摘 要

学科专业： 计算机应用技术

论文题目：

基于 LINUX 平台的服务器集群的架构和实现性研究

硕士研究生： 吕 其 元

导 师： 刘乃琦教授

服务器集群是一组用高性能网络连接起来的独立的服务器的集合。它们能够协同工作，共同处理 INTERNET 上日益增长的客户请求。它具有良好的可获用性，可伸缩性以及优秀的性能价格比，是当前服务器技术的热点。本文阐述了目前服务器技术的发展方向、服务器集群的背景。最先介绍了采用文档迁移方法实现的应用层服务器集群。然后对 LINUX 的中心式集群－LINUX VIRTUAL SEVER 和分布式集群－DPR 技术作了深入分析。最后在实验室测试评估的过程中，作者提出了服务器集群新方案。

关键字： 服务器集群 可获得性 可伸缩性 负载平衡 文档迁移 虚拟服务器
分布式集群 网络地址转换 IP 封装 直接路由

Abstract

Subject and specialty: Computer Application Technology

Title:

Research of Architecture and Implementation of the Server Cluster Based on LINUX Platform

Master Candidate: Lv Qiyuan

Tutor : Prof. Liu Naiqi

Server cluster is a set of autonomous servers connected by high performance networks, which can work co-operatively to process the more and more request from client with the explosive increment of Internet. Due to its high availability, high scalability and excellent performance-price ratio, server cluster has became a hot point of modern server technology. This paper formulated the development direction of server technology and the background of server cluster. It introduced an prototype of cluster implemented on application layer by means of document migration, and analyzed the centralized cluster-LVS and the distributed cluster-DPR of LINUX platform. During performance test of LVS via NAT, the author proposed new solution for server cluster.

Keyword: Server Cluster Availability Scalability Load Balancing Document Migration Virtual Server Distributed Server Cluster Network Address Translation IP Encapsulation Direct Routing

绪 论

我们知道，INTERNET 的前身 ARPANET 诞生时网络的规模很小。在 80 年代初，随着 TCP/IP 标准开始被广泛应用，网络技术日趋完善。网络规模的迅速扩大。今天 INTERNET 曼延及全世界，成了人类世界的信息基础设施，它正以前所未有的深度和广度影响人们的生活。在这个应用需求的作用下，作为网络灵魂的服务器引起了人们特别的关注。服务器的高可靠性、高性能、高吞吐能力、强大的存储能力、强大的网络功能和友好的人机界面是摆在 IT 业者面前的挑战。

近几年来，计算机网络在技术上飞速发展，使得通过高速网络连接的计算机进行协同工作成为可能。集群（Cluster）技术是近几年兴起的发展高性能计算机的一项技术。它将一组相互独立的计算机，由一网络互联，组成一个单一的计算机系统。作为目前计算机学科技术一热点之一，国内外的厂商推出了很多的产品。如清华同方的探索 108 ，还有 2000 年 9 月 1 日在上海推出的峰值速度为每秒 3000 亿次浮点数操作的集群式高性能计算机系统“自强 2000－SUHPCS”。集群系统最显著的特点是它具有良好的可获用性（Availability），可伸缩性（Scalability）及优秀的性能价格比。目前对集群技术需求最迫切，发展也最快的领域主要是 WEB 应用、科学计算、数据库应用。

本文是作者紧密追踪最新服务器集群发展情况，阅读了大量的国内外资料，分析总结了服务器集群所采用技术的过程中所撰写的。

下面简要地介绍本论文各章的内容。

第一章　介绍了目前应用的需求，服务器技术的发展水平和集群系统的一些基本概念。

第二章　介绍了服务器集群需要解决的问题，并对一种应用层的服务器集群进行了分析，这种集群通过在不同的服务器之间进行文档迁移来实现。

第三章　分析了 LINUX 操作系统的 IP 层的中心式集群的实现方案。找到了这种方案的精华：三种路由方式(网络地址转换，IP 隧道和直接路由)。

第四章　分析了 LINUX 操作系统的分布式集群的实现方案，抓住了这种方案的核心：集群中每一台服务器都充当请求转发器。

第五章　在试验室环境下对 LINUX 虚拟服务器 LVS 的 NAT 方案进行

了性能的测试。对它实现的源代码有一定了解。

第六章　在综合第三章和第四章的成果的前提下，作者提出了新的服务器集群解决方案。并在本章提出了下一步研究的可能方向。

第一章　概述和背景

1.1 目前应用对服务器性能的需求

近年来，信息化浪潮席卷全球，信息技术正以前所未有的深度和广度影响人们的生活。在这个过程中 INTERNET 功不可没，是它将全球的每一个角落串连起来成为一个小村庄。伴随 INTERNET 发展的是大大小小的商业网站如雨后春笋般层出不穷，也涌现了一大批千万用户级的 WEB 站点。在我国，国内各企业纷纷将建设企业网络作为一项重点投资项目进行规划。在这个大环境下面，作为网络灵魂的服务器不得不引起人们的特别关注。服务器作为新世纪的主流计算产品，在网络环境下为用户提供共享资源（包括查询、存储、计算等），在人类对信息的依赖越来越严重的背景下，它的高可靠性、高性能、高吞吐能力、大的存储能力、强大的网络功能和友好的人机界面将是人们关注的焦点。

随着 INTERNET 的指数性增长站点访问人数和访问频度的不断增加,越来越多的客户请求发送到服务器，服务器的负荷不断增加，响应时间越来越长。在目前的情况下如果要等待 8 秒以上才能进入网站或网页，那么将有三分之一的用户因为没有耐性而一走了之，网络延误等同把客户拒之门外。据统计[1]，今年全球因网络延误而损失的网上销售额达到 40 亿美元。因此目前网络发展对服务器提出的一个关键需求就是拥有不断的升级能力。

伴随着客户对系统安全性能要求的不断提高，以及客户应用系统尤其是关键领域的关键性应用对可靠性要求的不断增加，越来越多的应用要求能够有 7x24 的不间断服务。比如在军事，证券，银行等领域。如何建立和维护真正高可靠性的系统是摆在 IT 业者前的一个挑战。信息系统(IS)的主管人员不得不不断地寻找可以确保其关键性商务应用稳固运行的方法。今天，由于计算机技术的飞速发展，硬件服务器平台已经有了相当好的可靠性。据统计，一个独立系统的可靠性在99%以上，如果配备一些系统管理工具，则系统的可靠性可以达到99.5%～99.8%。

从表面上看，这样的可靠性已经很好了，对于一些可靠性要求不高的应用系统来说应该能满足要求了。但是如果一年按 365 天计算，那么每年仍然有 18～44 小时的停机时间，这对于那些任何停工都将产生严重的资产损失和名誉损失的关键性商务应用来说是不可以接受的。

为了满足这些可靠性要求极高的关键性应用，人们使用专用结构和广泛冗余的容错系统，通过以比常规系统昂贵若干倍的代价将系统的可靠性提高到99.9999%以上，也就是平均每年的停机时间降到半分钟！这种解决方案力求做到将故障发生的可能性降至最小，以致几乎不可能发生。虽然它的可靠性的确很高，

但花费的代价也很高。

人们在提高可靠性的过程中，除了了解到可靠性的提高是有代价的，可靠性越高，所需付出的代价就越大以外，还发现了一个重要的现象，那就是当系统的可靠性达到99.95%～99.99%时，故障的主要来源不再是硬件，而是环境和软件。关键应用系统首先要求计算机的性能不断扩展，然后是对系统的可靠性要求不断提高。

早期能够满足人们上述需求的系统主要是超级计算机和大型机，但是随着WEB应用、DSS、OLTP应用的发展和普及，人们很快又希望系统具有良好的可扩展性和高的性能价格比。在这样的大环境下面，服务器集群技术便应运而生。

集群（Cluster）技术是近几年兴起的发展高性能计算机的一项技术。集群技术是一组相互独立的计算机，由网络互联，组成一个单一的计算机系统，并以单一系统的模式加以管理。此单一系统为客户工作站提供高可用性的服务。在大多数模式下，集群中所有的计算机拥有一个共同的名称，集群内的任一系统上运行的服务都可被所有的网络客户所使用。

它使用特定的连接方式，将比超级计算机便宜许多的硬件设备结合起来，提供与超级计算机性能相当的任务处理能力。目前最为流行的方式是用高速网络传输设备将几台服务器相连，实现并行处理，屏蔽单点失效。而目前对集群技术需求最迫切，发展也最快的领域主要是WEB应用、科学计算、数据库应用。

另一个方面，集群技术也是一个发展高性能计算机的强有力的手段。目前世界各国都不再盲目追求高的峰值运算速度，而强调更好地利用高性能计算机，发挥高性能计算的潜能。利用集群技术实现高性能计算机正是适应了这一需求，其优势是可以利用普通的PC、工作站、服务器作为节点，系统造价低；可以实现很高的运算速度，完成大运算量的计算；具有良好的运行环境、程序开发环境，便于用户科学地开发并行应用程序。

1.2 目前服务器发展的情况

计算机技术的出现对现代科技与世界经济的发展以及人们的生活都产生了巨大的影响，其发展之迅猛是其它技术学科所不能比拟的。

就服务器而言，目前国际上服务器技术领域正发生着剧烈的变化。

随着技术的进步，服务器架构经历着巨大的变化，通常我们可以根据这一点将它们划分为以下几类

- 大型主机（MAINFRAME）

  这类机器以IBM的产品为代表。如IBM S/390，这类主机的特点是单台机器处理能力极强，但是它采用专用的设计和架构，价格昂贵，可扩展性差。无法作到平滑地升级和扩展。

- 对称多处理机（SMP）

  代表机器有 SUM ULTRA E10000，现在越来越多的操作系统支持对称多处理，象 WINDOWS NT 可以支持多达 32 个处理器。时下风流正行的 LINUX 也不乏对对称多处理的支持。对称多处理的计算能力不容小视。根据对称多处理的特点，它比较适合中小规模的信息处理系统。由于受到其体系结构的限制，其扩展性较小，单台的对称多处理机无法满足日益增长的信息服务处理需求。

- 大规模并行机（MPP）

  代表机器有 IBM SP2 ，这种机器拥有强大的计算能力和通信能力。缺点是价格不菲性能价格比不好。

- 集群系统(Cluster)

  集群系统是当前构造高性能计算机的一个热点，国内外的厂商推出了很多的产品，如清华同方的探索 108 ，还有 2000 年 9 月 1 日在上海推出的峰值速度为每秒 3000 亿次浮点数操作的集群式高性能计算机系统“自强 2000-SUHPCS”。它将多台独立的计算机通过网络连接起来，组成一个单一点工作节点，提供高性能的信息处理能力。最显著的特点是它具有良好的扩展性和可用性，可以极大的减少整个系统的关机时间，提供 7x24 的不间断服务。另外它还具有平滑的升级能力和优秀的性能价格比。

与服务器采用的架构同步的是，服务器的模式也经历着巨大的变化。

- 传统的终端/主机模式

  在这种模式下面，终端只是完成显示，功能极其简单，无须专门的维护，维护与管理在主机一端实现。它的缺点是主机价格昂贵，终端没有任何的处理能力。

- 客户机/服务器模式

  这种模式对计算产业的标准化和开发化有过极大的推动作用，它得益于分布式的网络计算模型，这种计算模型为客户机/服务器模式提供了巨大的灵活性。但是，随着网络规模和信息处理强度的急剧扩大，整个系统的维护和管理越来越困难，开销巨大。

- 服务器集群系统

  在集群系统中，服务器不再分布在各处，而是集中起来进行统一的管理和维护。它保持了客户机/服务器模式下的可开发性，可扩展性的优点，同时又具有终端/主机模式下的资源共享和集中，很方便管理。

1.3 集群（Cluster）概念

1.3.1 什么是集群

集群是用高性能网络或者 LAN 进行物理连接的计算机的集合。集合里的计算机又叫作节点（Nodes），它们是一些完整的独立的计算机系统。Cluster 中的节点可以是服务器，也可以是工作站，可以是 PC，也可以是大型机甚至是 MPPs (Massively Parallel Processors)。集群中的各个节点在保持本身计算机系统完备性的同时，还应该具有另一个更为重要的特征，即各个节点必须能够在一起协同工作，形成一个单一的，集成的系统资源。[3]

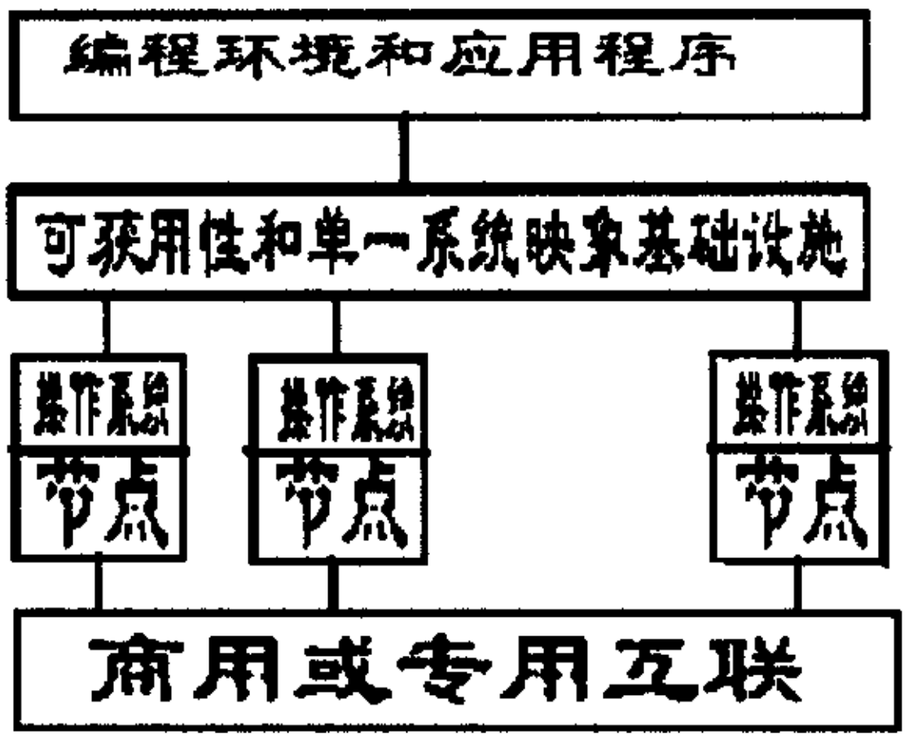

图 1.1 集群系统的典型体系结构

从图 1.1 中可以看出集群系统体系结构的特点

- Cluster 节点：每一个节点是一个完整的计算机。这隐含着完备的计算机系统,以及相应完备的外围设备。此外，在每一个节点上都驻留着一个完整的，标准的操作系统。每一个节点上允许有一个或者多个处理器。但是他们只能有一个操作系统的映像。
- Single-System Image
  整个集群是一个单一的计算处理资源，在这一点上集群系统和分布式系统有区别。集群系统是通过 SSI (Single System Image)技术来实现单一资源的特征。SSI 技术实现了集群的有效管理和简单使用。尽管到现在为止，大多数的集群产品还不能提供完整的 SSI 服务，但 SSI 却是集群的重要特征。
- 节点之间的互联
  每一个节点是用性能尽可能高的网络来连接，如 ETHERNET, FDDI 和 ATM 交换网等。当使用不同的网络连接时，必须提供标准的协议来实现节点之间的平滑通信。

- 增强的可获得性（Availability）
  可获得性表示系统对用户应用可使用的时间的百分比。集群技术提供了有效的、花费更低的解决方案。
- 更好的性能
  集群系统可以在很多的服务领域提供更好的性能。例如，一个集群系统可以作为一个超级 WEB 服务器，或者一个超级的数据库服务器。如果集群中的每一个节点可以支持 N 个客户，那么有 M 个节点的集群系统就可以支持 M*N 个客户。另外，集群技术在并行处理领域，还可以缩短单个任务的执行时间。
- 更好的灵活性和可伸缩性
  集群系统可以根据实际情况灵活地改变整个系统的配置，如增加或者减少节点等。这种平滑的可伸缩性不仅表现在系统规模的可伸缩，还表现在技术，服务的可伸缩。

NOW (Network Of Workstations)构成的集群系统已经成为科学和工程计算、企业日常事务处理、INTERNET 信息处理的主要基础，近年来，由于网络技术和工作站技术的迅猛发展，NOW 以其更好的性能价格比受到人们的青睐，并大有取代大型机甚至巨型机的趋势。

1.3.2 集群的优点

集群概念的提出带来了许多的优点，同时也带来了许多的挑战。其最主要的优点是可易用性，可获得性，可伸缩性和良好的性能价格比。

可易用性：因为集群的单个节点仍旧是传统的平台，所以用户可以在他们平时就很熟悉的环境下面开发和运行应用程序。同时这也可以让现有的许多程序可以不加修改的运行在处理能力强大的集群平台上。非常有利于保护用户已有的软件投资。

可获得性：可获得性包括可靠性和好的可用性等。在传统的系统中，如大型机和容错系统，通常是以高费用为代价来提供可获得性。相反，在集群系统中，却是用低费用的组件来提供较高的可获得性。集群系统可获得性的实现，其关键技术是开发共享组件可获得的软件。

可伸缩性：一个集群系统的处理能力可以简单地通过增加节点来加强。同时，集群的可伸缩性是多面的。包括资源的可伸缩性、应用的可伸缩性、和技术的可伸缩性等。SMP 提供了处理器的可伸缩性，在集群系统中的可伸缩性可以是计算机的各个组件，如处理器、硬盘、内存或者 I/O 设备以及软件组件等。

良好的性能价格比：集群系统良好的性能价格比是它受到人们青睐的重要因素。它可以把一些廉价系统组合在一起协同地工作，在总体上的性能却可以超过大型机甚至巨型机。美国亚特兰大的 EMORY 大学、CESDIS 哥德航天飞行中心

和加利福尼亚技术学院的科学技术人员、研究人员和实验室工作人员一起在1997年[2]建立了 16 个节点的 P-□200 微机集群 Beowulf 系统，只用 5 万美元而使系统达到每秒 10 亿次浮点运算的能力。使用大众化的 PC 通过 LAN 互联而达到超级计算机的功能和能力。

同时，集群技术可以保护用户在原有设备上的硬件投资，用户可以将新旧设备组合起来成为一个集群，达到提供更高的性能的目的。

1.4 可伸缩技术的概念

可伸缩技术是集群技术的一个重要的技术特征。同时，可伸缩技术又是一种内容十分丰富的技术，下面从多个方面对它进行论述：

1.4.1 资源的可伸缩性

资源的可伸缩性是指通过增加 CPU 数量或者进行更多的硬件投资来获得更高的系统性能和改善软件的效率。

系统大小的伸缩：加大一个计算机系统或者一个集群系统的最简单的办法是增加节点数目或者节点内处理器的数目。系统大小伸缩并不是仅仅简单的增加一台机器或者减少一台机器，它往往受到集群系统设计和节点互联方案的制约。

系统资源的伸缩：允许增加集群系统内接点的资源，如 Cache、内存、或者硬盘空间的大小，并以此来提高系统的性能和增加吞吐量。

系统软件的伸缩： 可伸缩系统的软件可以从下面三个方面进行改进

- 应使用最新版本的操作系统来提供更多的功能，比如多用户，多进程， 多线程支持、更大的用户空间支持和效率更高的内核。
- 使用优化的编译器。
- 使用效率更高的数学和工程库。

1.4.2 应用的可伸缩性

为了能够充分利用可伸缩系统的能力，基于 Cluster 的应用程序应该是可伸缩的。这样的应用程序才能在规模不断扩大的集群系统上有更好的性能。衡量应用程序的可伸缩性的两个标准是节点数量的可伸缩和面向问题规模的可伸缩。

节点数量的可伸缩：它表明在系统节点数量不断增加的情况下，系统性能有多大的改善。

问题规模的可伸缩性：它表明在系统有更大的数据处理量和负载时性能的表现如何。

1.4.3 技术的可伸缩性

技术的可伸缩性使得一个可伸缩的系统能够适应技术的改变。具体而言，它可以分为三类： 版本的可伸缩性，空间的可伸缩性和异构性可伸缩性。

版本的可伸缩：一个系统可以用更新的组件进行规模的扩展，如更快的处理器，更快的内存，更新的操作系统和性能更优的编译器等。并且，当系统变迁到新的版本后，它的计算处理能力应该相应的提高。此外，在系统中未作更新的部分应该做到尽可能少的改变，这样可以在对集群中的一个组件进行升级时可以尽量不要求整体的改变。对过去的并行计算机来说，这一点做得很不够，造成用户在对系统的更新后不得不重新开发新的用户应用程序。

