摘　要

近年来，随着高速计算机网络、数字压缩技术以及大容量存储技术的快速发展，基于网络的视频服务逐渐成为现实，基于网络的视频服务在娱乐、教育、广告、信息获取等各个方面都有广泛的应用。然而，这些系统中普遍存在的问题是，服务质量和服务水平受到网络带宽瓶颈和视频服务器瓶颈的限制，表现为网络拥堵，服务延迟、中断，甚至由于服务器负担过重而拒绝服务。

为解决上述问题，一些视频点播（Video-on-Demand，简称 VoD）系统通过提高硬件和服务器配置的方式来提高系统性能；有些 VoD 则通过调整系统结构，如采用了分布式或 P2P 体系结构把工作负载分配到多台服务器（或对等点 Peer）上，以集群协同工作的方式来提高系统性能；有些系统则通过采用流调度技术，如扩展指数广播（Extended Exponential Broadcasting，简称 EEB）、控制多播（Controlled Multicast，简称 CM）等，在不增加硬件系统性能情况下，通过广播或信道共享等机制来突破网络带宽瓶颈的限制，从而提高系统的服务用户数量。然而，无论采用何种方式，都有其各自的优缺点和适用范围。

本文在分布式体系结构的基础上采用单播、CM 和 EEB 等多种流调度方案，实现了一个可扩展的分布式 VoD 系统。对实时流协议（Real-Time Streaming Protocol，简称 RTSP）进行了扩展，使其能传输服务器的负载等信息。并针对该系统采用多种流调度方案的特点，提出一种对基于 Linux 虚拟服务器（Linux Virtual Server，简称 LVS）系统的传统动态反馈负载平衡算法的改进算法，使任务负载的分配更加合理。在节目存储策略方面，本文提出了以调整节目的流调度方案来代替节目存储调整的观点，并给出了一种节目流调度方案可自动调整的实现方法。另外，本文还介绍了如何提高客户端播放器解码能力，以及非线性编辑技术在节目制作中的应用等方面的研究内容。

最后，本文给出了一种系统仿真测试方法及其实现过程。测试结果表明，该系统能有效缓解传统 VoD 系统中服务器 I/O 或网络带宽的瓶颈问题，具有良好的可扩展性；改进算法比传统负载平衡算法能减少 10%～30%系统并发信道占用数，提高了系统支持的用户点播数量。

关 键 词：视频点播，分布式，负载平衡，流调度

Research of Service Mechanism and Algorithm for Video Service System

Abstract

Network video service systems came true in resent years, along with the fast developing of the fast computing network, digital compressed technology and the large capability storage technology,. Video services based network are widely applied to recreation, education, advertisement, information acquisition and other domains, like Video-on-Demand (VOD), distant education, online trading and initiative news etc.

But, there is one ubiquitous problem in these systems, that is these systems' quality of service are restricted by the bottleneck of network bandwith and video server, so that it causes network huddle, service delay and interruption, or worse that service requests would be refused because the servers are overload.

In order to solve the problem above, first, some VoD systems are improved by enhancing the systems' hardware. Second, some are improved by adjusting the systems' architecture, like adopting the distributed or P2P architecture, so that the work load can be distributed to multi servers or peers, which cooperate with each other and work as a cluster. Third, some VoD systems adopting new channel scheduling schemes, like Extended Exponential Broadcasting (EEB), Controlled Multicast (CM) etc., these technologies can break through the bottleneck of network and increase the amount of the users, by using broadcasting and steam sharing mechanisms. However, whatever mode they used to improve VoD system, in which there are advantages and disadvantages, and they applied to different fields.

An extensible distributed system would be present in this paper, which employs some Streaming scheduling schemes including unicast, CM and EEB. We extended the Real-Time Streaming Protocol (RTSP) and used it to transfer the load informations of server, so the servers can connect with each other and form as a distributed system. Aimed at this system had different Streaming scheduling schemes, we present an improved algorithm to traditional dynamic load balancing algorithm based on Linux Virtual Server (LVS), and it made the workload distribute more reasonable. In the programs storage aspect, we present a policy that the adjusting of programs storage can be substituted by the adjusting of programs Streaming scheduling schemes, and then we present an auto-adjusting implementation of the programs channel scheduling schemes. In addition, we introduce how to enhance the decode capability of VoD client player, as well as how to use DirectShow Editing Services (DES) to make a video clip.

Finally, we introduce the technology about implementation of simulation for VoD. We proceeded to test our VoD system with simulation program. The Result showed that the system could lighten the bottleneck problem in traditional systems, can reduce 10%～30% system's concurrent streams and serve more users, with the help of improved algorithm compared with traditional algorithm, this system can satisfy large scale of VoD requests.

Key Words: Video on Demand, Distributed, Load Balance, Streaming Schedule

独 创 性 声 明

本人声明所呈交的学位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。据我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得北方工业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谢意。

学位论文作者签名：吴志伟 签字日期：2007年5月20日

学位论文版权使用授权书

本学位论文作者完全了解北方工业大学有关保留、使用学位论文的规定，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，允许论文被查阅和借阅。本人授权北方工业大学可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后适用本授权书）

学位论文作者签名：吴志伟　　导师签名：袁志明

签字日期：2007年5月20日　　签字日期：2007年5月20日

学位论文作者毕业后去向：

工作单位：　　电话：

通讯地址：　　邮编：

1 引 言

随着宽带计算机网络、数字压缩技术及大容量存储技术的快速发展，视频点播服务逐渐成为现实。视频点播系统可以提供用户即点即播的视频服务，改变传统的视频播放方式，有效地解决了需求服务被动性、媒体信息的滞后性等技术问题，给用户使用带来方便。

然而数据量大、传输带宽高、实时性强的视频数据对视频服务器和网络性能提出了很高的要求，过多的用户经常会造成视频服务器 I/O 带宽和网络带宽的瓶颈，这使得视频点播服务的普及变得困难[1]，表现为网络拥堵、服务延迟、服务中断，甚至由于服务器负担过重而拒绝服务。因此，为解决上述问题，各种各样的技术被应用于 VoD 系统中，形成了不同结构、适用于不同网络条件和用户规模的 VoD 系统。

1.1 网络视频服务系统的研究现状

根据视频服务器在网络中的物理位置，VoD 系统可以分为集中式和分布式两种。集中式 VoD 对视频服务器主机性能要求很高，投资大，影响了 VoD 的普及和应用，因此在过去十多年中，对 VoD 系统的研究工作大部分关注的是分布式系统[2] 。与只有单个服务器的集中式 VoD 系统相比，分布式系统具有两个显著的优点：可扩展性和容错性。可扩展性指即系统可加入多个服务器节点而获得扩展，而容错性要求服务器的单点失败对客户来说是透明的。分布式 VoD 系统把视频节目按某种策略存储在多台视频服务器上，每台视频服务器为一定量的用户提供服务，如果要满足更多的用户点播需求，只需要适当的增加视频服务器的数量。因此，可采用多台廉价、普通性能的服务器，甚至可以采用 PC 机，来代替一台昂贵的高性能视频服务器，提高了系统的性价比，同时把用户分散到多台服务器上，可有效缓解视频服务器 I/O 和网络带宽瓶颈问题。本节主要分析这些系统的研究现状及目前比较流行的 VoD 系统的体系结构。

1.1.1 集中式的 VoD 系统

集中式结构的 VoD 系统如图 1.1 所示，该结构中视频节目集中存储在一台中心视频服务器里，由它向所有请求的用户提供流媒体服务。这类系统一般均采用高性能的视频服务器，从硬件上来提高服务器的性能。目前，国外比较著名的视频服务器产品主要有：nCUBE 公司的 MediaCUBE、FVC 的 V-Cache 和 SGI 的 Challenge 等。

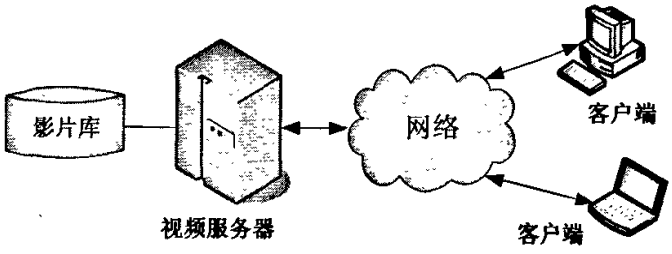

图 1.1 集中式的 VoD 系统

集中式结构最突出的优点是实现和管理简单，在用户数量不多的小规模应用上有一定的优势。但是在这种模式下的 VoD 系统所能提供服务的最大用户数，受到集中式视频服务器的磁盘读取速度、处理机能力、缓存容量、输出 I/O 速率和接入网络带宽的制约。采用这种结构的 VoD 系统，即便在高速接入网上，配置高性能的视频服务器，如耦合多处理器视频服务器，也不能满足大规模的点播应用要求，而且还会提高投资成本。

1.1.2 基于客户端 / 服务器的分布式 VoD 系统

VoD 系统通常采用客户 / 服务器模型[3]，由三部分组成：视频服务器、客户端和通信网络。分布式 VoD 系统的视频服务器如图 1.2 所示。

分布式 VoD 系统的视频服务器从系统结构上看，有以下特点：第一、系统由多节点构成；第二、多节点通过高速的内联网构成并行系统，并具有良好的可伸缩性和扩展性；第三、多节点并行地提供多条流的视频服务，提高了视频服务并发能力，可满足大量用户的服务请求；第四、对视频服务中的单条流来说，在某一时段，只能是由一个节点提供服务，但是其数据来源可以是其它节点。利用分布式视频服务器的这些特性，可从多方面提高其可靠性。

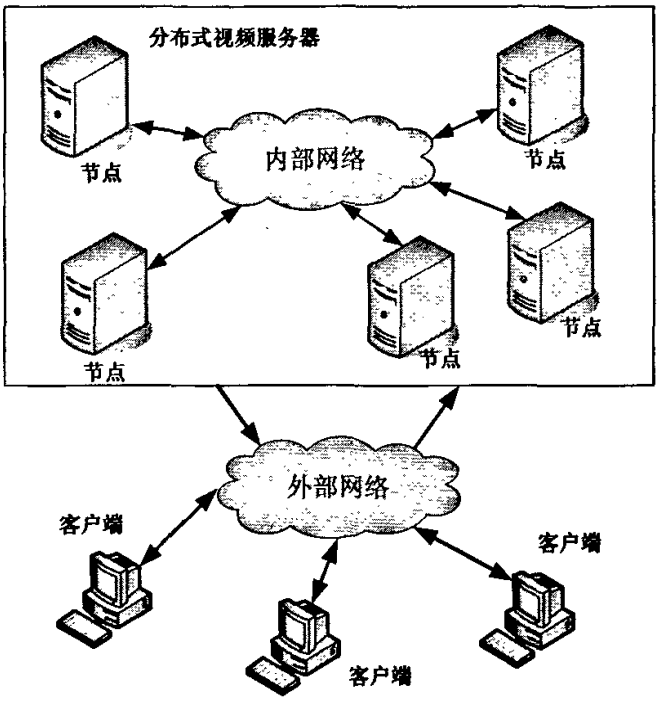

图 1.2 典型的基于客户 / 服务器的分布式 VoD 系统模型

目前典型的的分布式 VoD 系统一般可分为控制节点、服务节点和数据节点等组件，它们通过一个高速互连的内部网络连接起来，构成多节点视频服务器结构的服务器系统。当控制节点个数为 1 时，称之为总控节点。总控节点负责整个系统的工作。

根据系统优化方案的不同 VoD 系统可分为几类[4] [5]。一种划分是基于用户是否具有完全或部分控制流的能力，VoD 系统可分为 TVoD（True VoD）和 NVoD（Near VoD），TVoD 采用单播方案，而 NVoD 采用广播方案。另一种划分是根据数据的存储策略把 VoD 系统分为时间分片（time stripping），空间分片（space stripping）和无分片（non-stripping）三种。所谓分片（stripping）是指将一个完整的媒体文件分成一个个小块，按一定的原则存储到磁盘上。

采用数据分片的系统如文献[6]中提出的方案是将视频节目数据分割成独立的视频数据，分布在不同的视频服务器上，并根据一种推的调度算法为客户并行传送数据。这种方案提高了资源利用率，但是存在 3 点不足：第一是服务节点之间需要建立 RAID 阵列，复杂度大大增加；其次整个服务器系统鲁棒性降低，如果有一个服务节点出现故障，整个服务器系统将不能工作；第三是实用性和经济性不高，应用实现难度太大。

考虑到数据分片不仅会增加系统的复杂度并降低系统的可靠性，而且，数据分片在对交互性要求比较高的系统如 TVoD 系统中表现不佳[7]，因此下面我们主要讨论无分片的 VoD 系统。

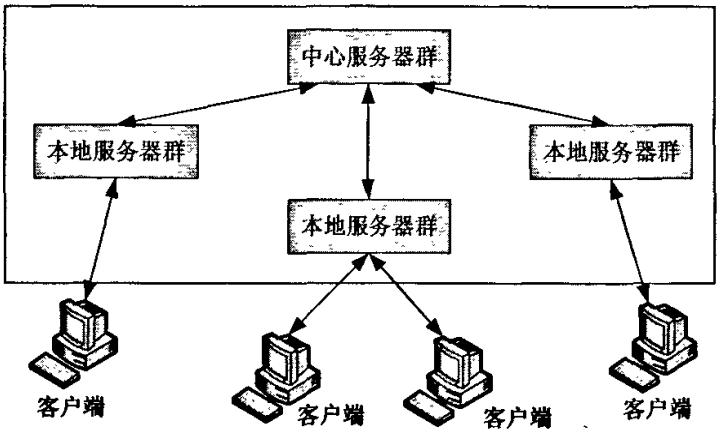

图 1.3 树型结构的 VoD 系统

很多系统采用中心服务器（或中心数据节点）一本地服务器的树形结构[8-10]，如图 1.3 所示。在这一体系结构中，树根节点即中心节点服务器（群），其作用是存档所有影片，有的中心服务器也兼顾向客户传输影片。中央服务器与本地服务器群之间有着高速网络。与之相连的下一层是一组本地服务器群，每一服务器群中包含若干台服务器，而每一个客户端都连接至某一本地服务器上。本地服务器群上存放了部分较为热门的影片，它们靠近最终用户，负责直接处理客户的服务请求，向客户传送视频流，从而达到分担中心服务器与网络负载、降低用户感知的延迟、使系统具有可扩展性等作用。中心服务器群中存储着本 VoD 系统所能提供的所有影片资料。当本地服务器群发现客户所请求的影片在本地无法找到时，就通过高速网络从中心服务器群下载所需影片，并直接传送给客户。这种结构的 VoD 系统适合应用于异构网络，外层的服务节点靠近用户，利用网络传输距离较短和速度较快的特点，可提高视频服务质量和增加服务用户数量。但这一结构对本地服务器影片存储策略和从中心服务器缓存影片的策略要求较高，策略不当往往会造成服务器间负载不均。

为简化系统，降低开发难度，本课题拟采用数据镜像，即为集群中所有的服务器都存储了相同的节目数据，无专门的数据节点。

A.Papagiannis 等[11]给出了一个类似的系统，它是一个低成本的可扩展的 TVoD（True VoD）集群系统，使用数据镜像来达到系统的负载平衡，但是该系统是基于单播方案实现的，没有采用新的信道调度技术，集群的负载平衡算法也没有具体给出。

A.Papagiannis 等[12]在上述系统的基础上增加了广播方案的支持，但它采用的是等分分块的广播方案，该方案没能很好的解决信道占用数与用户等待时间的矛盾，效果并不理想。另外，集群的负载平衡算法也没有具体给出。

1.1.3 基于 P2P 技术的 VoD 系统

近年来，随着 Napster，Gnutella 等 P2P 项目取得的巨大成功，利用 P2P 技术提供视频服务的尝试也倍受人们的关注。不同于传统的 B/S 和 C/S 模式，P2P 是一种分散的网络模式。在这种网络模式中每一个实体被称为对等体，既充当服务器为其他节点提供服务，又充当客户机享用其他节点提供的服务。从而减少甚至是克服了对中心服务器的依赖，能够避免因为过度依赖而产生的不良影响；而且随着网络中对等体数量的不断增加，网络所能提供的资源越来越丰富，性能也越来越强；同时能够有效地利用网络上大量闲置的资源，消除信息孤岛、增强互联网的分布和共享。

P2P 系统在具体实现中主要存在三种不同的结构[13]，即以 Gnutella 为代表的纯 P2P 结构，以 Napster 为代表的带核心服务器的混合 P2P 结构和以 Morpheus 为代表的带超级节点的 P2P 结构。其中纯 P2P 结构不依赖于任何中心服务器，动态发现网络中的其他对等方并与之交互信息；混合 P2P 结构中存在一个只具有服务功能的核心服务器来维护所有对等点的共享目录，并进行查询，但文件的共享交换过程是直接在对等方之间进行的；带超级节点的 2 结构类似于混合 2 结构，但超级节点的数目不只一个，可以向其中的一个发送资源更新信息及查询要求，超级节点之间是对等的。

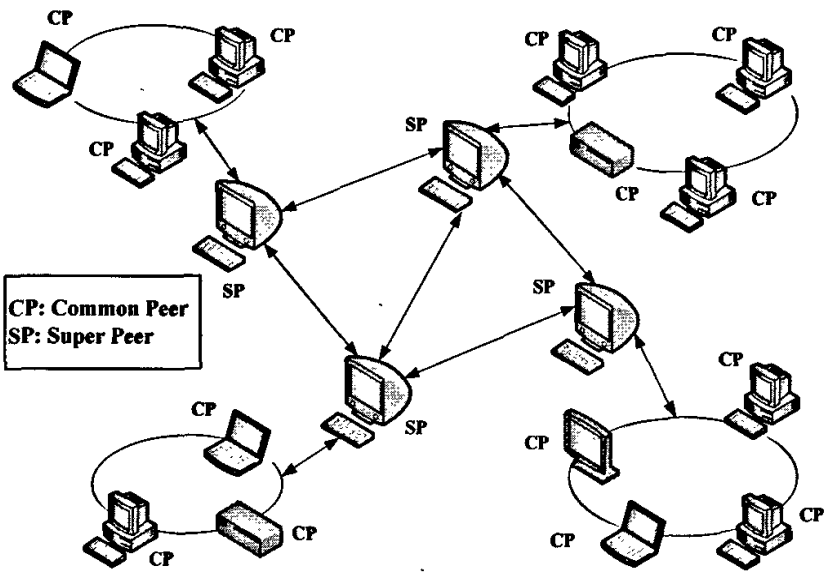

图 1.4 基于 P2P 技术的 VoD 系统

图 1.4 所示为带超级节点的 P2P 结构的 VoD 系统，它利用超级节点来存储影片目录信息、接受查询请求，将搜索到的影片提供者地址返回给用户节点，然后在影片需求者与提供者之间建立直接连接来进行多媒体数据流的传输。在该 P2P 系统框架结构中，各分布视频节点 Peer 通过相应的网关和路由器连接在一起，可以看作为一个人的虚拟服务器。每一个分布的视频节点 Peer 既可以作为虚拟服务器的一部分，来提供音、视频内容，又可以向这个虚拟服务器提出请求然后在中间件——P2P 应用程序的帮助下，从具体的某个节点处获得视频流来进行播放。

一般来说，基于 P2P 模式的 VoD 系统面临如下挑战[14]：

1) 节点搜索，主要是指当节点加入系统时，如何在组播树中快速、有效地搜索到合适的父节点；

2）系统容错，Peer 节点可能随时离开系统或失效，从而中断其子节点的服务。如何让被中断的节点能够快速、有效地进行中断恢复，是系统面临的核心问题；

3）协议开销，由于组播树的建立和维护依赖于控制协议，如何设计控制协议，使之具有良好的可扩展性也是系统的关键问题；

4）QoS 保证，主要是指在 Peer 节点存在离开或失效的前提下，如何保证节目的播放质量，如完整性、连续性等。

1.1.4 采用新的流调度技术的 VoD 系统

采用新的信道调度策略也是缓解上述网络带宽瓶颈和视频服务器瓶颈问题的另一个有效途径。当前比较新的信道调度方案主要有两类：适合于热门节目的分块广播（partitioned broadcasting）方案[1,15-17]和适合于普通节目的补块（patching）方案[18-21]。它们通过利用网络多播（multicast）技术实现多个用户对信道的共享，使得信道利用率得到提高。扩展幂级方案 EEB [16]是这些新信道调度方案中性能较好的一种，其基本思想是将节目按照一定的策略进行分块（逻辑上分块，非物理分块），并将各个分块在特定的信道上重复广播，通过利用客户端资源（I/O 带宽与缓存空间），很好地解决了传统轮播方案中信道数与用户等待时间的矛盾，如对于 120min 的节目，同样占用 8 个信道，4 阶 EEB 方案的平均用户等待时间为 27s，而轮播方案中的平均用户等待时间为 450s。

