率较高的元素，将不会选择他们作为新的用户的服务器。

StreamingScheduler 一直维护着一个可用的 StreamingHost 列表。当新的 Host 启动上报自身的信息时，会将该 host 加入到本列表中，当长时间没有与某个 Host 通信成功，会将该 Host 从列表中删除。

FileServerLogic 的任务主要是存储各个集群中的文件信息。由于不一定每个视频文件都存储在所有的服务器上，因此当用户发来请求时，还需要先通过 FileServerLogic 存储的信息来确认所需要的文件存储在哪些服务器上，然后再通过 StreamingScheduler 从这些服务器中进行选择。

4.5.3　流程层设计

该系统的流程层主要是作为与 Java 服务器的接口。流程层接收 Java 服务器发来的任务，任务中包含用户所请求的视频文件，用户本身属性等，然后流程层建立一个任务，与 StreamScheduer 和 FileServerLogic 交互，来获取一个包含用户所需要视频文件，并且适合为该用户提供服务的服务器地址，然后将该地址返回给用户，提供视频服务。

流程层的工作方式如下：

1. 启动服务程序后，建立与逻辑层服务器的连接，确认 StreamingScheduler 和 FileServerLogic 工作状况正常。
2. 扫描指定的临时文件存放目录，如果有任务文件存在，读取第一个。
3. 读入任务中的信息，包括用户所需要的文件，用户地址等。
4. 与 FileServerLogic 通讯，将文件信息发送给 Logic 层，要求取回包含该文件的所有服务器。
5. 如果没有找到包含该文件的服务器，则任务失败，将结果按一定格式写入一个结果临时文件。删除原任务文件，回到第 2 步循环执行。
6. 将所取得的服务器列表信息和用户地址等信息，一起发送给 StreamingScheduler，要求查找一个最合适的服务器。
7. 将返回的服务器信息写入一个临时文件，删除原任务文件。
8. 回到第 2 步，循环执行。

逻辑层是一个服务程序，启动之后就一直循环执行，直到被结束执行。因此在本系统中，该流程层需一直开启，不断处理 JAVA 服务端发来的请求。

在扩展应用的时候，流程层自身的代码基本不需要修改或继承，大部分情况下只需要将具体的任务文件传入流程层，即可顺利完成请求。但在某些特殊的情况下，任务的中间信息过于复杂，不是简单的顺序执行几个任务就可以的，就需要继承扩展流程层的代码，来实现具体的需求。

第 5 章　多云环境下的虚拟机配置算法

5.1　背景条件

本文的算法重点关注存在多个数据中心，在每个数据中心都拥有弹性的计算资源的情况。云计算是未来网络发展的趋势，越来越多的中小服务商以及个人都会将自己的计算环境甚至网络环境架构于云平台服务商的设施之上。而单一的云平台提供商也存在一定的风险，包括网络问题，数据可靠性问题，价格问题上面的风险，一个成熟的大规模的服务，不能够依赖于单一的云平台提供商，本文的出发点就是在多云平台，多数据中心环境下，如何能够提高服务质量，降低成本。

我们的服务架构于云平台之上，因此为用户提供服务的都是虚拟机，当用户数增多时，系统会申请更多的虚拟机提供服务。由于数据中心内部数据传输速度很快，因此每个数据中心中，系统会部署一套数据存储单元，该数据中心的虚拟机都可以共享这些数据，也就是可以提供这部分内容的流媒体访问。

当用户距离服务器的网络距离比较远的时候，比如一个在北京联通网络中，一个在深圳电信网络中，数据到用户端的延时和使用体验都会降低，同时由于丢包等增多，给服务器带来的压力也会增大。因此在用户密集的区域架设新的服务器会改善这一情况，但在新的位置新的数据中心开启服务，需要将原有的数据移动一部分过来，而且在数据中心租用存储空间也需要成本，所以本文的算法主要将解决这一问题。

为简化起见，将用户和服务器的位置放置在二维地图上，虽然实际的网络开销并不是与地理距离成正比。该位置已经综合考虑了不同网络之间的瓶颈，主干网与分支网等，而不是简单的地理位置的映射。通常大城市内，人口密集，用户数量也较多，而其他位置的用户可以认为随机的排布在地图上。而具体这张地图该如何对应实际用户，因为需要更多的数据支持，不是本文讨论的内容。本文的算法基于一个已经设计好的地图。

假设一个用户与服务器之间的距离为 x，则该名用户对服务器的资源消耗（包含 CPU，内存，带宽，以下统一称为服务器资源）则为 f(x)。函数 f 通常来说是一个增函数，因为用户距离增加后，丢包率等都会上升，整体上增加了服务器的负载。每条连接的服务质量与每台服务器的已消耗资源和总资源之比和用户与服务器的距离有关，如果已消耗资源占总资源比（t）大于某阈值（比如 80%），服务质量会降低。如果距离 x 大于某阈值，服务质量也会降低。总的服务质量不能过低，会影响应用的用户体验。

假设在全国各地有多个可用的云计算平台, 系统可以根据需要开启某个位置的云平台的服务。由于云计算是按需收费，如果不启用服务，则没有费用。每新开一个云平台上的服务，都需要增加额外的存储费用，而在同一个云平台上增加服务器，只需要增加虚拟机的费用。

作者在这些假设的基础上，设计了针对多云模型的负载均衡算法，目标为在用户数量增加的时候，能够尽可能的保证服务质量并降低成本，在用户数量减少的情况下能够减少成本消耗和浪费。

5.2　系统模型

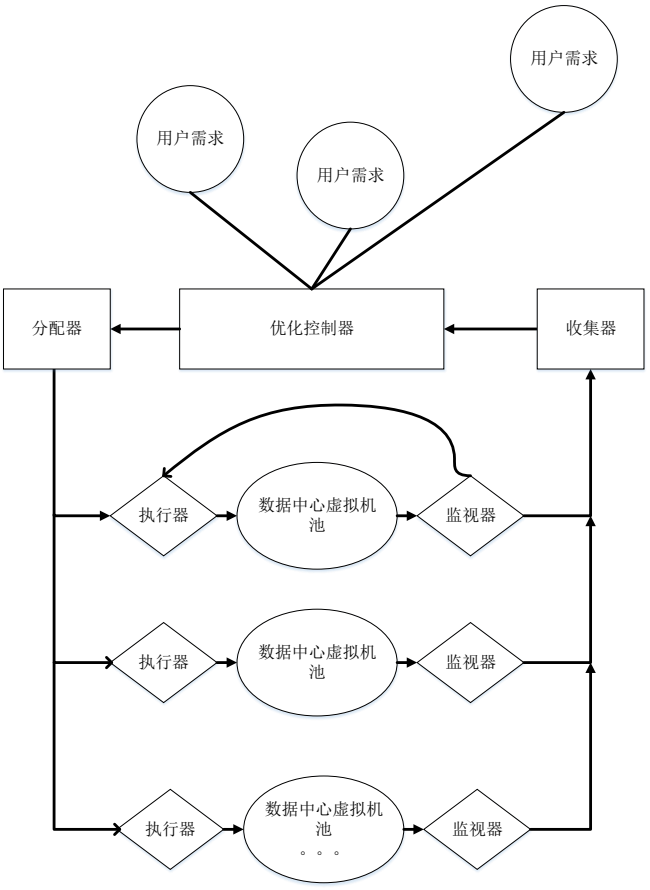

图 5.1　基于多云环境的服务优化模型

图5.1表示了该系统的模型。在每一个可用的云计算平台数据中心，都有一个监视器和一个控制器，可以负责监视所使用的虚拟机的状态，和控制虚拟机的开启关闭等。监视器手机虚拟机的资源占用率信息，并提交给优化器以及本地执行器。当同一个数据中心的虚拟机占用率不均衡时，可以在本地将连接到资源占用率较高的虚拟机的连接重置到资源占用率较低的虚拟机上。因此在同一个数据中

心内部，资源占用率可以基本保持均衡。在总体资源占用率不超过一定限制的情况下，可以保证里面的每一个的每一个虚拟机都在比较健康的状况中。优化控制器手机每个数据中心虚拟机平均的资源占用率，依次来决定是否应该新增或减少虚拟机的数目。