空间的可伸缩性：空间的可伸缩性在最初是指在多处理系统中，可扩充的处理器空间。对网络而言，它天然具有无限的空间可伸缩性。

异构可伸缩性：它使得系统可以通过集成不同的软件和硬件组件来扩充系统，它也被称为标准的，开放的体系结构和接口。

1.5 可伸缩性的设计原则

设计一个可伸缩的集群系统是一个复杂过程，通常在设计的过程中应该考虑下面 4 个原则：独立原则，平衡原则，可伸缩原则和延迟隐藏的原则。

1.5.1 独立的设计原则

这个原则要求在集群系统的设计中，保持各个组件间的独立性。即使不能够做到完全的独立，也应该在设计的时候尽量的减少各个组件之间的依赖，这里的组件是指在集群系统内所有的软硬件构成部分。

独立的原则为可伸缩性带来一个很明显的好处： 我们可以通过改善独立于其他组件的部分来扩充整个系统，而不用考虑和升级系统内的其他组件。比如当用户想增加系统的节点数，他并不需要考虑升级操作系统，编程环境和应用程序。

当对处理器进行升级时，同样可以不用升级其他组件而提高系统的性能。这也是异构性给系统带来的好处。这是因为组件并没有绑定在一个特殊的体系结构上面，从而赋予了这个组件可以内置于许多系统的能力，这样，可以极大地减少系统的开支。

对可伸缩集群系统的应用程序的设计，应该有以下的一些原则：

- ✧ 应用程序的算法应该独立于体系结构和平台。
- ✧ 编程语言应该是独立于机器的。
- ✧ 集群中的节点应该是独立于网络的，网络的接口应该独立于网络的拓扑

结构。

实现独立的设计原则有两个通用的技术，体系结构与实现相分离，使用标准的组件。

1.5.2 平衡的设计原则

在一个系统中，多数的组件是较新的和较快的组件，但有的组件却可能很慢，这个组件就成为整个系统的瓶颈。平衡的设计原则就是尽可能的减少在瓶颈环节的性能约束。此外，平衡设计还应该能够避免在单一的组件处的失效导致整个系统的崩溃。

1.5.3 可伸缩的设计原则

它表明在设计可伸缩系统的时候，可伸缩的设计从一开始就应该是一个主要的目标，它允许系统在要求高性能的时候可以扩大规模，同时也可以缩小规模以满足用户在系统花费上的预算要求，可伸缩设计的两个常用的方法是超前设计和向后兼容。

- ✧ 超前设计：

  使用超前设计技术，一个系统的设计不仅要满足当前的处理需求，它还要包括一些将来进行系统扩展时的附加特性的设计。虽然这些特性对当前是一种多余，但对系统将来的升级和性能改善打下了坚实的基础。

- ✧ 向后的兼容设计：

  超前设计是为了满足系统未来的技术和特性要求而进行的。与之互补的一种技术是向后兼容的设计原则。它要求在进行系统的软件硬件设计时，必须考虑到系统降级的情况。用于升级的组件也应该可以用在降级的系统中。

1.5.4 延迟隐藏的设计原则

在一般的集群系统中，一个远程的资源请求操作可能会有一个较长的处理延迟，在一些特定的场合中，必须对这种延迟进行处理。延迟隐藏的技术是将延迟隐藏在整个处理中。

第二章 应用层服务器集群分析

在本章和随后的几章将介绍目前服务器集群的发展情况并分析它们所采用的技术。这是作者查阅了大量零星的国内外资料，从中分析总结的结果。对作者来说，在这个过程中，不仅对集群技术有了深刻理解，还通过比较分析，在第六章提出了新的服务器集群的解决方案。对读者来说，也可以对目前的服务器集群技术有一个比较深入的了解。

本章将要介绍的是一种在应用层实现的集群技术，之所以在最初就介绍这种技术，是为了给第三章将要介绍的中心式 IP 层技术，第四章介绍的分布式 IP 层技术作一个比较的基础。这种安排可能会和这些技术的提出时间产生交叉。

2.1 服务器集群的总体方案讨论

集群不是多台计算机的简单集合，所有结点协调一致地工作，向用户提供单一的、集成的服务。为了实现这一点，服务器集群系统需要解决两个方面的问题。

第一：客户端发来的服务请求如何转发到集群内的真实机器上面。

第二：整个系统采用的任务调度算法，决定下一个服务请求交给集群中的哪一台服务器处理。

我们分析一下目前网络服务的最主要的模式就可以发现：在网络服务中，一端是客户程序，用户在客户端根据自己的不同需要发出请求。另一端是服务程序，服务器端是服务提供者精心设计的一个应用程序，它对客户端发来的各种请求作出相应的响应。在请求和响应的传输过程中，中间还有可能要经过各式各样的代理程序，代理程序通常完成一个转发过程。它将客户的请求发到服务器端，将服务器端返回的响应送到客户端，当然代理还可以完成一些其他的功能，比如缓存、安全等。

根据这个线索，可以找到解决上面第一个问题的出发点：我们可以在不同的层次上实现多台服务器的负载均衡。在服务器集群中解决网络任务分配问题的主要方法可以分为下面几类：：

● 基于客户端的方法

根据服务器的负载情况，用户在客户端作出服务请求发送给集群中哪一台服务器的判断。采用这种方法的典型的代表是 Berkeley 的 Smart Client。通常，客户端的 Applet 向一组服务器查询负载情况，选出负载最轻的服务器，再向该服务器发服务请求；当服务器失效时，Applet 将尝试其他服务器。

● 服务器端的 RR-DNS 方法

RR-DNS 的立足点是利用由域名到 IP 地址的解析过程实现客户请求的分配。通常客户在访问一个站点时，他使用的是这个站点的域名，这个域名要经过 DNS 服务器的解析才能获得这个站点的 IP 地址。对一个大的站点来说，它可以使用一个域名，为它的每一台服务器申请一个 INTERNET 全局 IP 地址。采用这种方法的基本思想是：通过 RR-DNS 服务器把域名轮流解析到这组 WEB 服务器的不同 IP 地址，将负载分到各台服务器上，从而提高整个系统的性能。NCSA 的可伸缩的 WEB 服务器系统就是最早基于轮转域名系统 RR-DNS (Round-Robin Domain Name System)的原型系统。

然而，该方法存在以下问题：

第一，域名服务系统是按层次结构组织的，各级域名服务器都会缓冲解析结果，包括客户端自己也有一个缓存。缓存的存在，导致了这些映射不能及时地反映整个站点负载的最新变化。从而妨碍 Round-Robin 方法在客户端生效，而导致不同 WEB 服务器间严重的负载不平衡。缓解这个问题的一个方法是设置合理的域名到 IP 地址映射的 TTL (Time-To-Live)值。这是一个两难问题， TTL 设置太长，不能及时地反映域名到 IP 的动态变化；TTL 设置过短，DNS 服务器的负载过大，客户端解析域名所花费的时间过大。

第二，由于用户访问请求的突发性和访问方式不同，即使 TTL 值为 0，各服务器间的负载仍存在较严重的负载不平衡，毕竟靠从域名到 IP 的映射来反映当前各台服务器的负载情况有些滞后。

第三，系统的可靠性和可维护性差。考虑下面这种情况：整个系统数台服务器中有些正在正常工作，但有一台或者数台服务器失效，也许是管理员正在对其进行维护。此时从整个系统的角度看，还能提供正常服务。但是，如果客户的域名解析结果是指向这些失效的或者正在维护的服务器，客户得到的结果只能“服务终止”。

●服务器端的应用层负载平衡调度方法

EDDIE、Reverse-Proxy 和 SWEB 都使用基于应用层调度的方法来建立一个可伸缩的 WEB 服务器。它们都将到达的 HTTP 请求转发到不同的 WEB 服务器，取得结果后，再返回给用户。该方法也存在一些问题：首先，从请求到达至处理结束，负载平衡器需要进行四次从核心到用户空间的切换，从用户到负载平衡器和负载平衡器到真实服务器的两次 TCP 连接，系统处理开销特别大，致使系统的伸缩性有限。其次，基于应用层的负载平衡器与应用协议密切相关，对于 HTTP、Proxy 和 SMTP 等应用协议，需要写不同的负载平衡器。本章将详细分析一种在应用层实现的服务器集群，讨论它的实现方式和性能特点。

●服务器端的 IP 层负载平衡调度方法

Berkeley 的 MagicRouter、Cisco 的 LocalDirector、Altheon 的 ACEDirector 和 F5 的 Big/IP 等都是使用网络地址转换方法。MagicRouter 是在 LINUX1.3 版本上应用快速报文插入技术，使得进行负载平衡调度的用户进程访问网络设备接近核

心空间的速度，降低了上下文切换的处理开销，但并不彻底，它只是研究的原型系统，没有成为有用的系统存活下来。在第三章和第四章将详细讨论 IP 层实现的服务器集群。这些系统已经运用到了当前的实际应用中。

2.2 设计构思及其相关背景

2.2.1 设计构思

有些构建分布式 WEB 服务器的技术存在着对一个中心资源节点的依赖，比如依赖于一个所谓的连接转发路由器或者 DNS 服务。系统要用这个资源节点来将客户端见到的单一 IP 服务分配到多个真正的 WEB 服务器上面。在本章中将要分析一种应用层技术，它采用文档迁移的方法实现负载在各台服务器之间的迁移和平衡。姑且把采用这种技术实现的服务器集群称为分布式协同 WEB 服务器。它能够有效地消除中心节点对整个系统的瓶颈限制，很方便地将系统的负载分配到各台服务器上面。在设计这个方案的时候要解决下面的问题：如果在应用层实现，那么它必须与同为应用层的 HTTP 协议语法一致，并且要和通常的客户端软件兼容。

2.2.2 相关背景

在人们对服务器集群的解决过程中，提出了各式各样的基于 DNS 的负载平衡调度方案，NCSA 的可扩展 WEB 服务器由一群同构的服务器组成，采用 RR-DNS 实现负载的平衡调度，并且使用 Andrew 文件系统来实现在这些服务器中的负载共享[4]。IBM 推出的可扩展 WEB 服务器是以 SP-2 并行系统为基础构建。它的核心是一群同构的 RS6000 工作站。这套系统使用 TCP 连接路由器来实现负载的分配[5]，不再基于 DNS 来调度客户的访问请求。但它的构建范围是仅仅局限在紧密耦合系统如：SP-2。

随着各式各样的异构 WEB 服务器的出现，DNS 调度的难度进一步增大。可以在一定程度上说这种方案失去可行性。现在人们提出了各种各样的 RR-DNS 改进方案，就是为了解决 WEB 服务器的异构问题和不规则的客户请求分布问题。两层 RR-DNS 调度就是其中的一种。它将客户请求分成两类，普通型和突发型，以便来处理不规则的客户请求。

另外，还有人提出了基于自适应的 TTL (存活时间 TIME-TO-LIVE)的调度算法。在 DNS 选择那些服务性能较差的服务器或者响应那些来自突发的热点用户的地址映射请求时，给这次映射分配的较短的 TTL。

在文献[6]提出的另一种解决方案是综合考虑多个调度指标如：CPU 使用率，磁盘空间，网络利用率等因素，从而找出最佳的任务作为迁移对象。有两个平衡

负载的方法：DNS 轮转(也就是我们通常说的 RR-DNS)和 HTTP URL 的重定向。DNS 轮转用作最初的任务分配，HTTP URL 重定向根据各服务器的负载情况动态的调整各台服务器的负载。使用 DNS 轮转的一个潜在的问题是有可能产生所谓的热点，而造成非常严重的负载不平衡。在文献[7]中有关于这个技术的详尽论述。

文献[8]对 CISCO 公司的 LocalDirector 系统有着详细的介绍。这个系统使用一个虚拟的服务器接受来自客户的所有请求，LocalDirector 在其中担任了一个智能路由器的功能，它将客户请求包的目标地址（对 LocalDirector 收到的客户包来说，其目的地址是这个集群系统的虚拟 IP 地址）的转换为集群服务器中的真实服务器的 IP 地址。这个系统的目标是提供一个通用的解决方案，以便处理 WEB 访问和其他访问。

在文献[9]中提出的快速包插入是一个用户级的技术。它在 MagicRouter 中被用来平衡负载。MagciRouter 可以让一群服务器不作任何修改而看起来是以一个单一的虚拟 IP 地址向外界提供访问服务。快速包插入技术在通过 MagicRouter 的数据包中修改网络地址。从已有的资料看这个系统有较好的容错性能和负载平衡性能。同所有的采用中心节点的方案一样，MagicRouter 最有可能成为整个系统的瓶颈。

在第四章，将会详细介绍一种称为分布式包重写的技术（DPR）。DPR 技术试图在 IP 层将客户请求转发到目标服务器上。这是一个分布式的算法。它试图通过取消所谓的中心节点来消除系统潜在的瓶颈。所有的机器同时作为 WEB 服务器和 IP 层的请求转发器。同时还使用 RR-DNS 以获得一个大致的平衡负载。

2.3 总体思路

首先，我们可以分析一下大多数的 INTERNET 站点的页面情况。通常一个站点的 WEB 页面的集合可以看作一个有向图，在这个有向图中每一个文档是一个节点，文档中的每一个超级连接是图中的有向弧，从一个节点指向另一个节点。如果存在一个将这个有向图分布在数台服务器上的方法，那么对图中的各个节点的访问负载就可以很平均的分配在这些服务器中，即使是突然改变 WEB 的访问模式。这样就有可能解决构造分布式 WEB 服务器所面临的一个主要问题：负载平衡问题。

其次，通过观察我们还发现：人们访问绝大多数的 WEB 站点时通常是从一个众所周知的入口点进入，就象我们访问雅虎时使用 http：//www.yahoo.com ，访问新浪时使用 http：// www.sina.com.cn 。

设想中的这种方案需要系统自适应地改变 WEB 页面中的超级连接，这样在系统运行过程中动态地修改站点文档有向图在各台服务器之间的分布，这种修改由集群中的 WEB 服务器自动地完成。为了以后叙述方便，把那些维护整个站点

入口的服务器称为 HOME 服务器，对系统中的其他服务器来说，站点的内部文档可会迁移到它们上面，以保持系统的负载平衡，可以把这些服务器称为 CO 服务器。这样 HOME 服务器和 CO 服务器就构成了分布式协同 WEB 服务器集群。它可能为我们解决随着 WEB 访问量的几何增长而来的对服务器的灵活性，可扩展性的需求提供方案。

分布式协同 WEB 服务器解决方案不再依赖于基于 IP 层的包操作技术，域名服务（DNS）和分布式文件系统。

- 网络级或者 IP 包级操作不再需要了，没有什么节点需要了解从客户到服务器的 IP 包流，这样就从根本上去除了传统系统中那个影响整个系统性能的中心节点，通常的情况下就是它最可能成为整个系统的瓶颈。
- 不再通过特制 DNS 来实现隐含的负载平衡，协同服务器利用超链直接将负载（客户的访问请求）从一台服务器转移到另一台机器上面，这种调度的粒度是非常细小的，达到了文档级。
- 这些协同的服务器不再需要在地理上集中或者在一个局域网段内部，它们可以在地理上分布，并且可以将网络流量分布在很多网段上。
- 添加一个新的 WEB 服务器是简单的，灵活的，低成本的，任何可以使用的机器都可可以添加到协同服务器中间来，根本不用考虑它相对于系统中已有的服务器的位置关系。

在后面的几节中，将分析这个系统的原理并将详细讨论在实现过程中可能遇到的问题。

2.4 设计原则

在这一节，将讨论和分析在设计过程中可能遇到的主要问题。

2.4.1 入口点假设

通常人们对一个 WEB 站点，只知道它的几个主要的访问点。这些入口点是一些广泛发布的 HTTP URL 地址。举个例子来说：一些著名的商业站点和门户网站通过互联网、新闻报纸、广播、电视等将站点的入口 URL 发布到世界各地，我们绝少看见内部文档的 URL 发布。因此在设计这个系统的时候，可以基于以下关于 WEB 文档的组织和访问模式的结论[10]：

- 一个 WEB 文档要么是一个人所共知的站点入口点，要么是内部文档。非入口点文档的访问一般是客户先经过入口点，然后顺着入口点内超级连接的指向来到这个 WEB 文档。
- 嵌在一个 WEB 文档中的图片 URL 是很少公布的，用户根本不用知道嵌

入文档的 URL，这是因为在取 WEB 文档时浏览器自动的将图片取回，无须用户的介入。同时还应该注意到图片消耗了网络很大一部分带宽，因为它们相对较大。

- 一个比较流行的技术时使用帧结构(FRAME)，它是由一个帧模板加上几个少为外界所知的 WEB 文档页面构成的。通常的帧模板文件很小的，可以很方便的保留在 HOME 服务器内部，帧的内部文档可能是很大的，它们可以迁移到其他的服务器上，以取得负载的平衡。
- 在这种情况下，用户可以根本不考虑采 WEB 站点内部的 URL 是指向哪里的，就像它们从一些通用的入口访问站点一样。

除了在上面所说的几种情况中外，还有几种情况不得不考虑，那就是 WEB 搜索索引和用户的书签。WEB 搜索索引是由很大的搜索引擎如 altavisa 和 Infoseek 等维护的。这些搜索引擎将特定站点内部的所有满足查询条件的页面公布出来。书签则是用户保存在它们计算机上的文档 URL 地址，这样用户可能以书签的方式访问给定的 WEB 站点内部的任何页面。不过，我们注意到，有很多的方法让这些访问必须经过那些公共入口，如可以在 COOKIE 中或者在 URL 中添加由 CGI 脚本或者 JavaScript 产生的标志、序列号等。

在后面的讨论中，我们假定大多数的访问请求遵循通常的模式，即由一个广为人知的入口点进入，只有少量的访问方式是来自于书签或者搜索引擎等。我们的目标是最优化通常的访问方式，对那些较为少见的方式来说，在一定的程度上其访问的响应时间是有所损失的。

2.4.2 文档迁移

在创建一个WEB站点的最初，所有的文档保留在HOME服务器上面。HOME服务器作为管理员或者开发人员创作这些文档时最初放置的地方，为了方便一致性管理和增强整个系统的鲁棒性（ROBUST），应该考虑在这个服务器上面保存一份文档的原始拷贝。此外，HOME 服务器还负责维护整个站点的入口，以便远端用户对这个站点有个一致的外观。除了那些入口文档以外，其他的文档和图片可以迁移到 CO 服务器上面。所谓的 CO 服务器是用来分担系统负载的计算机，理论上，一个 WEB 文档可以迁移到很多的 CO 服务器上面去，但是为了简化系统的设计和实现，可以先假设一个文档只能迁移到一台 CO 服务器上。