CM 方案允许不同时间请求同一个节目的用户共享一个视频流，这种共享是通过让后面到达的用户从为前面到达的用户分配的信道上接收数据实现的。同时，服务器为后面到达的信道开辟一个信道传送该用户错过的部分数据，这样该用户需要同时从两个信道上接收数据，并进行缓存。使用 CM 方案后，与仅使用单播时相比，系统的并发信道数大大减少，有些情况下可减少 60%的系统并发信道数。

上述两种新的流调度方案性能虽然好，但是同样存在一些问题。首先，由于它们是基于信道共享机制和组播的流调度方式，信道对用户来说不是独占的，因而这两种方案并不支持用户暂停、快进、快退、定位等交互操作（Video Cassette Recorder, 简称 VCR），对用户欣赏影片有一定影响。其次，这两种方案一般情况下只能应用于同构网络，对于异构网络，它们的应用具有一定的局限性。

1.2 本文的研究内容与方法

如第 1.1 节所述，无论是集中式 VoD 系统还是分布式 VoD 系统，都有各自的优缺点和适用范围，但总的趋势还是向分布式 VoD 的方向发展。

我校在 VoD 系统方面的研究有一定的基础，其中 02 级计算机应用专业研究生吕春所作毕业设计[22]，已经实现了一个采用单播、CM、EEB 等多种流调度方案的集中式 VoD 系统。该系统利用性能较优的新调度方案，有效地突破了网络带宽地瓶颈，使系统支持的并发用户数大大增加。但是该系统毕竟是只有单个服务器的集中式 VoD 系统，服务器的 I/O 带宽的瓶颈依然存在。

本文将在吕春研究的 VoD 系统（以后称原系统）的基础上，结合新的流调度技术和负载平衡技术，构建一个分布式的 VoD 系统，应用上述技术的优点更好地解决视频服务器 I/O 带宽和网络带宽的瓶颈问题，力图在系统支持并发用户数和服务质量（Quility of Service，简称 QoS）方面取得进展。本文的研究内容如下：

1）研究如何在现有系统的基础上构建一个可扩展的分布式的 VoD 系统，使其能有效缓解视频服务器 I/O 带宽和网络带宽的瓶颈问题，并具有结构简单、可动态扩展、适应性强等优点。

原系统是一个集中式 VoD 系统，虽然采用了多种新的信道调度方案，但由于存在服务器性能瓶颈，服务能力有限。因此，本课题拟在该系统基础上构建一个可扩展的分布式 VoD 系统，使用负载平衡机制把用户的请求合理地分配到各个服务器中，每台服务器满足一定量的用户请求数。如果要满足更多用户的视频服务请求，只需要适当地增加视频服务器的数量即可。合理设计 VoD 系统结构，不仅使用灵活，实现容易，而且还能巧妙解决接入网络带宽对最大用户数的制约，使 VoD 系统更适合于商业应用。因此，如何构建一个可扩展的分布式的 VoD 系统，使其具有上述优点是本课题首要解决的问题。

2）设计实现一种适合本系统的集群负载平衡算法。

目前的 VoD 系统集群系统多数是基于单播方案实现的，还没有采用 EEB，CM 等新信道调度方案的 VoD 集群系统的出现。这些系统的负载平衡机制一般采用动态反馈负载平衡算法，即根据各个服务器的负载情况（CPU 使用率，磁盘 I/O 带宽，活动连接数等指标）计算负载值，然后把任务分配给负载最轻的一个服务器。因此，本课题拟根据各种信道调度方案的特点，设计一种基于多种流调度方案的负载平衡算法，使用户请求能够根据各个服务器节目配置方案的不同，合理地分配到适当的服务器当中，以提高系统效能和资源利用率。

3）研究节目的存储策略，实现节目的流调度方案的自动调整。

原系统采用静态方法配置节目的调度方案。但节目的受欢迎程度往往是由观众而不是由管理员来决定的。而如果节目调度方案配置不当，如适用于热门节目的分块多播方案用在了冷门节目上，而热门节目却使用了适用于冷门节目的单方案，这样不仅多种信道调度方案的优势得不到发挥，而且会严重降低网络带宽利用率，造成系统资源的浪费。若采用动态配置，使节目的调度方案可以随着用户点播数量能够自动地得到及时的调整，可以更好地提高系统的整体性能和网络带宽利用率。

4）改善客户端播放器的解码问题。

原系统只能播放用 Divx3.11 编码的 avi 文件，而目前这类文件只占视频文件中的一小部分，很多视频文件由于格式不同而无法用于点播服务。目前的解决办法是先用其他视频转码工具把这些文件转成所需格式，然后才能用于视频点播。因此，客户端播放器的解码能力急需提高，这也是本课题的一个研究内容。

5）对系统进行仿真性能测试。

系统测试所需要解决的首要问题就是如何在实验室范围内对系统进行接近实际运行的测试。由于条件所限，不太可能做有几百人同时点播的真实测试；另一方面大规模的真实测试，测试数据难以反馈和统计，所人力物力消耗巨大。因此，采用仿真测试的方法，在一台主机上可以模拟几十个上百个用户点播，可有效解决上述问题，而且系统条件和测试方式也可以灵活配置，可以对系统进行更加全面的测试和分析。

6）采用非线性编辑技术制作视频节目。

这是在增加系统节目源方面对系统的扩展，通过非线性编辑，可制作出许多专门的视频剪辑，供“特殊观众”点播使用。如体育比赛的关键片段、影视节目或歌曲的精彩串联等，使系统可点播的节目更加丰富多彩。

其中，在上述内容中，把分布式体系结构和新的流调度方案结合起来，是本课题的一个创新之处；其次，对传统动态反馈负载平衡算法的改进，使之可根据不同的流调度方案对工作负载进行合理分配，也是本课题的一个创新之处。

1.3 论文结构

网络视频服务系统中服务机制及算法研究，本文从理论研究和应用实践两个方面对其进行论述，具体结构如下：

第一章，主要通过讨论当前流媒体技术与分布式体系结构和 P2P 结构的结合，以及当前视频点播系统的研究现状和典型应用，由此引出本文需要研究的问题和研究方法。

第二章，主要介绍分布式视频点播的相关研究，并对其进行了较为详细的分类讨论总结，比较各自的优缺点，提出针对本课题的具体应用方法。

第三章，详细介绍为实现本系统所采用的两个关键技术：系统的分布式结构的构建，以及为这个结构所设计的负载平衡算法。

第四章，详细介绍本课题的网络视频服务系统的设计和实现，包括系统体系结构、负载平衡算法、点播节目的信道调度方案的动态调整以及提高客户端解码能力等方面的设计和实现。

第五章，介绍了 DirectShow 非线性编辑技术的在视频节目制作中的应用，给出其详细实现过程。

第六章，详细介绍系统的仿真测试和性能研究，包括了模拟点播器的设计和实现以及对系统的各项性能测试和分析。

第七章，是全文的总结部分，并展望今后需要研究的内容。

2 VoD 系统的相关研究

本章将介绍与本课题研究相关的基础知识，主要有 RTSP 及其在 VoD 系统中的应用、负载平衡算法，这两个部分是构建分布式 VoD 体系结构的基础。此外，本章还介绍本课题所需要用到的视频编码解码以及视频节目制作中的非线性编辑等方面的知识。

2.1 RTSP 协议

目前，支持基于 IP 的流媒体网络协议主要有实时传输协议/实时传输控制协议(RTP/RTCP)、实时流协议 (RTSP) [23]、资源预留协议(RSVP)和会话描述协议(SDP)。在本课题的视频点播系统中实现了 RTP/RTCP、RTSP 和 SDP 协议。其中，除了服务器与客户端需要建立 RTSP 协议连接之外，各服务器之间也建立了 RTSP 协议连接，用于传递服务器负载和管理等信息，从而形成一个分布式的系统。本节主要讨论 RTSP 协议。这几种协议的层次如图 2.1 所示。

<table>
<tr><td rowspan="2">应用层</td><td>SDP</td><td></td></tr>
<tr><td>RTSP</td><td></td></tr>
<tr><td rowspan="2">传输层</td><td></td><td>RTP</td></tr>
<tr><td>TCP</td><td>UDP</td></tr>
<tr><td>网络层</td><td colspan="2">IP</td></tr>
</table>

图 2.1 协议层次图

RTSP 是一个比较新的协议，关于它的设想在 RFC2326 文件中有所描述。它最早是由 Real Networks 公司 Netscape Communications 公司和 Columbia 大学等联合提出的 Internet 草案。RTSP 协议用于建立并控制一个或几个时间同步的连续视频、音频流的连接。尽管用 RTSP 交叉传输连续媒体流和控制流是可能的，但通常它并不用于连续媒体流的传输。换言之，RTSP 充当多媒体服务器的网络远程控制。RTSP 连接没有绑定到传输层连接，如 TCP 连接，在 RTSP 连接期间，RTSP 用户可打开或并闭多个对服务器的可靠传输连接以发出 RTSP 请求。此外，也可使用无连接传输协议，如 UDP 协议。RTSP 控制的节目流可以用 RTP 作为传输协议，但 RTSP 操作并不依赖用于携带连续媒体的传输机制。RTSP 在语法和操作上与 HTTP/1.1 类似，因此 HTTP 的扩展机制大都可加入 RTSP，但它又具有许多与 HTTP 协议不同的特点，如有状态、客户端或服务器端

均可发出请求等。我们使用该协议实现视频服务器和用户端之间控制信息的交互，如初始化、查询节目单、播放节目、交互式控制等。

2.1.1 RTSP 协议格式

RTSP 协议定义了两种消息格式：RTSP 请求消息格式和 RTSP 应答消息格式。

请求消息的格式如图 2.2 所示。

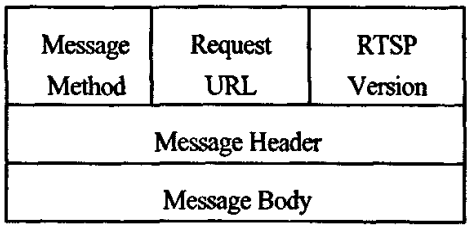

图 2.2 RTSP 协议请求消息格式

在请求行中有下面的一些域：

1. 方法<method>域有很重要的意义，它描述了请求的方法，主要的方法有 DESCRIBE、OPTIONS、PAUSE、PLAY、SETUP、TEARDOWN 等。
2. 统一资源地址<URL>域是用户请求访问的媒体数据的绝对路径,例如，www.ncut.edu.cn/vod/1.avi。
3. 版本<version>域是客户端使用的 RTSP 协议的版本号，现在版本号为 RTSP/1.0。

标题行(Message Header)是可选的，但客户一般都要在请求消息时插入许多标题行，每一标题行都包含两个部分：标题域名和相关的值。实体(Message Body)在一般情况下很少使用，它主要用于对标题行没有定义的标题进行扩展。下面是一个使用 PLAY 方法向媒体服务器请求播放一部节目的例子。

PLAY rtsp://video.example.com/vod/1.avi RTSP/1.0
Cseq: 835
Session:12345678
Range:smpte=0:10:22-;time=20050123T153600Z

这个请求消息包含一行请求行和 3 行标题行，整个消息共有 4 行 ASCII 文本。请求行（PLAY rtsp://vidio.example.com/vod/1.avi RTSP/1.0）用来告诉媒体服务器,客户应用程序使用 PLAY 方法想要播放的对象是“video.example.com/vod/1.avi”，使用的应用层协议是 RTSP/1.0。第一个标题行（Cseq：835）告诉服务器发送这个 PLAY 命令的顺序号为 835，第二个标题行（Session：12345678）告诉服务器该用户的会话身份识别符为

12345678，第三个标题行（Range:smpte=0:10:22-；time=20050123T153600Z）告诉服务器节目从 SMPTE 时间 0:10:20 开始播放直到节目结束，回放在 2005 年 1 月 23 号 15:36 开始。

应答消息的格式如图 2.3 所示。

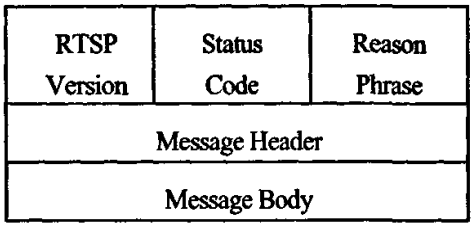

图 2.3 RTSP 协议应答消息格式

服务器接收到客户的 RTSP 请求消息之后进行分析，将分析和操作结果返回给客户机，具体的做法是发送一条 RTSP 响应信息，除了状态行之外，响应消息的格式与请求消息的格式相同，在状态行中，除了 RTSP 的版本号之外，还包含状态码<status code>和短语<phrase>，它们组合起来表示客户请求所获得的结果。例如，上面请求的媒体文件存放在视频服务器上，而且可发送给客户机，状态码和短语分别包含“200”和“OK”。

RTSP/1.0 200 OK

Cseq：835

Range：smpte=0:10:22-;time=20050123T153600Z

2.1.2 RTSP 协议在 VoD 系统中的应用

下面我们来举一个只有两台服务器（主服务器和从服务器）的简单分布式 VoD 系统，从它的点播流程中，来说明系统的服务器和客户端之间是如何利用 RTSP 协议来协同工作的。客户端首先连接到主服务器，通过 HTTP 得到所需视频数据的描述文件（description file），按照得到的文件名、地址发送 SETUP 消息，这时候的地址就是主服务器的地址。主服务器得到 SETUP 消息后，在数据库中查询客户需要文件所在的服务器的地址，并在 SETUPResponse 的消息中告诉客户端，具体是用到 Location 这一项表示用户需要多媒体务器的地址，并在 SETUPResponse 的消息中告诉客户端，具体是用到 Location 这一项表示用户需要多媒体文件的 URL。用户接收到 SETUPResponse，按照接收到的地址发送 PLAY 消息，并做好准备接收数据，该地址对应的播放服务器（其实这个地址也可能是主服务器或从服务器，因为它们也可以存放有多媒体数据）接

收到 PLAY 消息，发送回复消息 PLAYResponse，就开始给相应客户端发送数据，数据送完，客户端发 TEARDOEN 消息，服务器回复，这样一个完整的点播过程结束，如图 2.4 所示。

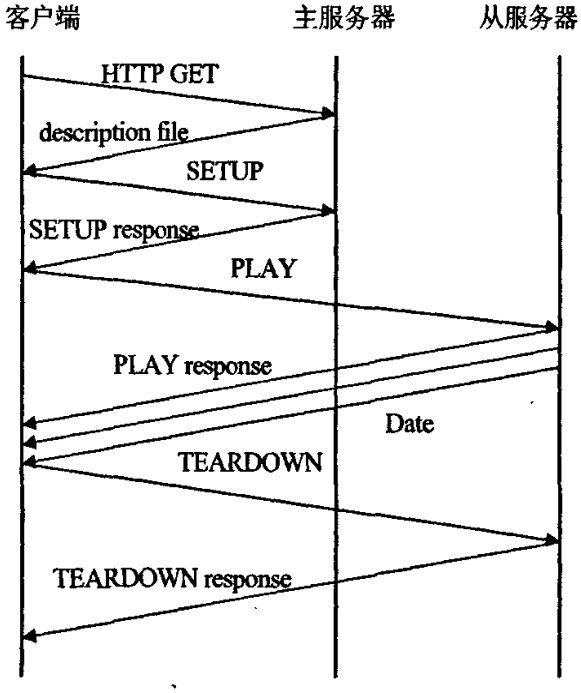

图 2.4 简单点播流程中的 RTSP 协议应答

该 VOD 系统利用了 RTSP 的灵活性，使客户端最开始都将 SETUP 命令发送到主服务器，而实际给客户提供数据的可能是别的服务器，实现了负载平衡，同时使得多媒体数据的管理也方便简单，只需要添加和删除文件的时候对数据库做相应的修改，这可以集成在一个管理系统中，利用数据库还可以开发更多其他的服务，比如点播计费等。

2.2 流调度方案

下面对本系统所采用的的 EEB 方案和 CM 方案进行简要介绍。这里 EEB 方案为分块广播方案，而 CM 方案属于补块方案。

2.2.1 EEB 方案

EEB 方案是为了解决传统轮播方案中信道数与用户等待时间之间的矛盾问题而提出的，它的基本思想是将节目按照一定的策略进行分块，并将各个分块在特定的信道上重复广播。分块方法如下（K 为块数）：

$$f(n) = 2^{n-\left\lfloor \frac{n}{m} \right\rfloor}$$

其中 m 为阶数，n=0，1，2，…，K-1

例如 3 阶 9 块时分块序列为：{1，2，4，4，8，16，16，32，64}，而 4 阶 8 块时分块序列为{1，2，4，8，8，16，32，64}。

首块的大小为：

$$L_1 = \frac{L}{\sum_{i=0}^{K-1} f(i)}$$

L 是节目长度，首块大小也就是客户的最大等待延迟。

服务器将各个分块在特定的信道上重复广播，图 2.5 给出了 3 阶 6 块 EEB 方案的广播示意图：

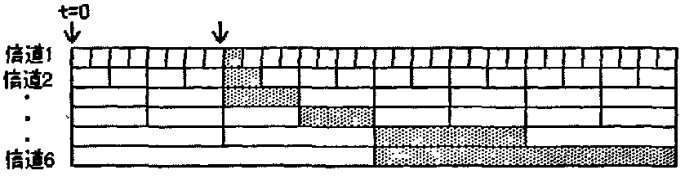

图 2.5 EEB 方案的广播示意图

t 时刻到达的用户最多等待第一个分块 D1 长度的时间，然后开始从信道上接收数据并同时进行播放。为保证播放的连续性，需同时从两个或两个以上的信道上接收数据，因此接收过程中需要进行数据缓存。图 1 的阴影部分给出了节目的接收过程。

EEB 方案通过利用客户端资源（I/O 带宽与缓存空间），很好地解决了传统轮播方案中信道数与用户等待时间的矛盾，如对于 120min 的节目，同样占用 8 个信道，4 阶 EEB 方案的平均用户等待时间为 27s，而轮播方案中的平均用户等待时间为 450s。

2.2.2 CM 方案

CM 方案允许不同时间请求同一个节目的用户共享一个视频流，这种共享是通过让后面到达的用户从为前面到达的用户分配的信道上接收数据实现的。同时，服务器为后

面到达的信道开辟一个信道传送该用户错过的部分数据，这样该用户需要同时从两个信道上接收数据，并进行缓存。为第一个用户分配的信道称为完全信道，而为后面的用户分配的信道称为部分信道。不难发现，越迟到达的用户占用部分信道的时间越长，为提高系统性能，CM 方案设定一个阈值 T，在分配了一个完全信道 T 时间后到达的用户将开辟一个新的完全信道。如图 2.6 所示，用户 C 到达的时间与用户 A 的时间间隔大于 T，则服务器为 C 重新分配一个完全信道，对于后面的用户(如 D)，服务器则为他们分配部分信道。

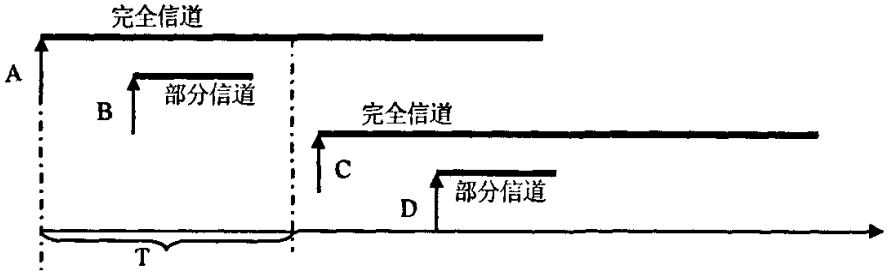

图 2.6 CM 方案示意图

这里设用户请求的到达服务 Poisson 分布，λ 为每分钟节目的请求强度。设定了阈值的 CM 方案的信道占用为 $\lambda\frac{2L+\lambda T^2}{2(\lambda T+1)}$，使得上式的最小阈值 T 为 $\frac{\sqrt{2LT+1}-1}{\lambda}$。此时信道占用数为 $\sqrt{2LT+1}-1$。