通过这样的控制系统，配合接下来的算法，可以让整个系统中的服务都保持一个较高的 QoS，并且降低总体的费用。

5.3　算法设计

本系统并不是在一个云平台都存储着所有的资源，通常资源会放置在初始位置上之后，按需复制到需求量比较大的云平台中。每个云平台中虚拟机的个数也是变动的，虚拟机的个数决定了这个云平台中虚拟机部分的费用。

因此，该系统能够控制的可变量为：

1. 是否开启或开启哪个云平台上的服务。
2. 每个云平台开启多少服务器。
3. 每个用户被分配到哪个服务器。

而该系统的约束条件包括：

1. 云平台总的数目以及位置，云平台都是预先建立好的，通常在大城市或电力，人力成本较低的地区，而不是随心所欲的可以放置在任何位置。
2. 总的服务质量必须在可接受的范围内。

该系统的优化目标为：总的费用最小，服务质量尽可能高。在服务质量非常低的时候, 会加重 QoS 的扣分权重, 在服务质量维持在很高水平的时候, 将重点参考服务费用, 而降低服务质量的权重, 在中间段区域, 会同时考虑这两个因素。

服务质量 (QoS) 将在 0 到 1 之间浮动，费用 (cost) 则为正数。

评价指标:

$$f(cost, QoS) = \begin{cases} \frac{0.7cost}{QoS^2}, & QoS \in (0, 0.7) \\ \frac{cost}{QoS}, & QoS \in [0.7, 0.9] \\ \frac{\sqrt{0.9}cost}{\sqrt{QoS}}, & QoS \in (0.9, 1] \end{cases} \tag{5-1}$$

该问题需要在运行过程中不断进行优化，而且由于问题的模糊性，这里作者并没有用单纯的优化问题的思路来解决，而是采用一定的经验性的处理方式，辅以仿真实验做参考。

该算法的思路可以描述如下：

1. 在当前状态下，确认总的服务质量是否接近阈值，是否具有可提升空间。

2. 如果服务质量有待提升，有两种选择，可以在已有的云平台上增加虚拟机，也可以在用户密集区域附近的云平台开始提供服务。分别计算出这两种方式的增加的费用和能够提高的服务质量，来确认如何选择。此处选择基于前面给出的评价指标。
3. 如果服务质量处于比较高的水平，确认是否有降低费用的空间。
4. 对于第二点中的增加虚拟机的方式，可以在根据新增用户所处的位置，为其选择合适的服务器位置，然后计算该云平台是否需要增加虚拟机，并计算出费用和服务质量。
5. 对于开设新的云平台的方式，由于可用云平台个数有限, 这里采用遍历的方式找出最优化的位置。逐个测试开启某个新的云平台后是否能够提升评价指标。找出最优的选择，返回新启用的云平台的位置和新的评价指标。
6. 在需要提高服务质量的时候，通过上面两种方式计算完成之后，根据评价指标的高低做出相应的选择。在需要降低成本的时候，选择资源占用率最小的云平台，计算减少虚拟机所能够减少的成本，以及对服务质量的影响。

每进行一次检测，系统会自动根据最优的策略进行用户连接的调整以及数据的传输。由于云计算的弹性能力，能够随时跟踪用户数量的变化，在保证服务质量的情况下降低运营成本。通过仿真结果，我们将看到该系统如何不断的在运行中优化资源配置，以达到成本和服务质量的平衡。

5.4　效果分析

5.4.1　数据设定

从算法的设计可以发现，本系统的算法设计重点考虑的就近服务大规模的用户的情况，具体效果如何，将通过仿真实验进行分析。

根据背景一节中的假设，用户数目一部分主要集中于大城市周边，也就是几个汇聚点，其他的可能分散在各个地方。我们首先看一下仿真环境下用户分布的情况。

图5.2表示了用户数目从 100 到 30000 的变化，地图被简化为了二维的 10000*10000 的点阵，每个红点表示有一个用户在那里。红点密集的区域表示用户数目多。需要说明的是，虽然通过仿真简化了整个的模型，随机产生的用户位置也过于固定，但是本系统的算法不依赖于用户实际的具体分布，这里使用这种模型来进行仿真是可行的。