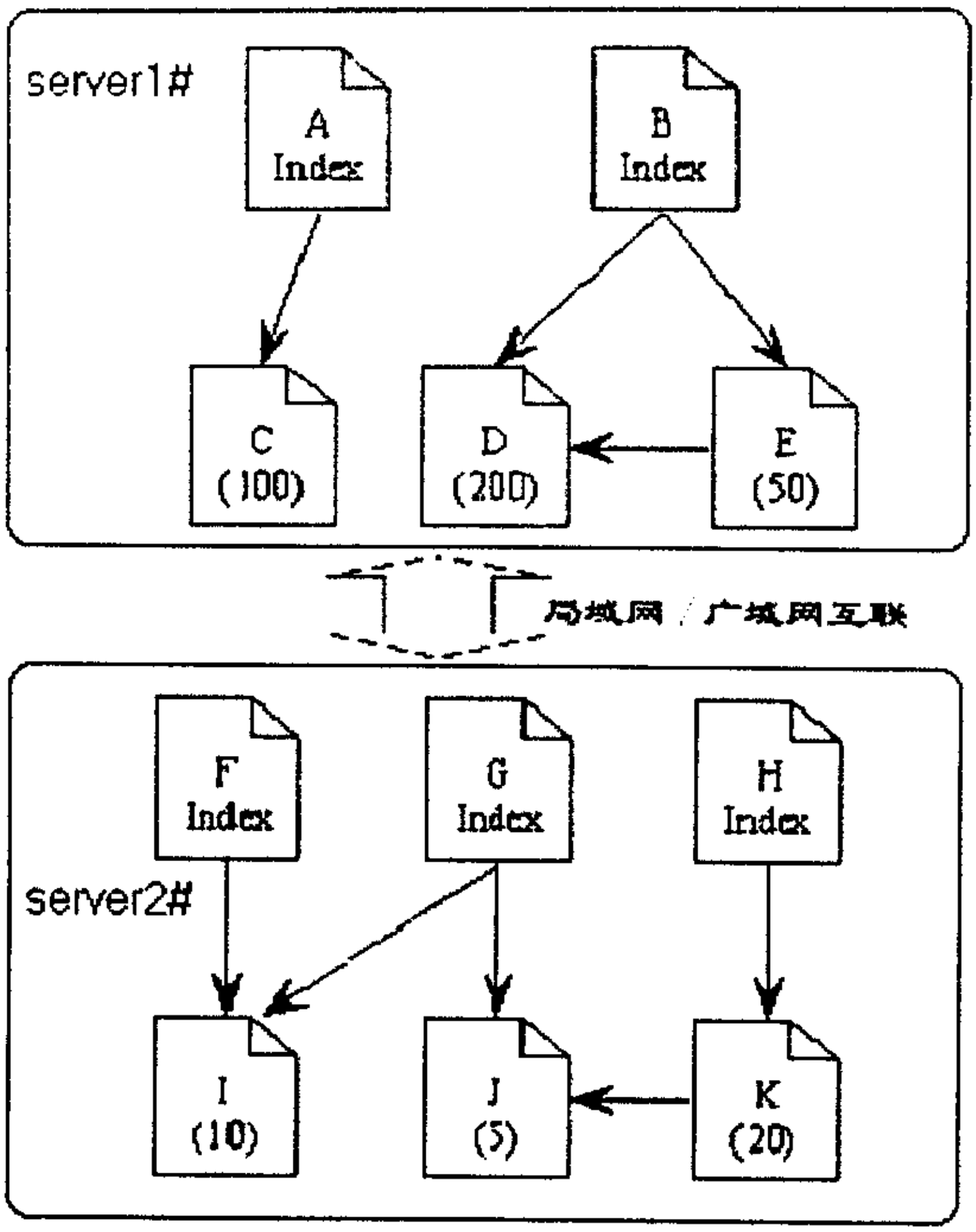

图 2.1　在文档 D 迁移以前的文档视图

在文档迁移的过程中，要处理的一个主要问题是修改文档中嵌入的超级链接，不仅为了保持文档、图片之间的正确指向，更重要的一点是方便客户请求（负载）在 HOME 服务器和 CO 服务器之间的转移。我们来看看图 2.1 和图 2.2 表示的情况：

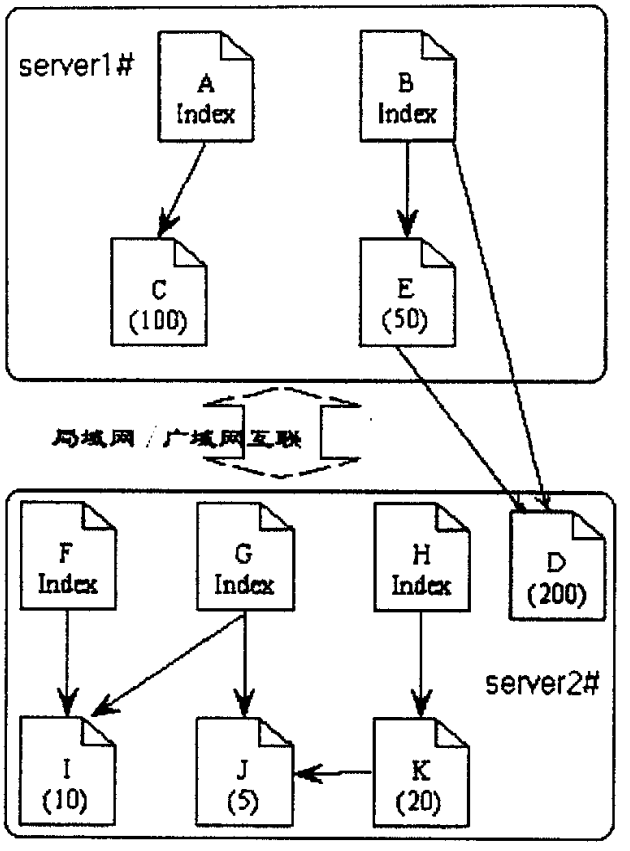

图 2.2 在文档 D 迁移后的文档视图

在图 2.1 中，文档 A-E 和 F-K 最初分布在两台不同的服务器上面，在括号内的数值表明了客户对这个文档的访问命中量也就是负载的情况。当第一台服务器过载了的时候，文档 D 被选中迁移到另外一台机器上面。在上面这个例子中，第二台服务器将成为文档 D 的 CO 服务器。

系统可以采用一个延迟迁移策略（Lazy Migration Policy），以便尽量减少文档的物理迁移，关于文档迁移的进一步讨论将在第四节中详细讨论。

2.4.3 关于文档视图

分布式协同 WEB 服务器的设计从一开始就考虑到系统的对称性，因为系统

中任何一台服务器都可能在保存自己文档的同时，充当其他服务器的CO服务器，因此在系统运行的过程中要维护两张关键的表结构，以便维持系统负载平衡。

首先，每一台服务器都必须保持本机文档的信息以及在这些文档之间的超级链接结构，所有的这些信息存储在本机上一张有向图中。各服务器负责本机的有向图的维护与创建，作到及时地反映本机存储的文档的信息。

其次，所有的服务器都之间将相互交流负载信息，这样才能实现在各服务器之间智能地调度负载。所以负载信息是整个系统的全局信息。总的来说，各个节点向系统提供本机的负载信息，除此以外还要根据其他节点提供的负载信息，维护一张整个系统的负载状况表。

本地文档视图（Local Document Graph　LDG）是一个由下面的组构成的一张表：

（Name，Location，Size，Hits，　LinkTo，LinkFrom，Dirty）

- Name ：Name 域就是文档的简单名字，通常是用户对它的称呼。这个名字应该和文件在磁盘系统上的位置相关，这样可以方便服务器找到它的存储位置。
- Location ：Location 域指明了该文档现在保存在哪一台服务器上，是在一台最初的 HOME 服务器还是在 CO 服务器。
- Size　：　Size 域指明了文件的大小
- Hits　：Hits 域表明了客户访问的命中量，它和上面的 Size 域通常用来计算负载指数，以便找出一个最佳的迁移文档。
- LinkTo ：　这个域表明了当前文档指向了哪些文档。
- LinkFrom ：这个域表明了有哪些文档指向了当前文档。
- Dirty ：Dirty 位用来表示 LinkTo 连接指向的文档是否迁移到了别的机器上面。这样，在适当的时候系统重新生成这个文档，并对这些指向修改这些指向其他文档的超级连接进行修改。表 2.1 显示了在文档 D 迁移到另外一台机器后的一台服务器上面的本地文档图。为了方便起见，诸如文件尺寸等参数没有在这个图中显示出来。

表 2.1 在图 2.2 所示的情况下 Server＃1 上的本地文档视图(LDG)

Name	Location	Size	Hits	LinkTo	LinkFrom	Dirty
A	#1	-	-	{C}	Nil	0
B	#1	-	-	{D，E}	Nil	1
C	#1	-	100	Nil	{A}	0
D	#2	-	200	Nil	{B，E}	0
E	#1	-	50	{D}	{B}	1

本地文档视图是各服务器扫描磁盘上的文档分析后计算出来的。这张表设计成动态结构的(比如以链表的形式组织)，以便在管理员修改文档的内容以后能够及时地更新这张视图。在第五节将分析在一个文档迁移后如何更新本地文档视图，设计时还使用一个 hash 函数来加快这张表的查找速度，同时还注意到有必要采用一个优化的 hash 函数，因为服务器处理的每一个请求都将查找这张文档视图，它的使用率实在是太高了。采用优化的 hash 函数能够减少系统的响应时间。现在还有一个问题是这张表的大小，在设计时假定它是能够完全装入内存的。如果不能够完全装入内存，应该考虑采取一定的措施，加快这张视图的访问速度。

全局负载表 （Global Load Table GLT）存储了整个系统中的服务器状态。每一台服务器选择迁移文档的时候，都必须参考这些信息。所以集群中每一个节点都要保存一个全局负载表的拷贝。对整个系统来说，同时还要花大力气来保持全局负载表的一致性，全局负载表由下面的多元组构成：

（Server，LoadMetric）

- Server： 它是某一台服务器的名字或者地址。
- LoadMetric：它是目前某一台服务器的负载的度量，举个例子来说，LoadMetric 可以是目前每秒钟处理的客户请求数。在负载平衡的计算时它是最重要的指标。每一台服务器要及时地更新它自己当前的 LoadMetric。

各台服务器计算出自己的 LoadMetric 后，一个很重要的问题是服务器之间采用什么方式及时地通报这些负载信息。

通常的情况下，主要的网络流量是来自客户的请求和从服务器返回的响应。如果能够交流负载信息而不引起额外的网络流量，那就是最理想的了。这样就不会再浪费网络的带宽，特别是在当前网络带宽是一种稀缺资源。在这里，可以充分利用 HTTP 传输头捎带传输这些负载信息。

HTTP 协议允许在 HTTP 头中加入扩展的信息，对不能解释这些信息的机器来说，它们将被忽略。这些扩展头可能插在从客户到服务器的请求中，也可能插在从服务器到客户的响应之中。这样我们就可能利用目前的 HTTP 协议来传送任意数量的双向信息。在这个设计方案中 HTTP 传送头很频繁地在服务器之间完成文档的迁移。这种传送头是通告各服务器负载信息的绝佳机会。这样完成负载信息的交流就根本不再需要额外的信道。

在 HTTP 协议头中捎带负载信息并不是对所有情况都合适。通常在各个节点之间需要很频繁地交流负载信息，这时需要采用另外的手段来实现。在设计时为每一台服务器考虑一个特殊的 pinger 线程，它负责监视有哪些负载信息需要立即更新，一旦发现有需要更新的负载信息，它将引发这些负载信息的更新。

图 2.3 描述了在这个应用层服务器集群中的功能模块和数据结构，显示了从客户到服务器的访问请求和在 HOME 服务器、CO 服务器之间的交互。

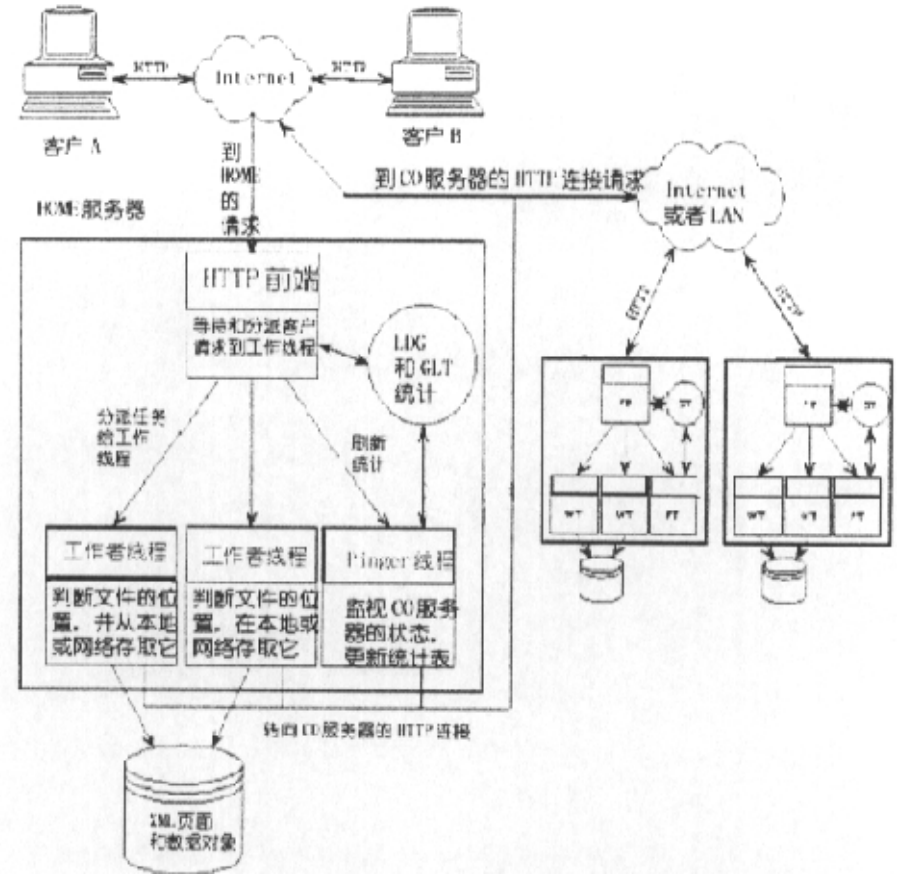

图 2.3 应用层的分布式 WEB 服务器集群的架构图

2.5 文档迁移进程

在本节中，我们将要描述文档迁移的衡量尺度和文档迁移进程的详细情况，和前面提到过的延迟迁移策略（Lazy Migration Policy），超级链接的修改、负载信息的更新和一致性问题等。

2.5.1 文档迁移的选择标准

在决定哪一个文档应该被迁移的时候，有好几个条件和因素需要考虑。在这

些因素中，设计时选择了其中一些，以便尽量少的文档被迁移，同时尽量地维持负载平衡。下面描述了选择迁移文档的算法。

算法 1 迁移文档的选择算法

输入： HOME 服务器上面一个给定的本地文档视图，以及一个负载阀值 T。

输出：本算法选择出一个将要被迁移到 CO 服务器上面的文档。

步骤 1 ：将本地文档视图中所有的文档加入候选集 C 中。

步骤 2 ：将那些广为人知的站点入口从 C 中去除。如果 C 是空集，那么算法返回 Nul。

步骤 3 ：将那些多元组中负载值（Hits）小于阀值 T 的文档从 C 中去除。如果 C 为空集，返回步骤 1。

步骤 4 ：对这些文档来说，指向它们的文档有些迁移到了其他的 CO 服务器上面，有的仍旧驻留在 HOME 服务器。我们在 CO 中选择那些有最少的来自 CO 服务器的链接数的文档。

步骤 5 ：如果在步骤 4 中被选中的文档数超过 1 个，从中选择一个有最小的 LinkTo 数的文档。

算法结束

显而易见，那些广为人知的入口点提供了整个站点的外貌。用户可以从这些入口进入站点内部，如果这些入口被迁移到了 CO 服务器，那么 HOME 服务器就必须将这些 URL 重定向到这些 CO 服务器。因此必须由上面算法的步骤 2 维护站点的入口和避免那些很麻烦的请求重定向。

步骤 3 之所以那么重要是因为我们想尽量少的迁移文档，（被迁移地文档的数量越少越好），换一句话说，就是一个文档的访问频率要高到一定程度，值得迁移进程的开销，如果一个文档不是经常被访问，那么即使将它迁移到其他的服务器上面，它对整个系统负载平衡的意义也不大。

如果一个文档从它的 HOME 服务器迁移到 CO 服务器上面，那么所有有超级链接指向它的文档必须更新它们的超级连接和连接信息表，步骤 4 选择最少的 LinkFrom 数的文档，以便尽量减少这些工作。步骤 5 选择了尽量少的 LinkTo 文档，为的是在以后的操作中，为它创造条件，以便保持一致。

2.5.2 延迟迁移策略(Lazy Migration Policy)

HOME 服务器使用上面的算法选择出来了一个或者多个需要进行迁移的文档，那么进行下面的步骤：

从全局负载表中选择一个负载值最轻的服务器，这可以扫描一下负载表就可以得出结果。这台服务器将作为 CO 服务器，选出的文档将以它作为目的地进行迁移。

HOME 服务器相应地修改本地文档视图，尤其是被迁移文档的 Location 域

将被更新，以反映它的新位置。在这个文档的多元组中 LinkFrom 域中的文档，需要相应的修改它们指向这个被选中文档的超级连接，设置 Dirty 位为 1。这样在下一次这些 Dirty 位置 1 的文档被访问的时候，服务器要重新生成这些文档(修改指向被迁移文档的超级链接)。

到目前为止，上述被选中的文档只是在逻辑上的迁移，物理上的迁移将等到这个文档的确收到一个访问请求。可以称这种迁移策略为延迟迁移策略（Lazy Migration Policy)。

为了更好地说明物理迁移过程，需要更加详细地分析这个应用层服务器集群系统的行为。当客户的一个访问请求来到以后，有两种情况：一种情况是请求的文档恰好在生成它 HOME 服务器上；另一种情况是这个文档已经迁移到 CO 服务器上。

如果收到的客户请求是访问一个已经迁移的文档，那么再分析一下下面这两种情况：

- 这个 CO 服务器的磁盘上还没有这个文档的物理拷贝。在这种情况下面，CO 服务器将向 HOME 服务器发起一个 HTTP 会话（HTTP SESSION)，获得该文档的拷贝，并保存在本机的磁盘上。然后再将向客户传送这份文档，响应客户的请求。
- 如果在 CO 服务器上面已经有了这个文档的拷贝，那么到此为止，这个文档的物理迁移已经进行了，在 CO 服务器上面的拷贝就可以直接的返回到客户。

2.5.3 文档分析和重建

为了修改文档中嵌入的超级链接，我们考虑使用 HTML 语法分析器从一个 HTML 源文件中建立一个简单的分析树。在这个分析树上对那些需要修改的连接进行修改。因为分析和重建一个 HTML 文件是一件很费时的任务，将它的进行推迟到最后一刻进行，以避免不必要的重复和浪费。在文档的多元组中的 Dirty 位就是用来指示这个文档是否需要分析和重建。

当一个客户请求来到，检索一个文档时，如果 Dirty 位没有置位，那么可以认为它是‘干净的’，不需要修改。这时可以直接从磁盘上送出一份拷贝到客户端。如果文档的 Dirty 位已经置位，表示这个文档需要分析和重建，当一个新的拷贝写到磁盘上时，我们将 Dirty 复位。

2.6 性能分析和评价

2.6.1 不同类型的 WEB 站点数据

真实站点的 WEB 数据是可以划分为不同的类型的。可以分为以下几个类型：

- MAPUG Mailing List Archive ： 它是由 1534 个格式化的邮件消息组成的。其中包含了 28998 个超级连接和总共 5918k 的数据量。这些 Email 信息的组织是根据时间，主题和作者组织起来的。一共有 6 个位图文件在其中起到导航按钮的作用。应为这 6 个位图按钮几乎在每一个文档中都使用。它们的请求率是非常高的，成为通常我们所说的热点。
- 2.SBLogWEB Statistics ： 它是由一个统计程序自动产生的。这个文档集一共有 402 个文档。57531 个连接，总共的大小是 8468k 字节。这个测试集的文档视图看起来像一颗树，有少量的总结和索引页作为树内的节点，大量的页面占据的是叶子位置。在这个测试集中有一个 JEPG 图片在所有的文档中作为标志图片，使用率非常高，这也是一个典型的热点。
- Lod role -playing adventure guide： 它有 349 个文档。1433 个超级连接，750k 的数据量。它有一个典型的特点是图片特别的多。一共有 200 个图片。选择这个测试集的原因是目前 WEB 的发展，其图片的使用量正在增大，同时复杂度也增大了。虽然这些图片不再像前几个测试集那样成为热点，但是还是有一些索引文档成为了这个系统的热点。
- Sequoia benchmark data ： Sequoia 2000 storage benchmark 包含了 130 幅在卫星上的高分辨率辐射计测得的光图像。这些图像都被压缩过的，它们的大小一般在 1—2.8M 之间。在前端有一个 HTML 文件，它们分别指向每一个光栅图像文件。

2.6.2 可扩展性和热点文档问题

热点文档尤其是一些很受欢迎的文档和图片限制了整个系统的可扩展性。因为在这种集群系统中，哪一个文档放在哪一台服务器上面是可以控制的。极度的热点文档肯定是要迁移到不相邻的 CO 服务器上面，因为这种情况下系统的为了更有效的分担负载，肯定将这些文档尽量的分布到最多的服务器上面。对 LOD 和 Sequoia 类型的站点来说由于没有热点文档的限制，可能取得较好的系统扩展性。