2.3 负载平衡

负载平衡是集群系统中的重要技术[24]，本质上讲，网络负载平衡是分布式作业调度系统的一种实现。平衡器作为网络请求分配的控制者，要根据集群节点的当前处理能力，采用集中或分布策略对网络服务请求进行调配，并且在每个服务请求的生命周期里监控各个节点的有效状态。已有的研究表明，在集群系统中采用负载平衡系统可以显著地提高集群系统的性能[25]。

一般的说，平衡器对请求的调度具备以下的特征：

1）网络服务请求必须是可管理的

2）请求的分配对用户是透明的

3）最好能够提供异构系统的支持

4）能够依据集群节点的资源情况进行动态分配和调整

负载平衡器在集群的各个服务节点中分配工作负载或网络流量。可以静态预先设置或根据当前的网络状态来决定负载分发到哪个特定的节点，节点在集群内部可以互相连接，但它们必须与平衡器直接或间接相连。

网络平衡器可以认为是网络层次上的作业调度系统，大多数网络负载平衡器能够在网络的相应层次上实现单一系统映像，整个集群能够体现为一个单一的 IP 地址被用户访问，而具体服务的节点对用户而言是透明的。这里，平衡器可静态或动态配置，用一种或多种算法决定哪个节点获得下一个网络服务请求。

2.3.1 网络平衡原理

在 TCP/IP 协议中，数据包含有必要的网络信息，因而在网络缓存或网络平衡的具体实现算法里，数据包的信息很重要。但由于数据包是面向分组的（IP）和面向连接的（TCP），且经常被分片，没有与应用有关的完整信息，特别是和连接会话相关的状态信息。因此必须从连接的角度看待数据包——从源地址的端口建立到目的地址端口的连接。

平衡考虑的另一个要素就是节点的资源使用状态。由于负载平衡是这类系统的最终目的，那么及时、准确的把握节点负载状况，并根据各个节点当前的资源使用状态动态调整负载平衡的任务分布，是网络动态负载平衡集群系统考虑的另一关键问题。

一般情况下，集群的服务节点可以提供诸如处理器负载，应用系统负载、活跃用户数、可用的网络协议缓存以及其他的资源信息。信息通过高效的消息机制传给平衡器，平衡器监视所有处理节点的状态，主动决定下一个任务传给谁。平衡器可以是单个设备，也可以使一组平行或树状分布的设备。负载平衡的组件如图 2.7 所示。

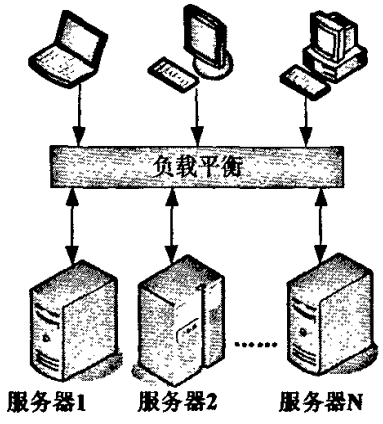

图 2.7 负载平衡组件

平衡算法设计的好坏直接决定了集群在负载均衡上的表现，设计不好的算法，会导致集群的负载失衡。一般的平衡算法主要任务是决定如何选择下一个集群节点，然后将新的服务请求转发给它。有些简单平衡方法可以独立使用，有些必须和其它简单或高级方法组合使用。而一个好的负载均衡算法也并不是万能的，它一般只在某些特殊的应用环境下才能发挥最大效用。因此在考察负载均衡算法的同时，也要注意算法本身的适用面，并在采取集群部署的时候根据集群自身的特点进行综合考虑，把不同的算法和技术结合起来使用。下面介绍一下基本的网络负载平衡算法：

2.3.2 轮询调度

轮询调度算法就是以轮询的方式依次将请求调度不同的服务器[26]。虽然轮询 DNS 方法也是以轮询的方式将一个域名解析到多个 IP 地址，但轮询 DNS 方法的调度粒度是基于每个域名服务器的，域名服务器对域名解析的缓存会妨碍轮询解析域名生效，这会导致服务器间负载的严重不平衡。这里，轮询调度算法的粒度是基于每个连接实体，同一用户的不同连接都会被调度到不同的服务器上，所以这种细粒度的 轮询调度要比 DNS 的轮询调度优越很多。但是，轮询调度总是假设所有服务器处理性能均相同，不管服务器的当前连接数和响应时间。所以该算法不适用于服务器组中处理性能不一的情况，而且当请求服务时间变化比较大时，轮询调度算法容易导致服务器间的负载不平衡。

2.3.3 加权轮询调度

加权轮询调度算法可以解决服务器间性能不一的情况，它用相应的权值表示服务器的处理性能，服务器的缺省权值为 1。加权轮询调度算法是按权值的高低和轮询方式分配请求到各服务器[27-28]。权值高的服务器比权值低的服务器先收到连接，并能处理更多的连接，相同权值的服务器处理相同数 目的连接数。加权轮询调度算法比较简单和高效。当连接请求的服务时间变化很大时，单独的加权轮询调度算法依然会导致服务器间的负载不平衡。

2.3.4 最小连接调度

最小连接调度是把新的连接请求分配到当前连接数最小的服务器。最小连接调度是一种动态调度算法，它通过服务器当前所活跃的连接数来估计服务器的负载情况[27-29]。调度器需要记录各个服务器已建立连接的数目，当一个请求被 调度到某台服务器，其

连接数加 1；当连接中止或超时，其连接数减 1。当各个服务器有相同的处理性能时，最小连接调度能把 负载变化大的请求分布平滑到各个服务器上，所有处理时间比较长的请求不可能被发送到同一台服务器上。但是，当各个服务器的处理能力不同时，该算法并不理想，因为 TCP 连接 处理请 求后会进入 TIME_WAIT 状态，TCP 的 TIME_WAIT 一般为 2min，此时连接还占用服务器的资源，所以会出现这样的情形：性能高的服务器已处理完所收到的连接，连接处于 TIME_WAIT 状态，而性能低的服务器可能忙于处理所收到的连接，还不断地收到新的连接请求。

2.3.5 加权最小连接调度

加权最小连接调度是最小连接调度的超集，各个服务器用相应的权值表示其处理性能。服务器的缺省权值为 1，系统管理员可以动态地设置服务器的权值。加权最小连接调度在调度新连接时尽可能使服务器已经建立的连接数和其权值成比例[27-29]。最小连接调度没有考虑连接请求的服务时间，也没能根据服务器当时的响应情况动态自动调整服务器的权值，所以，该算法依然会导致服务器间的负载不平衡。

2.3.6 动态反馈负载平衡算法

当客户访问集群资源时，提交的任务所需的时间和所要消耗的计算资源是千差万别的，它依赖于很多因素。例如：任务请求的服务类型、当前网络带宽的情况、以及当前服务器资源利用的情况等等。一些负载比较重的任务需要进行计算密集的查询、数据库访问、很长响应数据流；而负载比较轻的任务请求往往只需要读一个小文件或者进行很简单的计算。

对任务请求处理时间的不同可能会导致处理结点利用率的倾斜（Skew），即处理结点的负载不平衡。有可能存在这样情况，有些结点已经超负荷运行，而其他结点基本是闲置着。同时，有些结点已经忙不过来，有很长的请求队列，还不断地收到新的请求。反过来说，这会导致客户长时间的等待，而集群整体的服务质量下降。因此，有必要采用一种机制，使得平衡器能够实时地了解各个结点的负载状况，并能根据负载的变化做出调整。

具体的做法上采用了基于负反馈机制的动态负载均衡算法，该算法考虑每一个结点的实时负载和响应能力，不断调整任务分布的比例，来避免有些结点超载时依然收到大量请求，从而提高单一集群的整体吞吐率。

在集群内，负载均衡器上运行服务端监控进程，监控进程负责监视和收集集群内各个结点的负载信息；而每个结点上运行客户端进程，负责定时向均衡器报告自身的负载状况。监控进程根据收到的全部结点的负载信息来进行同步操作，既对将要分配的任务按照权值得比例重新进行分布。权值得计算主要根据各个结点的 CPU 利用率、可用内存以及磁盘 I/O 状况计算出新的权值，若新权值和当前权值的差值大于设定的阀值，监控器采用新的权值对集群范围内的任务重新进行分布，直到下一次的负载信息同步到来之前。均衡器可以配合动态权值，采用加权轮询算法来对接受的网络服务请求进行调度。

2.4 视频编码与解码

如今的电影是越来越好看，拍摄、制作的特技效果越来越精彩，人们期待的程度越来越高，盗版也越来越多；同时电影制作公司的防盗版技术做的越来越高明。但有句老话叫："道高一尺，魔高一丈"，无论你的电影多精彩、防盗技术多高明，总是有人偏偏能够把你"盗"出来，而且还"盗亦有盗"，在保持"原版原味"的条件下，占用的空间变得越来越小，操作越来越灵活、简单，越来越方便传播。而且这种技术随着版本不断更新，画质越来越贴近原版、压缩速度越来越快、压缩 / 播放进程对计算机的需求越来越低。目前可用的压缩编码目前有很多，现在比较流行的有 DivX，XviD，rmvb 等等。

2.4.1 DivX 编码 / 解码器

DivX 实际就是视频压缩技术，它由微软 Mpeg4 v3 修改而来，使用 Mpeg4 压缩技术，并同时分离视频和音频。DVDRip 的核心部分便是由 DivX 对 DVD 音视频进行压缩，生成 Mpeg4 视频格式文件（也就是我们常见的 AVI 格式）。 Mpeg4 原本是美国禁止出口的编码技术，于是微软将这个 Codec 仅仅用于 Windows Media 流媒体技术（即网络常见的 ASF 格式文件），不再用于 AVI 文件。ASF 文件虽然有一些好处，但是既没有编辑软件，播放时又不能精确定位，播放器也只有唯一的 Windows Media player，自己动手制作电影光盘的黑客们对此很不满意，于是一位黑客 CEJ 修改了微软的 MS Mpeg4 v3，解除了用于 AVI 文件的限制（其实 Mpeg4 v3 不能用于 AVI 没有技术上的原因，纯粹是微软在里面加了个锁），并调整了一些压缩参数。这样，就诞生了最早我们熟悉的 Mpeg4 编码器 DivX。

目前的 DivX 编码器已经引入了一些新的概念和很多新的技术。如 Variable Bitrate（动态比特率）。它能分配较多的数据量给高速运动的场景。相应减少静态的场景的数据量。从而达到更大的压缩比；引入了 Psychovisual Enhancements（心理视觉增强）的概念，它的依据是：人的视觉系统在一幅图像中总是对某些特定的部分敏感度非常低而对其它特定的部分有较高的敏感度，因此就可以减少那些人们视觉敏感度低的部分的数据而分配给视觉敏感度高的部分；此外还有像 GMC（全局动态补偿）、Bidirectional Ecoding（双向编码）、Pre-Processing（预处理）等一系列的新技术。通过这些技术，DivX 比早期 Mpeg4 编码器有了更好的视觉效果和更高的压缩比，从而对 DVD 形成了更大的挑战。2001 年，DivX4 被推出，从 DivX4 开始，原来的"DivX 3.11;-)"笑脸符号就不再使用，而改成 DivX4.X 了，目前版本已经出到 5.10 以上。

2.4.2 Xvid 编码 / 解码器

DivX 广泛流行，成为 DVDRip 的标准，问题是，它的基础技术是黑客非法盗用微软的，只能在地下里流传却上不了台面，无法进行更广泛的产品化，更无法生产硬件播放机。在这种情况下，一些精通视频编码的程序员（包括原 DivX 3.11 的开发者）成立了一家名为 DivXNetworks Inc.的公司，简称 DXN。DXN 发起一个开放源码项目 ProjectMayo，目标是开发一套全新的、开放源码的 MPEG4 编码软件。特别是完全符合 ISO MPEG4 标准的 OpenDivX CODEC 吸引了许多软件高手参与，并很快开发出 OpenDivX 编码器和解码器原型，之后又开发出更高性能的编码器 Encore 2 等等。这一时期，主要编码工作是 DXN 的人在做，而许多技术难关的解决得力于来自开放源码社会的帮助。

就在一切都看起来进展顺利的时候，好戏上演了。ProjectMayo 虽然是开放源码，但不是依据 GPL（通用公共许可证，一种开放源码项目中常用的保障自由使用和修改的软件或源码的协议）。DXN 在设计授权协议时留了一手，2001 年 7 月，就在 Encore 2 基本成型，差不多可以产品化的时候，DXN 另搞了一个 DIVX.COM 网站，封闭了源码，发布了他们自己的 DivX 4。DivX 4 的基础就是 OpenDivX 中的 Encore 2，但利用了 DivX 的牌号，可以说出乎意料地摆了所有人一道。由于 DXN 不再参与，ProjectMayo 陷于停顿，Encore2 的源码也被 DXN 从服务器上撤下。经过激烈的争论，DXN 虽然承认 Encore 2 在法律上是开放的，但仍然拒绝把它放回服务器。开放源码社会就这样被狠狠地涮了一回。

OpenDivX 尚不能实际使用，而 DivX 4（以及后续的收费版本--DivX 5）等等都成了私有财产，许多人为打破微软垄断而无偿付出的智慧和劳动仅仅是帮助了 DXN 发财，这种结果当然是不能被接受的。为此，整个 0dayz 组织永远的拒绝了 DXN 公司的 DivX4\5，而原 OpenDivX 开发组中的幸存者，逐渐重新聚拢开发力量，在最后一个 OpenDivX 版本的基础上，发展出了 XviD。

劫后余生的 XviD 又度过了近 1 年时间，它继承并发展了 OpenDIVX Encore 2，性能得到极大提高，被认为目前世界上速度最快的 MPEG4 CODEC。XviD 重写了所有代码，并吸取前车之鉴依照 GPL 发布（注意不再是 LGPL，所以谁要是想用它做成产品而不开放源码是非法的）。不过，因为 MPEG4 还存在专利权的问题，所以 XviD 只能仿照 LAME 的做法，仅仅作为对如何实现 ISO MPEG-4 标准的一种研究交流，网站上只提供源码，如果要使用就要自己编译源码或者到第三方网站下载编译好的可运行版本。

2.4.3 AVI 文件格式

DivX 和 Xvid 编码器都可以用来压缩制作 AVI 文件，早期的 AVI 文件大多是基于 DivX 编码的，但是随着 Xvid 的出世和越来越壮大，用 Xvid 压缩制作 AVI 文件已成主流。本系统支持 AVI 文件的读取、传输和播放，因此在这里介绍一下 AVI 文件的格式。

AVI 文件是 Microsoft 公司指定的一种 RIFF(Resource Interchange File Format)文件格式，AVI 文件的基本结构是块（chunk），块有块头和块体两部分构成，块头有 8 个字节，前 4 个字节时标示字段“RIFF”，后 4 个字节是一个无符号的长整数，即块体的长度，紧随其后的是块体，其所含字节数是块体长度的值，块体可能是基本数据构成的数组，也可能是再嵌套一个具有同样块头、块体结构的子块，这种嵌套结构具有递归性质，因此，可能形成多层子块结构，但是不管如何嵌套，最底层的子块的块体必须是有基本数据构成的数组。

```
RIFF  'AVI ' {                    //RIFF 标识，AVI 类
   LIST   'hdrl' {                //第一个一次构造块
      avih( )                     //AVI 文件的头
      LIST   'strl' {             //二次块，视频流
         strh( )                  //视频流的头
         strf( )                  //视频流格式块
       }
      LIST   'strl' {             //二次块，音频流
         strh( )                  //音频流的头
         strf( )                  //音频流格式块
       }
   }
   LIST   'movi' {                //第二个一次构造块，记录块
      LIST   'rec ' {             //二次构造块
         ...                      //音频或者视频数据块序列
      }
      idxl ( )                    //可选的索引块
   }
}
```

图 2.8 AVI 文件格式

典型的 AVI 文件结构层次描述如图 2.8 所示，视频流和音频流分别是 BMP 格式和 WAV 格式。

2.5 非线性编辑

非线性编辑（DirectShow Editing Services，简称 DES）[30]，是一套基于 DirectShow 核心框架的编程接口。DES 的出现，简化了视频编辑任务，弥补了 DirectShow 对于媒体文件非线性编辑支持的先天性不足。但是，就技术本身而言，DES 并没有超越 DirectShow Filter 架构，而只是 DirectShow Filter 的一种增强应用。我们可以从图 2.9 中了解到 DES 在我们整个多媒体处理应用中的位置。

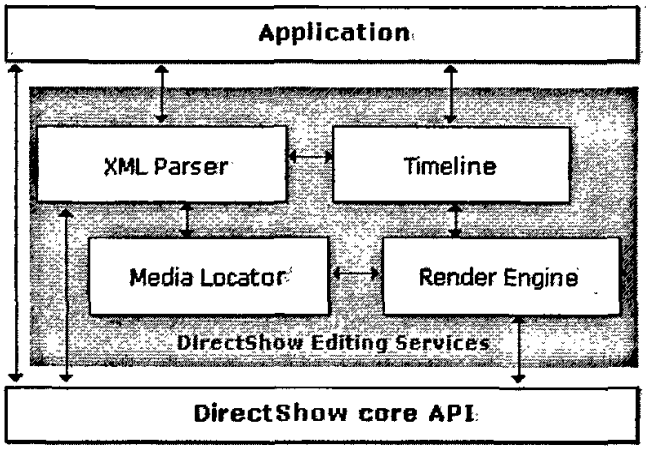

图 2.9 DES 的系统结构

下面，我们举个例子来看一下 DES 能够给我们带来些什么。假如我们现在有三个文件 A、B 和 C，使用这三个文件做成一个合成的文件。我们想取 A 的 4 秒钟的内容，紧接着取 B 的 10 秒钟的内容，再紧接着 C 的 5 秒钟的内容。如果仅仅是这样，我们直接使用 DirectShow Filter 是不难实现的。一般情况下，应用程序会维持各个文件的编辑信息，由应用程序根据这些信息动态创建 / 控制功能单一的 Filter Graph，以顺序对各个文件进行处理。但是，如果我们的"创意"是随时改变的，我们现在想让 C 在 B 之前出现，或者我们想取 A 的不同位置的 10 秒钟内容，或者我们想给整个合成的文件加上一段美妙的背景音乐。如果我们仍然直接使用 DirectShow Filter 去实现，情况就变得很复杂了。然而，对于 DES，这真的是小菜一碟。我们只需将所有的编辑信息以 DES 提供的接口告诉 DES，其它的如 Filter Graph 的创建 / 控制输出，就完全交给 DES 来负责吧。这时候，DES 创建的 Filter Graph 带有各个 Source 输出的控制功能，一般比较复杂。

使用 DES，我们可以得到如下的便利：

1）基于时间线（Timeline）的结构以及 Track 的概念，使得多媒体文件的组织、编辑变得直观而高效；

2）支持即时的预览；

3）视频编辑项目支持 XML 文档的形式保存；

4）支持对视频 / 音频的效果处理，以及视频之间切换的过渡处理；

5）可以直接使用 DES 提供的 100 多种 SMPTE 过渡效果，以及 MS IE 自带的各种 Transform、Transition 组件；

6）支持通过色调、亮度、RGB 值或者 alpha 值进行图像的合成；

7）自动对源文件输出的视频帧率、音频的采样率进行调整，直接支持视频的缩放。

接下来，我们来看一下 DES 的结构（Timeline 模型），如图 2.10 所示：

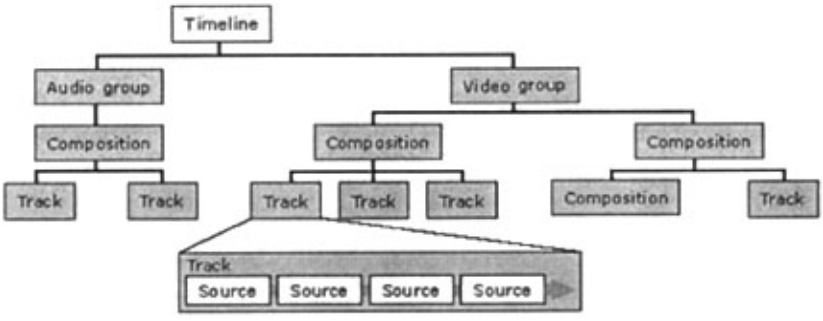

图 2.10 DES 的内部结构模型

这是一个树形结构。在这棵树中，音视频文件是叶结点，称作为 Source；一个或多个 Source 组成一个 Track，每个 Track 都有统一的媒体格式输出；Track 的集合称作为 Composition，每个 Composition 可以对其所有的 Composition 或 Track 进行各种复杂的编辑；顶级的 Composition 或 Track 就组成了 Group；每个 Group 输出单一格式的媒体流，所有的 Group 组成一个 Timeline，Timeline 表示一个视频编辑的项目，它是这棵树的根节点。一个 Timeline 项目必须至少包含一个 Group，最典型的情况一般包含两个 Group：Audio Group 和 Video Group。图 2.11 是一个典型的基于 Timeline 的 Source Track 编排。