从图上可以直观的看出，在地图上有一些人口比较密集的区域，而其他区域分布则较为稀疏，以此来模拟实际环境下全国用户的分布情况。

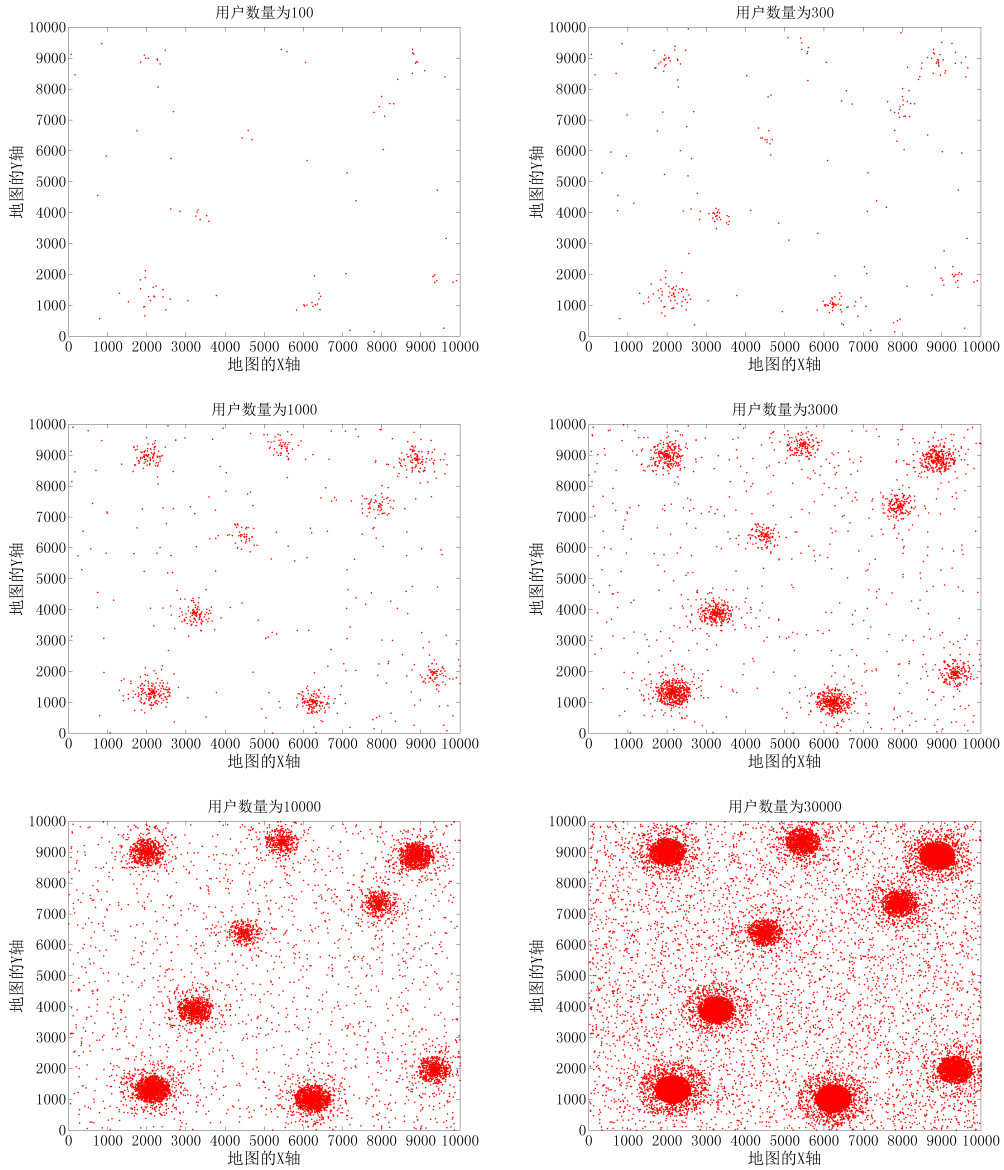

图 5.2　用于数量从 100 到 30000 增长时的分布情况

实验环境下，一共有 18 个位置的云平台可供调用，可用的云平台的位置如图5.3。

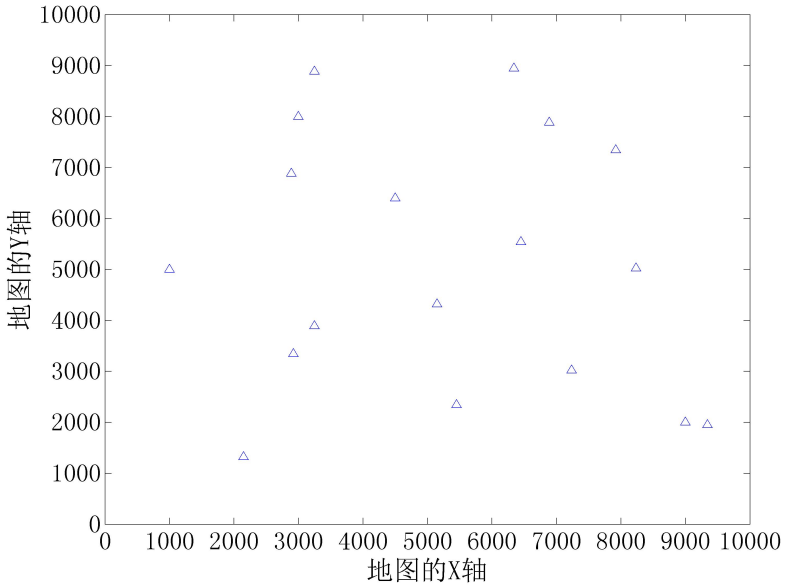

图 5.3　所有可用云平台位置示意图

5.4.2　在多云中动态配置和使用单一云平台的对比

开始时刻随机挑选一个云平台开始提供服务，随后跟随着用户数量增长，可以在已有的云平台上扩展资源提供更好的服务，也可以新增云平台提供服务。

先测试用户一直增长的时候，系统如何进行扩展来保证服务质量。下图为用户数增长的过程中，平均服务质量 (QoS) 的变化曲线如图5.4：

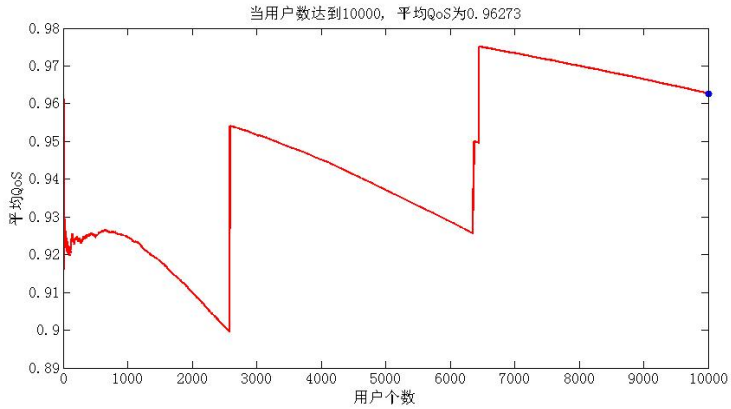

图 5.4　随用户数增长，QoS 的变化曲线

曲线出现了多次转折点，是由于新增云平台之后，用户连接被重置，更多的用户可以连接到距离较近，时延较小的服务器中，服务质量会有较大提升。整个

过程中最低的服务质量为 0.9。

系统的总费用随着用户数量增长肯定是不断增长的，因此我们主要考察每用户的平均费用，也就是总费用除以用户数量。当用户规模较小的时候，平均成本过高，因此我们从 200 用户之后才开始统计。

图5.5为平均每用户服务成本的变化曲线。

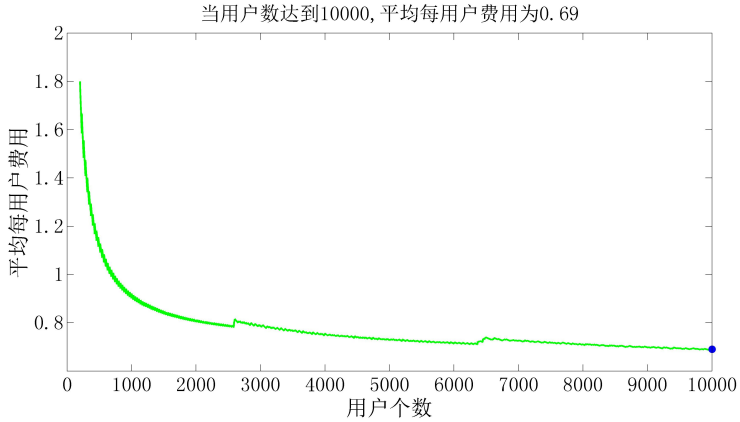

图 5.5　随用户数增长，每用户费用的变化曲线

平均费用基本趋势是递减的，达到 10000 用户时每用户平均费用 0.6901。下降过程中也有很多波动，主要是新开虚拟机以及新启用云平台之后的成本提升。最终，在达到 10000 用户时，启用了 4 个云平台，虚拟机数目分别为 73, 66, 68, 98。

启动的云平台如图5.6。

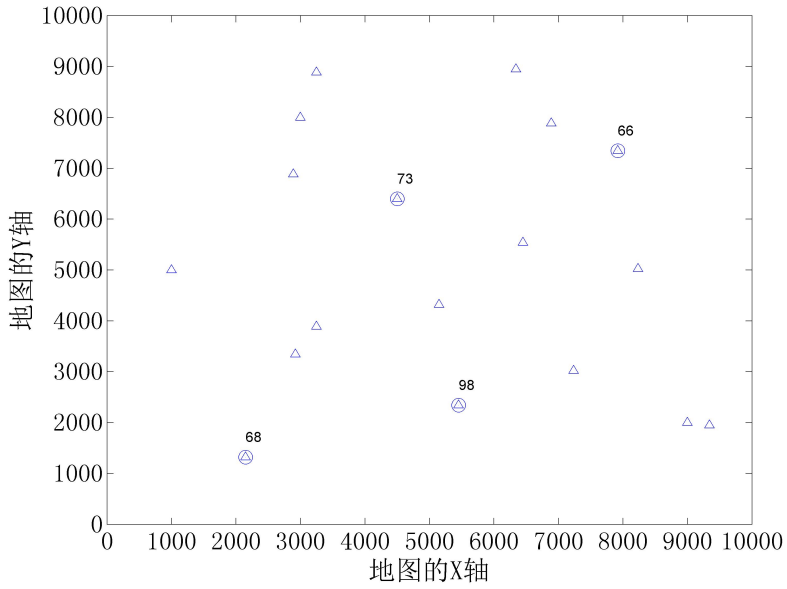

图 5.6　用户数达到 10000 时，启用的云平台状况

与不使用我们的算法的方式进行对比，如果只使用最初的一个云平台进行服务，服务质量的对比曲线表现如图5.7：

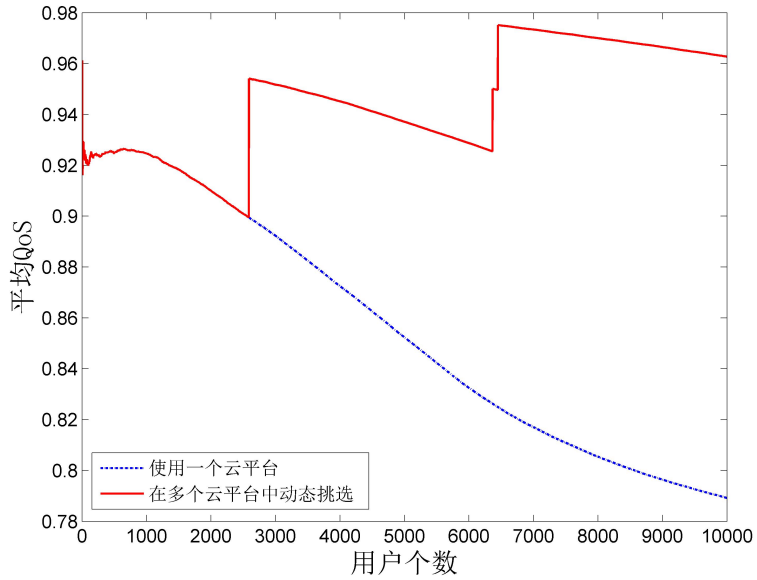

图 5.7　使用多云动态配置和单一云平台，QoS 变化曲线的对比

对比可以发现，在多个云平台中动态进行配置，QoS 的表现要远远好于只使用一个数据中心。这个结果非常合理，因为只使用一个数据中心，用户与数据中心的平均距离较远，导致 QoS 降低。而我们更感兴趣的是，平均每用户费用对比如何？