但是对 sblog 和 Mapug 这种类型的站点，由于有极度热点文档的存在，即使它会尽可能的分布到不同的服务器上面，这些文档的极度过热还是有可能将这几台服务器推向了极限。因此对这种有极度热点文档的站点，可以考虑将它们复制到多台 CO 服务器上面。这样可那能能够克服由于这些极度热点文档对集群规模的限制。

2.6.3 调度标准的讨论

CPS 和 BPS 是关联在一起的。这是由于除了传送有效数据的 IP 包以外，通过 TCP 连接完成 WEB 事务通常还要交换一些信息来完成一些额外的事务如连接的建立和撤除。这样，一个小文件组成的站点可能有较高的 CPS，较低的 BPS。因为在这种情况下更多的网络带宽花在那些额外的信息传递上面了。

真实的 WEB 事务是相当小的。我们选择 CPS 作为系统负载平衡的调度尺度，而不是 BPS。从另外一个方面来说，也有一些系统可能使用较大的文档，对这些系统来说，BSP 可能是一个较好的调度指标。连接的额外花费被大的文件分担了。这种情况下用 BPS 就不太适合了。

2.6.4 评价

这个应用层的 WEB 服务器集群是动态地改嵌在 WEB 文档的 HTTP URL 来实现将客户的请求分布到多个协同的服务器上面。 因为它能够有效地的避免在中心式系统中的瓶颈，不论是磁盘访问速度还是局部的网络带宽，同整个系统在性能上的飞跃相比，修改文档内部的超级连接这一点额外的负担微不足道。在没有典型的热点文档的情况下，可以在十数台服务器的规模下面保持整个系统的基本线性，负载可以很均匀的分布在这些服务器中间。

不论在分布式环境下面还是集中环境下面，这种应用层的集群是构建可扩展 WEB 服务器系统的一个可行方案，它可以将一群分散的服务器集中起来作为一个联合的 WEB 服务器，以便客户访问一个大规模的图像和科学数据，这些数据和图片的产生和存储可能是一群在地理上分布的机器。

在这种应用层 WEB 服务器集群系统中还有很多的问题需要进一步的研究，比如对文档的迁移策略和一致性策略进行进一步的研究，同时还可以对地理分布系统的调度参数进行进一步的研究。还有对那种存在热点效应的站点，将它的热点文档复制到多个的机器上面，再来研究整个系统的并行性和扩展性。

第三章 LINUX 虚拟服务器分析

针对随着 INTERNET 飞速发展带来的对服务器可伸缩性和可靠性的要求，遍布世界各地的 LINUX 开发者正在大力推动一个项目，那就是 LINUX VIRTUAL SERVER PROJECT，有关这个项目的详细信息可以在[11]获得。LINUX VIRTUAL SERVER PROJECT 的进行，使得在 LINUX 内核中能够通过三种实现方法，将一组服务器构建成提供可伸缩的、高可用网络服务的服务器集群，称之为 LINUX 虚拟服务器（LVS）。在 LVS 集群中，服务器集群的结构对客户是透明的，客户访问集群提供的网络服务就像访问一台服务器一样。客户程序不受服务器集群的影响，不需作任何修改。系统的伸缩性通过在服务器集群中透明地加入和删除一个节点来达到，通过检测节点或服务进程故障和正确地重启系统获得高可用性。

3.1 LINUX 发展简史和性能特点

本小节将介绍目前风流正行的 LINUX 的发展情况和它的优点

3.1.1 LINUX 发展简史

1991 年：21 岁的芬兰人 Linus Torvalds 想看看 Intel 386 存储管理硬件是怎样工作的，即兴写出了一种新的操作系统，命名为"LINUX"。然后在 INTERNET 软件新闻组将程序张贴出来，让人免费下载。到 1991 年 12 月，全球范围共 100 人加入 LINUX 新闻组的电子邮件清单。代码行数达到了 1 万行。

1992 年 ：全功能台式 LINUX 操作系统可在 Intel x86 芯片上运行。增加了图形用户操作界面。 此时有 4 万行代码。

1993 年 ：100 多个编程人员先后对代码进行了修改。Torvalds 捐献出源代码，让 5 个核心小组阅览和查对。

1994 年 ：增设了网络功能。代码行数达到了 17 万。

1995 年 ：修改代码可在 Intel、Digital 和 Sun SPARC 芯片上运行。LINUX 杂志开始发行。代码行数增加到 25 万行。

1996 年 ：可同时在几种芯片上运行。代码行数已经达到 40 万行。

1997 年 ：许多国家开始发行 LINUX 月刊。每周都有 LINUX 新版本出现。这时整个系统的代码行数增加到 80 万行。

1998 年 ：约有 10000 编程人员参与新闻组，进行测试和代码修正。代码量更是增加到了 150 万行。

1999 年 ：100 多家厂商推出不同版次的 LINUX 产品；以 LINUX 产品为研发内容的公司开始成为股市新贵，LINUX 的春天到来了。

LINUX 这一新兴的操作系统，正在改变着我们的生活，从高端的服务器市场，到低端的桌面市场，以及新兴的嵌入式操作系统，LINUX 的影响可谓无所不至。而目前 LINUX 最成功的应该还是在服务器市场，IDC1998 年的调查显示，LINUX 已经占据了服务器市场出货量的 25%，排名在 NT 之后的第二位而它的增长率却高达 212%[12]。

3.1.2 LINUX 系统的性能特点

LINUX 的特点适合服务器应用，从 LINUX 本身来讲：

- 兼容性好：

  网络兼容性出色,能够和多种网络系统联接并操作，兼容旧标准，支持新标准。 硬件兼容性出色，从 x86 的新旧硬件平台，到 Alpha、SPARC 等架构都能运行，并表现出很好的性能。
- 扩展性好，定制容易。
- 成本低廉。
- 稳定性得到业界充分认可。
- 网络服务软件多且全。

从服务器应用的特点来讲，LINUX 满足并且适合：

- ✧ 对系统安全稳定性要求高。
- ✧ 对系统资源控制能力的要求高。
- ✧ 对 I/O 及进程处理能力要求高。
- ✧ 对用户的管理能力要求高。
- ✧ 对远程管理和控制能力要求高。
- ✧ 图形化界面使用率低，对图形界面要求不高。
- ✧ 对中文的显示能力要求不高。

由于 LINUX 系统具有如此的优良性能和特点，通常人们可以用来作为以下服务：

- ◆ 适合 ISP/ICP 的 INTERNET 服务器。
- ◆ 由 Apache 搭建 WEB 服务器。
- ◆ 通过 IPAlias 及 Virtual Box 技术提供 Virtual Hosting 服务。
- ◆ 通过远程管理和控制功能管理用户、空间、权限等。
- ◆ DNS、Sendmail 等服务均可通过工具远程配置。
- ◆ 辅以 FTP、Telnet 等功能方便专业用户特殊需要。

3.2 LINUX 虚拟服务器（LVS）集群的体系结构

LVS 采用基于 IP 层负载平衡调度技术，在操作系统核心空间中将 IP 层上的 TCP/UDP 连接请求均衡地转移到不同的服务器上，且负载平衡器自动屏蔽掉服务器的故障，从而将一组服务器构成一个高性能的、高可用的虚拟服务器。为此，在设计时需要考虑系统的透明性、可伸缩性、高可用性和易管理性。LVS 集群的体系结构如图 3.1 所示，它由 3 个主要组成部分：

- 负载平衡器（Load Balancer）：
  它是整个集群对外面的前端机，也即是客户所能够见到的整个系统的外貌。它负责将客户的请求发送到集群内部的一组真实服务器上执行，从客户的角度看来所有的服务都是来自于同一个 IP 的服务器。
- 服务器池（Server Pool）：
  它们是一组真正执行客户请求的服务器，提供 INTERNET 上的绝大多数的服务，如 WEB、MAIL、FTP 和 DNS 等。
- 后端存储（Backend Storage）：
  它为服务器池提供一个共享的存储区，这样很容易使得服务器池拥有相同的内容，提供相同的服务。

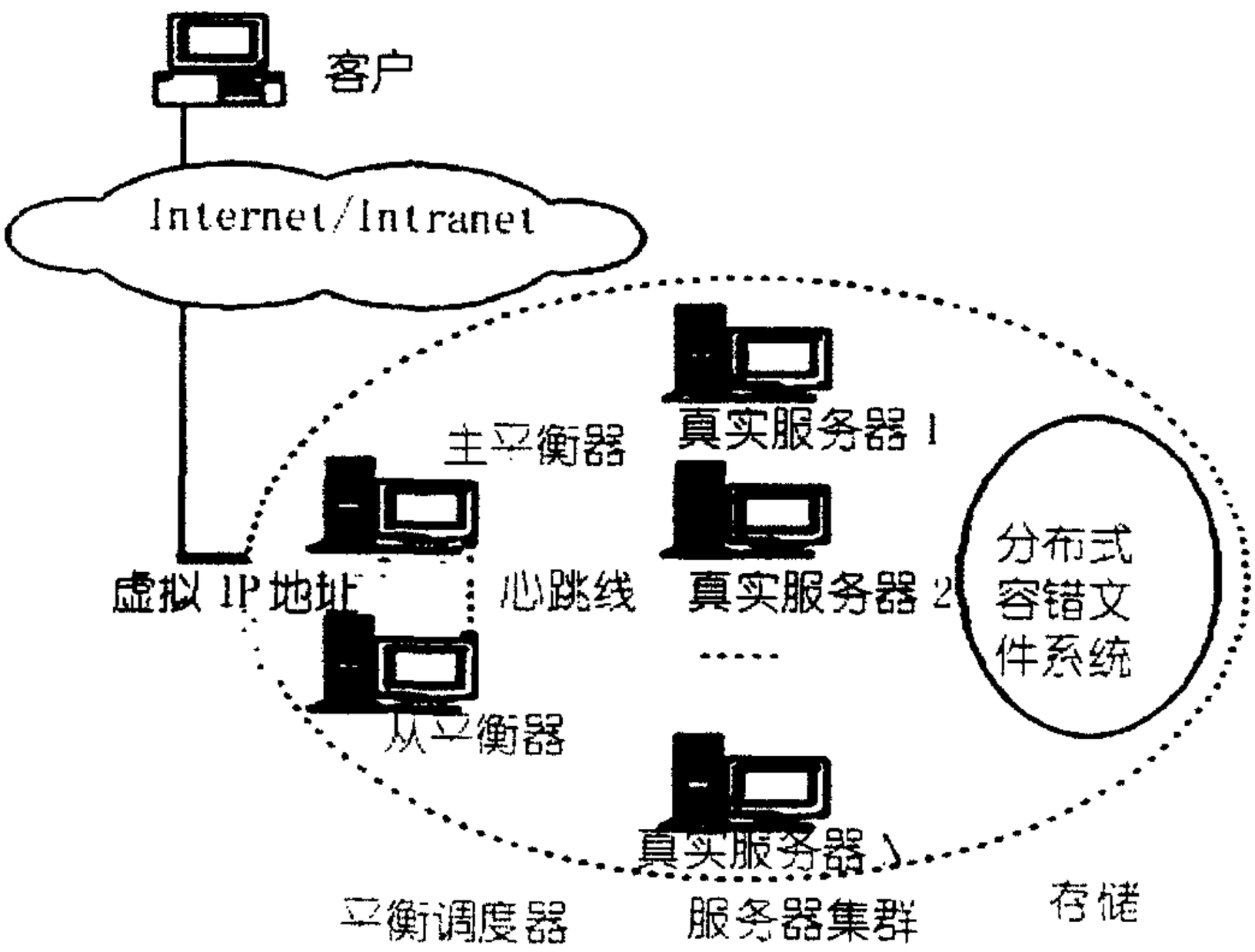

图 3.1 LVS 集群的体系结构

LINUX 虚拟服务器（LVS）负载平衡器采用基于 IP 层负载平衡调度技术。当客户的连接请求到达时，平衡器根据调度算法和当时的负载情况从服务器池中选出一个服务器，将这个连接请求转发到被选中的服务器，同时将这个转发和调度记录在一张路由表中。对 LVS 来说，所有的操作都是在操作系统核心空间中完成的，它的调度开销很小，所以它能调度很多服务器，而本身不会成为系统的瓶颈。服务器池的结点数目是可变的。当整个系统收到的负载超过目前所有结点的处理能力时，可以在服务器池中增加服务器来满足不断增长的请求负载。对大多数网络服务来说，结点与结点间不存在很强的相关性，所以整个系统的性能可以随着服务器池的结点数目增加而线性增长。

集群系统的特点是它在软硬件上都有冗余。系统的高可用性可以通过检测节点或服务进程故障和正确地重启系统来实现。通常，在负载平衡器上由资源监视进程来时刻监视各个服务器结点的健康状况，当服务器对 ICMP 包不可达或者网络服务在指定的时间内没有响应时，资源监视进程通知操作系统内核，将该服务器从调度列表中删除或者失效。这样，新的服务请求就不会调度到坏的结点。

现在前端的负载平衡器有可能成为系统的单一失效点。为了避免平衡器失效导致整个系统不能工作，可以设立平衡器的备份。两个心跳后台进程（Heartbeat Daemon Process）分别在主、从平衡器上运行，它们通过串口线和 UDP 等心跳线来相互汇报各自的健康情况。当从平衡器不能监听到主平衡器的心跳时，从平衡器会接管主平衡器的工作从而开始提供负载调度服务。一般通过 ARP 欺骗（Gratuitous ARP）来接管集群的 Virtual IP Address。当主平衡器恢复时，将自动变成从平衡器，或从平衡器释放 Virtual IP Address，主平衡器收回 Virtual IP Address 并提供负载调度服务。然而，主平衡器的接管会导致已有的调度信息丢失，需要客户程序重新发送请求。

后端存储通常用容错的分布式文件系统，如 AFS、GFS、 Coda 和 Intermezzo 等。这些系统会考虑文件访问的伸缩性和可用性。各服务器访问分布式文件系统就像访问本地文件系统一样。然而，当不同服务器上的应用程序同时访问分布式文件系统上同一资源时，应用程序的访问冲突需要消解才能使得资源处于一致状态。这需要一个分布式锁管理器（Distributed Lock Manager），它可能是分布式文件系统内部提供的，也可能是外部的。

3.3 采用网络地址转换（NAT）实现的虚拟服务器（VS/NAT）

LVS 实现了三种 IP 负载均衡技术，它们分别为通过网络地址转换实现的集群（VS/NAT）、 通过 IP 隧道技术实现的集群（VS/TUN）和采用直接路由实现的集群（VS/DR）。在后面几小节将仔细分析它们的工作原理和它们的优缺点。在这里先作以下约定：称客户的 Socket 和服务器的 Socket 之间的数据通信为连接，无论它们是使用 TCP 还是 UDP 协议。

3.3.1 网络地址转换(NAT)

提出网络地址转换[13]解决 32 位的 IP 地址在 INTERNET 的爆炸性增长面前的日益枯竭，除了在这方面的用途以外，我们还可以用网络地址转换技术来实现服务器的集群，当然这是当初提出 NAT 时没有想到的。

以 IP 为基础构建的 INTERNET 面临两个主要问题是 IP 地址的耗尽和路由表规模的急剧增大，人们提出了不少针对这些问题的短期和长期解决方案，短期的解决方案有 CIDR（无类互连域路由）。长远一点的解决方案包括各种各样的增大 IP 地址空间的 INTERNET 协议之提议，如现在正在试验的 IPv6。在那些长远解决方案成熟以前 IP 地址重用将是对付 IP 地址强劲需求的好方法。网络地址转换利用了在给定时间，一个末梢区域中只有很少的主机在与外界联系的事实，（一个末梢区域是诸如公司网络的区域，它只处理流进或者流出本域的流量）。在很多公司的信息网络中，从各主机担当的功能看来，许多主机从不与内部网以外的主机通信。因此，在一个诸如公司网络的末梢区域中，只是其 IP 地址的一个子集才需要转换为外部通信所必须的全局 IP。

网络地址转换有个缺点，它让 IP 地址丧失了端到端的特性，但它用增加网络状态的方法弥补了这个缺点。有很多方法可以减小这个潜在的缺陷，实际上，面向连接的协议本质上在每一跳就进行了地址的重用。

网络地址转换最大的优点在于它可以在现有的体系下很容易的实现，而不需要改变任何主机和路由器（小部分不常用的应用需要一些改变），这样 NAT 可以很快地实现并测试，如果不受其他因素的影响，网络地址转换可以在其他更为复杂和长远的解决方案成熟以前为人类的信息化助一臂之力。

NAT 的工作原理是报文头（目标地址、源地址和端口等）被正确改写后，客户相信它们连接一个 IP 地址，而不同 IP 地址的服务器组也认为它们是与客户直接相连的。

为了让 NAT 能正常地工作，人们将 IP 地址空间划分为可重用部分和不可重用部分：可重用部分用在诸如公司网络的末梢区域的内部，不可重用部分用做全球唯一性地址。通常人们把可重用地址称为局部地址，不可重用地址称为全局地址。任何给定的地址要么是局部地址，要么是全局地址，两者之间没有重叠的部分。

人们把 10. 0. 0. 0/255. 0. 0. 0 这一个 A 类网地址、172. 16. 0. 0/255. 240. 0. 0 这 1 6 个连续的 B 类网地址、192. 168. 0. 0/255. 255. 0. 0 这 256 个连续的 C 类网地址用作可重用部分。也就是说这些地址只能在内部网上使用，不能在 INTERNET 上作为全局地址。当内部网的主机要访问外部网或者要接受来自外部网访问时就必须进行网络地址转换。

3.3.2 VS/NAT

VS/NAT 的体系结构如图 3.2 所示。网络客户访问集群服务器提供的服务时，他将请求包的目的地址填为集群系统的虚拟地址（这个地址是为负载平衡器使用的外部网地址），负载平衡器检测到这个包以后，发现这个包的目的地址及端口号与服务规则表中描述的某一项服务相符。它通过一个调度算法从真实服务器当中选择一台来处理这个请求。负载平衡器将描述这个连接的表项加入Hash 表中，同时改写这个包的目的地址和端口为所选择的服务器地址及端口并将其转发到这个服务器，属于这个连接的后续包来到时，可以在那张 Hash 表中找到这个真实服务器，这些包同样被负载平衡器改写目的地址及端口号后被转发到真实服务器。源自真实服务器的响应包返回负载平衡器时，它们将被重写源地址和端口号（变为虚拟服务器的地址及端口号）。连接释放或者超时过后，在 Hash 表中的记录将被删除。对改写后的报文，应用增量 Checksum 算法调整 TCP Checksum 的值，避免了扫描整个报文的开销。这样，用户所看到的只是在 Virtual IP Address 上提供的服务，而虚拟服务器的结构对用户是透明的。

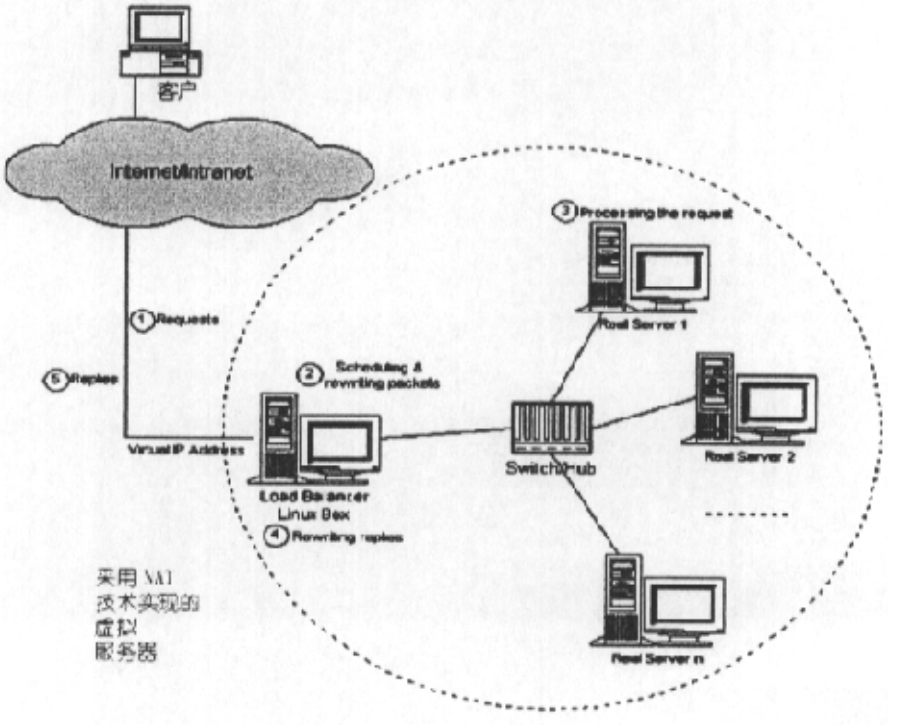

图 3.2 采用网络地址转换实现的集群服务器架构

3.4 采用 IP 隧道技术实现的虚拟服务器(VS/TUN)