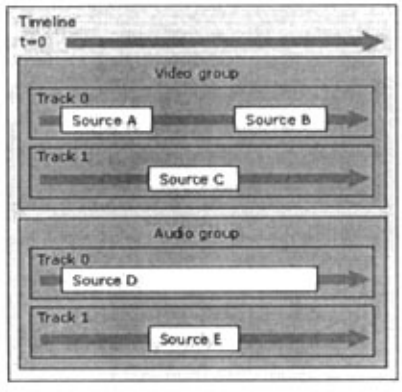

图 2.11 基于时间线的媒体轨道模型

其中，箭头方向即是 Timeline 的方向。这个 Timeline 由两个 Group 组成，每个 Group 中包含两个 Source Track。在 Group 中，Track 是有优先级的（Track 0 具有最低的优先级，依次类推）。运行时，总是输出高优先级的 Track 中的 Source 内容。如果此时高优先级的 Track 中没有 Source 输出，则让低优先级的 Track 中的 Source 输出。如上图中 Video Group 的输出顺序为 Source A->Source C->Source B。而对于 Audio Group，它的所有 Track 的输出只是简单的合成。

我们再看一个典型的 Track 之间加入了 Transition 的 Timeline 结构，如图 2.12 所示：

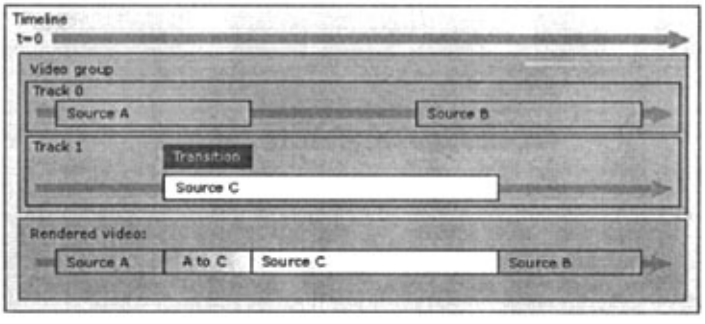

图 2.12 加入过渡处理的轨道模型

图中，Video Group 中是两个 Track 以及 Track 上几个 Source 的编排；Rendered video 中表示这个 Group 最终输出的效果。我们可以看到，在 Track 1 上有一个 Transition，表示这个时间段上从 Track 0 过渡到 Track1 的效果。一般，Transition 位于高优先级的 Track 上。Transition 也是有方向的，默认是从低优先级的 Track 过渡到高优先级的 Track。当然，我们也可以改变 Transition 的方向。如图 2.13 所示，第一个 Transition 是从 Track 0 到 Track 1，第二个 Transition 是从 Track 1 到 Track 0。

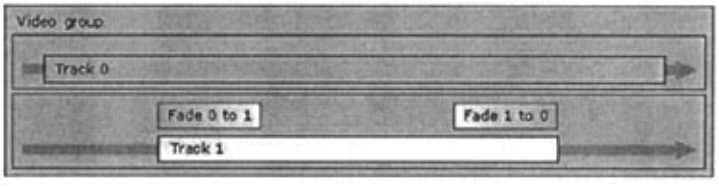

图 2.13 过渡的两种方向

3 网络视频服务系统的关键技术研究

本章将介绍为构建一个分布式 VoD 系统所采用的两个关键技术：服务器间的连接以及它们的数据传输，即系统的分布式结构的构建，这是首要解决的技术；其次，是为这个结构所设计的负载平衡算法，该部分将研究如何使工作任务在服务器间的分配更加合理，从而使它们更好地协同工作。

3.1 分布式结构的构建

分布式的构建从服务器间的连接入手，而服务器间的连接可从原系统的服务器和客户端的 RTSP 连接入手。研究和理解原系统的连接和数据传输机制，将有助于利用现有的成熟的资源，构建服务器间的连接。

原系统是一个集中式的 VoD 系统，服务器端和客户端采用 RTSP 连接来传输控制和交互操作等信息。RTSP 协议构建于 TCP 协议之上，因而它是一种面向连接的可靠的传输方式。而服务器和客户端的视频数据传输则采用 RTP 协议，RTP 协议构建于 UDP 协议之上，是一种无连接的协议，适合于传输数据量大而又实时性高的数据。原系统的服务器和客户端的连接交互如图 3.1 所示，其中 RTSP Server 和 RTSP Client 分别对应于 Socket 编程的 Server 端和 Client 端。

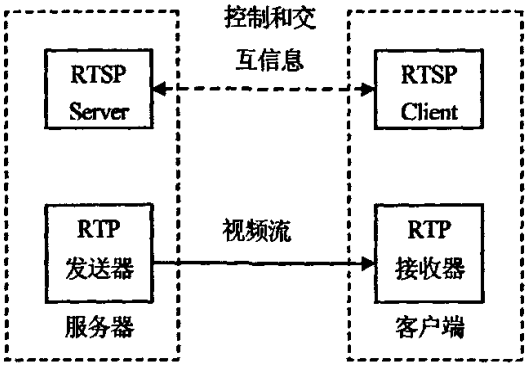

图 3.1 服务器和客户端的连接交互

在服务器端，RTSP Server 可接收多个用户（客户端 RTSP Client）的连接请求，并建立一个用户队列，管理这些 RTSP 连接。从而，服务器和客户端形成一对多的关系，即一台服务器可为多个用户提供服务。如图 3.2 所示。

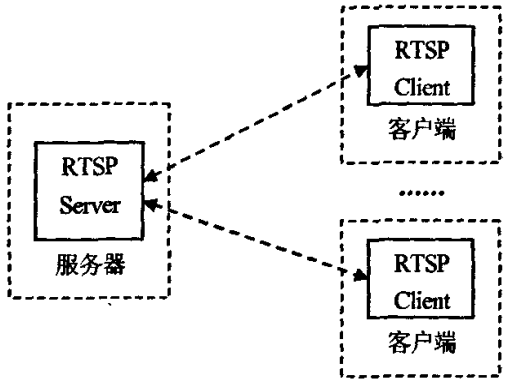

图 3.2 服务器和客户端一对多的关系

利用 RTSP Server 可以与多个 RTSP Client 建立连接的观点，我们把这种一对多的思想推广到服务器之间。为此，我们在服务器端添加了一个 RTSP 客户端，叫做 Server Client，它的创建、连接、通信方式和作用均与 RTSP Client 类似，只是在传输信息的内容和处理上与 RTSP Client 有不同。Server Client 用于与目标服务器（可以是除自身外的其他任一服务器）进行 RTSP 连接，通过它，服务器间可以交换负载状况和管理控制等信息。

虽然任意两台服务器之间均可以建立连接，但是它们之间的连接也不是盲目的，应当遵循一定的原则，使系统的结构较为简单，这样利于管理也方便扩展和改进。在中小规模的 VoD 应用中，我们可以采用“主一从服务器”的连接方式，以其中一台服务器作为主服务器，其他服务器作为从服务器，与主服务器进行 RTSP 连接，从而构成集群系统，如图 3.3 所示。

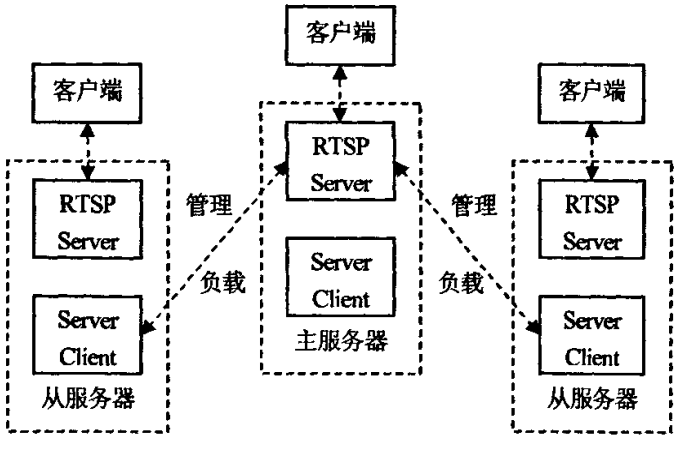

图 3.3 主一从服务器结构的连接方式

在该结构中，主从服务器在软件的设计上可以无差别，都有一个 RTSP Server 端和一个 Server Client 端，都可以为用户提供视频服务。但是，作为从服务器的 Server Client 端需与主服务器的 RTSP Server 端进行连接，传递负载、管理等信息。主服务器除了为客户端建立一个用户队列之外，还建立一个（从）服务器队列，用于管理与之相连的各从服务器。由于客户端的 RTSP Client 和从服务器的 Server Client 均与主服务器的 RTSP Server 相连，为区分彼此，客户端和从服务器向主服务器发送连接请求时，虚传送一个标志位，表明该连接是用户还是服务器。

主从结构的 VoD 系统构造简单，管理方便。由于所有服务器均与主服务器相连，主服务器可随时了解整个系统的运行状况，负载平衡和控制管理都很方便。同时，从服务器只作为视频服务器使用，无需参与其他事务，可专注于视频服务。这种分工上的明确也使得系统在扩展上非常方便，当系统的服务能力达到上限时，只需开启一台服务器，与主服务器进行连接，主服务器便把它纳入到集群当中，并为其分配工作负担。

在故障恢复方面，当其中一台从服务器因故障而不能提供服务时，主服务器会把该服务器上的工作负担转移到其他服务器上，使中断的服务在短时间内继续进行，保证了系统的可靠性和服务质量。但同时，这种结构也为系统带来了一个比较大的风险，当主服务器发生故障时，整个系统便瘫痪了，此时的解决办法是从其他服务器中找出一台作为主服务器，然后重新构建集群系统。

主从结构（二层结构）的连接思想可以推广到更多层次上，构建树型的 VoD 系统，如图 3.4 所示。

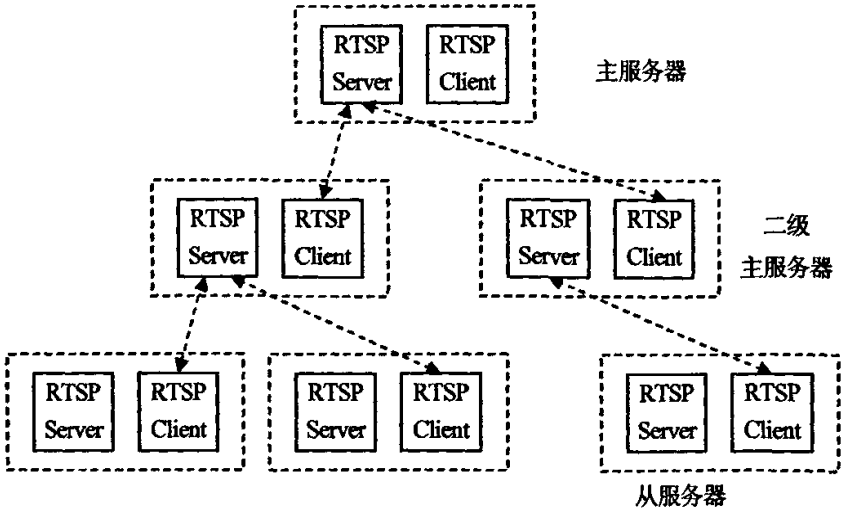

图 3.4 树型服务器结构的连接方式

树型结构的 VoD 系统由于层次更多，不同的层次可设置在不同的网络中。例如主服务器和二级主服务器可设置在高速内部网络中，而从服务器可设置在靠近用户的网络中，系统在调度时可根据各服务器与用户距离的远近（或网络带宽的大小），优先选择靠近用户的一台服务器，为用户提供服务。这样的结构更接近于实际比较复杂的网络状况，如内容分发网络（Content Dilivery Network，简称 CDN），因而更适合在这些网络中进行传输。

但同时，树型 VoD 系统的结构更复杂，管理和调度也更复杂，尤其在服务器连接策略（层次设置策略）、节目存储策略和负载平衡策略上，要考虑的维数更多，而且这几种策略相互影响、相互制约，要使系统达到一个比较好的状态，需要对这几种策略进行权衡。

考虑到其中的复杂程度和个人能力的限制，本系统主要依据主从服务器的结构来构建，并设计系统的服务机制。

3.2 负载平衡算法研究

根据上一节的主从服务器系统结构，采用了以下两种负载平衡算法：

3.2.1 传统动态反馈负载平衡算法

目前分布式 VoD 系统所采用的负载平衡算法大多为基于 LVS[32]的动态反馈负载平衡算法，这些算法的基本思想是：

首先考虑服务器之间可能存在的性能差异，赋予每一台服务器一个权重系数，系数越大，表示服务器的性能越高。其次考虑服务器当前的负载状况，主要为服务器的几个性能指标（均是百分比）：CPU 使用率 P，内存使用率 M，一定时间内磁盘 I/O 操作次数占所有服务器同类操作次数的百分比 D，当前服务用户数占所有服务器服务用户总数的百分比 S，并为这些指标分配权重系数分别为∂p、∂m、∂d、∂s（$\partial i<1$，$\Sigma \partial i=1$），则服务器负载 L 为：

$$L=\partial_p *P+\partial M *M+\partial_d *D+\partial_s *S。$$

假设选出服务器 j 来处理新的请求，那么服务器 j 应满足

$$Lj/Wj=\min（Li/Wi，i=1，2，...）。$$

其中，L 为服务器的负载，W 为服务器的权重系数。

例如，在视频点播系统中，服务器性能指标可采用以下权重系数（0.2，0.2，0.4，0.2），这里认为服务器的磁盘 I/O 操作次数较其它指标重要一些。若当前的系数不能很

好地反映当前应用的负载，系统管理员可以对系数不断地修正，直到找到贴近当前应用的一组系数。

为采集服务器的各项性能指标，在 VoD 服务器端程序添加一个负载参数采集模块，运行在每个服务器上。此模块实时收集自身负载信息，并周期性地传送给中心服务器。关于参数采样周期，虽然很短的周期可以更确切地反映各个节点的负载，但是过于频繁地采集（如每秒多次）会给平衡器和节点带来负担，也可能增加不必要的网络负荷。另外，由于采集模块是在采集时刻进行负载计算的，平衡器反映出来各个节点的负载信息会出现剧烈的抖动，平衡器无法准确捕捉节点真实的负载变化趋势。为此，本系统对于 CPU 使用率和内存使用率，每秒采样一次，然后取 5 秒钟内的平均值；对于磁盘 I/O 操作次数和连接数，则每 5 秒钟采样一次，取 5 秒钟内的总和；各服务节点以 5 秒钟为周期向中心服务器发送负载信息。

3.2.2 新负载平衡算法

传统负载平衡算法只是单纯地根据各个服务器的负载状况来分配工作负担，没有考虑到单播方案资源耗费大，而 EEB 方案是分块广播方案，没有用户数目限制的特点。这种算法在系统仅使用单播方案时是高效的，能使系统各服务器间负载达到很好的平衡。但是，对于采用多种信道调度方案的分布式 VoD 系统，由于传统算法对各种信道调度方案不加以区别分析，它有可能不会把用户的服务请求分配给提供 EEB 服务的服务器，而是分配给提供单播服务的服务器，由该服务器新开辟一条信道为用户服务，这就会造成系统资源和网络带宽的白白浪费，如图 3.5 所示。

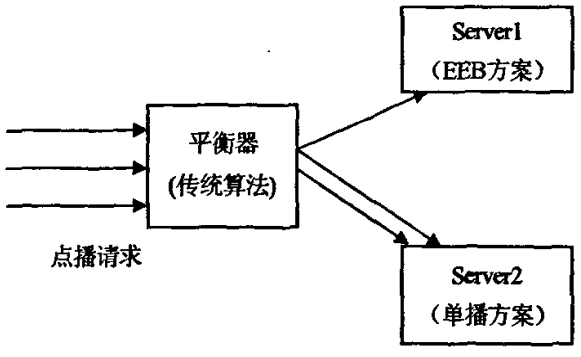

图 3.5 传统负载平衡算法示意图

因此，我们提出一种改进的负载平衡算法，基本思想是：在传统负载平衡算法的基础上把各种信道调度方案的特点考虑进去，即对于同一个节目，当系统中各服务节点采

用不同的信道调度方案时，把用户的服务请求分配给采用 EEB 方案的服务器，如图 3.6 所示。目标服务器收到用户请求后，即把该用户加入广播组，这样用户便可以在指定的广播信道上获取节目数据，不会增加服务器和网络的负担，这将有利于系统整体性能的提高，从而使系统能够支持更大的并发用户数量。

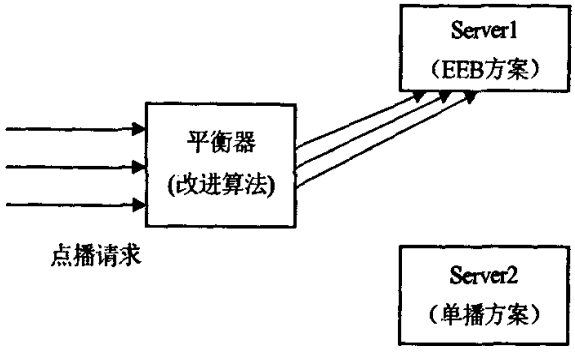

图 3.6 改进负载平衡算法示意图

改进负载平衡算法的流程如下：

1）用户点播节目时，通过客户端向中心服务器发送一个点播请求，该点播请求包含了用户的认证信息和节目的 URL 等信息；

2）中心服务器收到点播请求，首先对用户进行认证。在确定了用户的合法性和点播的有效性后，中心服务器根据当前各个视频服务器和自身的状态，在考虑节目信道调度方案和负载等因素后，选择出一个最合适的服务器（目标服务器）。具体方法是：首先中心服务器向各视频服务器发送查询通知，各视频服务器将返回各自指定节目的信道调度方案以及负载状况等信息；接着中心服务器判断所有服务器中，是否有能提供用户请求节目的 EEB 方案的服务器，如果有，则确定该 EEB 方案服务器为目标服务器；如果没有，则通过第 2.1 节介绍的负载计算方法，找出所有服务器中负载最轻的一个，作为目标服务器；假如此时所有服务器都处于重载状态，则向用户发送一个拒绝服务信息；

3）若目标服务器是中心服务器本身，则接受点播请求并开始为用户提供视频服务；若不是，则向用户发送一个转移服务的信息（内容包括目标服务器的 IP 地址和端口等信息），通知该用户把点播请求发送给目标服务器；

4）若用户（客户端）收到转移服务信息，则再次形成一个请求点播信息，发送给目标服务器。目标服务器收到点播请求后，直接为该用户提供视频服务。

改进算法是在传统算法的基础上增加一层对节目信道调度方案的判断，并无过多的数据计算和数据传输，因此改进算法在时间复杂度和空间复杂度上与传统算法相差无几，实际运行效果良好。

在分布式 VoD 系统中，为了负载分流和提高系统的容错能力，往往把同一个节目复制到多个甚至所有服务节点上，由多个服务节点提供同一节目的服务。一开始所有节目可能都采用单播方案，而对于一些热门节目，随着点播用户数量的增加，系统管理员或服务程序会把一些节目设为多播方案（例如 EEB）进行传输。但是出于节省系统资源和 EEB 方案没有用户数目限制上的考虑，对于同一个节目，只需把其中的一个服务节点该节目设为 EEB 方案即可，其他服务节点该节目调度方案保持不变。此时，上述改进算法便会发挥其效能了。

3.2.3 两种负载平衡算法的性能分析

为便于分析，定义如下参数：

N：服务器上的节目总数；

λ_i：第 i 个节目的用户点播请求强度（为 Poisson 分布参数）；

S：系统并发信道占用总数；

n：采用 EEB 方案节目的个数，n<N；

di：第 i 个节目的 EEB 分块数。

从节目流行度服从 Zipf[33]分布可以得到 $\lambda_i = \frac{1}{i^{\alpha}} \times \left(\sum_{j=1}^{N} \frac{1}{i^{\alpha}} \right)^{-1}$，$\alpha$ 为 Zipf 参数，一般取 0.271。当点播用户总数为 M 时，第 i 个节目的点播数为 $M^* \lambda_i$。下面分三种方案进行分析（均不考虑用户中途退出和节目播放完毕而停止的情况）：