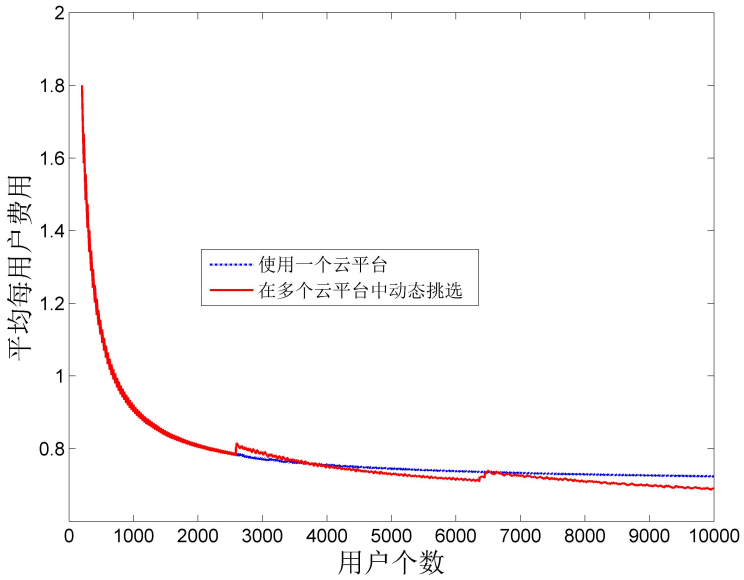

图 5.8　使用多云动态配置和单一云平台，每用户费用变化曲线的对比

图5.8表示了每用户费用的变化曲线。对比可以发现，使用我们的算法来平衡的开启云平台进行服务之后，虽然每新开一个云平台的服务会增加不小的存储成本，但是最终的每用户成本仍然降低了，并且服务质量有了质的提升。

5.4.3　引入价格系数之后的实验结果

在实际情况中，不同地点不同服务商的云计算平台的价格并不相同，处于人口密度较小或电力成本较低的地区，云计算的成本也会降低。因此，我们的算法在进行选择的时候，可以引入成本系数来优化选择。对实验环境下的 18 个云平台，分别给予一个不同的系数，从 0.8 到 2 随机生成。实际的虚拟机及存储费用都需要乘以这个系数。在有了这个系数之后，系统并不会总是选择最近的可用云平台来连接新的用户，而是会综合考虑费用问题。当然不同地点的云计算价格差别并不大。相对来说人口密集区域的成本会偏高，但是我们的算法并不依赖具体的成本规律，只是根据数值进行优化计算。在每一个用户选择连接哪个云平台时，不仅会考察距离，也会考虑该平台的使用成本，依此做出选择。图5.9和图5.10是引入了价格因素的服务质量曲线和费用变化曲线。

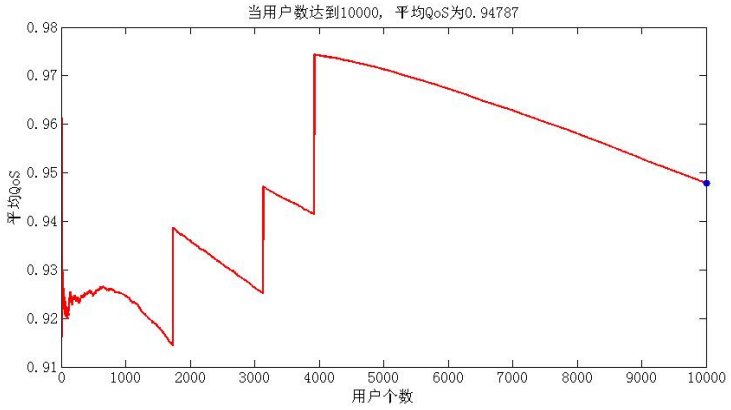

图 5.9　引入价格系数后，QoS 变化情况

结果说明该算法在引入价格差别之后，仍然能够提供优化的选择，在成本控制方面可以做得更好。由于不同云平台的价格产生了变化，因此数据中心的启用和虚拟机分布情况也有了变化，如图5.11。

5.4.4　变化存储费用的实验结果

使用更多的参数进行测试，会发现更多有意义的结果。在上面的仿真实验中，我们设置的每个数据中心固定的存储成本为一台虚拟机成本的 10 倍。这并不总是符合实际情况，如果系统中的视频数据量较小，或者说用户所需要的视频更加集

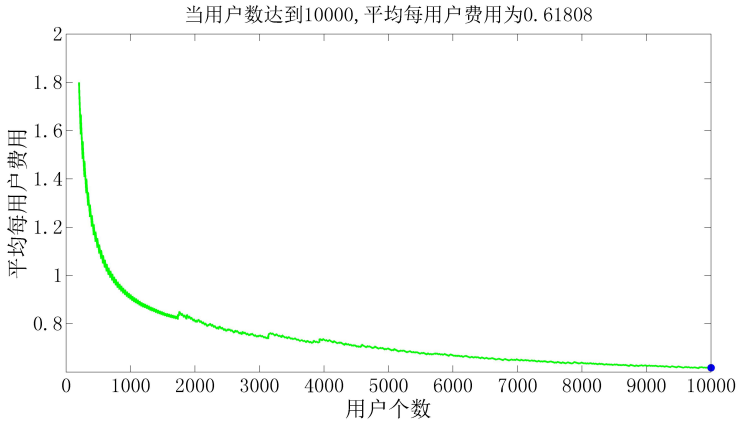

图 5.10　引入价格系数后，每用户费用变化情况

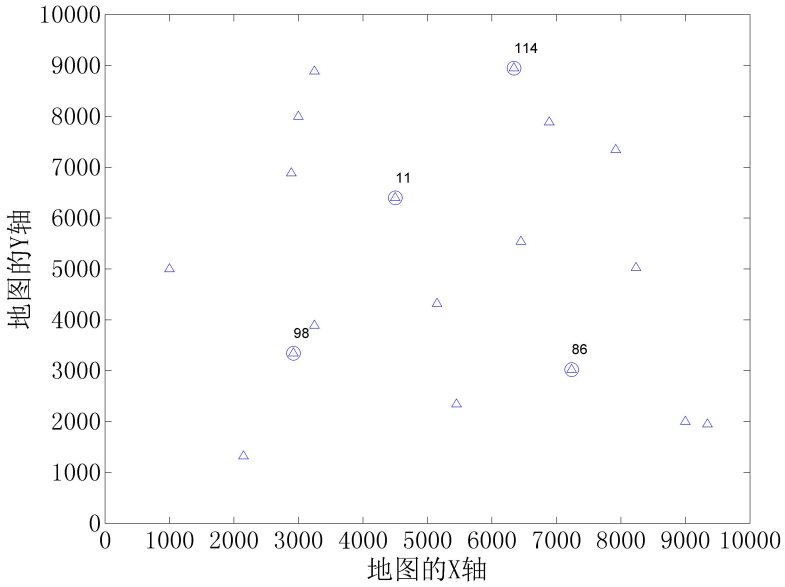

图 5.11　引入价格系数后，最终数据中心使用情况

中，则所需要的存储空间会减少，存储费用会降低。如果视频文件总量增加，用户所需要的视频更加分散，所需要的存储空间会增加，存储费用会增加。

下面我们测试并分析一下在存储费用和每个虚拟机费用比例不同的情况下，算法表现如何。

1. 测试存储费用增加的情况。将存储费用设置为单个虚拟机费用的 25 倍，并且不使用价格系数，即所有数据中心的价格一直。在这种情况下，服务质量和费用的变化曲线参加图5.12和图5.13。