在 VS/NAT 的集群系统中，请求和应答的数据包都需要通过负载平衡器，当实际服务器的数量增加到一数量时，负载平衡器将成为集群系统的新瓶颈。许多 INTERNET 服务都有这样的特点：请求报文较短而响应报文往往包含大量的数据。如果能将请求和响应分开处理，即在负载平衡器中只负责调度请求而响应直接返回给客户，将极大地提高整个集群系统的吞吐量。

3.4.1 什么是 IP 隧道技术

IP Tunneling（IP 隧道）是一项将 IP 数据包封装在 IP 数据包中的技术，它把目的地址是某一 IP 的数据包封装起来，并重定向到另一 IP 地址。IP 隧道技术现已广泛的应用在 Extranet，Mobile-IP ，IP-Multicast，Tunneled Host or Network.请参见文档[14]获得详细的资料。

3.4.2 VS/TUN

利用 IP 隧道技术将请求报文封装转发给后端服务器，响应报文能从后端服务器直接返回给客户。但在这里，后端服务器有一组而非一个，所以不可能静态地建立一一对应的隧道，而是动态地选择一台服务器，将请求报文封装和转发给选出的服务器。这样，我们可以利用 IP 隧道的原理将一组服务器上的网络服务组成在一个 IP 地址上的虚拟网络服务。VS/TUN 的体系结构如图 3.3 所示，各个服务器将 VIP 地址配置在自己的 IP 隧道设备上。

通过 IP Tunneling 技术的虚拟服务器与采用 NAT 技术实现的虚拟服务器相比，其最大的差别在于前者的负载平衡器与真实服务器之间采用 IP 隧道技术转发网络客户请求，而后者采用是通过网络地址转换。当网络客户使用虚拟服务器提供的服务时，它发送目的地址是虚拟 IP（虚拟服务器使用的 IP）的数据包。负载平衡器检查这些包的目的地址和端口号，如果这些与虚拟服务器提供的服务匹配的话，它通过调度算法选择一个真实服务器，并将这个连接添加进记录转发连接的 Hash 表中，然后负载平衡器用一个 IP 数据包封装这个请求包并转发它到被选中的真实服务器，属于这个连接的后续包来到后，可以在 Hash 表中找到相应的真实服务器，仍旧由负载平衡器封装这些包并转发到真实服务器。真实服务器接收到封装的包后，他解封装这些包并处理用户请求，请求结果由真实服务器直接发送到用户，在连接终止或者超时的情况下，连接记录将从 Hash 表中删除，整个工作流程如图 3.4 所示：

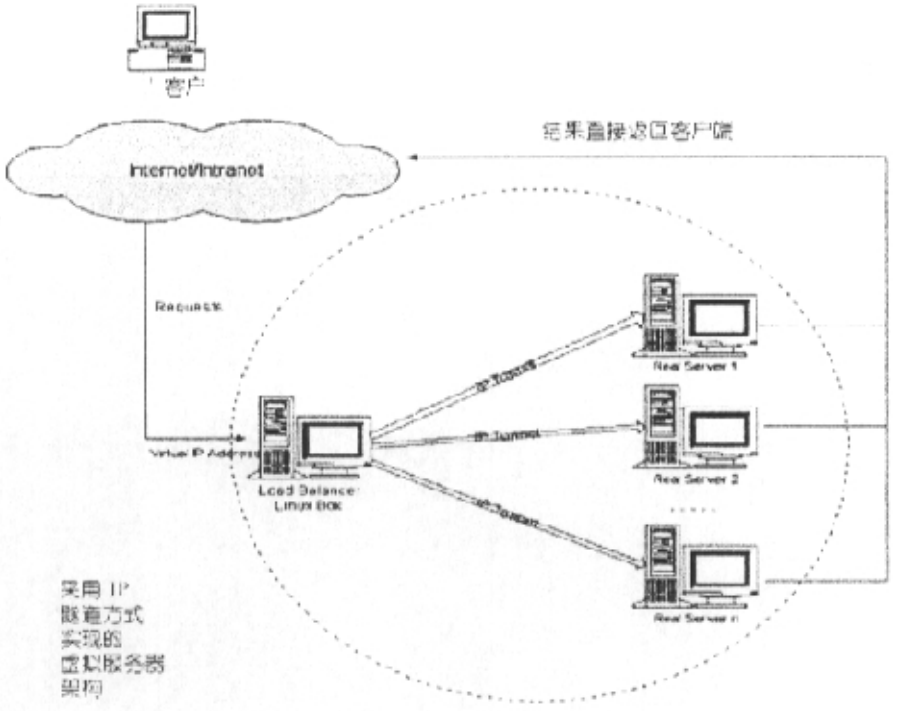

图 3.3 采用 IP 隧道技术的虚拟服务器（VS/TUN）架构

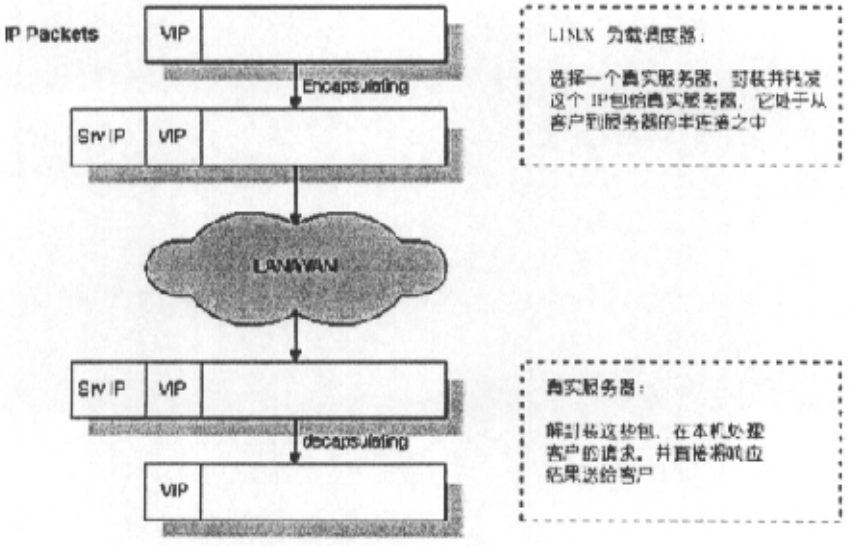

图 3.4 VS/TUN 的客户请求包转发流程

VS/TUN 的连接调度和管理与 VS/NAT 中的一样，只是它的报文转发方法不同。

3.5 采用直接路由技术实现的虚拟服务器(VS/DR)

这种 IP 层负载均衡方法与 IBM 的 NetDispatcher 中的方法类似。它的体系结构如图 3.5 所示：平衡器和服务器组都必须在物理上有一个网卡通过不分段的局域网相连，如通过交换机或者高速的 HUB 相连。虚拟服务器的 IP 地址（VIP）为平衡器和服务器组共享，平衡器配置的 VIP 地址是对外可见的，用于接收虚拟服务的请求报文；所有的服务器把 VIP 地址配置在各自的 Non-ARP 网络设备上，它对外面是不可见的，只是用于处理目标地址为 VIP 的网络请求。

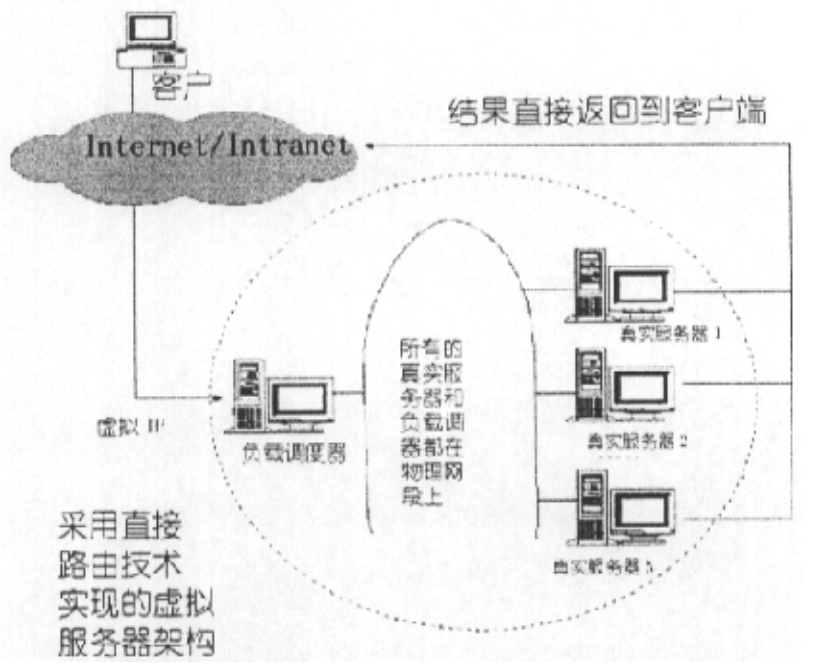

图 3.5 采用直接路由技术的虚拟服务器（VS/DR）架构

VS/DR 的连接调度和管理与 VS/NAT 和 VS/TUN 中的一样，它的报文转发方法又有不同，将报文直接路由给目标服务器。在 VS/DR 中，平衡器根据各个服务器的负载情况，动态地选择一台服务器，不修改也不封装 IP 报文，而是将数据帧的 MAC 地址改为选出服务器的 MAC 地址，再将修改后的数据帧在与服

务器组的局域网上发送；因为数据帧的 MAC 地址是选出的服务器，所以服务器肯定可以收到该报文，发现 VIP 地址被配置在本地的网络设备上，所以就处理这个请求，然后根据路由表将响应报文直接返回给客户。

考虑到直接路由技术的特点：负载平衡器只是简单的将数据帧的 MAC 地址改变为选中的真实服务器的 MAC 地址，然后将其发送到 LAN 上。因此在采用这种技术的虚拟服务器上要求所有的真实服务器必须在同一个 LAN 网段上。

整个系统工作的流程如下：

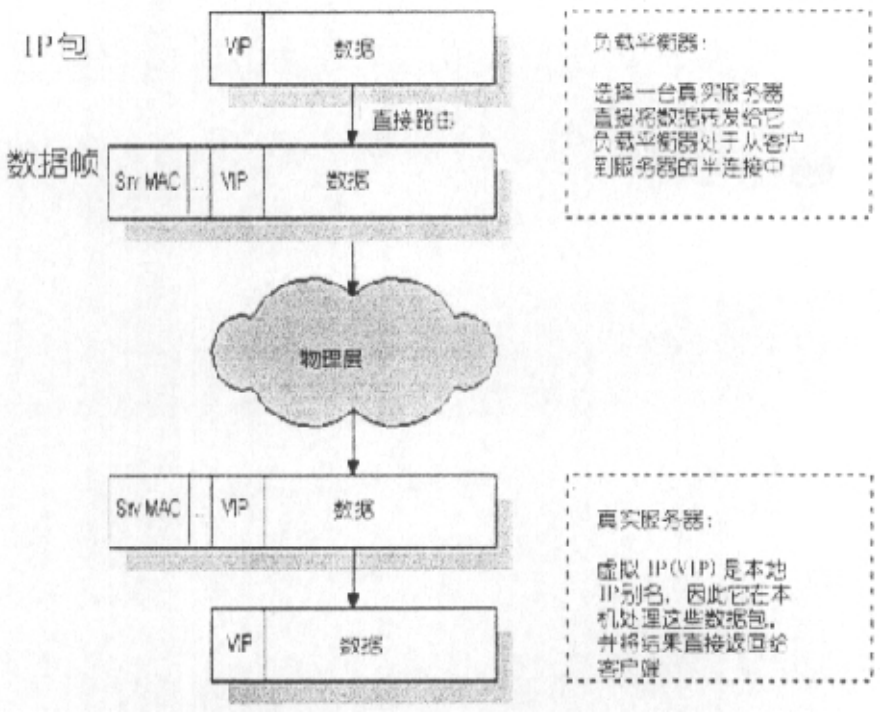

图 3.6 VS/DR 架构下请求包的转发流程

3.6 LINUX 虚拟服务器采用的三种方法之比较

三种 IP 负载均衡技术的优缺点归纳在表 3.1 中：

● VS/NAT

VS/NAT 的优点是服务器可以运行任何支持 TCP/IP 的操作系统，它只需要一个 IP 地址配置在平衡器上，服务器组可以用私有的 IP 地址。缺点是它的伸缩能

力有限，当服务器结点数目升到 20 时，平衡器本身有可能成为系统的新瓶颈，因为在 VS/NAT 中请求和响应报文都需要通过负载平衡器。在 Pentium 166 主机上测得重写报文的平均延时为 60us，假设 TCP 报文的平均长度为 536 Bytes，则平衡器的最大吞吐量为 8.93 MBPS。再假设每台服务器的吞吐量为 600KBPS，这样一个平衡器可以带动 16 台服务器。

表 3.1 LVS 三种路由方法之比较

	VS/NAT	*VS/TUN*	*VS/DR*
服务器要求	无	支持 IP 隧道	支持 Non-ARP 设备
服务器网络	私有	LAN/WAN	LAN
服务器数量	少（10-20）	多（100）	多（100）
服务器网关	负载平衡器	自身路由器	自身路由器

● VS/TUN

在 VS/TUN 的集群系统中，负载平衡器只将请求调度到不同的实际服务器，实际服务器将应答的数据直接返回给用户。这样，负载平衡器就可以处理巨量的请求，而不会成为系统的瓶颈。即使负载平衡器只有 100MbPS 的全双工网卡，虚拟服务器的最大吞吐量可以达到几 GBPS。所以，VS/TUN 可以极大地增加负载平衡器调度的服务器数量，它可以用来构建高性能超级服务器。VS/TUN 技术对服务器的要求是所有的服务器必须支持“IP Tunneling”或者“IP Encapsulation” 协议。目前，VS/TUN 的后端服务器主要运行 LINUX 操作系统。因为“IP Tunneling”正成为各个操作系统的标准协议，所以 VS/TUN 也会适用运行其他操作系统的后端服务器。

● VS/DR

同 VS/TUN 一样，VS/DR 平衡器只处理客户到服务器端的连接，响应数据可以直接从独立的网络路由返回给客户。这可以极大地提高 LVS 集群系统的伸缩性。同 VS/TUN 相比，这种方法没有 IP 隧道的开销，但是要求负载平衡器与实际服务器都有一块网卡连在同一物理网段上，服务器网络设备或者设备别名不作 ARP 响应。

连接调度

负载平衡调度是以连接为粒度的。在 HTTP 协议（非持久）中，每个对象从 WEB 服务器上获取都需要建立一个 TCP 连接，同一用户的不同请求会被调度到不同的服务器上，所以这种细粒度的调度完全避免了用户访问的突发性引起的负载不平衡。通常有以下 4 种调度算法：

- 轮转调度(Round-Robin Scheduling)
- 加权轮转调度(Weighted Round-Robin Scheduling)

- 最小连接调度（Least-Connection Scheduling）
- 加权最小连接调度（Weighted Least-Connection Scheduling）

轮转调度算法是假设所有服务器处理性能均相同，依次将请求调度不同的服务器，算法简单，但不适用于服务器组中处理能力有差异的情况。为此使用加权轮转调度算法，用相应的权值表示服务器的处理能力，将请求数目按权值的比例分配到各服务器。权值高的服务器先收到连接，权值高的服务器比权值低的服务器处理更多的连接，相同权值的服务器处理相同数目的连接数。

最小连接调度是需要记录各个服务器已建立 TCP 连接的数目，把新的连接请求发送当前连接数最小的服务器。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平滑分布到各个服务器上，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 2 分钟，此时连接还占用服务器的资源，所以会出现这样情形，性能高的服务器已处理所收到的连接，连接处于 TIME_WAIT 状态，而性能低的服务器既要忙于处理所收到的连接，还要收到新的连接请求。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。假设每台服务器的权值为 Wi(i=1..n)，TCP 连接数目为 Ti(i=1..n)，依次选 Ti/Wi 为最小者的服务器为调度对象。

LINUX 虚拟服务器在 LINUX 内核实现了三种 IP 负载调度技术和四种连接调度算法。通过负载平衡调度、故障检测技术和集群管理等，将一组服务器组成一个高性能、高可用的网络服务。该系统具有良好的伸缩性，支持几百万个并发连接。无需对客户机和服务器作任何修改，可适用任何 INTERNET 站点。

第四章 LINUX 的 IP 层分布式集群技术

为了建造可扩展的 WEB 服务器，系统设计者把目光转移到了集群技术。但正如前面谈到的一样：在设计一个集群系统的时候，首要的挑战是如何将客户的连接请求高效，均衡地分配到系统中的各台真实服务器上面。在前一章谈到的三种集群技术在本质上有一点是相同的，那就是它们都采用了一个中心节点对客户的请求进行路由和转发。在有些文献[8]中将这个中心节点称为 TCP 路由器。在这种体系结构下面，这个中心节点的 IP 地址通过 DNS 广播出去后，所有的客户请求都发向这个中心节点。其实这种集中式的解决方案本身就不是一个可扩展的系统，在高负荷的情况下，TCP 路由器本身就成了瓶颈。

为了避免这个隐含的瓶颈，人们提出了一种称为分布式包重写的技术（Distribute Packet Rewriting DPR）解决这个问题，DPR 技术同样有着将大量的客户请求交给一群 WEB 服务器处理的思想。DPR 技术和中心节点方案的最大区别在于 DPR 系统中的所有服务器都承担了接收，判断，转发，和处理客户请求的任务，而在中心节点方案中，所有的接受，判断，转发任务都是由一台中心节点来完成。另外一个区别是，DPR 技术使用 RR-DNS 将集群中每一台机器的 IP 地址都广播出去，这样集群系统中每一台机器都可以接收来自客户的请求。

4.1 分布式包重写技术的背景

不断增长的 WWW 服务对 INTERNET 资源和 WEB 服务器的要求越来越高，现在人们不得不考虑 WEB 服务器的高可用性。单台高性能的 WEB 服务器对这个问题的解决能力有着先天的不足，现实的需求要求我们的 WEB 服务器有着内在的可扩展性。在人们将目光转向集群技术的时候，各式各样的解决方案呈献在我们的面前。

早期在一个集群中处理分布和分配用户的连接请求是通过 Round Robin DNS （RR_DNS）机制来实现的，它将一个域名映射到多个 IP 地址之上。由于 DNS 协议的非及时性（如它的 cache 机制）RR-DNS 在解决系统的负载平衡和容错性上的表现并不理想。

对在第三章中提到的 LVS 类中心连接路由方案来说，存在一个中心节点来将客户的连接请求转发到集群中的一台台服务器上。除了上面提到的 LVS 系统外，这类方案还有 BERKELEY 大学 NOW 项目开发的“MagicRouter”[15]，它基于包过滤技术将网络请求分配到集群系统中的各台机器。正如图 4.1 显示的一

样，MagicRouter 充当了一个交换中心，它将客户的 WEB 服务请求转发到集群中各个节点上去。实现这个功能需要 MagicRouter 把客户发来的包转发（重写）到被选中的服务器上。实际上，包重写过程打破了人们的高性能幻觉，这种由一个路由器和一群服务器构成的 WEB 服务实际上是达不到高性能的。对 MagicRouter 的工作来说，其重点是缩短重写所花的时间，而并不是如何实现负载的平衡。另外还有一些这类方案：包括 Cisco 公司的 LocalDirector 和 IBM 公司的 Interactive Network Dispather。

在 1996 年 Dias 等提出了一个和 MagicRouter 有一点小小差别的解决方案[16]。前端一个 TCP Router 将客户的 WEB 服务请求转发到后端的一群服务器。TCP Router 和前面提到的 MagicRouter 有两点差别：首先，正如图 4.2 中所显示的一样，从服务器到客户端的重写过程被省去了。这一点在响应数据很多的时候显的特别重要。作到这一点通常要对服务器操作系统的内核进行修改。在 MagicRouter 解决方案中则没有这个要求。第二点，TCP Router 是根据各个节点的状态来决定下一个请求是转发到哪一个节点。这就是说 TCP Router 必须保存各个节点的连接数。