方案 1 当系统中所有节目均采用单播方案时，改进算法和传统算法无差别。当用户点播总数为 M 时，系统的并发信道数为 $S = M$。

方案 2 当系统中采用 EEB 方案节目的个数为 n，其余节目为单播，且使用改进负载平衡算法，则当用户点播总数为 M 时，系统的并发信道数为

$$S = M\left(1 - \sum_{i=1}^{n} \lambda i\right) + \sum_{i=1}^{n} di \qquad (3.1)$$

其中 $\sum_{i=1}^{n} di$ 为 n 个 EEB 方案节目的分块数之和，$0 \leqslant \sum_{i=1}^{n} \lambda i \leqslant 1$，为这 n 个 EEB 方案节目的点播概率。

方案3 当系统中采用EEB方案节目的个数为n，其余节目为单播，且使用传统负载平衡算法时。由于传统算法在任务分配的时候不考虑各信道调度方案的特点，因此它的效率有起伏。最好的情况：即把所有点播EEB方案节目的服务请求，都由相应EEB方案的服务器来提供服务，此时方案3的并发信道占用数同方案2；而最坏的情况：即把所有点播EEB方案节目的服务请求，都开辟单播信道进行传输，此时方案3的系统并发信道占用数为$S = M + \sum_{i=1}^{n} di$。在实际运行中，方案3在减少系统并发信道占用数方面的性能介于方案1和方案2之间。

在以M为横坐标轴，S为纵坐标轴的坐标系（即“点播用户数－系统并发信道数”坐标系）中，公式(1)是一条以$-\sum_{i=1}^{n} \lambda i$为斜率的直线，减小其斜率可以使系统并发信道数S减小，而S的减少意味着能节省更多的系统资源和网络带宽资源，从而使系统能同时为更多的用户服务。减小直线(1)的斜率有以下两个途径：

1）增大n，即增加采用EEB方案的节目的个数，使$\sum_{i=1}^{n} \lambda i$增大，则斜率$-\sum_{i=1}^{n} \lambda i$减少。

2）由λ_i的计算公式知增大Zipf参数α也可使λ_i增大，从而使$\sum_{i=1}^{n} \lambda i$增大，斜率$-\sum_{i=1}^{n} \lambda i$减少。

4 网络视频服务系统的设计与实现

本章将从系统的点播流程、动态扩展、服务器端和客户端的设计实现等方面，详细描述系统的总体结构及其搭建过程；将从系统的负载平衡机制、节目的存储策略及其流调度方案的自动调整等方面，详细描述系统的服务机制和所用到的一些关键算法；另外，还将从客户端的解码能力方面描述如何提高系统的服务性能。

4.1 系统的总体结构

本文提出的网络视频服务系统是一个分布式 VoD 系统，由一台中心服务器和一些视频服务器所组成，它们通过快速以太网交换机相连，构成集群系统，如图 4.1 所示。中心服务器是整个系统的核心，负责整个系统的管理工作。中心服务器一方面作为 Web 服务器的载体，负责生成节目导航信息，提供节目检索、节目点播等交互界面，是用户点播节目的唯一入口；另一方面，它还是控制服务器，监视各个视频服务器的状态，协调各个服务器间的通信，负责节目的流行度统计、负载平衡、动态扩展等功能的实现。一般情况下，中心服务器完成系统的管理并不需要花费太大的工作量，因此出于提高系统利用率和成本上的考虑，中心服务器还可以作为一个视频服务节点，兼顾向客户提供视频流服务。

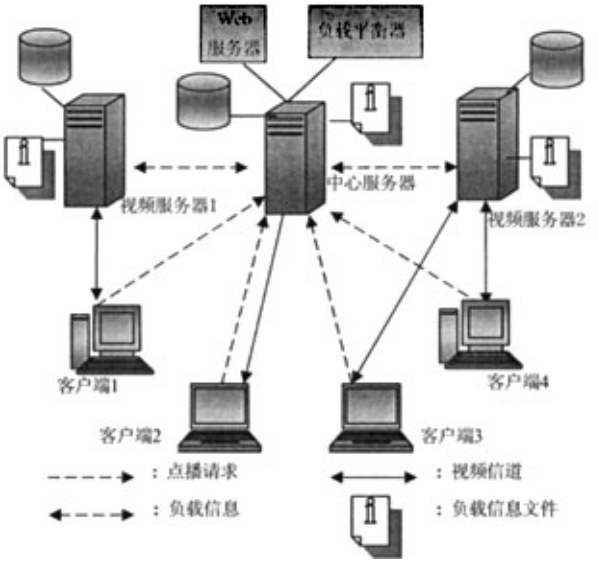

图 4.1 系统总体结构

系统从原集中式 VoD 系统经过扩展改进而来，设计思想在于负载分流和动态扩展。当有用户点播请求时，中心服务器通过其 Load Manager 模块中的负载平衡算法，选择出一台最合适的视频服务器，向用户提供视频流服务，让负载科学地分配到各个视频服务器上。在这样的体系结构中心服务器和视频服务器耦合度小。当总负载超过系统能提供的上限时，可以在不改变已有系统的基础上，通过动态添加视频服务器来满足需求，具有良好的可扩展性。在硬件配置上中心服务器和其他视频服务器可以没有区别，本系统均采用普通 PC 机作为服务器便可使系统发挥很好的效能。

在软件上系统主要由 VoD 服务器和 VoD 客户端组成，采用基于 IP 协议的适合多媒体传输的协议进行信息交互和数据传输，包括实时流协议（RTSP）、实时传输协议（RTP）和会话描述协议（SDP）。系统使用 RTSP 实现服务器和用户之间控制信息的交互，如初始化、节目播放、交互操作等，但它自身不参与媒体数据的传输，而是依靠底层的 RTP 协议来实现媒体数据的传输。SDP 协议用于描述网络上传输的数据的特定信息，这些信息包括媒体类型、信道调度策略、播放的地址和端口等，系统在 RTSP 服务器与客户端传输节目信息时使用了该协议。系统的实现模型如图 4.2 所示。

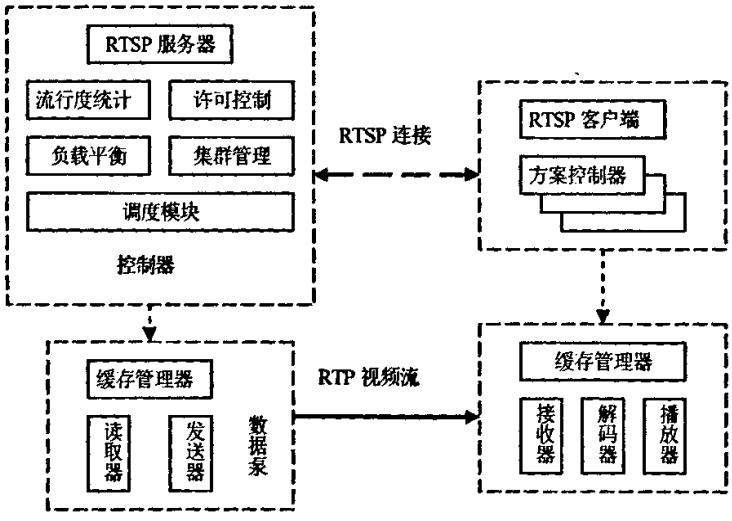

图 4.2 系统实现模型

系统采用了单播、CM 和 EEB 等多种流调度方案，适合对不同流行度的节目进行分类调度，以利于突破服务器性能瓶颈和网络带宽瓶颈，从整体上提高系统的性能。系统的服务器端和客户端都有一个方案控制器，其中，服务器的方案控制器决定每个节目的调度策略，它可以根据每个节目的点播情况，动态地改变节目的调度策略；客户端方案控制器则根据服务器中节目调度策略的不同，采用相应的接收策略。

4.1.1 点播流程

系统采用 B/S 和 C/S 的混合点播结构，方便用户使用，点播时，用户只需等待几秒钟的启动延时便可收看到节目。点播的流程如下：

1）首先用户通过浏览器（例如 IE）浏览或者查询节目信息。当用户决定点播一个节目的时候，浏览器会通过一个 ActiveX 控件，启动 VoD 客户端，并同时向客户端传送用户的认证信息及节目的 URL；

2）客户端启动后，把用户的认证信息及节目的 URL，根据 RTSP 协议的格式打包，形成一个点播请求信息，并发送给中心服务器；

3）中心服务器收到点播请求，首先对用户进行认证。在确定了用户的合法性和点播的有效性后，中心服务器根据当前各个视频服务器和自身的状态，在考虑用户与服务器的位置、网络和负载等多种因素后，选择出一个最合适的服务器（目标服务器）；

4）若目标服务器是中心服务器本身，则接受点播请求并开始为用户提供视频服务；若不是，则向用户发送一个转移服务的信息（内容包括目标服务器的 IP 地址和端口等信息），通知该用户把点播请求发送给目标服务器；

5）若用户（客户端）收到转移服务信息，则再次形成一个请求点播信息，发送给目标服务器。目标服务器收到点播请求后，直接为该用户提供视频服务。

6）客户端收到服务器发送来的视频数据后，经过短暂的初始缓存，便可启动播放器播放节目。

4.1.2 动态扩展

动态扩展是指在不改变原来系统的正常运行的情况下，通过添加视频服务器的方式提升系统的性能。

本系统的中心服务器和其它视频服务器之间是一对多的松耦合关系，中心服务器管理和维护多个视频服务器的状态。当要向系统添加新的视频服务器时，先向中心服务器发出注册请求，注册请求包括该新服务器的 IP 及所能提供视频服务的节目信息等。中心服务器收到注册请求后，把该视频服务器纳入管理范围，在以后处理点播请求的时候就会合理地把部分点播请求分配给它处理，使负载得到分流。如果要停止一台视频服务器的服务，只要向中心服务器发出注销请求即可，不会影响系统的正常运行。中心服务器和视频服务器这种松耦合的关系增强了系统的可扩展性，让视频服务器真正做到“即插即用”。

为实现动态扩展，在启动服务器程序时，先选择服务器的启动方式：作为主服务器还是作为从服务器加入集群，如图 4.3 和图 4.4 所示。若选择前者，则服务器直接正常启动（启动视频服务功能），并开启负载平衡功能和集群管理功能。

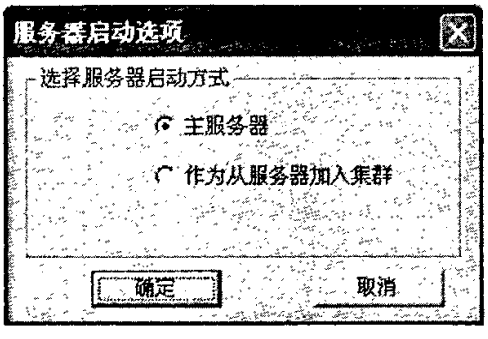

图 4.3 启动主服务器

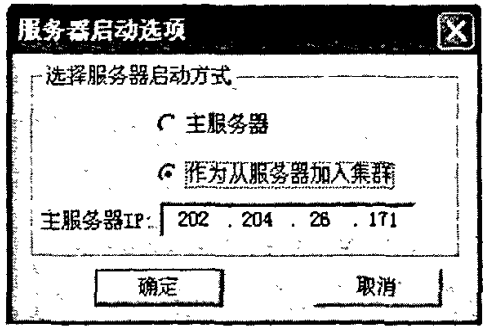

图 4.4 启动从服务器

若选择作为从服务器加入集群，则先输入主服务器的 IP 地址，并向主服务器发送连接请求，建立 RTSP 连接，然后启动视频服务功能，而负载平衡功能和集群管理功能处于未启动状态，表明该服务器仅作为视频服务器，加入到集群当中。

主服务器的集群管理功能是通过服务器链表 ServerLink 的建立和维护来完成的，它保存有各视频服务器当前的运行状态等信息，对它的查询便可了解整个集群的运行状态。主服务器会周期性地向各视频服务器发送查询指令，使 ServerLink 得到更新。ServerLink 中有一个重要的数据结构 STATUS_SERVER，用于保存服务器的各种相关信息，它的定义如下：

```
typedef struct _status_server
{
  SOCKET sockID;
  ULONG sessionID;        //会话号
  CString addr;           //服务器地址
  int port;               //端口号
  int iCPURate;           //CPU 使用率
  int iMemoryRate;        //内存使用率
  ULONG ulDiscIOCount; //磁盘读写次数
  int iConnectCount;      //连接数
  float fLoad;            //负载因子 L=λp*P+λs*S+λm*M+λd*D
  float fWeigh;           //服务器性能权重系数
  float fRatioLW;         //  fLoad / fWeigh 的比值
  bool bAvailable;        //服务器是否能提供服务
  long lProgramID;        //用户请求的节目 ID
```

```
    int   iSchemeID;          //节目的调度方案编号
    int   iState;             //节目当前的调度状态
    SERVER_STATUS curstate; //服务器的状态，为枚举类型：轻载、一般、重载
    UINT curseq;
    BYTE statusid;
    LONG statisticID;
    struct _status_server *next;
}STATUS_SERVER;
```

4.1.3 服务器端的设计与实现

本系统是在原集中式 VoD 系统的基础上经过扩充改进，增加了分布式机制和负载平衡机制而来。本节主要介绍系统对 RTSP 协议的扩展，使其支持对服务器运行状况信息的解析和传递，从而使各视频服务器能够纳入中心服务器的管理，形成整体（VoD 集群）。关于本系统服务器端的调度机制等其他部分，与原系统保持一致，详细情况可参考文献[31]。

如第 2.1 节所述，RTSP 协议只提供协议框架，系统可根据具体的应用对 RTSP 协议进行扩展。在原系统中，使用 RTSP 协议作为流式传输的控制协议，主要是控制节目流的开始、暂停和停止等交互操作，因此已经实现了 RTSP 协议的 OPTIONS、DESCRIBE、PAUSE、SETUP、PLAY、TEARDOWN 等方法。如 DESCRIBE 方法，是客户端发送给服务器的请求消息，本系统中用于给服务器发送用户的信息，并且从请求的 URL 中检索描述信息，DESCRIBE 的请求应答对组成 RTSP 的初始阶段，格式如下：

客户发给服务器的请求：

```
DESCRIBE rtsp://vod.ncut.edu.cn/first.avi RTSP/1.0
CSeq: 1
```

服务器返回给客户的响应：

```
RTSP/1.0 200 OK
CSeq: 1
Date: 23 Jan 2005 15:35:06
Content-Type: application/sdp
Content-Length: 376
```

一次完整的点播过程需要 RTSP 协议的 DESCRIBE、SETUP、PLAY 和 TEARDOWN 方法在服务器和客户端之间传递。为使服务器之间能够使用 RTSP 协议传递负载等信息，我们也采用类似的格式定义自己的方法：

1）Request Loadbalance 方法：

中心服务器需要查询各视频服务器负载信息时，向各视频服务器发送查询请求：

Request Loadbalance

Programname = 奥迪

其中，Programname 是用户请求的节目名称。

当各视频服务器收到 Request Loadbalance 查询请求后，先按照 Programname 查询本机上该节目的调度方案，然后读取本机的负载等信息，按照如下格式向中心服务器发回消息：

LoadBalance Parameters:

CPURate = 5

MemoryRate = 60

DiscIOCount = 2850

ConnectCount = 32

ServerWeigh = 0.3

ProgramID = 108

SchemeID = 4

State = 播放中

其中，前四个参数为服务器性能指标，ServerWeigh 为本服务器的权重系数，ProgramID 为用户请求的节目在本服务器上的 ID，SchemeID 为该节目在服务器上所采用的调度方案，State 为该节目当前的调度状态。

中心服务器收到各视频服务器返回的消息后，将分别解析，得到各个参数的值，然后再分别保存到服务器队列 ServerLink 的结构体中，以进行下一步的负载计算或任务调度。

2）Transfer Server 方法：

作用是通知用户把请求转发到目标服务器，该方法在中心服务器进行负载平衡时，作任务调度用。格式如下：

Transfer Server=202.204.26.175

ProgramID=32

其中等号后面的值为目标服务器的 IP 地址。用户（客户端）接收到这个消息后，便重新生成一个点播请求消息，发向目标服务器。

图 4.5 所示为 VoD 服务器运行时的系统监控界面。在该界面中可以监控所有在本服务器上点播的用户的节目调度情况。

图 4.5 VoD 服务器运行界面

4.1.4 客户端的设计与实现

与服务器端相对应，客户端也对 RTSP 协议的扩展做好了准备。本节主要介绍客户端是如何解析扩展后的 RTSP 协议，并根据协议的指示来执行的。客户端的接收、缓存及播放等模块与原系统保持一致，详细情况可参阅文献[31]。

当客户端收到 Transfer Server 消息，先从消息中解析出目标服务器的 IP 地址和请求节目的 ID 号，接着与中心服务器断开 RTSP 连接，然后重新形成一个 DESCRIBE 请求，发往目标服务器。接下来的过程与一次正常的点播一致。

整个服务转发的点播过程会在很短的时间内完成，这与不进行服务转发的点播所需的时间相差无几。对用户来说，这一过程是透明的，他不知道节目数据从哪而来，感觉就像只有一台服务器向他提供服务一样。客户端的播放界面如图 4.6 所示：

图 4.6 VoD 客户端播放器界面

4.2 负载平衡的设计与实现

中心服务器的 Load Manager 模块是系统的负载平衡器。平衡器作为网络请求分配的控制者，主要根据集群节点的当前处理能力，采用一定的策略对网络服务请求进行调配，并且在每个服务请求的生命周期里监控各个节点的有效状态。而负载平衡算法是其中的关键，它起到任务调配的作用。本系统主要采用了两种负载平衡算法：传统负载平衡算法及其改进算法，它们的实现基础是对服务器负载指标的采集，这节我们将介绍服务器负载指标的采集和系统负载平衡的实现。

4.2.1 服务器性能指标的获取

我们通过本机系统服务（Native API）来探测本机系统信息：利用 NT（Windows 2000，XP，2003）下 ntdll.dll 中没有公开的 API 函数：NtQuerySystemInformation。该函数为我们提供了丰富的系统信息，同时还包括对某些信息的控制和设置。在这里我们主要用这个函数来获取系统的性能信息，所获取的信息保存在一个名叫 SYSTEM_PERFORMANCE_INFORMATION 的性能结构体中，该结构体为我们提供了 70 余种系统性能方面的信息，非常丰富，定义如下：

```
typedef struct _SYSTEM_PERFORMANCE_INFORMATION
{
  LARGE_INTEGER  IdleTime;                    //CPU 空闲时间；
  LARGE_INTEGER  ReadTransferCount;           //I/O 读操作数目；
```

```
LARGE_INTEGER   WriteTransferCount;          //I/O 写操作数目;
LARGE_INTEGER   OtherTransferCount;          //I/O 其他操作数目;
ULONG           ReadOperationCount;          //I/O 读数据数目;
ULONG           WriteOperationCount;         //I/O 写数据数目;
ULONG           OtherOperationCount;         //I/O 其他操作数据数目;
ULONG           AvailablePages;              //可获得的页数目;
ULONG           TotalCommittedPages;         //总共提交页数目;
ULONG           TotalCommitLimit;            //已提交页数目;
ULONG           PeakCommitment;              //页提交峰值;
.....           .....
.....           .....
ULONG           SystemCall;                  //系统调用次数;
}SYSTEM_PERFORMANCE_INFORMATION,
    *PSYSTEM_PERFORMANCE_INFORMATION;
```

由此，通过这个结构体我们便可按照如下方式来获取几个系统的性能指标：

一开始先装载 NtQuerySystemInformation 函数：

```
NtQuerySystemInformation = (PROCNTQSI)GetProcAddress(
                                        GetModuleHandle("ntdll"),
                                        "NtQuerySystemInformation") ;
```

然后获取 SystemPerformanceInformation：

```
status = NtQuerySystemInformation(SystemPerformanceInformation,
                                  &SysPerfInfo,sizeof(SysPerfInfo),NULL);
```

接下来便是获取系统各项性能指标：

1） CPU 使用率：在每个时钟周期内（设为 1s）计算一次前当 CPU 的空闲度：

```
//当前 CPU 空闲度 = 空闲时间 / 系统时间
dbIdleTime = dbIdleTime / dbSystemTime;
```

其中 dbIdleTime 和 dbSystemTime 分别是本次获取的 dleTime / SystemTime 与上个周期获取的 dleTime / SystemTime 相减的结果。

然后计算当前的 CPU 使用率：

```
//当前 CPU 使用率 =100 － （当前 CPU 空闲度 ×100）/ 处理器个数
 dbIdleTime = 100.0 - dbIdleTime * 100.0 /
                  (double)SysBaseInfo.bKeNumberProcessors;
```