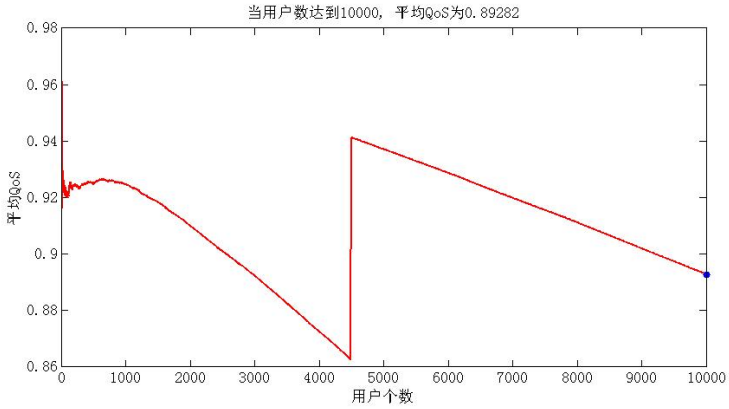

图 5.12　增加存储费用后，QoS 变化情况

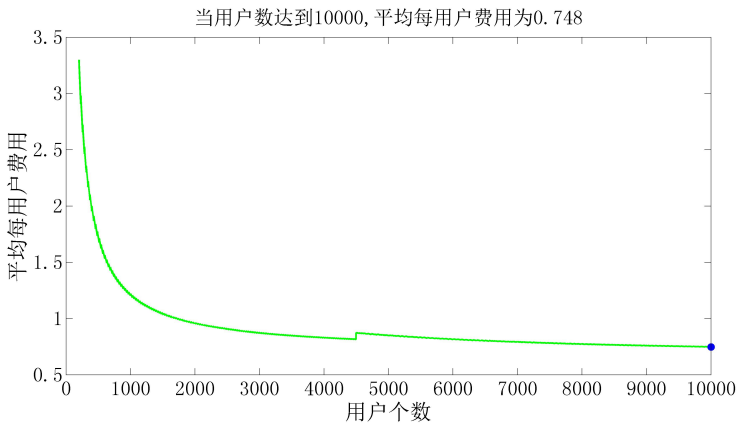

图 5.13　增加存储费用后，每用户费用变化情况

在这种情况下，系统更倾向于减少使用新的数据中心，因为新增数据中心所带来的存储成本增加相比于上一个测试更大。使用的数据中心情况如图5.14。可以发现，系统最终只使用了两个数据中心，少于存储成本更低的时候。因此，对于数据量非常大的服务来说，在没有大量用户的情况下，使用多个数据中心通常是不合算的。但是使用更少的数据中心的问题是服务质量会难以保证，比如本次实验中，在用户数达到 10000 的时候，QoS 的值为 0.893，远远小于存储成本较低的时候的 0.96。

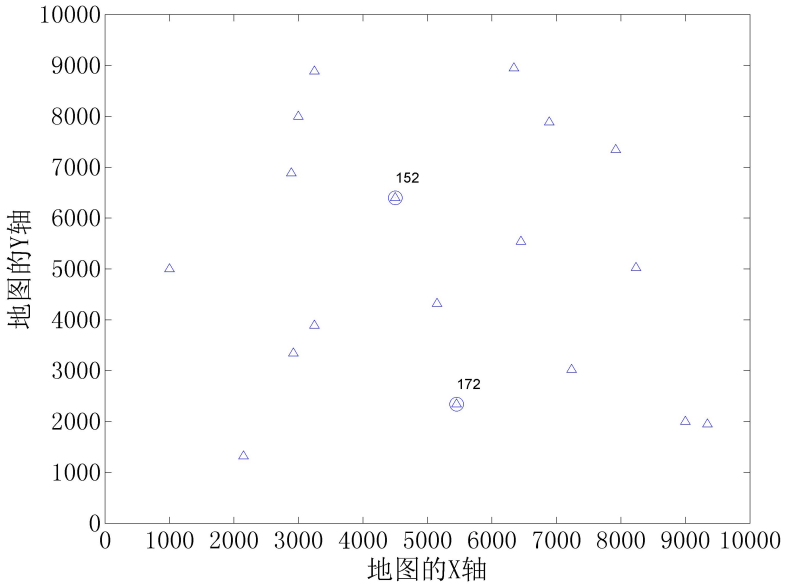

图 5.14　增加存储费用后，最终使用的数据中心情况

2. 保持存储费用为虚拟机费用的 25 倍，引入上一届相同的价格系数。同样让用户数从 0 增长到 10000 进行试验。

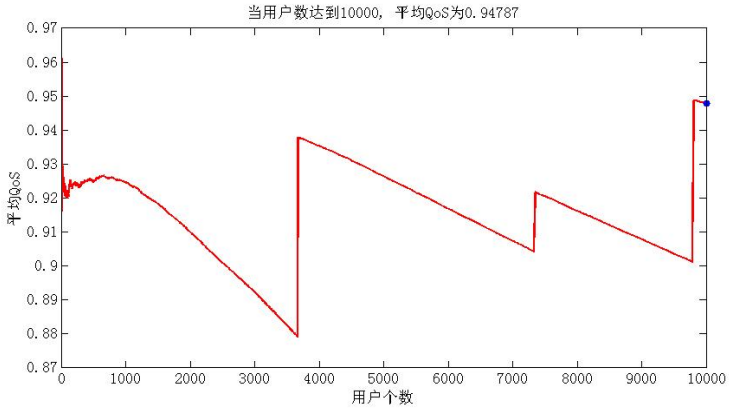

图 5.15　增加存储费用，且引入价格系数后，QoS 变化情况

服务质量和费用的变化曲线如图5.15和图5.16。QoS 变化曲线中的每一个明显转折点，都是一次重新分配用户连接的过程。由于有了一些价格较低的数据中心，在选择时可以适当抵消一部分存储费用增长带来的问题，因此相比于上一组数据，在引入价格系数时，系统会更加积极的启用新的数据中心。在用户数量达到 10000 后，启用的数据中心情况如图5.17。结果证明确实在这种情况下，所使用的虚拟机比没有价格差异的时候更多。