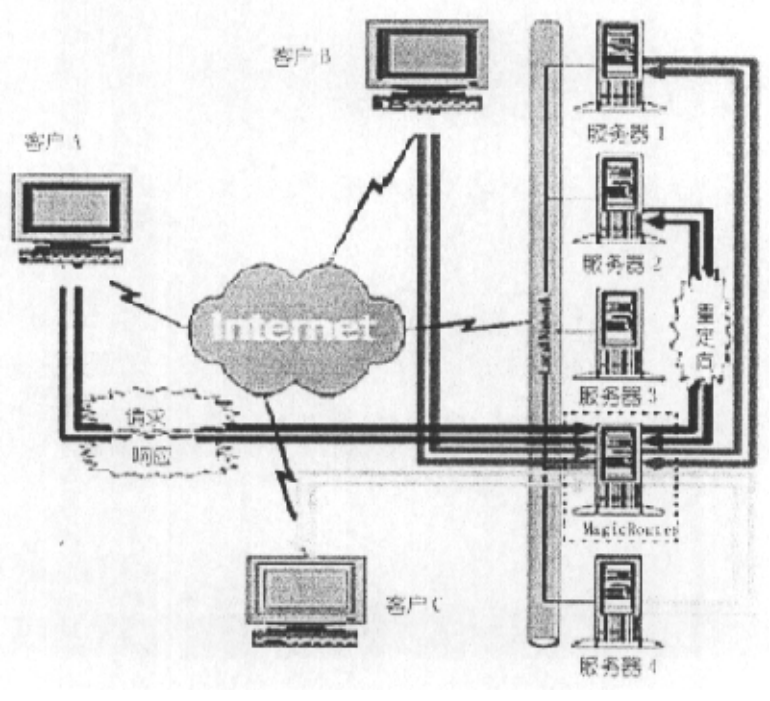

图 4.1　有两次包重写过程的中心式集群架构

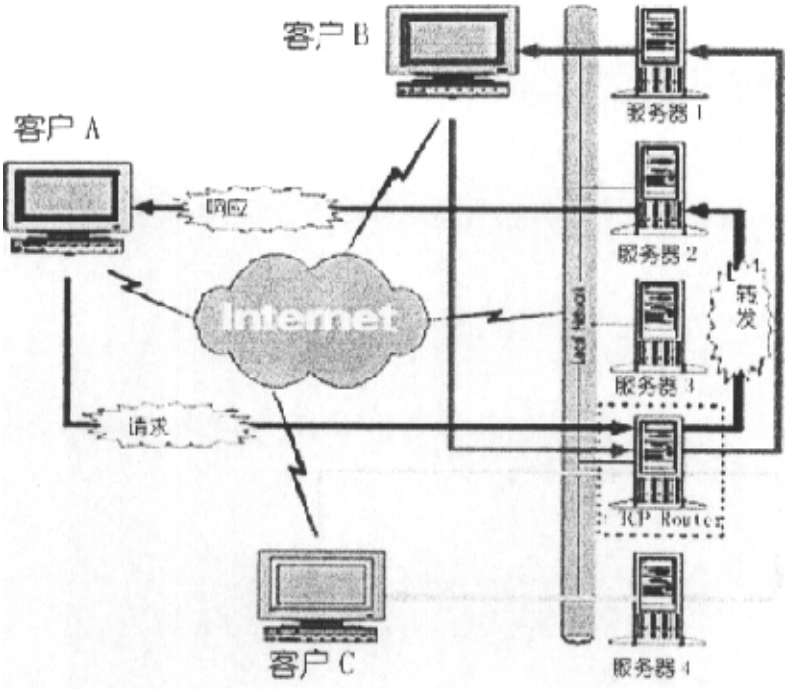

图 4.2 只有一次包重写过程的中心式服务器集群架构

4.2 分布式连接路由

以上所有的连接路由技术（通常被称为第四层交换）都使用一个中心节点来处理客户发来的连接请求。与此相对，分布式包重写技术（也被称为分布式路由）将这个任务分配给数台机器执行。如图 4.3 所示，使用 DPR 技术，分布式系统中的所有机器都参与了连接路由。这种分布式的方法，有可能提供一个更佳的可扩展性和容错性。在中心式连接路由方案下，存在对这个节点的过度依赖。

DPR 是 IP 层的转发机制。集群中服务器具有根据客户端发来的第一个包（SYN）就可以将这个连接转发到集群中的其他服务器的能力。这就是说转发的决定是根据 SYN 包中的信息（包中的 SRC/DST IP 地址和 SRC/DST 端口）和机集群中各机器的状态（各机的负载大小）做出。根据这些信息，一个装备了 DPR 的服务器要么转发一个连接到别的服务器，要么把这些包放在自己的堆栈中，然后送到上面的应用层。从具体的实现方式上说，DPR 技术可以考虑两种版本：一种是有状态的，另一种是无状态的。无状态的 DPR 除了使用包中的信

息以外不需要任何信息。它使用一个 Hash 算法将一个一个的 IP 包转发到另一台服务器上去。有状态的 DPR 技术保持了一张连接转换表，它指明了一个已知连接被转发到了哪一台服务器上。

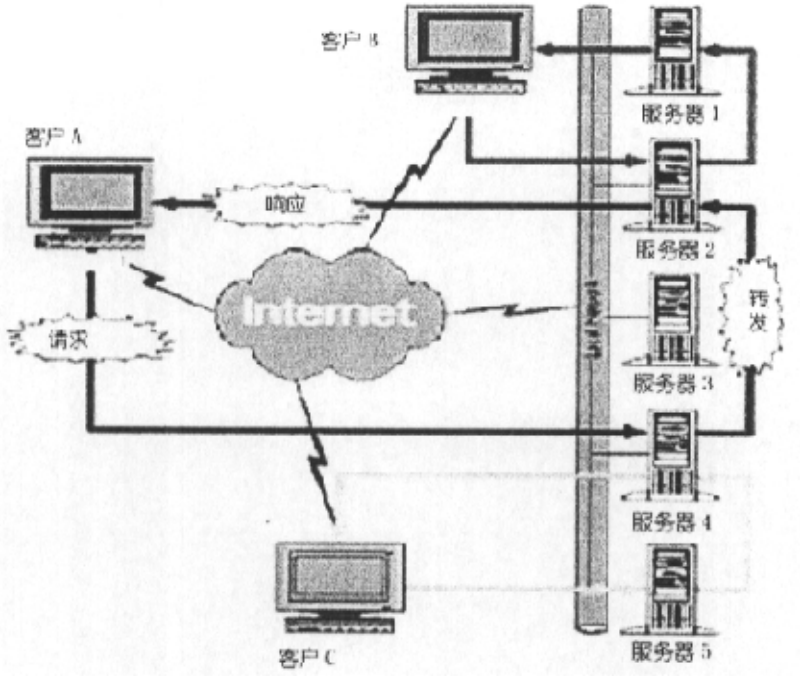

图 4.3 分布式连接路由服务器集群架构

在文献[17]中，试验中的 DPR 采用了一个随机算法判断一个连接是由自己来处理还是转发到别的机器。对包中的 TCP 源端口运用一个 Hash 函数，就可以做出这个判断。这个实现完全是一个无状态的方案。它不考虑目当前服务器的负载状态。在本文中，将要分析一个有状态的方案。这个方案在决定将包转发到其他服务器上的时候将要考虑到整个系统中各服务器的负载情况，这将有可能取得更好的吞吐量和平均响应时间。

4.3 DPR 的一种实现方式探讨

对设计中的 DPR 集群系统来说，集群中的每一台服务器不仅要像通常的服务

器一样响应客户的访问请求，还要具备判断，转发这些请求的能力（通过包重写机制）。这些服务器的 IP 地址通过 RR-DNS 方式发布出去，系统中所有的服务器都可以收到客户的请求。这些请求要么被转发到别的机器上，要么由收到包的主机自己来处理。在前一种情况下，服务请求可以转发到任意机器上，处理的结果将直接返回客户端。

4.3.1 DPR 技术主要过程

为了更好的说明这种包重写技术，假设在图 4.3 中的服务器 4 收到了从客户 B 来的原始请求，进一步假设根据当前各服务器的负载信息和其他的评判标准，服务器 4 作出判断，如果本机不处理这个请求，而是由服务器 2 来处理这个请求，这对整个系统来说是最优的。因此要把这个请求转发到服务器 2 上。这时服务器 4 充当了一个路由器，它把包重写后转发到服务器 2。服务器 2 响应这个请求时，使用服务器 4 的 IP 地址作为响应包的源地址。这样就可以很有效地屏蔽后面的转发过程。对客户 B 来说，它所见的仍旧是服务器 4，根本没有感觉到服务器 2 的存在。它的后续请求包仍然会以服务器 4 作为目的地。注意到在这个过程中服务器 4 实际上是充当了客户 B 访问请求的路由器，它同样可以为其他客户的请求服务，例如，它将客户 C 的请求转发到服务器 5 上面。

在一个具体的实现方案中，有必要让集群中的各服务器了解转发包和直接从客户发来的包的差别。进一步来说，如果服务器 4 转发一个请求到服务器 2，那么服务器 4 必须让服务器 2 知道客户 B 的 IP 地址，以便它正确地对请求作出响应。为了解决这些问题我们使用了 IP-IP 封装技术[18]。

使用 IP-IP 封装技术，服务器 4 将客户 B 发来的 IP 包封装为另一个 IP 包。然后再重新转发到服务器 2。服务器 2 可以从这里推断出这个包是从服务器 4 路由而来，也可以正确的将响应包正确的返回到客户端。因为他可以在封装包内找到客户 B 的 IP 地址。

为了能够完成 DPR 的无状态转发，每台机器都必须保持一张随时更新的表，以便了解集群中其他机器的 IP 地址和负载情况。主机不间断的通过多播 IP 广播自己的负载情况。这些信息被服务器用来决定收到的连接请求是转发到别的服务器上还是由自己来处理。同时每台机器还要保持一张路由表，以保存重定向的连接信息。

对任何一个集群系统来说，一个很重要的策略是它的负载调度策略。在实现中可以考虑使用下面描述的负载均衡算法。当客户 B 有一个新的连接请求（也就是一个 TCP 连接的 SYN 包）到达服务器 4 时，服务器首先检测自己的负载情况，如果负载小于一个特定的阀值如 Maxload，那么这个请求就由服务器 4 自己来处理。反之，服务器 4 将在连接转发表中创建一个表项，记录这次转发的目的地址以及其他一些信息。然后将这个 SYN 包转发到集群中其他的服务器上面。连接的

后续请求包将根据连接转发表中的信息转发到正确的服务器上。对于某一服务器的最大负载 Maxload 值，可以根据参考一些配置情况，如 CPU 速度，内存大小，磁盘容量。

在转发连接请求时可以使用两种方法来选择目标服务器。第一种是比较各服务器当前的负载指数，选中的目标服务器是负载最轻的。第二种是比较各服务器的负载指数的倒数，倒数越大，这台服务器越可能被选中。后一种方法的优点是能够避免可能出现的拥塞。前一种方式可能造成这么一种情况发生，在小段时间内客户的请求数突然增加，其增加速度超过了系统负载信息的刷新速度，所有的请求都被转发到当前宣称自己是最轻负载的机器上面，造成这台机器的临时过载。影响系统对这些请求的服务质量。

我们可以使用以下一些标准来衡量一台机器的负载情况：

- 在某一给定时刻，每一台机器打开的 TCP 连接数。
- 每一台机器的 CPU 利用率。
- 集群中每一台机器重新转发的 TCP 连接数。
- 集群中每一台机器的活动 socket 数目。

同时，考虑到上述几个判断标准的不同权重，还可以产生不同的算法。不同的权值通常产生不同的调度效果。

4.3.2 内核空间实现路由功能

当一台机器接收到一个 IP 包时调用函数 ip-receive（）。为了实现 DPR 技术需要对这个函数作一些修改，以便完成转发连接的功能。在这个函数中要对 IP 包的内容进行检查，如果它包含了一个 TCP 包并且这个包中 TCP 端口是 80(或者其他任意 WEB 服务所绑定的端口)，我们就知道这个 TCP 连接是一个直接从客户来的 HTTP 连接。如果这个包中包括了 SYN ，那么可以进一步的判断这是一个新的连接请求。此时，服务器需要对这个连接请求作出是否转发的判断。正如前面所介绍的一样，这个判断是根据当前集群系统中所有的服务器的负载情况作出的。如果本机的负荷小于阀值或者即使超过阀值但它还是集群中负载最轻的机器，那么这个请求就将由本机来服务，此时不必更新连接转发表。如果目前的负载情况已经超过了阀值，系统中还存在其他负载较轻的服务器，此时需要更新连接转发表，这个连接的请求包将转发到其他的服务器，这个目标服务器的选择，可能根据前面提到的两种选择方法。如果在这个 TCP 包中没有包含 SYN，那么这是一个请求的后续包。如果这个请求是由本服务器处理的，那么将这个包交给协议的上层处理就行了。如果这个连接是转发到其他服务器进行处理的，此时需要根据连接转发表中的信息，将这个包转发到它的最终服务器上。如前面所述的，这个转发过程可以采用 IP-IP 封装技术。需要进行转发。那么在此时就将它转发到其他的机器上面。

对服务器来说，如果收到的 IP 包，在包头的协议字段的值指明这个 IP 包是一个 IPinIP 包，服务器可以据此判断这个包是从其他的服务器上转发而来的，需要本服务器进行处理，此时解开这个 IPinIP 包，将其中的 TCP 包送到上层处理。另外，我们还可以查看收到的 IP 包，如果它的源地址是集群系统中某一服务器的 IP 地址，那么就可以判断这是一个转发包。需要进行 IPinIP 的解包处理。

集群系统中的服务器处理来自其他服务器转发的客户请求和通常的情况一样。对上层的 HTTP 应用程序来说，包的来源透明的。上层应用产生的响应数据将经由 TCP 层送到 IP 层。

当一台机器要发送 IP 包时，需要调用函数 ip-send（），为了实现 DPR 技术，需要修改这个函数，因为对采用这种架构的服务器在向客户送回响应数据包时，要区别两种情况：如果这个到客户的连接是直接来自客户的，那么和通常的情况没有区别，响应包中的源地址就是这台服务器的 IP 地址。如果这个到客户的连接是经由集群系统中的其他服务器转发的，那么，为了让客户对整个站点有一个一致的外貌，需要在返回客户端的回包中将其源地址设置为转发服务器的 IP 地址。

4.3.3 用户模式下实现集群的负载平衡

对于服务器集群来说，完成客户连接请求的转发只是万里长征第一步，对它来说还要考虑集群系统的负载平衡，这涉及到如何评价服务器的负荷，如何在各台服务器之间广播负载信息。整个系统采用什么样的调度策略。

为了取得本机的负载情况，需要设计一个进程或者线程来计算当前的负载值，计算的依据是 CPU 利用率，打开的 TCP 连接数，活动的 Socket 数目以及正在服务的转发数等。

为了让系统中的其他服务器了解本机的负载状态，需要第二个进程或者线程广播本机的负载情况。在通常的情况下，网络带宽是一种稀缺资源。为了减少负载信息广播过程对带宽资源的占用，各服务器在广播自己的负载信息时采用组播 IP 。在文档[19]中指出，可以采用 224.*.*.*的地址实现组播。

集群中的服务器还需要一个进程或者线程来收集其他服务器的负载信息，它监听各个机器的广播信息，一旦接收到了新的负载值，调用一个系统调用更新在内核中的链表。

最后，还需要一个进程或者线程负责清除负载情况表和连接转发表。如果间隔一定的时间还没有收到从某一服务器来的负载情况报文，也就是说没有听到这台服务器的心跳声，在内核负载表中关于这个服务器的表项将被清除。以免转发客户的连接请求到没有运行的机器上面。这个实施方案也给集群系统带来了方便的维护性，在维护性关机的时候，其它的服务器是不会将客户的连接请求转发到这台机器上的。

使用 IP-IP 技术将连接转发到其他的服务器，还可以方便的实现将服务器分布在几个网络上。我们只需要稍微改动一下广播负载情况的进程就可以。如果不止一个网络上的机器参加 DPR，这个进程就不能只在本网络内广播服务器的负载情况，他应该在所有由机器参与 DPR 集群的网络上广播各台机器的负载情况，这些参与 DPR 集群的服务器所在的网络可以在广播进程初始化的时候通过一个配置文件的形式维护。

4.4 DPR 技术的讨论

前面提出了采用 DPR 技术的分布式连接路由实现的集群的初步原理。DPR 可以在没有中心节点的情况下将连接重新转发到分布的服务器上。与以前的系统采用一个特定的节点将连接转发到它们的目标不一样的是，DPR 让这个分布式系统中的所有机器都参与了连接的路由。在文献[17]中，对 DPR 技术的主要意图有所阐述，同时还论证了这个技术的可扩展性。

关于这个模型需要指出的是将连接的重写转发任务交给系统中的机器，这有可能造成一个它们难以负担的额外工作。然而随着系统体系结构的发展，这种额外的工作将可忽略。因为 I/O 接口硬件和网络接口硬件功能正在迅速地增强。在通用计算机系统越来越多的智能 I/O 指令（I^2O）正在成为软件和硬件接口的标准。随着系统功能的增强，加上这个趋势的加速，象 DPR 这样的解决方案将更加有吸引力。

DPR 的功能并没有完全的取代中心连接路由器（如 Network Dispather 或者 LoaclDirector）的功能。这种以单一 IP 表现的连接路由器将完成包的重写，负载平衡和潜在的网关功能。DPR 表现出来的并不是一个单一的 IP。它也不会完成网关的功能。但是，我们知道仅仅是简单的 RR-DNS 就可以给客户一个单一名称的幻觉。一个标准的路由器就可以完成网关的功能。

DPR 技术和以前的集中式方案相比，它的优点是：系统重写转发连接的能与系统的规模成正比，如果任何一个节点失效，整个系统也不至于崩溃。对小规模系统来说 DPR 技术也有它特别的用处，例如，如果一台 WEB 服务器需要的处理能力由一台机器变成了两台，在集中模式下，需要购买两台额外的机器，一台新的主机外加一个连接路由器，即使连接路由器的大部分处理能力都被浪费了。DPR 技术可以做到最有效的处理分布式服务器的规模。

第五章 VS/NAT 实现分析与测试

为了具体分析在 LINUX 服务器集群（LVS）的实现，我在它的三种解决方案中选取了 VS/NAT 作为具体的例子进行深入探讨。与其他两种方案相比，VS/NAT 技术的优点是真实服务器可以运行任何支持 TCP/IP 协议的操作系统，这就意味着真实服务器不一定非要使用 LINUX/UNIX 系统，它还可以是 MS WINDOWS 系列操作系统。同时，真实服务器可以使用内部网地址，只需要为负载平衡器分配一个合法 IP 地址。从总体上说 NAT 方案对预期访问量比较小的网站来说，是比较合适的。

5.1 LINUX 虚拟服务器的 NAT 方案实现分析

由于 LINUX 的 TCP/IP 由核心实现，实现虚拟服务器需要对核心加以改造。

5.1.1 LINUX 内核的 NAT 机制：

所谓 NAT 即网络地址转换。因为 IPv4 协议带来的 IP 短缺问题和一些安全方面的原因，越来越多的网络使用内部 IP 地址(例如 10.0.0.0/255.0.0.0，172.16.0.0/255.240.0.0 和 192.168.0.0/255.255.0.0)，这样的地址是不能在 INTERNET 上使用的。当内部网络的主机需要访问 INTERNET 或被 INTERNET 访问时，网络地址转换就有了其必要性。

IP 地址通过网络地址转换被成组映射到其它地址。当映射是 N 到 N 时，称作静态网络地址转换；当映射是 M 到 N 时(M>N)，称作动态网络地址转换。网络地址端口转换是对基本的 NAT 的一个扩充，通过它将大量网络地址和它们的 TCP/UDP 端口转换到一个单独的网络地址和它的 TCP/UDP 端口中。这种映射是 N 到 1 的映射，LINUX 的 IP 伪装(IP Masque)就是通过它实现的。关于网络地址转换在第一章已经有了较为详细的描述。