2） 内存使用率：通过 GlobalMemoryStatus 函数来获取内存的使用情况：

```
//先声明一个内存状态变量，并设置它的长度
MEMORYSTATUS MemStat;
MemStat.dwLength = sizeof(MEMORYSTATUS);
//然后读取内存状态，其中 MemStat.dwMemoryLoad 的值便为内存使用率
GlobalMemoryStatus(&MemStat);
m_ulNewUsges = MemStat.dwMemoryLoad;
```

3） 磁盘的 I/O 操作次数：与获取 CPU 使用率的情况类似。在每个时钟周期内（设为 5s），记录一次磁盘读写操作的次数：

```
//读写操作次数 = 读操作次数 + 写操作次数
ULONG ulDiscReadAndWriteCount = SystemPerfInfo.ReadOperationCount +
SystemPerfInfo.WriteOperationCount;
```

然后与上个周期内的磁盘读写操作次数相减，即可得到本周期内的磁盘 I/O 操作次数：

```
m_ulDiscIOCount = ulDiscReadAndWriteCount - m_ulOldDiscReadAndWriteCount;
```

关于服务器当前（RTSP）连接数的获取，由于每个服务器端均创建有一个 RTSP 用户队列，因此只需获取这个队列的长度便可得到服务器当前的（RTSP）连接数。

服务器的负载指标采集和权值设置界面如图 4.7 所示，采集到的数据将通过服务器间的 RTSP 连接，汇总到中心服务器中，由它进行统一管理。

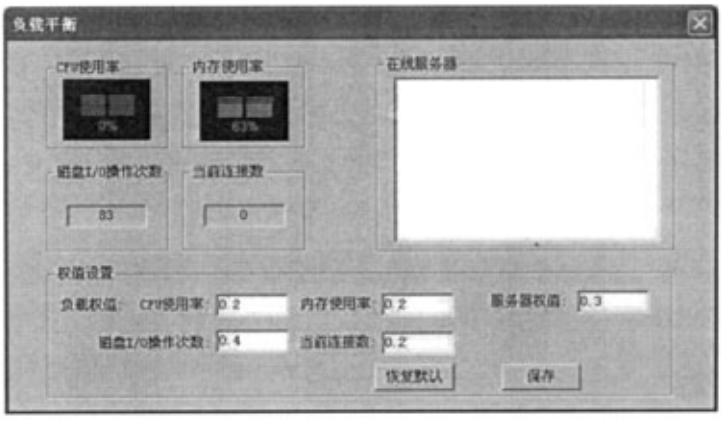

图 4.7 服务器负载指标采集和权值设置

4.2.2 负载平衡的程序实现

系统的负载平衡体现在任务分配上：把用户的点播请求，合理地分配到各个服务器中，平衡各自的负载。中心服务器是整个集群的点播入口点，任务分配便是由它来完成的，分配的时机发生在用户向中心服务器发送点播请求的时候。

用户向服务器发送点播请求是通过 RTSP 协议的 DESCRIBE 方法来实现的。在集中式 VoD 系统或单一的视频服务器（仅提供视频服务，如分布式 VoD 系统的视频服务器）中，当服务器收到用户的 DESCRIBE 请求时，便直接进行点播的准备工作，与用户进行后面的 SETUP 和 PLAY 等交互。但是在中心服务器中，由于它需要进行任务分配，因此我们对服务器收到 DESCRIBE 请求时，扩展了一下它的工作步骤。

服务器收到 DESCRIBE 请求后，首先判断自己是否是中心服务器，方法如下：

如第 4.1.2 节所述，服务器启动的时候需要选择启动方式：是作为主服务器还是作为从服务器，这时可以设置标志位 bHost；如果 bHost 为真，则还需要判断服务器链表 ServerLink 长度是否大于 0，即是否已经有其他服务器加入到集群中。只有这两个条件满足，才进行负载平衡工作，否则直接进行点播的准备工作。

需要进行负载平衡工作时，将调用 LoadBalance()函数来完成这一功能，流程如图 4.8 所示。LoadBalance()的执行结果有两个：1）接收用户请求，由本机提供服务；2）向用户发送转移服务消息，由其它服务器提供服务。

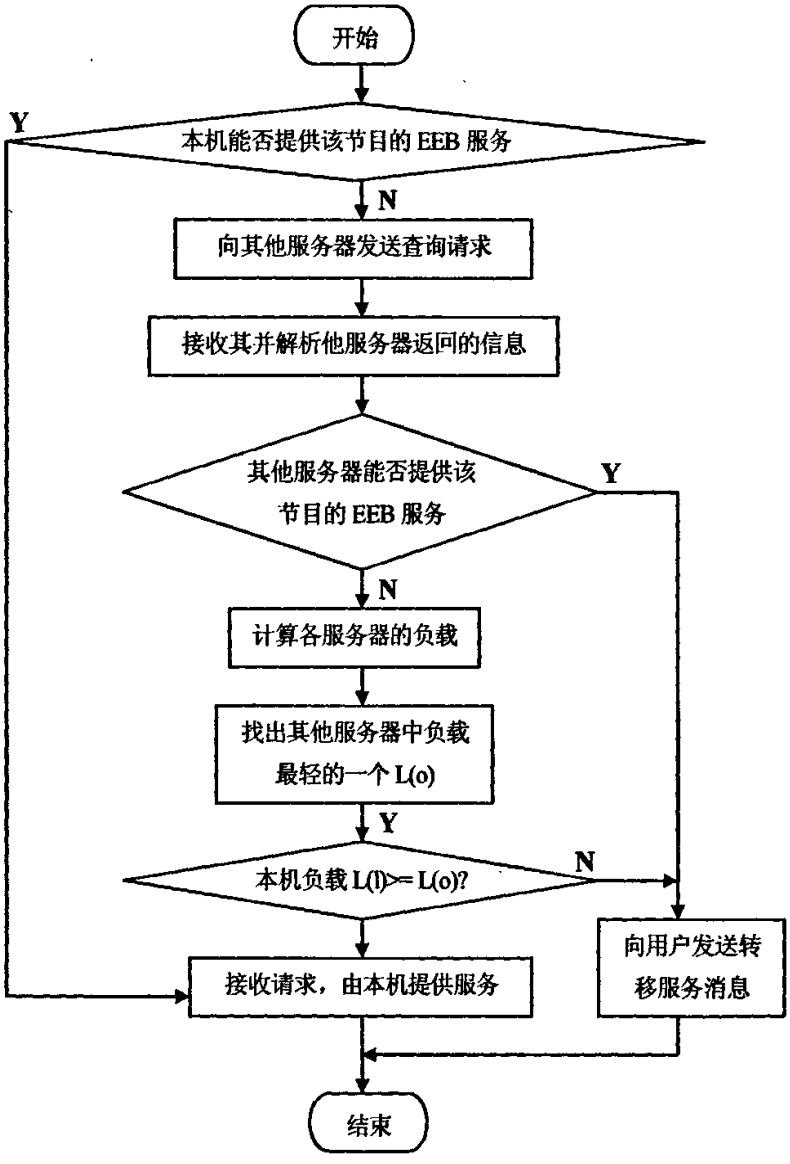

图 4.8 负载平衡流程

当所有的节目方案都设为单播方案时，新负载平衡算法将退化为传统负载平衡算法，表明该函数具有很强的灵活性。

4.3 节目的存储策略及其流调度方案的自动调整

视频节目的存储策略对于 VoD 系统的服务质量有很大的影响，合理的存储策略可以提高系统服务质量，不合理的视频节目存储策略会影响系统的服务质量甚至导致服务不可用的情况。

有了好的节目存储策略，还需要好的节目的流调度方案与之配合，才能使 VoD 服务器的性能得到进一步的提高。前面提到，本系统实现了 EEB 和 CM 等新的流调度方案，因此，如何使节目的流调度方案可根据节目的流行度而自动调整，这也是本节所需要介绍的内容。

4.3.1 存储策略

系统中作为服务节点的视频服务器在配置和性能上可能有差别，另外视频节目在视频服务器上的分布也可以有差别，这正是存储策略所要考虑的[34]。视频节目的存储策略有多种，如按照对待流行节目的方法来分有两种，即无区分和有区分(对待流行与非流行节目)的存储策略。有区分的存储策略主要考虑两个因素:节目流行度和视频服务器的处理性能。存放流行节目的视频服务器会收到比较多的用户请求，这些视频服务器的负载会比较重，因此，优先将流行节目存放到多个性能高的服务节点上(各节点都拥有该节目的拷贝)，而将非流行节目存放到少数几个甚至一个性能较低的服务节点上。每个调度器都拥有一个节目存储配置表，节目的存储配置是调度器进行负载调度的依据之一。

文献[35]给出了一种基于点播率的存储和动态调整技术。基于点播率存储和动态调整技术的核心思想是以节目点播概率为参数，根据一定的规则来周期性调整整个系统中的节目文件存储，以使用户点播请求合理分配到各个视频服务器中，实现整个系统的负载均衡。该方法将一些高点播率的节目复制到多个视频服务器中，并且随着节目点播率的变化而动态调整节目文件在视频服务器间的分布，从整体上提高这些节目总的 I/O 带宽输出量，避免引起部分服务器负载过重，发生系统拒绝服务的现象。

文献[35]给出的动态调整技术是基于节目采用单播技术进行调度的，但事实上，频繁地调整节目在各服务器上的存储会大量增加服务器的负担。而采用信道共享技术可以很好地解决由于点播人数过多而使服务器负担过重的问题。因此，对于热门节目，我们只需在一台服务器把该节目设为 EEB 或 CM 方案即可，而无需移动或复制节目。

在大规模的 VoD 系统中，如采用中心服务器群一本地服务器群的面向内容分发网络应用的 VoD 系统，节目的存储策略还是相当重要的，不能简单依靠修改节目的流调度方案来完成节目的存储和调度。在本系统中，节目的存储基本上还是靠人工来完成的，如何使节目按照一定的策略在各服务器间自动调整，将是本系统下一步的研究内容之一。

4.3.2 流调度方案的自动调整

首先要确定的是方案调整的时机。当发现有的节目由冷变热或由热变冷时，能不能立即把这些节目调整为其他方案来进行调度呢？为此，我们做了个试验：

在一台服务器（HP Proliant DL580，Intel Xeon 2.0GHz CPU，1GB 内存、146G×2 Raid 硬盘）上设置了两个用 EEB 方案调度的节目，总的分块数是 15 个，然后启动服务器端程序。此时我们发现服务器的 CPU 使用率超过了 90%，处于暂时“瘫痪”状态，连最基本的鼠标键盘响应都慢了好多，这一过程持续约半分钟。造成这种现象的原因是由于 EEB 是数据中心方案，它在服务器程序启动的时候便进行初始化，然后开始调度。但是在初始化的过程中，需要消耗大量的服务器资源（如节目总分块为 15 个时，服务器需要在短时间内，先根据 EEB 的公式计算这些分块的大小，再同时启动 15 个数据读取线程和 15 个发送线程，服务器的瞬时工作量非常大）。因此，如果立即调整节目的调度方案，那么服务器上正在点播的其他节目的点播将大受影响。因此，处于保证服务质量的考虑，我们不赞成立即调整节目的调度方案，而使用“下次启动时调整”的方式；或者利用系统可动态扩展的优点，新开启一台服务器加入到集群中，然后在新开启服务器中采用调整后的方案来调度节目。

接下来要确定的是方案该如何调整。前面提到（第 1.1.4 节），EEB 方案适合于热门节目，CM 方案和单播方案适合于普通节目。那么，该如何区分节目的冷热程度呢。文献[36]给出了一种区分方法：通过λ（用户的请求强度）计算出每个节目的λi，并将 T 值（CM 方案的阈值）预先设好，此时发现使用 CM 方案时，系统的信道数占用大于使用 EEB 方案的信道占用数（即分块数），那么就将该节目视为热门节目，改为 EEB 方案调度。文献[36]虽然没有给出使用 CM 方案和使用单播方案的区分，但是也可以通过类似计算信道占用数的方法得出。

但是经过实际运行我们得知，节目的冷热程度并不需要区分得太细：用户的口味随时在变，节目的点播率也随时在变，我们不可能跟踪得很精确，节目的调度方案也不好立即调整；同时也由于 EEB 方案在初始化时需要消耗大量计算机资源，因此一台服务

器不能设置太多的 EEB 节目。由此，我们建议使用如下方案：根据节目点播率的统计，一般情况下，热门程度排前千 20%的节目采用 EEB 方案，前 20%～50%的节目采用 CM 方案，后面的节目采用单播方案；对于单台服务器来说，一般设置 2～3 个 EEB 节目，5～10 个 CM 节目比较合适。

关于如何得出节目点播率的统计，由于中心服务器的数据库上记录有所有节目的点播信息，因此通过一个查询结果排序的 SQL 语句便可得出。

4.4 提高客户端播放器的解码能力

前面第 1.2 节提到，原系统的客户端播放器只支持用 DivX 3.11 压制的 AVI 文件，当前大多数 AVI 文件是用 Xvid 来压制的。因此，客户端要想播放用 Xvid 压制的文件，必须引入 Xvid 解码器，并以此为突破口，用同样的方法引入更多的解码器，提高客户端播放器的解码能力，使其能播放更多的其他格式的文件。

Xvid 解码器可以到其官方网站上去下载，关于它的安装和使用具体情况请参阅文献[37]。

在引入新的解码器的同时，我们需要对它们进行管理和使用，为此我们使用了视频压缩管理器（Video Compress Manager，简称 VCM），它提供了一个访问接口，通过该接口可以使用系统已经安装了编码/解码器去执行压缩和解压视频数据、使用应用程序默认的 renderers 去压缩、解压或者画视频数据等任务。因此，对 VCM 的编程是提高客户端播放器解码能力的关键。

4.4.1 VCM 应用基础

VCM 界于应用程序和编码/解码驱动程序之间，后者负责对数据帧进行压缩和解压缩处理。

当应用程序去调用 VCM 的时候，VCM 把这个调用动作封装到一个消息中。然后通过使用 ICSendMessage 函数，把这个消息发送到适合的编码器或解码器。VCM 接收到编码器或解码器的返回值后，就把控制返回给应用程序，由此便可进行下一步的压缩和解压缩处理。

系统使用注册表中的“项”来定位 VCM 的驱动程序。这些“项”用两个四字符码（two four-character codes）的形式来表示，中间使用一个点来分割（比如 VIDC.DIVX）。第一个 4 字符码（four-character code）由系统定义，如表 4.1 所示：

表 4.1 四字符码说明

四字符码	说明
"VIDC"	编码器和解码器的 ID
"VIDS"	视频流 Renderers 的 ID
"TXTS"	文本流（text-stream）Renderers 的 ID
"AUDS"	音频流控制器

第二个四字符码由驱动程序定义。比较典型的是，第二个四字符码用于描述这个驱动程序可以处理的数据类型。

当打开一个 VCM 驱动程序的时候，应用程序就指定了这个驱动程序和驱动程序可以处理的数据类型。一般来说，这个信息来源于数据流的头部。系统将尝试去打开指定的数据，如果打开失败了，系统就在注册表中查找可以处理该数据的其他驱动程序。

当在查找驱动程序的时候，系统会尝试用与这四字符码匹配的其他驱动程序“项”来处理这个数据类型。例如，一个应用程序指定了一个 XVID 的压缩器，系统在注册表寻找 VIDC. XVID 这个项。如果没有找到，它就打开每个驱动程序去看是否可以处理该数据。在这个例子中，系统在注册表中如果不能找到 VIDC. XVID “项”，它将打开所有带“VIDC“的指定的驱动程序去处理这些数据。

4.4.2 视频数据解码流程

有了上一节的技术准备之后，便可进行视频数据的解码，流程如图 4.9 所示：

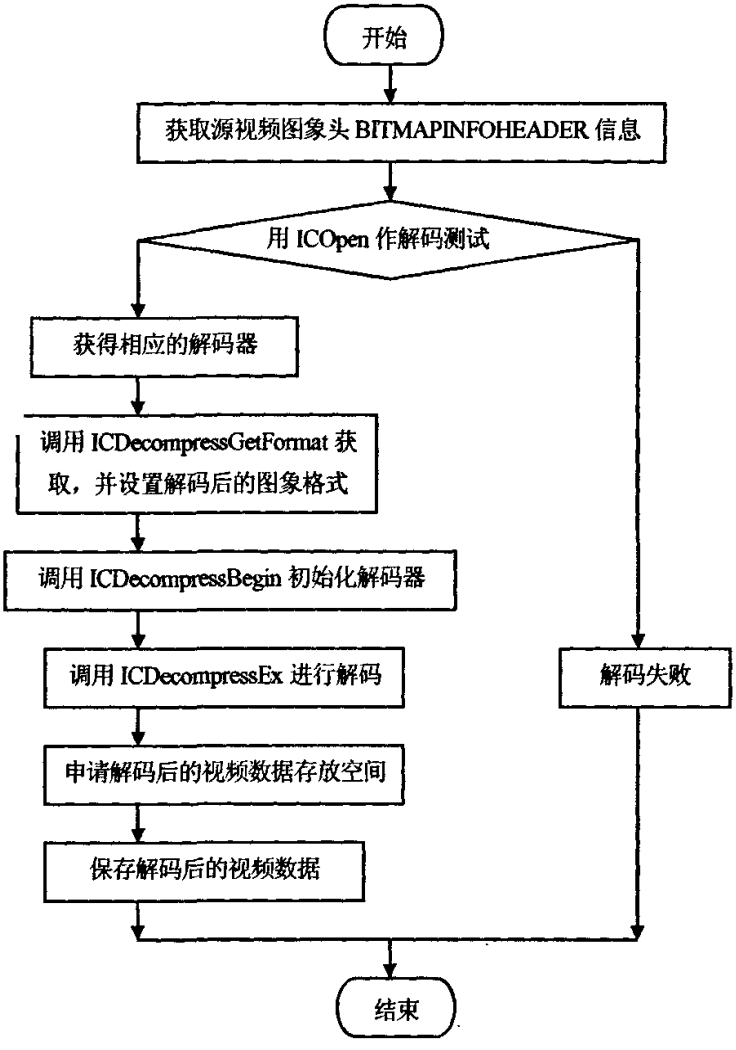

图 4.9 客户端播放器解码流程

该部分的作用是客户端播放器从缓冲区中取出一帧接收到的视频数据，然后交由解码器解码的过程。解码后的数据是二进制位图格式，可直接输出到 Video Renderer 进行回放。ICDecompressEx 的作用是去解压一个帧数据，只要缓冲区中有数据，在整个点播期间，客户端会重复调用这个函数，直到所有的数据帧都解压完毕。

在回放过程中，如果视频图像滞后（比如图像比声音延后），我们可以使用 ICDECOMPRESS_HURRYUP 标志去加速解码。使用加速解码，在解码过程中可能不会对整个视频帧都解码，而是取其中关键的一部分数据进行解压。所以当使用这个标志的时候，Video Renderer 最好不要尝试把解压的数据回放出来。

5 非线性编辑技术 DES 在视频节目编辑中的应用

本节从制作体育比赛剪辑入手，详细介绍非线性编辑技术在视频节目制作中的应用及实现过程，该部分可作为网络视频服务系统附加功能的扩展。

5.1 视频节目的编辑需求

本文引入非线性编辑技术是因为笔者参与了实验室另一个课题的开发工作："乒乓球比赛临场技战术统计和分析系统"，其中就有一块功能是从一场或多场乒乓球比赛视频中，检索出一些视频片段，实现定位播放并可保存到文件中。在这里，检索是指根据运动员、比赛进程和技战术等特定条件，来确定这些视频片段在视频文件中的位置，即开始时间和结束时间（用整数秒表示），如图 5.1 所示。接下来要做的就是按顺序把这些片段播放出来或存成文件。用这种方法可以制作出许多有用的视频节目，可作为 VoD 系统的一个重要的节目来源。

	脚本编号	开始时间	结束时间
▶	628	88	106
	629	106	117
	632	159	165
	633	165	173
	636	193	208
	639	256	277
	640	208	268
	641	277	293

图 5.1 视频的时间片段

本文就不过多介绍视频片段是如何检索出来的了，本文将要介绍的是这些片段的开始时间和结束时间检索出来之后，是如何利用 DES 技术，来实现视频片段的定位播放和保存到文件中的。

5.2 DES 在视频节目制作中的应用

用 DirectShow 实现播放功能非常方便。该功能模块主要是由 GetCurrentPosition()函数得到编入点和编出点的时间，为后面的编辑提供媒体的起始时间。下面主要介绍用 DES 进行预览和保存的实现方法。