3. 测试存储费用降低的情况，设置存储费用仅为虚拟机费用的 2 倍，测试用户数量增长后的变化。可以预见到，在存储费用较低的情况下，系统会更加积

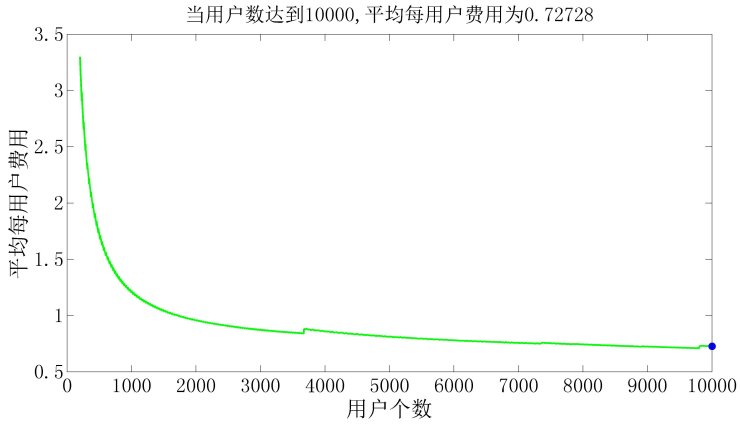

图 5.16　增加存储费用，且引入价格系数后，每用户费用变化情况

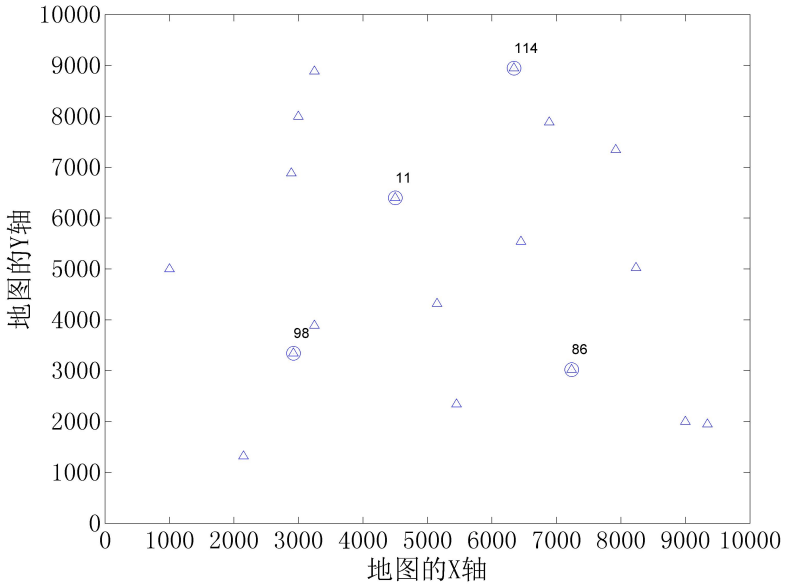

图 5.17　增加存储费用，且引入价格系数后，最终使用的数据中心情况

极的使用更多的数据中心，这样用户会有更多的机会连接更近的服务器，因此 QoS 的整体表现会更好。实际情况如图5.18和图5.19。

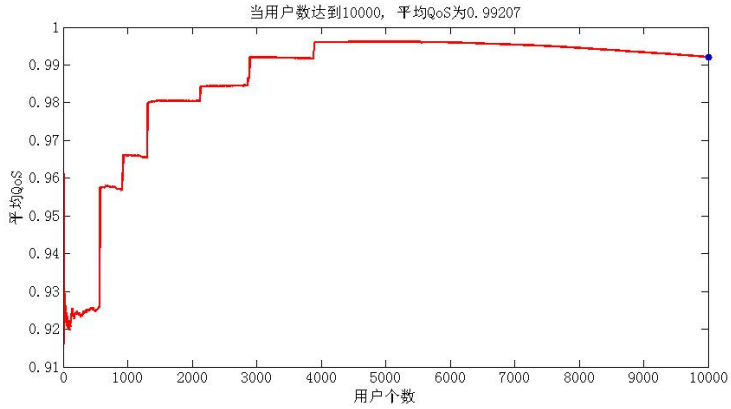

图 5.18　降低存储费用后，QoS 变化情况

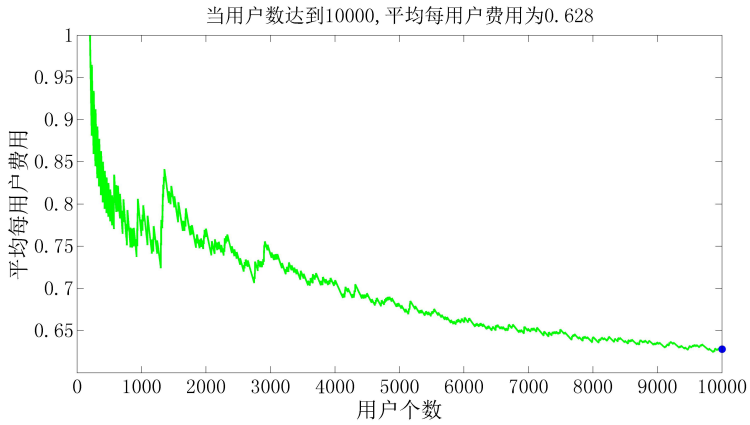

图 5.19　降低存储费用后，每用户费用变化情况

通过 QoS 的变化情况可以发现，系统进行了多次的重新分配，几乎启用了所有的数据中心。而平均费用的波动也相对更加剧烈，因为频繁的启用新的数据中心导致的成本变化更加频繁。最终的数据也验证了这一点，如图5.20。

4. 在保持存储费用是虚拟机费用两倍的前提下，引入价格系数，测试用户数量增长时的表现。

   图5.21和图5.22分别是服务质量和成本的变化曲线。图5.23是最终的数据中心使用情况。

   分析结果可以发现，相比于使用不同的数据中心不存在价格差异的情况，在这一次测试中，所启用的虚拟机要更少。这是因为在存在价格差异的时候，系统会在 QoS 没有较大变化的情况下优先选择价格更便宜的数据中心。因此很多价格较高的数据中心没有被利用起来。代价就是这种情况下，最终在 10000 用户时的 QoS 只有 0.962，小于没有价格差异情况下的 0.99。不过这