5.1.2 IP MASQ 分析

通过对 IP 伪装的背景和实现的分析，可以发现，IP 伪装与实现虚拟服务器

有一定的相似之处，LINUX 内核中关于 IP 伪装的代码可能对我们的问题有所帮助。IP 伪装是一种多用途的技术，其中之一是扩展 IP 地址空间。当一个组织有一定数量的计算机需要与 INTERNET 连接，而又无法申请到足够数量的 IP 地址时，就可以使用一台主机直接与 INTERNET 连接，即所谓的负载平衡器，而其余的主机则与负载平衡器连接而行成一个内部网络，在这个内部网络中每台主机有一个内部的 IP 地址，如 192.168.0.1，192.168.0.2……而负载平衡器则绑定两个 IP 地址，一个是内部的 IP 如 192.168.0.1，一个是 INTERNET 上的 IP 如 202.115.16.8，所有的内部 IP 只在内部网络有效，在 INTERNET 上则不可见，内部网络中的任一台主机既可使用内部 IP 访问堡垒内的各台主机，也可使用 INTERNET 上的主机的 IP 访问 INTERNET，在后一种情况，内部主机事实上是由负载平衡器上的不同端口号加以区分的，而堡垒外的主机则只能对负载平衡器进行访问，至少形式上是只对负载平衡器进行访问。这种情形和我们的问题的相近之处在于对于堡垒外的主机而言整个内部网络象是一台主机，这启发我们利用 IP 伪装机制。

下面我们来分析一下 LINUX 源代码中的 INCLUDE/NET/IP_MASQ.H

```
.
.
/*
通常情况下面，LINUX 系统在端口分配的时候还不会用到大于 32k
（32000 以上的端口）。取一定的剩余空间，因此，将 61000 以上的端
口作为伪装后的端口。
*/
#define PORT_MASQ_BEGIN 61000
#define PORT_MASQ_END   (PORT_MASQ_BEGIN+4096)
#define MASQUERADE_EXPIRE_TCP      15*60*HZ
#define MASQUERADE_EXPIRE_TCP_FIN  2*60*HZ
#define MASQUERADE_EXPIRE_UDP      5*60*HZ
#define MASQUERADE_EXPIRE_ICMP     125*HZ
/*
* 在进行端口转发的时候。需要确定此次转发的后续包的最大时间间
  隔。
* 在超过这个时间间隔后，这个转发将失效。
* 这里定义的就是 TCP ,TCP_FIN ,UDP 和 ICMP 的超时时间。

*/
```

在 LINUX 的 IP 伪装中一个关键性的数据结构是 struct ip_masq，它被用于描述一个 NAT 变换，该结构定义的形式如下：

```
struct ip_masq {
    struct list_head m_list, s_list, d_list;
                                        /* 基于 HASH 表的 d-linked 链
                                                              表头*/
    atomic_t  refcnt;                       /* 引用计数 */
    struct timer_list timer;                /* 失效计数器 */
    __u16     protocol;                     /* 当前使用的协议 */
    __u16     sport, dport, mport;          /* 源、目的及伪装端口 */
    __u32     saddr, daddr, maddr;          /* 源、目的及伪装地址 */
    struct ip_masq_seq out_seq, in_seq;
    struct ip_masq_app *app;                /* 绑定 ip_masq_app 对
象 */
    void      *app_data;                    /* 应用程序私有数据 */
    struct ip_masq *control;                /* 主控制连接数 */
    atomic_t  n_control;                    /* 已控制的伪装数 */
    unsigned  flags;                        /* 状态标志 */
    unsigned  timeout;                      /* 超时 */
    unsigned  state;                        /* 状态信息 */
    struct ip_masq_timeout_table *timeout_table;
}
```

其中的 sport，dport，mport，saddr，daddr，maddr 描述了这样一种变换关系：当一台内部 IP 为 saddr 的主机，用它的 sport 端口送出一个到 INTERNET 上的一台 IP 为 daddr 的主机的 dport 端口的 IP 请求时，负载平衡器挡获这个请求(IP 包)，为了访问在 INTERNET 上的目的主机，用自己在 INTERNET 上的 IP 即 maddr 取代 IP 包头部的源地址 saddr，在自己的空闲端口中分配一个端口即 mport，取代 IP 包头部的源端口 sport，并将修改过的 IP 包发往 INTERNET；对应的，当负载平衡器收到一个从 INTERNET 上地址为 daddr 的主机从 dport 端口返回针对负载平衡器(maddr)的 mport 端口的应答(IP 包)时，负载平衡器再用内部地址 saddr 与内部主机端口 sport 取代 IP 包头部中的目的地址 mpaddr 与目的端口 mport，并将修改过的 IP 包发向内部网络。

为了保存这种变换关系，LINUX 内核维持三张双向 Hash 表：ip_masq_m_table，ip_masq_s_table，ip_masq_d_table 分别以 maddr:mport, saddr:sport, daddr：dport 为索引关键字。struct ip_masq 则是其中的节点。当内部主机提出对 INTERNET 上的主机的访问请求时，负载平衡器将转换信息记录到这三个 Hash 表中，若在指定的时间内收到对该请求的响应则按记录的信息进行相应的转换，否则认为请求失败并删除对应表项。

5.1.3 虚拟服务的 Hash 映射表

为了使虚拟服务器与 IP 伪装协同工作，需要为 struct ip_masq 添加一个指向 struct ip_vs_dest 的指针。struct ip_vs_dest 定义如下：

```
struct ip_vs_dest {
    struct list_head n_list;          /* 链表头 */
    __u32          addr;              /* 真实服务器 IP 地址 */
    __u16          port;              /* 服务的端口号 */
    unsigned       flags;             /* 目的状态标志 */
    unsigned       masq_flags;        /* 伪装标志 */
    atomic_t       activeconns;       /* 活动连接数 */
    atomic_t       inactconns;        /* 未活动连接数 */
    atomic_t       refcnt;            /* 引用计数 */
    int            weight;            /* 服务器权重 */
                                      /* 下面是针对虚拟服务 */
    __u16          protocol;          /* 协议类型(TCP/UDP) */
    __u32          vaddr;             /* 虚拟服务的 IP 地址 */
    __u16          vport;             /* 服务的端口号 */
};
```

如同 IP 伪装，虚拟服务器也使用 Hash 表进行高效的映射。与 IP 伪装不同，由于虚拟服务器的专用性，只需要按外部主机的 IP 与端口进行检索，因此只须建立一个 Hash 表：

```
struct list_head ip_vs_table[IP_VS_TAB_SIZE];
```

其节点为改造后的 struct ip_masq 结构。

当用户访问由集群服务器提供的服务时，向虚拟 IP 地址发出的请求包（负载平衡器的外部地址）到达负载平衡器。负载平衡器检查包的目的地址和端口号。如果对照虚拟服务器的服务规则表，与某项虚拟服务器的服务相对应，就会从集群中按照一定的调度算法选择出一台真实的服务器，连接会被记录到 Hash 表中。随后这个包的目的地址和端口会被改写到选出的真实服务器上，包被重定向到真实服务器。属于这个连接的后续包可以根据 Hash 表中的记录被重写和重定向到服务器上。当响应的包返回的时候，负载平衡器将根据虚拟服务再次重写包的源地址和端口。如果连接终止或超时，连接记录会从 Hash 表中删除。

下面，我们来分析 LINUX2.2.16 版源代码 LINUX/net/ipv4/ip_masq.c 中的一个函数

```
struct ip_masq * ip_masq_new(int proto, __u32 maddr, __u16 mport,
```

```
__u32  addr, __u16 sport, __u32 daddr, __u16 dport, unsigned
mflags)
{
  struct ip_masq *ms, *mst;      /* 两个指向 ip_masq 结构的指针*/
  int ports_tried;
  atomic_t *free_ports_p = NULL;
  static int n_fails = 0;
  int prio;
   if (masq_proto_num(proto)!=-1 && mport == 0) /*检查协议和伪
                                          装端口号的有效性*/
{ free_ports_p = ip_masq_free_ports + masq_proto_num(proto);
   if (atomic_read(free_ports_p) == 0)
     {
       if (++n_fails < 5)
          IP_MASQ_ERR(  "ip_masq_new(proto=%s):  no  free
              ports.\n", Masq_proto_name(proto));
       return NULL;
     }
  }
prio = (mflags&IP_MASQ_F_USER) ? GFP_KERNEL : GFP_ATOMIC;
ms = (struct ip_masq *) kmalloc(sizeof(struct ip_masq),
     prio);
                                          /*申请内存空间*/
 if (ms == NULL) /* 处理未能成功申请内存的请况*/
     {if (++n_fails < 5)
      IP_MASQ_ERR("ip_masq_new(proto=%s):     no     memory
          available.\n", masq_proto_name(proto));
       return NULL;
     }
MOD_INC_USE_COUNT;
sysctl_ip_always_defrag++;
memset(ms, 0, sizeof(*ms));
INIT_LIST_HEAD(&ms->s_list);
INIT_LIST_HEAD(&ms->m_list);
INIT_LIST_HEAD(&ms->d_list);
init_timer(&ms->timer);
ms->timer.data     = (unsigned long)ms;
 ms->timer.function = masq_expire;
```

```
ms->protocol    = proto;
ms->saddr           = saddr;
ms->sport    = sport;
ms->daddr    = daddr;
ms->dport    = dport;
ms->flags    = mflags;
ms->app_data     = NULL;
ms->control      = NULL;
atomic_set(&ms->n_control,0);
atomic_set(&ms->refcnt,0); /*设置 ip-masq 结构的各个字段*/
```

这里需要完成的工作是为 ip_masq.c 添加适当的代码，以便在进行 IP 转换时先在 ip_vs_table 中查找映射关系，若找到，则完成虚拟服务器所要求的转换，否则，做一般的 ip_masq 转换。

5.1.4 调度分析

为了使虚拟服务器有效地工作，需要对集群中的主机进行合理的调度。有四种调度算法可供选择：

a) 循环调度

循环调度算法以循环方式将网络连接定向到不同的服务器。它不考虑实际的连接数或响应时间，将所有真实服务器看作是完全相等的。尽管循环 DNS 也是以这种方式工作，但它们之间仍有很大区别。循环 DNS 将单一域名解析为不同的 IP 地址，调度的单位是基于主机的，DNS 的缓存机制也妨碍了算法的实现，这些将会在真实服务器中产生明显的负载不均衡。而虚拟服务器的调度单位是基于连接的，因为有更好的调度单位，它比循环 DNS 的优越得多。

b) 加权循环调度

加权循环调度将真实服务器看作具有不同处理能力的机器。每台服务器可以被单独指定一个权值，这是一个整数，指出它的处理能力，缺省的权值是 1。例如真实服务器 A，B，C 的分别具有权值 4，3，2，则一次良好的调度过程将是在一次调度周期（对权值之和取模）中出现 ABCABCABA 序列。在加权循环调度的具体实现中，如果虚拟服务器的设置发生改变，调度序列将根据新的权重来生成。网络连接将以循环方式按照调度序列重定向到不同的真实服务器。

加权循环调度无需计算每台真实服务器的网络连接数，调度所需的消耗也比动态调度算法小，因此可以支持更多的真实服务器。

不过，如果请求的变化幅度很大，就有可能导致动态的负载不平衡。简单的说，有可能出现大部分的长时间请求都被定向到同一台真实服务器的情况。循环调度是加权循环调度的一个特例，在这种情况下所有的权值是相等的。修改虚拟服务器的设置后重新生成调度序列所需的消耗是微不足道的，也不会增加实际调度的消耗。所以，没有必要单独实现循环调度算法。

c) 最少连接调度

最少连接调度算法将网络连接重新定向到建立连接数最少的真实服务器。这是一种动态调算法，因为它需要动态计算每台服务器的活动连接数。对由相似的真实服务器组成的虚拟服务器而言，最少连接调度有利于平滑的分发变化较大的请求负载，因为所有的长期请求不会都被定向到同一台服务器。表面上看，最少连接调度即使在真实服务器的处理能力之间存在较大差异的情况下仍可以很好发挥作用，因为更快的服务器会得到更多的网络连接。但实际上 TCP 的 TIME_WAIT 状态使得它无法很好的工作。TCP 的 TIME_WAIT 状态通常是 2 分钟，对繁忙的服务器来说，2 分钟内通常有成千上万的连接，例如，服务器 A 工作能力两倍于服务器 B，A 正在处理数千个请求，并将它们保持在 TCP 的 TIME_WAIT 状态，而服务器 B 却在吃力等待数千个连接完全建立。因此最少连接调度对工作能力差异较大的主机群而言无法较好的平衡负载。

d) 加权最少连接调度

加权最少连接调度是最少连接调度的一个超集，通过它你可以为每台真实服务器指定一个性能权值。具有较高权值的服务器会得到更大百分比的活动连接数。虚拟服务器管理员可以为每台真实服务器指定一个权值，网络连接被调度到各台服务器，在这种情况下，各台服务器的活动连接数在总连接数中所占的百分比是与它的权值成比例的。缺省的权值是 1。

加权最少连接调度是这样工作的：

假定有 n 台真实服务器，每台服务器 i 具有权值 Wi(i=1 到 n)，活动连接数 Ci(i=1 到 n)，A 是 Ci 的总和，下一个网络连接会被重定向到服务器 j，则有

$$(Cj/A)/Wj = \min \{ (Ci/A)/Wi \} \quad (i=1 \text{ 到 } n)$$

因为 A 在最小值查找中是一个常数，没有必要用 A 去除 Ci，所以表达式可以这样优化

$$Cj/Wj = \min \{ Ci/Wi \} \quad (i=1 \text{ 到 } n)$$

加权最少连接调度算法与最少连接调度算法相比需要作额外的

比例分配。当集群服务器具有相同的处理能力时，为了将调度所需的消耗减小到最低限度，最少连接调度和加权最少连接调度两种算法都实现在程序中了。

为了统计各主机的连接数，在 ip_vs_dest 中设置了 activeconns 和 inactconns 字段，activeconns 记录当前活动的连接数目，inactconns 记录当前非活动的连接数目.

```
/*
 *  下面是一个原始的调度实现。Crude m_host scheduler
 *  这个实现需要扩充，以便能够提供其他的调度选择
 *  这个函数的调用者必须将  m-entry 锁定（lock）.
 */
static struct ip_masq_mfw_host * __mfw_sched(struct ip_masq_mfw *mfw, int force)
{
    struct ip_masq_mfw_host *h = NULL;

    if (atomic_read(&mfw->nhosts) == 0)    /*可供调度的主机数为 0 ， 返回 Null
值*/
        goto out;

    /*
     *  这里就是真正的调度策略所在。
     *  当 pref_cnt 为 0，表项移动到链表的尾。
     *  并且它的 pref_cnt 值将设为从用户态空间传来的 h->pref
     *  当它的 pref 等于 0 时，它表明本主机不参与调度。
     */

    h = list_entry(mfw->hosts.next, struct ip_masq_mfw_host, list);
     /*从链表中取出下一个主机节点*/
    if (atomic_read(&mfw->nhosts) <= 1)
        goto out;

    if ((h->pref && atomic_dec_and_test(&h->pref_cnt)) || force) {
        atomic_set(&h->pref_cnt, h->pref); /*设置主机节点的 h_pref  值为
h_pref*/
        list_del(&h->list);/*删除这个主机节点*/
        list_add(&h->list, mfw->hosts.prev); /*插入该节点的后续节点 */
    }
```

```
out:
    return h;
}
```

5.2 构建一个集群服务器

为了掌握 LINUX 虚拟服务器的真实性能情况。需要在实验室环境下搭建一个真实的集群服务器，以便检测它的性能。

5.2.1 负载平衡节点的内核配置

对整个集群系统来说，其负载平衡节点是一个关键环节，特别是对这种采用 NAT 方式进行转发请求/响应包的系统。

在构建这个集群系统的过程中，考虑到 LINUX 内核不断推陈出新的发展速度，现在已经推出了 LINUX-2.4.0-test12 版内核。新版内核的功能不断的添加和完善，但是它可能存在的 bug 多一些。因此，选定了一个相对较新的内核版本 2.2.16 作为集群的负载平衡器的内核。

由于通常的内核不支持集群系统，因此在编译内核之前要打上补丁。

在 HTTP://WWW.LINUX-VS.ORG 站点下载针对 2.2.16 版内核的补丁，ipvs-0.9.15-2.2.16.tar.gz。

```
cd  /usr/src/LINUX
cat     /usr/src/ipvs-0.9.15-2.2.16/ipvs-0.9.15-2.2.16.patch|patch
-pl
```

在 LINUX 核心源代码中加入虚拟服务器支持代码，并重新配置核心：

在“Code maturity level options”菜单中选中如下项目：

[*] Prompt for development and/or incomplete code/drivers

即在配置核心时显示开发中或未完全完成的代码/驱动程序的配置选项；

在“Networking options”菜单中选中如下项目：

[*] TCP/IP networking

[*] Network firewalls

此项防火墙支持

[*] IP: firewalling

此项 IP 防火墙支持

[*] IP: masquerading(new)

此项支持 IP 伪装，选中。支持 IP masque 的特定协议将作为一个模块编译

[*] IP: ICMP masquerading(new)

支持对 ICMP 的伪装，同样一些特定的协议将作为一个模块编译

[*] IP: masquerading special modules support (new)

此项为选中一些特殊的模块

[*] IP: masquerading virtual server support (EXPERIMENTAL

基于 IP 伪装的虚拟服务器支持，括号中的“EXPERIMENTAL”表示此功能仍在开发阶段

(12) IP masquerading VS table size(the Nth power of 2) (new)

上面的数据表明了 IP 伪装地址转换表的大小为 4096 个表项

<> IPVS: round-robin scheduling(NEW)*

调度算法之一：循环调度

<> IPVS: weighted round-robin scheduling(NEW)*

调度算法之二：加权循环调度

<> IPVS: least-connection scheduling(NEW)*

调度算法之三：最少连接调度

<> IPVS: weighted least-connection scheduling(NEW)*

调度算法之四：加权最少连接调度

5.2.2 激活集群路由（IPCHAINS）

集群系统需要使用 IPCHAINS 来完成两个功能。系统内的真实服务器组成的一个内部网，对外界的用户来说是不可见的。因此，为了实现一定的安全性，需要使用 IPCHAINS 来完成防火墙的配置。同时，内部机器和外边机器之间的信息交换需要通过负载平衡器，因此需要 IPCHAINS 来启用它的路由功能。

IPCHAINS 是用来构建，维护和检查 LINUX 系统内核的 IP 防火墙规则的工具，这些防火墙规则可以划分为 4 类，IP Input Chain, IP Output Chain, IP Forwarding Chain 和用户自定义链表（Chains）。对每一类 Chains 来说，都维护着一张独立的规则表，这些规则的任意一条都有可能指向一个用户自定义的链表(User-Defined Chains)。

一个防火墙规则指明了一个 IP 包的判断标准和一个目标，如果当前包不匹配这个判断标准，该链表（Chain）的下一条规则将用来做判断。如果恰好匹配这个判断标准，那么这个包所受到的下一条判断标准将由匹配规则的目标来决定。这个目标可能是一个用户定义链表（User-Defined Chain）的名字，也可能是系统预置的五个值之一，它们可能是 ACCEPT, DENY, REJECT, MASQ, REDIRECT

或者 RETURN。

ACCEPT 意味着让这个包通过。DENY 将这个包丢弃，与 DENY 相比，REJECT 丢弃这个包后还要向源端返回一个 ICMP 包指示有丢弃发生。MASQ 只是对 Forward 链表和用户自定义链表才是合法目标，通常，只有在内核编译时定义了 CONFIG_IP_MASQUERADE 标志时，才有来自本地的 IP 包将被伪装，来自外部的包将被解除伪装。REDIRECT 对 Input Chain 和用户自定义链表（User-Defined Chains）才是合法的，同时还需要在编译内核时设置了 CONFIG_IP_TRANSPARENT_PROXY 标志。在这个目标的作用下，IP 包将被重定向到本地套接字，即使它们是要发往远端的机器。如果 REDIRECT 指明的端口是缺省值 0，那么通常将该 IP 包重定向到本地与它相同的目的端口上。如果 IP 包在匹配一个链表时已经到达其尾部或者在匹配成功的规则，其目标是一个 RETURN，此时上一个（调用）链表的下一条规则将用来尝试匹配。