5.2.1 时间线的构建

用 DES 实现剪辑后的预览或保存功能首先必须构建时间线模型。首先调用了系统提供的一个虚接口。这个虚接口，称为 IAMTimeline。我们要做的事情就是，遵循图 2.10 中时间线的结构定义自己所需要的属性和函数，并且建立出我们的时间线对象。最基本的属性包括有图 2.10 中所提到的组（Group），集合（Composition），轨道（Track）和媒体源（Source）。

(1) 首先创建一个时间线对象

IAMTimeline*pTL=NULL;

hr=CoCreateInstance(CLSID_IAMTimeline,NULL,CLSCTX_INPROC_SERVER,IID IAMTimeline,(void* *)&pTL);

这时，我们面对的是一个空的时间线框架，接下来所作的是根据自己的需要为我们的“树形”时间线结构填上“枝叶”。

(2) 接下来使用接口方法 IAMTimeline::CreateEmptyNode 创建各种 DES 对象。包括：IAMTimelineGroup（视频组 pVideoGroup，音频组 pAudioGroup）、IAMTimelineComp（视频 pVideoComp,音频 pAuioComp）、IAMTimelineTrack（视频 pVideoTrack,音频 pAuioTrack）、IAMTimelineSrc（视频 pVideoSrc,音频 pAuioSrc）。

(3) 接着在组中加入轨道

pVideoComp→VTracklnsBefore(pVideoTrackObj, -1);

pVideoTrackObj→Querylnterface(IID_IAMTimelineTrack,(void **)&pVideoTrack)。

(4) 这是最关键的一步，设置媒体源的剪切时间和其在时间线上的时间，然后将其放到相应的轨道上。

5.2.2 预览功能的实现

创建好时间线后，创建基本渲染引擎 IRenderEngine，它的作用是通过已经建立好的时间线构建滤波器图（FilterGraph）供预览或者输出文件。所以，我们需把时间线的信息传递给它。接下来的过程很简单了，调用 ConnectFrontEnd 连接时间线部分所建立的滤波器（Filter），调用 RenderOutputPins。至此，滤波器图已成功建立，只要调用 IMediaControl 接口的 Run()函数即可进行预览了。

//创建基本渲染引擎

IRenderEngine *pRenderEngine = NULL;

hr=CoCreateInstance(CLSID_RenderEngine, NULL,CLSCTX_INPROC_SERVER,IID_IRenderEngine, (void**)&pRenderEngine);

```
hr=pRender->SetTimelineObject(pTL); //确定要渲染的时间线
hr=pRenderEngine->ConnectFrontEnd( );//构建 graph 的前端
//将前端出来的引脚根据媒体类型分别连接到音视频渲染器，完成 graph 的构建
hr=pRenderEngine->RenderOutputPins( );
IGraphBuilder *pGraph=NULL;
IMediaControl *pControl=NULL;
hr=pRender->GetFilterGraph(&pGraph);
hr=pGraph->QueryInterface(IID_IMediaControl, (void **)&pControl);
hr=pControl->Run();      //运行 Filter Graph
```

5.2.3 保存功能的实现

创建好时间线和前端后，前端输出的是非压缩的音频流和视频流，而我们要保存的是压缩数据，所以必须向滤波器图中加入 MPEG-2 音频编码器和视频编码器以及复用器。

第一步，将视频编码器、音频编码器和复用器以及文件写入程序滤波器加入到滤波器图中，例如加入”mpeg video encoder”：

```
hr=AddFilterByCLSID(pGraph,LSID_VIDEO_ENCODER,L"mpeg video
encoder",&pVideoEncoder);
```

第二步，得到组的个数及输出引脚指针，根据引脚的媒体类型将其连接到相应的编码器上。首先通过时间线函数 GetGroupCount()获得组的个数，然后再通过控制引擎 pRenderEngine 的函数 GetGroupOutputPin()获得各个组的引用针脚，并通过 GetMediaType()函数获得引用针脚输出的媒体类型。假如 GetMediaType()函数返回 TRUE 则引脚输出的是视频流，否则是音频流。这时，就可以通过 ConnectFilters()函数把输出针脚与相应的视频/音频编码器相连接，例如：

第三步，将视频编码器和音频编码器滤波器连接到复用器滤波器上：

```
hr=ConnectFilters(pGraph,pVideoEncoder,pMux,TRUE);
hr=ConnectFilters(pGraph,pAudioEncoder,pMux,TRUE);
```

第四步，连接复用器和文件写入程序滤波器：

```
hr=ConnectFilters(pGraph,pMux,pfilewriter,TRUE);
```

第五步，创建 MPEG 输出文件，调用 IFileSinkFilter 的 SetFileName()函数。

最后调用 IMediaControl 接口的 Run()函数即可进行保存了。

非线性编辑的运行界面如图 5.2 所示，对话框左半部分为操作区，右半部分为预览区。提取视频时，可选择下拉框中列出的已安装在系统上的解码器。

图 5.2 非线性编辑运行界面

6 系统的仿真测试及性能研究

为了在实验室范围内对系统进行接近实际运行的测试，系统采用了一个客户请求仿真器，它按照 Poisson 分布（用户请求发生模型）和 Zipf 分布（节目流行度模型）产生节目点播请求。本节将介绍在仿真测试环境下对系统的各项性能测试，以及由测试结果得出的初步结论。

6.1 视频点播的仿真实现

针对图 4.2 所示的视频点播系统实现模型，设计并实现了一个仿真器[38]，用作模拟客户端的点播请求，对系统能进行比较全面的测试。仿真器主要包括事件发生器、流行度模型、RTSP 客户队列等几个模块，如图 6.1 所示。

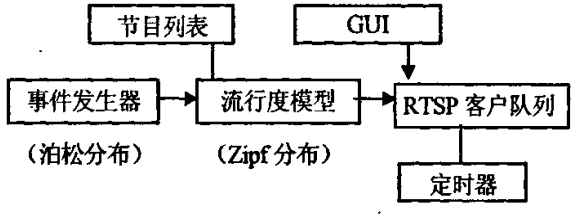

图 6.1 仿真器程序模型

其中事件发生器产生一系列服从 Poisson 分布的点播事件；流行度模型定义了节目流行度的分布，这里采用 Zipf[1] 分布，即节目 i 的流行度 $P_i = C/i^{\alpha}$，其中 $C = \left(\sum_{j=1}^{N} 1/i^{\alpha}\right)^{-1}$，$\alpha$ 为 Zipf 参数，一般取 0.271；RTSP 客户队列保存了当前正在点播的用户请求。仿真器可以实现自动点播和手动点播，在手动点播时，可以根据需要创建 RTSP 客户并进入客户队列，并随时停止点播。而在自动点播中，由事件发生器产生的点播事件根据流行度模块定义的节目分布生成点播请求，这些请求创建 RTSP 客户并进入 RTSP 客户队列，RTSP 客户负责与服务器进行通信，并将节目长度信息告知定时器，定时器将在节目播放完毕后自动向服务器发出停止点播命令。这样便可以模拟用户从点播到停止的完整点播过程。仿真器只是负责与服务器连接并模拟用户向服务器发送点播请求，但当它接收到服务器发来的节目数据时便立即丢弃，并不做缓存和解码等处理，因此不会消耗很多客户端资源。

在自动点播时，仿真器有两种产生点播请求的方式：轮转点播和按 Zipf 分布点播。轮转点播即按顺序依次点播节目列表中的节目，此时每个节目的点播几率是对等的。而按 Zipf 分布点播，则对节目的流行度进行区分，列表中的节目排位越靠前其被点播的几率越大。

按 Zipf 分布点播的实现如下：我们用流行度数组 P 来表示节目在每一轮次被点播的概率，初时的时候先计算每个节目的 λ 值，即 λ_i。在每个点播轮次内，每个节目的 P[i]值将加上 λ_i 值，即

$P[i] = P[i] + \lambda_i$;

表示各节目的流行度叠加。这样，上一轮次没有被点播的节目的 P[i]值，表明该节目在这一轮次被点播概率将增大。而上一轮次被点播的节目，其 P[i]值将－1，即

P[i] = P[i] - 1;

表明该节目已被点播过一次，下一轮次它被点播的概率将降低。

在每个点播轮次，我们将选择流行度总和。即 P[i]值最大的一个作为当前要点播的节目，由此便实现了按 Zipf 分布自动点播。

另外，仿真器还可以统计当前的点播数和点播带宽，方便进行网络带宽利用率和系统支持的点播数量等方面的统计。这样，就可以采用仿真器对服务器系统进行模拟 VoD 点播和性能测试了。仿真器的运行界面如图 6.2 所示。

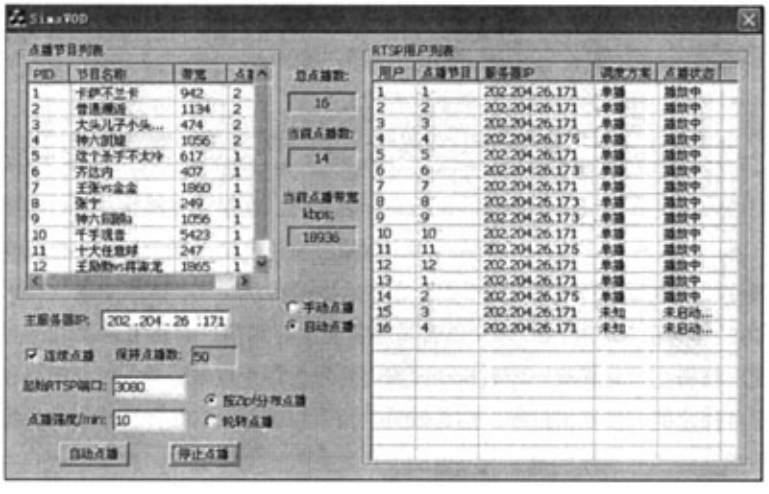

图 6.2 仿真器运行界面

6.1.1 仿真测试的可行性分析

假设 VoD 系统结构如图 6.3 所示，所有的设备均由一个快速以太网交换机连接。以下服务器 Server1、Server2、Server3 分别简称为 S1、S2、S3，客户端 Client1、Client2、Client3 分别简称为 C1、C2、C3，交换机简称为 S。

由交换机及网线、网卡的工作特性，并经测试验证，整个网络是双工工作的。即假如 S1→C1（从 S1 到 C1）有数据传输，占用的是 S1→C1 的网络带宽，而 C1→S1 的网络带宽并不受影响。同样，其他点对点之间，只要不是以 S1 为起始点，它们之间的网络带宽并不受影响，例如 S2→C1、S2→C2、S3→C2、S3→S2 等不受影响，而 S1→C2、S1→C3，S1→S2 等受影响。

为什么 S1→C1 有数据传输时，其他以 S1 为起始点的点对点之间的网络带宽有影响呢？这是因为整个系统是由交换机连接起来的，某两点之间的通路 A→B（A 和 B 均可是服务器或客户端 PC），是由起始点 A 到交换机 S 的通路 A→S，和交换机 S 到终点 B 的通路 S→B 所组成，因此 A、B 之间的网络带宽由 A→S 和 S→B 的网络带宽来决定。A→S 和 S→B 任意一段网络的带宽降低了，A→B 的网络带宽都会降低。例如有两个通路 S1→C1，S1→C2，都以 S1 为起点，它们之间有一段通路是共用的，即 S1→S（起始点到交换机之间的通路）。假如 S1→C1 有数据传输，由于数据须经过共用通路 S1→S，S1→S 带宽降低了，那么 S1→C2 的带宽将会受到影响；同理，S1→C2 之间有数据传输，S1→C1 的带宽也会受到影响。因此我们可以说，有相同起始点的通路之间的网络带宽是相互影响的，这些通路的带宽总和等于从起始点到交换机的网络带宽。

因此，由以上分析，由仿真器在一台客户端 PC 上模拟多个用户向服务器点播节目，其效果和相同数量个用户在不同的客户端 PC 上向服务器点播节目的效果是一样的。

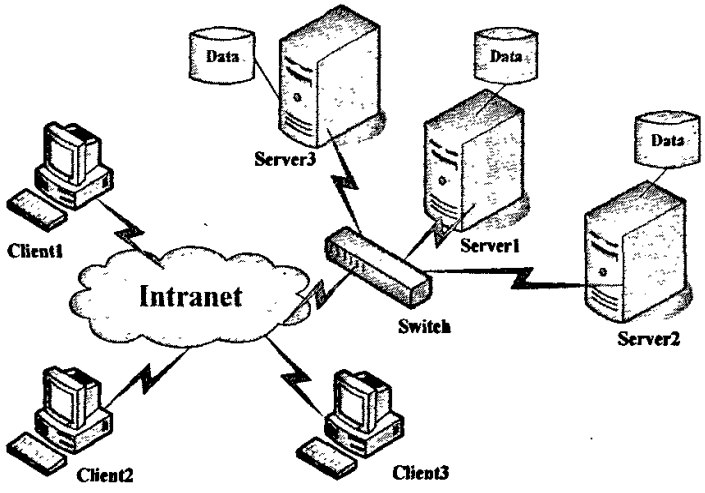

图 6.3 系统各组件的连接

6.1.2 测试计划

系统测试所需要解决的首要问题就是如何在实验室范围内对系统进行接近实际运行的测试。由于条件所限，不太可能做有几百人同时点播的真实测试；另一方面大规模的真实测试，测试数据难以反馈和统计，所人力物力消耗巨大。因此，采用仿真测试的方法，在一台主机上可以模拟几十个上百个用户点播，可有效解决上述问题，而且系统条件和测试方式也可以灵活配置，可以对系统进行更加全面的测试和分析。

系统的测试计划如下：

1）单服务器节点测试。该测试分两次进行，分别使用性能较好的服务器和性能较差的服务器进行测试。主要测试单台服务器最多能支持多少个单播流，以及在这个状态下服务器的 CPU 使用率、内存使用率等情况，再计算此时所有单播流的总带宽，并与网络实测带宽相比较。通过比较服务器之间性能差别对服务器支持最大并发信道数的影响，以及网络带宽使用率的比较，由此得出制约 VoD 系统性能发挥的瓶颈。

2）多服务器节点测试。希腊的 A. Papagiannis[11]等是这样对他们的 TVoD 系统进行多服务节点的测试的：他们使用两个或三个服务器（均是 P4 1.5GHz CPU，512 MB 内存，40GB 硬盘）。他们认为，如果把各个服务器节点使用快速以太网交换机连接起来，那么理论上网络带宽就可以根据服务器节点的数量而成倍数的增长，系统支持的最大并发信道数与服务器节点的数量是一种线性增长的关系。他们的试验也证实了这种假设：只有一个服务器节点时，系统支持的单播流上限是 90 个，而有两个服务器节点时

是 172 个，有三个服务器节点时是 241 个。文献[11]最后的结论是，他们的系统能最大的利用网络带宽，在 100Mbps 的网络带宽下，服务器并不存在性能瓶颈。因而提高网络带宽是提高系统性能的一个有效途径，比如换成 Gbps 的以太网。我们计划也来做一个类似的试验，验证一下我们的系统，也验证一下文献[11]的结论。

3）负载平衡算法性能测试。前两个计划均是对服务器的性能所作的测试，以验证本系统可采用多台普通 PC 机代替昂贵的高性能服务器的思想。而第三个测试计划则是分别对系统实现的传统负载平衡算法和新负载平衡作性能测试，通过比较以验证新负载平衡算法理论的有效性。

具体的测试步骤如下：

A. 把所有节目都设为单播方案进行测试；

B. 把最流行的 2～5 个节目设为 EEB 方案，其余节目为单播方案，采用传统负载平衡算法进行测试；

C. 把最流行的 2～5 个节目设为 EEB 方案，其余节目为单播方案，采用新负载平衡算法进行测试；

由这三个步骤的测试结果，得出用户强求强度 λ 与系统平均并发信道数曲线图，然后进行分析。

6.2 测试结果及系统性能分析

测试按照上一节的测试计划进行，测试的结果及初步结论如下：

6.2.1 单服务器节点测试

分别对三台服务器进行测试：

- Server1：HP Proliant DL580 服务器（Intel Xeon 2.0GHz CPU，1GB 内存、146GB×2 Raid 硬盘）；
- Server2：普通 PC 机（Intel P4 2.4GHz CPU，512MB 内存，80GB SCSI 硬盘）；
- Server3：清华同方服务器 （Intel P4 1.7 GHz CPU，256MB 内存，80GB SCSI 硬盘）。

测试参数确定如表 6.1 所示。网络带宽的测试工具为 NetIQ 公司的 Chariot v5.4，经测试，实验室内部网络带宽稳定值为 94Mbps 左右。

表 6.1 测试参数

参数	默认值	范围

节目数 N	10	N/A
节目带宽/Kbps	1056	247~5423
Zipf 分布函数	0.271	N/A
并发信道数	40	0—90

测试的时候在每个测试 PC 机内运行一个仿真客户端和一个真正的 VoD 客户端。仿真客户端在模拟客户点播，真正的 VoD 客户端在接受并播放节目。当发现所播放的节目有剧烈抖动、长时间停止等现象时，说明此时的系统已达到极限。实验结果如图 6.4、图 6.5 所示：

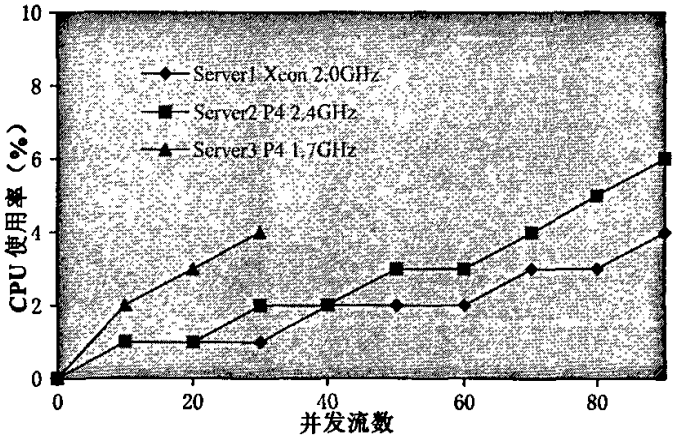

图 6.4 单服务器节点测试—CPU 使用率

在整个测试过程中，服务器的 CPU 使用率都很低，均不到 10%。Server3 由于内存较小，在有 30 个左右并发流的时候（点播带宽约为 36Mbps），因内存不足而无法支持更多的并发流数。

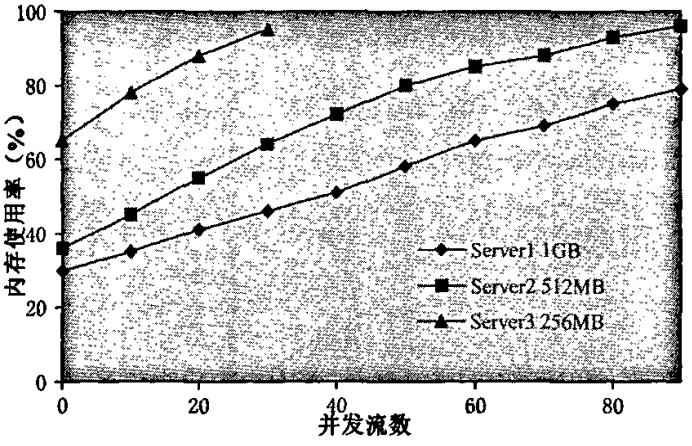

图 6.5 单服务器节点测试—内存使用率

Server3 在有 30 个左右的并发流时，内存使用率达到了 95%，出现了内存瓶颈，表现为新点播的节目启动很慢，已点播节目出现抖动。可能的原因为由于内存不足，操作系统需以硬盘空间作为缓存，从而增加了这部分的存储时间。Server1 和 Server2 的内存使用率也很高，在有 80 个左右的并发流时（点播带宽约为 89Mbps），1GB 内存的 Server1 内存使用率约为 70%，而 512MB 内存的 Server2 内存使用率则达到了 90%。

测试表明本 VoD 系统对服务器的内存大小要求较高，而对服务器的 CPU 性能要求不高。Server1 和 Server2 的点播极限分别为 87 个和 85 个并发流，点播带宽分别为 93Mbps 和 92.2Mbps，网络带宽利用率分别为 98.9%和 98.1%，均已达到网络带宽的极限。这表明，在 100Mbps 的网络带宽环境下，普通 PC 机与价格昂贵的服务器在视频点播服务方面的表现差别不大，并没有出现服务器性能瓶颈，完全能胜任视频点播的服务要求。内存偏小会限制服务器性能的发挥，因此服务器的内存最好能达到 1GB 或以上。目前内存的价格不高，扩充很方便，用扩充了内存的普通 PC 机作为视频服务器具有很高的性价比。