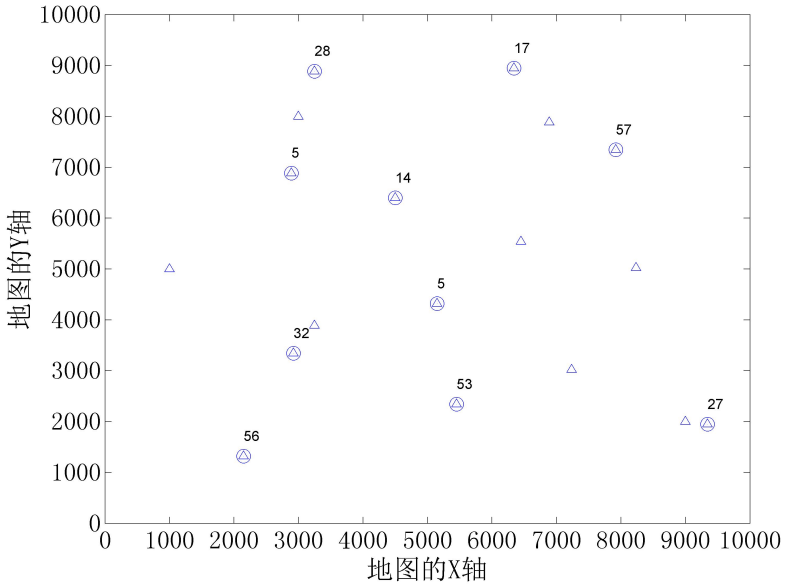

图 5.20　降低存储费用后，最终使用的数据中心情况

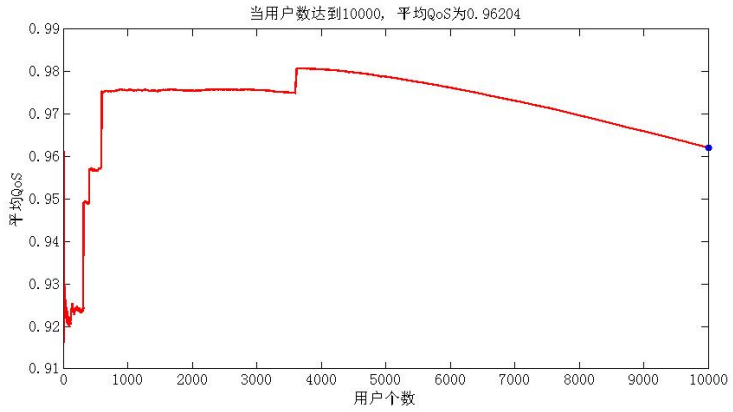

图 5.21　降低存储费用且引入价格系数后，QoS 变化情况

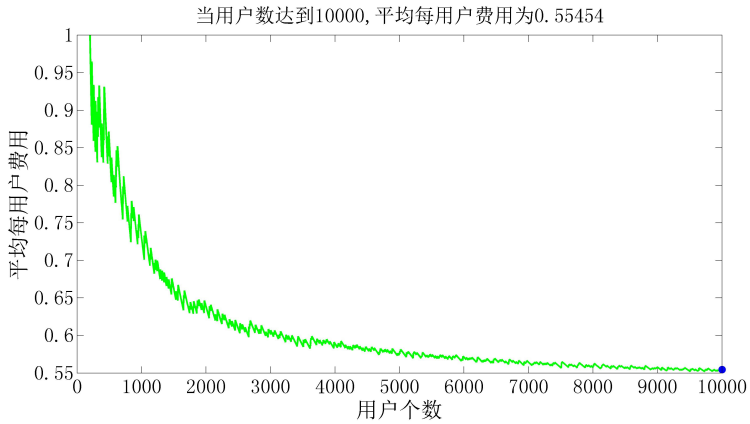

图 5.22　降低存储费用且引入价格系数后，每用户费用变化情况

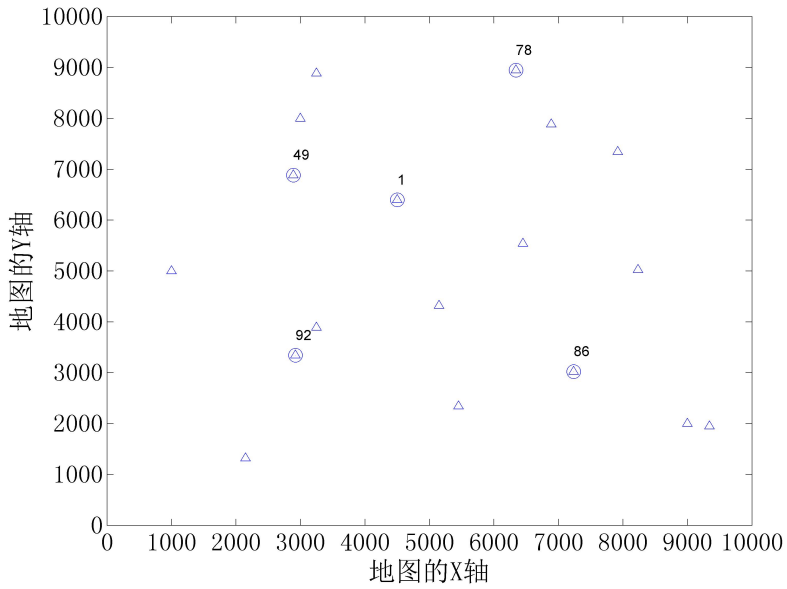

图 5.23　降低存储费用且引入价格系数后，最终使用的数据中心情况

本身也是符合我们最开始的假定，就是 QoS 比较高（高于 0.9) 的情况下，不过分关注 QoS 的微小差异。

5.5　综合分析与结论

在本章中，作者提出了一种多云环境下动态配置使用不同云计算平台的算法。在多云环境中使用假设视频流媒体服务，能够降低初建时的费用，能够在用户数量波动的情况下更好的提供优质服务。而本文所关注的，是优化对用户的服务质量，同时降低成本。

多组仿真实验证明，该算法能够很好的提高服务质量，降低服务成本。与仅仅使用一个数据中心对比，使用多个数据中心的服务，能够大大提升平均服务质量，并且成本上也略低于前者。

在不同数据中心的价格有一定差异的情况下，该算法更为有效，能够在保证服务质量的前提下，选择价格更低的数据中心，从而降低每用户的平均成本。当作者调整存储成本和虚拟机成本的比例时，结果更加有趣，也为搭建视频服务提供了一定的参考：

1. 当服务存储成本提高的时候，系统更倾向于使用较少的数据中心，因为启用新的数据中心的固定成本提高了。同理，在存储成本降低时候，系统更倾向于使用较多的数据中心。存储成本通常是和存储的数据量正相关，因此如果一个视频服务中，有大量的视频文件需要保存，则使用较少的数据中心更为合适，而对于那些用户观看视频更为集中，数据量相比于观看人数更少的视

频服务来讲，使用分布在各个地区的多个数据中心更为合适。

2. 在存储成本提高的时候，引入价格差异，会促使系统启用更多的数据中心，最终 QoS 得到了提高，且每用户成本降低了。具体的数据参见图5.12，图5.13，图5.15，图5.16。也就是说，在存储成本较高时，如果存在一些价格相对较低，但相对偏远的数据中心，能够在服务质量和每用户成本两方面都有所提升。
3. 与上面不同的是，在存储成本很低的情况下，引入价格差异后，虽然每用户成本降低了，但 QoS 的表现出现了一定的下滑。这是由于在存储成本较低时，本身就更容易建立起较多的数据中心提供服务，因此 QoS 的水平整体较高，因此在评价指标中，QoS 的权重下降，因此更关注于降低成本，而损失了一定的服务质量，但最终的服务质量依然在预期的 0.9 以上。针对具体的应用，管理者应该具体考察其服务质量的重要性，设定好评价指标函数，才能够避免过于关注成本因素造成的不符合预期的服务质量下滑。

第 6 章　总结与展望

6.1　论文工作总结

本课题主要研究了教育行业垂直云平台的架构与设计。教育行业垂直云平台不同于传统的 IaaS，PaaS，SaaS 的划分方式，而是从功能和服务对象上针对教育行业用户开发，包含了 IaaS 和 SaaS 的部分轻量级功能，重点设计了视频流媒体服务模块，开发了从云到移动端的整套解决方案。该系统云端基于云计算中间件 elop 软件包。elop 中包括元素层，逻辑层，组织层和流程层四层，在这四层的基础上可以继续进行二次开发扩展该云平台功能。

同时，作者研究了在多云环境下动态配置虚拟机个数的方法。目前云计算正处于快速发展时期，越来越多的云计算服务商开始出现，国内的盛大，阿里等互联网厂商纷纷进入这一行业。要将视频服务部署在云端时，面对如此多的服务商和不同地点的数据中心，本课题提出了一个配置方案。根据用户所在位置估计用户距数据中心的距离，从而估计出该服务的延时和丢包率对服务质量的影响，以及该连接所消耗的服务器资源。在用户数量增加的情况下，系统尝试计算各种情况的服务质量和服务成本，然后根据预设的评价指标选择一个最好的方案进行配置。

通过仿真实验结果可以看出，动态的在多个云平台数据中心中选择服务器，比只使用一个服务器的效果好得多。当每一个数据中心的价格不同时，该算法也能够很好的选择价格较低且位置合理的数据中心，从而在保证服务质量的同时降低成本。服务的主要成本包括存储成本和虚拟机成本。存储成本对于每一个启用的虚拟机来说是固定成本，虚拟机成本是根据用户数量动态变化的。当存储成本与单个虚拟机成本的比例升高时，系统会倾向于使用较少的数据中心，当该比例降低时，系统会倾向于使用更多的数据中心提供服务。

同时，在存储成本上升时，不同云计算平台间的价格差异有利于系统选择更多的数据中心提供服务，从而提高 QoS，而在存储成本降低时，该价格差异则会阻碍系统选择较多的数据中心，但成本会控制的更低，管理者需要根据实际情况设置评价指标，来保证服务质量。

作者的主要贡献是实现了一个可用的教育云平台系统示例，为 elop 中间件做了一个二次开发的示例。对多云环境下虚拟机的配置算法的研究，也是具有创新性的。完全基于云计算环境下的视频流媒体服务已经具有可行性，而在不同的云计算提供商的平台之下动态进行配置，也是降低成本，保证服务可用性的一条重

要途径。而多云环境下的类似问题还没有被广泛研究，本课题在这个方向上也做出了一定的贡献。

6.2　工作展望

在网络技术飞速发展的今天，教育行业相对来说变革较慢，传统的 IT 架构对于教育行业的辅助有限，成本偏高且效果一般。而云计算的出现对于教育行业的现代化是一个重要契机。下一步，该教育云平台架构还需要继续丰富功能，添加更多交互功能，制作教学开发工具等。

elop 软件包在云计算领域具有很大的潜力，在未来，还需要继续丰富其包含的功能，增强可用性和快速迭代的能力，在云计算发展突飞猛进的今天占据一席之地。

多云环境是未来发展的一大趋势，在多个云服务中动态的进行选择是一个降低成本，提高服务可用性，降低风险的一项重要方案。本课题已经对视频流媒体服务在多云环境下的部署进行了一定的研究，在未来还需要进行更多服务种类，不同情况下的研究及实验。

参考文献

[1] Bhardwaj S, Jain L, Jain S. Cloud computing: A study of infrastructure as a service (IaaS). International Journal of engineering and information Technology, 2010, 2(1):60–63.

[2] Vaquero L M, Rodero-Merino L, Caceres J, et al. A break in the clouds: towards a cloud definition. ACM SIGCOMM Computer Communication Review, 2008, 39(1):50–55.

[3] Cusumano M. Cloud computing and SaaS as new computing platforms. Communications of the ACM, 2010, 53(4):27–29.

[4] La H J, Kim S D. A systematic process for developing high quality saas cloud services. Proceedings of Cloud Computing. Springer, 2009: 278–289.

[5] 物联网行业试水语音识别“云平台”. http://www.d1net.com/cloud/plat/84533.html.

[6] Cao J, Chen S, Wan Y, et al. Enabling distributed computing systems with elop™. Proceedings of 2012 Third International Conference on Networking and Distributed Computing (ICNDC). IEEE, 2012. 49–53.

[7] Neiger G, Santoni A, Leung F, et al. Intel virtualization technology: Hardware support for efficient processor virtualization. Intel Technology Journal, 2006, 10(3):167–177.

[8] Rosenblum M. VMware’s Virtual Platform™. Proceedings of Hot Chips, 1999. 185–196.

[9] Dowty M, Sugerman J. GPU virtualization on VMware’s hosted I/O architecture. ACM SIGOPS Operating Systems Review, 2009, 43(3):73–82.

[10] Watson J. Virtualbox: bits and bytes masquerading as machines. Linux Journal, 2008, 2008(166):1.

[11] Barham P, Dragovic B, Fraser K, et al. Xen and the art of virtualization. ACM SIGOPS Operating Systems Review, 2003, 37(5):164–177.

[12] Mergen M F, Uhlig V, Krieger O, et al. Virtualization for high-performance computing. ACM SIGOPS Operating Systems Review, 2006, 40(2):8–11.

[13] Kivity A, Kamay Y, Laor D, et al. kvm: the Linux virtual machine monitor. Proceedings of the Linux Symposium, volume 1, 2007. 225–230.

[14] Cherkasova L, Gardner R. Measuring CPU overhead for I/O processing in the Xen virtual machine monitor. Proceedings of the USENIX annual technical conference, 2005. 387–390.

[15] Cherkasova L, Gupta D, Vahdat A. Comparison of the three CPU schedulers in Xen. Performance Evaluation Review, 2007, 35(2):42.

[16] Menon A, Cox A L, Zwaenepoel W. Optimizing network virtualization in Xen. Proceedings of USENIX Annual Technical Conference, 2006. 15–28.

[17] Armbrust M, Fox A, Griffith R, et al. A view of cloud computing. Communications of the ACM, 2010, 53(4):50–58.

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

[19] Buyya R, Yeo C S, Venugopal S, et al. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation computer systems, 2009, 25(6):599–616.

[20] Kondo D, Javadi B, Malecot P, et al. Cost-benefit analysis of cloud computing versus desktop grids. Proceedings of IEEE International Symposium on Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE, 2009. 1–12.

[21] Sultan N. Cloud computing for education: A new dawn? International Journal of Information Management, 2010, 30(2):109–116.

[22] Chen Z, Han F, Cao J, et al. Cloud computing-based forensic analysis for collaborative network security management system. Tsinghua Science and Technology, 2013, 18(1):40–50.

[23] Bafeng H, Ming Z, Chuanjun Y, et al. Research and application of the darwin streaming server. Computer Engineering, 2004, 19.

[24] Yin H, Liu X, Zhan T, et al. Design and deployment of a hybrid CDN-P2P system for live video streaming: experiences with LiveSky. Proceedings of the 17th ACM international conference on Multimedia. ACM, 2009. 25–34.

[25] Tao S, Guérin R. Application-specific path switching: A case study for streaming video. Proceedings of the 12th annual ACM international conference on Multimedia. ACM, 2004. 136–143.

[26] Tao S, Xu K, Estepa A, et al. Improving VoIP quality through path switching. Proceedings of INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE, volume 4. IEEE, 2005. 2268–2278.

[27] Chan S H G. Distributed servers architecture for networked video services. IEEE/ACM Transactions on Networking (TON), 2001, 9(2):125–136.

[28] Wu Y, Wu C, Li B, et al. Cloudmedia: When cloud on demand meets video on demand. Proceedings of 2011 31st International Conference on Distributed Computing Systems (ICDCS). IEEE, 2011. 268–277.

[29] Sampaio A, Mendonça N. Uni4Cloud: an approach based on open standards for deployment and management of multi-cloud applications. Proceedings of the 2nd International Workshop on Software Engineering for Cloud Computing. ACM, 2011. 15–21.

[30] Lucas Simarro J, Moreno-Vozmediano R, Montero R S, et al. Dynamic placement of virtual machines for cost optimization in multi-cloud environments. Proceedings of 2011 International Conference on High Performance Computing and Simulation (HPCS). IEEE, 2011. 1–7.

[31] Cao J, Hwang K, Li K, et al. Optimal multiserver configuration for profit maximization in cloud computing. IEEE Trans. Computers, Special Issue on Cloud of Clouds, 2012. 1087–1096.

[32] Li J, Chinneck J, Woodside M, et al. Performance model driven QoS guarantees and optimization in clouds. Proceedings of ICSE Workshop on Software Engineering Challenges of Cloud Computing, 2009. CLOUD'09. IEEE, 2009. 15–22.

[33] Bolte M, Sievers M, Birkenheuer G, et al. Non-intrusive virtualization management using libvirt. Proceedings of the Conference on Design, Automation and Test in Europe. European Design and Automation Association, 2010. 574–579.

[34] Bai S, Wu H. The performance study on Several distributed file systems. Proceedings of 2011 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC). IEEE, 2011. 226–229.

[35] Chapman M. Rdesktop-A remote desktop protocol client, 2010.

[36] Seid H A, Lespagnol A. Virtual private network, June 16, 1998. US Patent 5,768,271.

[37] 李建设, 吴庆波. 基于 OpenSSL 的 VNC 安全性研究及实现 [J]. 微计算机信息, 2005, 11:6–6.

[38] 金汉均, 仲红, 汪双顶. VPN 虚拟专用网安全实践教程. 清华大学出版社, 2010.

[39] Chen W, Cao J, Li Z. Customized virtual machines for software provisioning in scientific clouds. Proceedings of 2011 Second International Conference on Networking and Distributed Computing (ICNDC). IEEE, 2011. 240–243.

[40] Mein G, Pal S, Dhondu G, et al. Simple object access protocol, September 24, 2002. US Patent 6,457,066.

[41] Fielding R. Representational state transfer. Architectural Styles and the Design of Netowork-based Software Architecture, 2000. 76–85.

致　谢

首先要衷心感谢我的导师曹军威副研究员，在研究生的这三年中，曹老师在科研方向上给了我很多的指导，帮我创造了各种学习的机会，让我能够快速的进入云计算这一复杂的领域进行研究，并不断的在研究过程中给与我指导和支持。同时，曹老师严谨求实，努力工作的态度也不断地感染着我，在工作，生活的方式与态度上，曹老师也言传身教，帮助我迅速的成长。

同时感谢陈震老师，黄铠教授，两位老师在我科研中也给我极大的帮助。感谢实验室的王震，李俊伟和万宇鑫师兄，给我提供了大量的科研材料，并创造很好的研究平台，让我能够顺利进行研究。感谢在科研过程中合作过的各位师兄，师弟，同学们，没有你们也就没有我的成果。

还必须要感谢我的父母，在我 19 年的漫长学生生涯中，是我最坚实的后盾，在生活上、精神上给我支持与鼓励，教给我做人的道理，教会我坚持与努力。感谢我的女朋友，为我的研究生生活增添了许多色彩，在我科研的路上不断地给我鼓励和帮助，是我能够顺利完成科研工作的重要力量。

最后感谢所有对我提供帮助的亲人、老师、朋友们，谢谢你们！

本论文使用 ThuThesis 编写，特此致谢。

声　明

本人郑重声明：所呈交的学位论文，是本人在导师指导下，独立进行研究工作所取得的成果。尽我所知，除文中已经注明引用的内容外，本学位论文的研究成果不包含任何他人享有著作权的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。

签　名：_____ 日　期：_____

个人简历、在学期间发表的学术论文与研究成果

个人简历

1988 年 10 月 16 日出生于山东省胶南市。

2006 年 8 月考入清华大学自动化系，2010 年 7 月本科毕业并获得工学学士学位。

2010 年 9 月免试进入清华大学自动化系攻读控制科学与工程硕士学位至今。

发表的学术论文

[1] Chen W, Cao J, Li Z. Customized virtual machines for software provisioning in scientific clouds. Proceedings of Networking and Distributed Computing (ICNDC), 2011 Second Inter-national Conference on. IEEE, 2011. 240–243.(EI 检索, 检索号 20114514497722)

[2] 陈伟，曹军威，钱翰. 基于虚拟组织的桌面云安全访问与共享机制研究, 集成技术, 1(4), 25-29, 2012.

[3] Chen W, Cao J, Wan Yuxin. QoS-aware Virtual Machine Scheduling for Video Streaming Services in Multi-Cloud. Tsinghua Science and Technology, Special Section on Cloud Computing(已经录用，EI 检索）