6.2.2 多服务器节点测试

测试服务器组由三台普通 PC 机（1GB 内存，其它配置同 Server2）组成。负载平衡算法的平衡参数设置为（0.1，0.3，0.4，0.2），这主要是考虑到在 VoD 系统中，服务器磁盘的工作量比较大，所以把磁盘 I/O 操作次数的权值设得较高。另外由前面的试验得知，本系统对服务器的 CPU 性能要求不高，而对服务器的内存大小要求较高，所以适当降低 CPU 使用率的权值而提高内存使用率的权值。测试中，各服务器能达到很好的负载平衡，测试结果如图 6.6、图 6.7 所示：

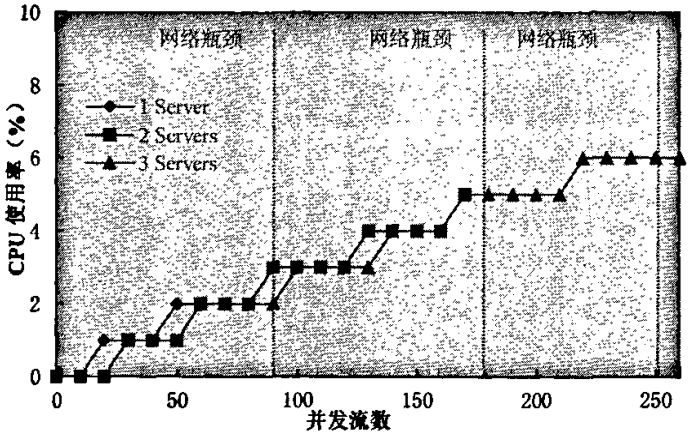

图 6.6 多服务器节点测试—平均 CPU 使用率

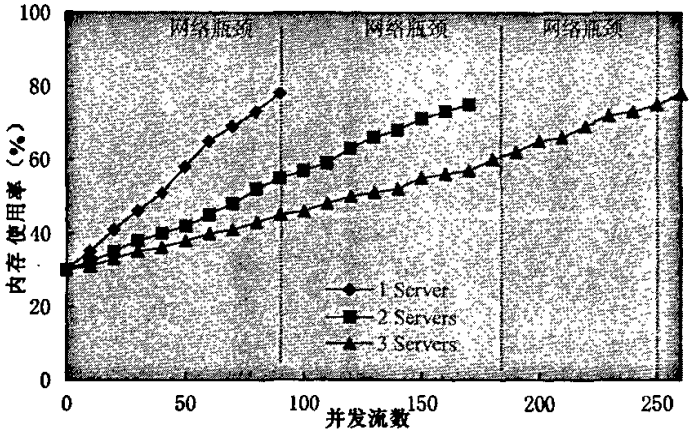

图 6.7 多服务器节点测试—平均内存使用率

从测试结果中可以看出，系统各服务器在整个测试期间的平均 CPU 使用率都很低，而平均内存使用率较高，这与单服务器测试的情况类似。当达到点播极限时，单服务器节点能支持 87 个并发流，当有 2 个和 3 个服务器节点时，系统分别能支持 170 个和 252 个并发流，系统所支持的最大并发流数与服务器节点的个数基本上成线性增长关系。这表明，系统的负载平衡机制并没有过多地耗费系统资源和网络带宽资源，系统可以通过添加服务器来增加系统支持的用户数量，具有良好的可扩展性。

6.2.3 负载平衡算法的测试

此部分的测试根据第 3.2.3 节的分析进行。测试的参数设置如表 6.2 所示。

表 6.2 负载平衡算法测试参数

名称	默认值	范围
节目带宽/kbps	576	N/A
节目数 N	10	5，10，15，20
节目长度 L/s	5400	N/A
Zipf 分布参数	0.271	N/A
节目请求强度 λ	5	1—10
EEB 方案分块数	9	N/A
EEB 方案阶数	2	N/A

测试使用 3 台服务器，配置同第 6.2.1 中的 Server1，节目总数为 10 个，采用 EEB 方案的节目均为流行度最高的几个节目。测试时使用客户端仿真程序对每种情况分别进行 20min 的测试，并在测试开始 10min 后每隔 2min 记录服务器的并发信道数，将记录的结果取平均值。

当 EEB 方案节目个数 n=2，流行度因子 α =0.271 时的实验结果如图 6.8 所示，方案 1 能支持约 250 个的用户点播，方案 3 能支持约 280 个的用户点播，而方案 2 能支持约 350 个用户点播。

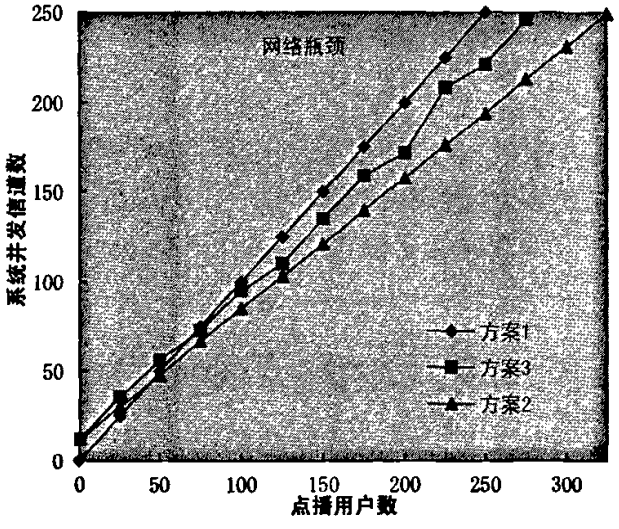

图 6.8 多服务器节点负载平衡算法性能比较（N=10，α =0.271）

减少系统并发信道占用数意味着能节省更多的系统资源和网络带宽资源，从而能同时为更多的用户服务。那么如何才能进一步减少系统的并发信道占用数呢？由公式

(3.1)我们知道它是一条以$-\sum_{i=1}^{n}\lambda i$为斜率的直线。在 M 一定时减少斜率$-\sum_{i=1}^{n}\lambda i$可以减少系统并发信道占用数 S。由于$\lambda_i>0$，所以增大 n，$\sum_{i=1}^{n}\lambda i$增大，$-\sum_{i=1}^{n}\lambda i$减少，即增加 EEB 节目的个数可以减少系统的并发信道占用数。图 6.9 所示为有 4 个 EEB 节目和 6 个单播节目，$\alpha=0.271$ 时，三种方案的实验结果。此时方案三能支持最多 320 个左右的用户点播，而方案二能支持超过 400 个用户点播。

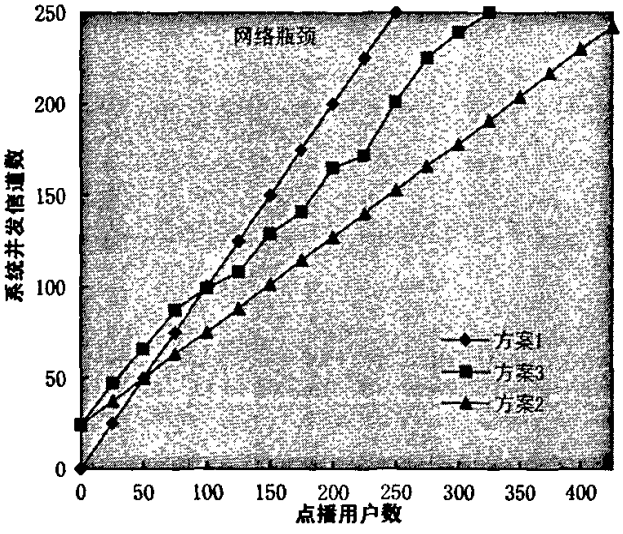

图 6.9 多服务器节点负载平衡算法性能比较（N=10，$\alpha=0.271$）

另外，由式$\lambda_i=\frac{1}{i^{\alpha}}\times\left(\sum_{j=1}^{N}\frac{1}{i^{\alpha}}\right)^{-1}$知增大 Zipf 参数$\alpha$也可使$\lambda_i$增大，从而使$\sum_{i=1}^{n}\lambda i$增大，$-\sum_{i=1}^{n}\lambda i$减少。即增大节目的流行度也可以使系统的并发信道占用数减少。图 6.10 所示为有 2 个 EEB 节目和 8 个单播节目，$\alpha=0.5$ 时，三种方案的实验结果。此时方案三能支持最多 300 个左右的用户点播，而方案二能支持约 370 个用户点播。

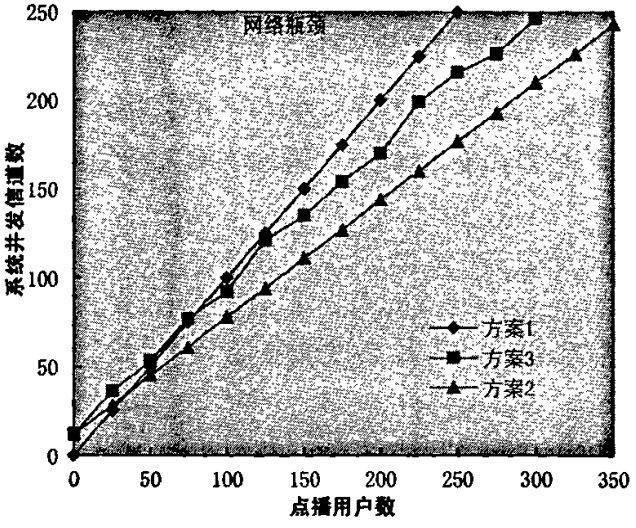

图 6.10 多服务器节点负载平衡算法性能比较（N=20，α =0.5）

测试结果表明，改进算法比传统负载平衡算法能增加 10%～30%系统支持最大并发点播用户数，尤其当系统提高采用 EEB 方案的节目个数，或采用 EEB 方案的节目流行度增大时，改进算法在系统支持最大点播用户方面比传统算法拥有更好的效能。

7 结 论

本文在分布式体系结构的基础上采用单播、CM 和 EEB 等多种流调度方案，实现了一个低成本、可扩展的分布式 VoD 系统。系统对 RTSP 协议进行了扩展，采用了中心服务器一视频服务器的分布式体系结构，由中心服务器对视频服务器进行统一管理，实现了负载的分流和平衡，能有效地解决集中式 VoD 系统的瓶颈问题。系统采用普通 PC 机作为服务器，在 100Mbps 网络带宽环境下能充分地利用网络带宽，未出现服务器性能瓶颈，能够满足大规模 VoD 应用的要求。

我们对基于 LVS 系统的动态反馈负载平衡算法进行了改进，使集群的任务负载分配更加合理。性能测试结果表明系统的性能良好，改进算法比传统负载平衡算法能减少 10%～30%系统并发信道占用数，提高了系统支持的用户点播数量。

在现有节目存储策略的基础上，我们实现了节目流调度方案的可根据节目的冷热门程度来自动调整，使得节目的方案配置更加合理。本文给出了提高客户端解码能力的实现方法，并把非线性编辑技术引入到 VoD 系统的视频节目制作当中。这些措施，进一步提高了系统的性能和服务质量，丰富了系统节目的来源。

但是，由于个人精力和时间有限，目前的系统还存在不少问题，需要在下一步的研究中来解决，主要有：

1）文中给出的负载平衡机制只是根据热门节目采用 EEB 方案，其余节目采用单播方案的实现，而实际上热门节目和冷门节目只占少数，更多的是流行度一般的节目，这些节目更适合用 CM 方案传输。因此下一步的工作是对系统和算法做进一步改进，使最流行的几个节目使用 EEB 方案，冷门节目使用单播，普通节目使用 CM 方案，这样的信道调度策略更合理一些，以进一步提高系统的性能。

2）目前的系统只适合在局域网内点播，为使系统能够适应不同网络状况、不同地域和用户规模的应用，还需要对系统在 CDN、服务机制和存储策略方面做进一步的研究。

3）目前本系统还没有一套建立故障恢复机制，而该部分对系统的服务质量来说具有重要意义。因此，如何为系统设计和实现一套完善的故障恢复机制，提高系统的容错能力，也是下一步的研究内容之一。

参 考 文 献

[1] Kien A. Hua and S. Sheu. Skyscraper Broadcasting: A New Broadcasting Schemefor Metropolitan Video- on-Demand Systems. ACM SIGCOMM'97 Conference Proceedings,pp. 89-100, 1997.

[2] Y.B. Lee. "Parallel Video Servers: A Tutorial". IEEE Multimedia, pp. 20-18, April-June 1998.

[3] 谢四莲, 彭宇行, 刘永逸. 并行 VOD 系统的可靠性研究与实现. 计算机工程与应用[J].2004,7,227-229.

[4] D. Meliksetian, F.F. Yu, C.Y. Roger. "Methologies for designing video servers". IEEE Transactions on Multimedia, pp. 62-69, March 2000.

[5] D. Sitaram and Asit Dan. Multimedia servers, Applications, environments, and design. Morgan Kaufmann Publishers, January 2000.

[6] 万元元. 一种基于推的并行视频服务器方案及其算法分析. 电子工程师[J].2001, 27(5).

[7] M. Reisslein, K. W. Ross, S. Shesthra. Striping for Interactive Video: Is it Worth It?. Proc. IEEE Int. Conference on Multimedia Computing and Systems (ICMCS'99), pp II-635-640, June 1999.

[8] S. Chan and F. Tobagi. Distributed servers' architecture for networked video services. IEEE Transactions on Networking, 9(2):125–136, April 2001.

[9] 陶阳, 彭宇行等.并行视频服务器中的 DLB 策略.计算机工程[J] . 2003, 7:39–41.

[10] 顾铁成, 陈道蓄. 分布式视频服务系统中接入控制问题的研究. 计算机工程[J]. 2002, 12:27–29.

[11] A. Papagiannis, D. Lioupis, S. Egglezos. Design & Implementation of a low-cost Clustered Video Server using a network of personal computers. IEEE 4th Int. Symp. on Multimedia Software Engineering (MSE 2002), Newport beach California, USA, Dec 2002.

[12] A. Papagiannis, D. Lioupis, S. Egglezos. A Scalable, low-cost VoD server with Multicast Support. IEEE Int. Conf.. on Information Technology: Research & Education (ITRE 2003), Newark New Jersey, USA, Aug.10-13 2003.

[13] 张谢华, 夏士雄, 张欢. 基于 P2P 分布式 VOD 系统的研究. 计算机应用与软件[J] .2005,11,137-139.

[14] 刘亚杰, 窦文华. 一种 P2P 环境下的 VoD 流媒体服务体系. 软件学报[J] . 2006,4,876-884.

[15] S.Viswanathan. Pyramid broadcasting for vod service. In proceedings of the SPIE Multimedia Computing and Networking Conference. Frebruary 1995,66-77.

[16] 鄢仁祥, 高远. 一种新的视频点播方案——扩展幂级方案. 计算机研究与发展[J]. 2002, (7):869-875.

[17] L.Juhn, L.Tseng. Harmonic broadcasting fro video-on-demand service.IEEE Transactions on Broadcasting, September 1997, 268-271.

[18] S.W.Carter and Darrel D.E.Long. Improving video-on-demand server efficiency through stream tapping. In:Porc.ICCCN 97,Las Vegas, NV, Septempber 1997.IEEE Computer Society Press, 200-207.

[19] L.Gao and D. Towsley. Supplying instantaneous video-on-demand services using controlled multicast. In Proc. IEEE Int. Conf. Multimedia Computing and Systems, vol. 2, Florence, Italy, June 1999.117-121.

[20] K Hua, Y Cai. Patching: A multicast technique for truevideo-on-demand services. The ACM Multimedia, Bristol, UK, 1998.

[21] Ying Cai, Kien A. Hua and Khanh Vu. Optimizing Patching Performance. In Proc.of ACM/SPIE Multimedia Computing and Networking, Jan. 1999, pages 204-215.

[22] 吕春, 刘娟, 赵会群. 大型体育赛会 VoD 系统关键技术的研究与实现. 北方工业大学学报[J]. 2004,9,22-26.

[23] H Schulzrinne, A Rao, R Lanphier. Real time streaming protocol (RTSP) [M]. RFC 2326, 1998.

[24] 唐丹,金海,张永坤. 集群动态负载平衡系统的性能评价.计算机学报[J].2004,6:803-811.

[25] Kayama K., Shimizu K., Ashihara H., Zhang Y., Kameda H.. Performance evaluation of adaptive load balancing policies in distributed systems. In: Proceeding of Singapore International Conference on Networks/ International Conference on Information Engineering'93, Singapore, 1993, 606-611.

[26] Katz E D, Butler M, McGrath R. A Scalable HTTP Server: The NCSA Prototype [J]. Computer Networks and ISDN Systems, 1994,8(5): 155-163.

[27] Dahlin A, Froberg M, Walerud J, et al.EDDIE: A Robust and Scalable Internet Server [J]. http://www.eddieware.org/, 1998-05.

[28] Dias D, Kish W, Mukherjee R, et al. A Scalable and Highly Available Server [J]. In Proceeding of COMPCON 1996, IEEE-CS Press, Santa Clara, CA, USA, 1996,11(3):85-92.

[29] Zhang W, Jin S, Wu Quanyuan. Linux Virtual Server: Server Clustering for Scalable Network Services [J]. Beijing: Proceeding of World Congress Conference 2000, 2000, 9(7):21-25.

[30] 陆其明, 金邦飞. DirectShow 开发指南. 清华大学出版社[M]. 2003,12.

[31] 吕春. 大型体育赛会 VOD 系统关键技术的研究与实现[D]. 硕士, 北方工业大学, 2005.

[32] The Linux virtual server project [EB/OL]. http://www.LinuxVirtualServer.org, 2002.

[33] G Zipf. Human Behavior and the Principle of Least Effort [M]. New York: Addison-Wesley, 1949.

[34] 王永亮, 刘峰, 张春. VoD 系统的负载均衡存储策略及调度算法研究.电视技术[J].2004(11).

[35] 夏绍春, 易波, 刘威. 一种视频服务器的负载均衡方法. 计算机应用[J]. 2004,3.

[36] 胡玉琦, 鄢仁祥, 高远. VODStar 视频点播系统的设计、实现及性能测试. 计算机研究与发展[J]. 2003,11,1643-1649.

[37] Xvid org: Home of the Xvid codec[EB/OL]. http://www.xvid.org/, 2006.

[38] 王荣生. RTSP 协议在视频点播系统中的应用. 计算机应用与软件[J]. 2004, 9, 149-150.

申请学位期间的研究成果及发表的学术论文

在学期间发表的论文：

[1] 吴杰伟,张雨佳,赵会群.一种低成本可扩展的分布式 VoD 系统体系结构研究.计算机应用研究[J].2007,第 9 或第 10 期（已录用）.

[2] 张雨佳,赵会群,吴杰伟.数据挖掘技术在排球比赛技战术分析中的应用研究.计算机应用[J].2006,12,3027-3029.

[3] 郑福泽,高洪歌,吴杰伟,赵会群.视频技术在乒乓球比赛技战术分析中的应用研究. 第十二届全国图象图形学学术会议论文集[M].2005.776-780.

致　谢

论文即将搁笔，随着论文的装订，心中些许的快慰伴随着淡淡的失落挥之不去，意味着三年研究生学习的结束，在此，对所有一直关心和帮助过我的老师、同学、朋友、父母和亲人表达最诚挚的谢意。

本文是在恩师赵会群教授的悉心指导下完成的，在研究生三年中得到赵老师的指导是我一生的荣幸，在三年中赵老师给我提供了充分的研究空间和学习氛围，在生活和学习上给予我无私的关心和帮助，使我有机会直接参与高水平的科研课题，掌握相关领域的最新动态，并安排我参加相关的学术会议也专题讲座，充实了我的知识体系，拓展了我的知识面，使我在思想和科研方法上逐步走向成熟。赵老师严谨的治学态度、诲人不倦的学者风范对我产生深远影响，将是我一生效仿的楷模。

由衷感谢三年来所有给我上过课和帮助过我的的老师，各位老师丰富的专业知识和严谨的治学态度给予了我学习和生活极大的帮助，激发了我的创新意识和开拓进取的精神。

感谢三年来与我一起学习和生活的师兄弟姐妹，感谢张雨佳、吕春、郑福泽、潘林的帮助和指导，在与你们互相学习共同探讨中受益匪浅，感谢三载同窗共同学习和生活过的好友同学，一起度过了三年愉快的时光。

特别感谢我的父母和亲人近二十年来对我学业的一贯支持和勉励，是他们的默默奉献让我得以完成我的学业，照亮我的前程。

最后感谢在百忙之中抽出时间评议审阅论文的所有尊敬的老师。