为了让负载平衡器能够完成路由转发功能，需要激活 packet-forwarding 和 defragmenting，确定/etc/sysconf/network 中由这两行：

```
FORWARD_IPV4=yes
DEFRAG_IPV4=yes
```

这些行将使/etc/rc.d/rc.sysinit 在路由启动时执行：

```
echo 1 > /proc/sys/net/ipv4/ip_forward
echo 1 > /proc/sys/net/ipv4/ip_always_defrag
```

激活 IP masquerading，用这个命令：

```
IPCHAINS -A forward -j MASQ -s 192.168.1.0/24 -d 0.0.0.0/0
```

5.2.3 集群系统内真实服务器配置（ipvsadm）

在做好各项准备工作以后，最重要的就是在负载平衡器使用 ipvsadm 来配置虚拟服务器和内部的真实服务器。

ipvsadm 的命令有如下的两类：

```
ipvsadm COMMAND [protocol] service-address [scheduling-method]
        [persistence options]
ipvsadm command [protocol] service-address server-address
        [packet-fowardiing-method] [weight options]
```

前一格式的命令设置虚拟服务和分配服务请求到各真实服务器的调度算法。第二种格式的命令配置一个完成虚拟服务的真实服务器。在这里可以指出这个真实服务器的包转发方法，相对于集群系统内其他服务器的权重。

5.2.4 虚拟服务器组成

LB：负载平衡器
N1：真实服务器结点 1(由 LB 担任)
N2：真实服务器结点 2
N3：真实服务器结点 3

其中，N3 采用了非 LINUX 操作系统以验证 NAT 技术对支持 TCP/IP 协议的不同平台的支持。

表 5.1 试验集群系统的硬件配置

服务器	CPU	内存	硬盘	网卡
LB(N1)	Intel P166MMX	48M	2.1G	NE2000 兼容 10M 以太网卡
N2	Intel P133	48M	800M	NE2000 兼容 10M 以太网卡
N3	Intel PII 300	64M	10G	NE2000 兼容 10M 以太网卡

表 5.2 试验集群系统的软件配置

服务器	操作系统	系统内核	WWW 服务程序
LB	Slackware 4.0(LINUX)	LINUX 2.2.16(*)	Apache 1.3.6
N2	Redhat 6.0(LINUX)	LINUX 2.2.16	Apache 1.3.6
N3	PWindows 98 SE	N/A	Apache 1.3.9(Win32)

(*)：经过修改的 LINUX 核心，加入了内核级的虚拟服务器支持。

表 5.3 试验集群系统的网络设置

服务器	真实 IP	内部 IP	网关	子网掩码
LB	202.115.16.23	192.168.1.1	202.115.16.1	255.255.255.0
N2	N/A	192.168.1.2	192.168.1.1	255.255.255.0
N3	N/A	192.168.1.3	192.168.1.1	255.255.255.0

安装过程

三台电脑均通过双绞线连接到 Intel Express 510T 交换机。按照如上表所

示网络配置修改各机的网络设置。其中 LB 机需要使用同一块网卡绑定 202.115.16.23 与 192.168.1.1 两个 IP，可通过在 LINUX 内核中添加 IP 别名支持（IP_ALIAS）来实现。

修改负载平衡器的 LINUX 系统核心，按照前面的介绍，在 LINUX 核心源代码中加入虚拟服务器支持代码，并重新配置核心之后重新编译生成新的内核。此内核将用于 LB 机，结点机无需修改核心。

真实服务器上 WEB 服务程序安装，三台结点机均安装 Apache Web Server，存放相同的网页。我们在每台结点机上均放置了有目录索引的数千个 GIF 图形文件（总大小约 100M）和一个约 50M 的压缩文件，用于测试集群服务器在处理随机访问和顺序访问两种情况下的性能表现。

在 LB 机上设置虚拟服务，加入调度算法：

```
ipvsadm -A -t202.115.16.23:80 -s wlc
```

即在 202.115.16.23 上提供 WWW 服务，采用最少连接调度算法。

加入真实服务器（结点机）列表，设置转发方式：

```
ipvsadm -a -t 202.115.16.23:80 -r 192.168.1.1
ipvsadm -a -t 202.115.16.23:80 -r 192.168.1.2 -m
ipvsadm -a -t 202.115.16.23:80 -r 192.168.1.3 -m
```

其中 LB 机自身又兼作真实服务器，在加入列表时无需设置转发方式。

虚拟服务器的实际运行及运行中的调度

若以上各步均已正确无误的完成，各真实服务器上的 WEB 服务进程也已经正常运行，此时就可以通过浏览器访问虚拟服务器 202.115.16.23 了。

5.3 实测数据及其分析

5.3.1 实际运行情况

下面是实际运行中某一时刻的虚拟服务器组成及活动连接情况（原始数据由 ipvsadm 工具输出，以下已经过整理）：

```
Prot LocalAddress:Port        Scheduler Flags
TCP  www.cs.uestc.edu.cn:www     wlc
```

这里显示服务类型为 TCP，服务端口为 WWW（80），调度算法为加权最少连接

调度（WLC）：

RemoteAddress:Port	Forward	Weight	ActiveConn	InActConn
192.168.1.1:www	Local	1	64	49

这里显示目前集群中只有一个真实结点，就是负载平衡器自身。因为本机无需再转发 IP 包，所以转发方式为 Local（本地）。当前有 64 个活动连接，49 个非活动连接。

负载平衡器的最近 1 分钟，5 分钟和 15 分钟的负载情况分别为 2.50，1.89，1.32。

下面是用 ipvsadm 工具加入了另一台结点机(192.168.1.3)后的立刻记录下的连接情况：

RemoteAddress:Port	Forward	Weight	ActiveConn	InActConn
192.168.1.3:www	Masq	2	22	125
192.168.1.1:www	Local	1	35	8

可以看到 192.168.1.3 的转发方式为 Masq 即通过 NAT 转换过再转发，该结点权重为 2，目前已有 22 个活动连接和 125 个非活动连接。

下面是新结点加入并运行一段时间后记录下的连接情况：

RemoteAddress:Port	Forward	Weight	ActiveConn	InActConn
192.168.1.3:www	Masq	2	54	66
192.168.1.1:www	Local	1	30	11

可以看到，在 WLC 算法调度下，结点机 3 的活动连接数基本上是结点机 1 的两倍，这是由它的权重决定的。

负载平衡器的最近 1 分钟，5 分钟和 15 分钟的负载情况分别为 1.41，1.24，1.23。

下面是加入了第三个结点并运行一段时间后记录下的连接情况：

RemoteAddress:Port	Forward	Weight	ActiveConn	InActConn
192.168.1.2:www	Masq	2	42	61
192.168.1.3:www	Masq	2	42	57
192.168.1.1:www	Local	1	29	9

负载平衡器最近 1 分钟，5 分钟和 15 分钟的负载情况分别为 0.11，0.16，0.47。

在整个测试过程中，包括在加入和删除结点时，WWW 服务均未中断。

5.3.2 对方案的评价及展望

NAT 技术的优点是，真实服务器可以运行任何支持 TCP/IP 协议的操作系统，并且可以使用内部的 INTERNET 地址，只需要为负载平衡器分配一个合法 IP 地址。它的缺点是，通过 NAT 实现的虚拟服务器的可伸缩性不太好，当服务器结点数目达到 25 或更多时，负载平衡器就可能成为整个系统的瓶颈，因为请求包与响应包都需要由负载平衡器来重写。假定包的平均长度是 536 字节，重写一个包的平均延迟约为 60 微秒[20]（对 586 级芯片而言，如果采用更高级的处理器，这个数值可以稍稍减小一点），负载平衡器的最大吞吐量就是 8.93MB/秒。假定真实服务器的平均吞吐量是 400KB/秒，负载平衡器可以调度 22 台真实服务器，这在大多数情况下已能满足需要。

第六章 作者的新方案以及研究展望

综合消化第三章、第四章和第五章的知识，作者提出了两种新的服务器集群解决方案。在第三章提出的 LVS，它具有多种的路由转发方法，网络地址转换、IP 封装和直接路由。第四章提出的 DPR 技术，不存在 LVS 的中心节点，避免了整个系统可能出现的中心瓶颈。这就给作者思路，是否可以综合两种方案的优点，提出新的解决方案。

6.1 通过网络地址转换实现的 DPR 技术

DPR 技术的核心是集群中每一台服务器都参与了路由的转发。在这个路由的转发过程中，并没有指明一定要采用什么路由转发技术。经过仔细分析，发现网络地址转换也可以完成这个功能。

6.1.1 基本原理

网络地址转换（NAT）通常用于内部网地址和外部网地址之间的转换，起到隐藏内部主机的功能。但是在理论上它并不排斥外部网地址和外部网地址之间的转换。

对设想中的这种采用 NAT 作为路由方法的 DPR 集群系统来说，集群中的每一台服务器都分配了一个外部网 IP，这些服务器的 IP 地址通过 RR-DNS 方式发布出去，系统中所有的服务器都可以收到客户的请求。像在第四章提到的采用 IP 封装技术的 DPR 一样，这些服务器同样要响应客户的访问请求，具备判断、转发这些请求的能力。这些请求要么通过 NAT 转发到别的机器上，要么由收到包的主机自己来处理。在前一种情况下，服务请求可以转发到集群中的任意服务器上，处理的结果和采用 IP 封装技术的方案不一样，前者将通过第一台服务器作网络地址转换后才返回客户端，后者则是直接返回客户端。

集群中的任意一台服务器收到客户发来的服务请求后，它根据当前各服务器的负载情况，选择一台机器来处理这个请求。如果由自身来处理，它将连接这个 IP 包放入上层协议就可以了。如果这个请求需要其他的机器来处理，它通过一个调度算法从其他服务器当中选择一台来处理这个请求。它还将描述这个连接的表项加入一个 Hash 表中，同时改写这个包的目的地址和端口为所选择的服务器地址及端口并将其转发到这个服务器，属于这个连接的后续包来到时，可以在那张 Hash 表中找到这个服务器，这些包同样被这台服务器改写目的地址及端口号后

被转发到被选中的服务器。源自真实服务器的响应包返回第一台服务器时，它们将被重写源地址和端口号。连接释放或者超时过后，在 Hash 表中的记录将被删除。这样，用户所看到的只是在第一台服务器上提供的服务，后面的转发过程和第二台服务器的处理对用户是透明的。

6.1.2 实现方案

前一小节提出的采用网络地址转换实现的分布式集群方案，可以充分利用现有 LVS 项目的一些成果。在 LVS 的 NAT 方案中考虑的是一个外部 IP 地址和很多的内部 IP 地址之间的转换。但是，NAT 技术从理论上是不排斥外部网地址和外部网地址之间的转换。

假设一个很小的服务器集群。它有 4 台服务器，分别分配的 IP 地址是 202.115.16.1、202.115.16.2、202.115.16.3、202.115.16.4。现在我们看看怎么利用现有的 LVS 技术实现 NAT 方案的分布式集群。

首先，为每一台机器绑上一个内部网 IP，如 192.168.1.1。这样每台机器就拥有了两个 IP 地址。这需要在 LINUX 内核编译时启用 IPAlias 支持。

然后，在 202.115.16.1 上设置虚拟服务（假设为 WEB 服务），并加入调度算法：

```
ipvsadm -A  - t 202.115.16.1:80 -s wlc
```

即在 202.115.16.1 上提供 WWW 服务，采用最少连接调度算法。

然后，将加入真实服务器（结点机）列表，设置转发方式：

```
ipvsadm -a -t 202.115.16.1:80 -r 192.168.1.1
ipvsadm -a -t 202.115.16.1:80 -r 202.115.16.2 -m
ipvsadm -a -t 202.115.16.1:80 -r 202.115.16.3 -m
ipvsadm -a -t 202.115.16.1:80 -r 202.115.16.4  - m
```

此时集群中每一台机器都作为了 202.115.16.1 的真实服务器。

第三，对集群中的其他三台服务器进行与 202.115.16.1 相同的操作。在这里我们可以发现，每台机器都担任了 VS/NAT 中的负载平衡器的功能，同时还担任了真实服务器的功能。

集群系统中的四台机器可以通过RR-DNS以域名WWW.CS.UESTC.EDU.CN发布出去，客户以这个域名访问时，可能解析到的 IP 是这四个 IP 中的任意一个。由于这几台机器都充当了负载平衡器，也充当了真实服务器，具体由那一台服务器来处理客户请求，就与当时各机的负载情况有关了。

6.1.3 面临的问题

在这个实现方案中，可能存在一个问题。那就是客户的请求可能在某两台服务器之间循环，比如202.115.16.1将请求转发202.115.16.2处理。然而，202.115.16.2

根据自身的负载情况又将请求转发到 202.115.16.1。解决这个问题有待于制定更详细的转发规则。需要通过修改 LINUX 内核源代码来实现。

限于时间和精力的关系。这个实现方案没有在实验室进行测试。

6.2 通过直接路由实现的 DPR 技术

同前面提到的一样，我们可以采用直接路由技术来实现分布式集群中的路由转发功能。

这种实现方案的连接调度和管理与前面讨论的采用 IP 封装技术的方案没有区别。只是它的报文转发方式不同，它将报文直接路由给目标服务器。在这个方案中，集群中的服务器根据当前各服务器的负载情况，动态地选择一台服务器，不修改也不封装 IP 报文，而是将数据帧的 MAC 地址改为选出服务器的 MAC 地址，再将修改后的数据帧在与服务器组的局域网上发送；因为数据帧的 MAC 地址是选出的服务器，所以服务器肯定可以收到该报文。

为了实现这个方案，需要在每台服务器上启用 LINUX 的 IP 别名支持。绑上其他服务器的 IP 地址作为 IPAlias。并将这些 IP 别名设为 Non-ARP 设备。

服务器 B 收到服务器 A 直接路由来的客户请求后，由于服务器 A 的 IP 地址作为别名被置在 B 的本地的网络设备上，服务器 B 就处理这个请求，然后将响应报文直接返回给客户。

本方案和通过网络地址转换实现的 DPR 技术相比，除了在转发包的方式上有所不同以外，它的实现方法和可能面临问题都可以参考 6.1 节

6.3 进一步研究展望

在前几章，本论文对服务器集群的请求转发和调度进行了一定程度的讨论和研究。除此之外，服务器集群还有许多问题需要进一步地研究。本节将简单讨论一些可能的领域。

6.3.1 集群服务器后端存储问题

从第三章 LVS 的架构就可以看出，它包括三个部分：负载平衡器，服务器池和后端存储。后端存储为服务器池提供一个共享的存储区，这样很容易使得服务器池拥有相同的内容，提供相同的服务。

6.3.1.1 存储的一致性

后端存储需要解决一致性问题。在 LINUX 虚拟服务器项目中的三种实现方案

中，都没有对集群服务器的内容一致性问题提出详细的解决方案，在实际的应用过程中，都是以手工方式将同一文件复制多份，分别存放在不同的服务器上面。这种方式效率很低。在理论上有方案提出了采用容错的分布式文件系统，如 AFS、GFS、Coda 和 Intermezzo 等来解决这个问题。各服务器访问分布式文件系统就像访问本地文件系统一样。在下一步的研究中可以探讨这些分布式文件系统在 LVS 中的具体实现方案和实际效率。

6.3.1.2 存储的容量和速度问题

随着 INTERNET 的飞速发展，集群服务器需要更大的和更快的后端存储。下一步的研究可以考虑新的存储技术在集群中的应用。

最近存储局域网（SAN）和光纤通道（FibreChannel）引起了人们极大的兴趣。它们由于能够突破许多分布式网络中的存储瓶颈，因而被认为是新的用于高端存储方案的工业标准。

光纤通道是设计来专门解决服务器一存储设备接口限制的。一条光纤通道总线（没有 SAN）中的一个基本的服务器一存储设备 I/O 连接可以大大改进网络和存储访问的性能。光纤通道的高带宽改进非常适全于任何高端应用如成像、视频、OLTP、数据库、CAD/CAM 等对系统存储性能影响大的信息传输。而且使用光纤通道扩展的连接距离也便于进行远程备份、存档和用于灾难恢复目的的镜像。

光纤通道的物理速度是非常快的。当前光纤通道的速率可以达到 100MBPS，将来的可以达到 200MBPS，400MBPS（速度快）。根据使用的媒质，光纤通道可以连接远达 10 公里外的节点。

6.3.2 应用层要求与第四层交换的冲突问题

在具体的应用中，比如在 WEB 应用中，一个客户对服务器的访问，通常需要记录会话过程的状态，比如记录在一个在线商店的购物状态。但是 HTTP 协议是一个无状态协议。这个问题的解决一般是在服务器端保持存了许多 SESSION 变量和 APPLICATION 变量，以消除 HTTP 协议的无状态对应用要求的影响。

但是，前面提到的服务器集群解决方案中。调度的粒度是一个 TCP/UDP 连接，同一个 SESSION 的连接，可能会被负载平衡器转发到不同的服务器上面。由于 WEB 服务器之间没有 SESSION 和 APPLICATION 变量的同步，客户的请求将无法正确地处理。

在下一步的研究中，可以考虑解决这种问题的方法。简单地说，可以通过会话状态保存在客户端的办法来解决这个问题。

6.3.4 对象概念的引入

近 10 年来，面向对象技术的研究不断深入，面向对象的程序设计方法逐渐

被人们所接受。它可以较自然地模拟客观世界。从而使问题空间和解空间尽可能的一致。使得软件产品具有易理解性，易修改性和可重用性。

在服务器集群中的调度粒度是客户连接请求。可以在下一步的研究中考虑将对象技术引入，将连接对象作为调度的基本单位。同时，集群中的真实服务器也具备对象的特征，也可以考虑用对象的方法来封装这些服务器。

6.3.5 服务器集群的安全性问题

在 LVS 的几个方案中，负载平衡器转发客户请求给内部服务器。由于它的位置是一个绝佳的防火墙。我们可以在下一步的研究中充分挖掘这个位置的潜力，增强集群系统的安全性。

在 LINUX 操作系统下面，有很多的包处理工具，如早期版本的 IPCHAINS 和后来版本的 IPTABLES。在设计服务器集群的时候，可以利用这些现有的开发成果。

下一步的工作还可以测试整个系统的抗 DOS 攻击能力。

致　　谢

整个的研究生学习阶段，到现在基本上就宣告结束了。在此期间，作者得到了很多老师和同学热心，无私的帮助，在此我表示衷心的感谢。

近三年来，我的导师刘乃琦教授以其深厚的理论造诣、丰富的实践经验和对学科发展前沿敏锐的洞察力，为我的研究工作提供了强有力的支持，师恩难报，希望能借此机会，向刘老师致以最由衷的敬意和感激之情。

感谢实验室众多的同学多年来给予我热情指导、关心和帮助。陈文胜、刘奇志、雷昊峰、郭雪松、韩新等在学习和生活上的支持与关心，让我感激不尽。正是他们的陪伴和鼓励让我克服了一个又一个的难关。

感谢我的父母和妹妹。是他们在我漫长的求学生涯中给予了支撑生活和生命的爱。☺

参 考 文 献

1. “WEB 交换机为数据中心加速”。*On line at* http://www.computerworld.com.cn/99/week/9923/9923c01.asp。计算机世界报，1999
2. 黎康保，计算机世界报 http://www.computerworld.com.cn/2000/week/0008/0008c01.asp 2000.3
3. 朱世进，《基于 PVM/Cluster 任务迁移和任务调度的研究》电子科技大学硕士毕业论文。2000.1
4. Eric Dean Katz, Michelle Butler, and Robert McGrath. A scalable HTTP server: The NCSA prototype. Computer Networks and ISDN Systems, 27:155--164, 1994.
5. D. M. Dias, W. Kish, R. Mukherjee, and R. Tewari. A scalable and highly available web server. In Proceedings of IEEE COMPCON Conference on Technologies for the Information Superhighway, pages 85--92, February 1996.
6. Daniel Andresen, Tao Yang, and Oscar H. Ibarra. Toward a scalable distributed WWW server on workstation clusters. Journal of Parallel and Distributed Computing, 42:91--100, 1997.
7. D. Andresen, T. Yang, O. Egecioglu, O. Ibarra, and T. Smith. Scalability issues for high performance digital libraries on the world wide web. In Proceedings of ADL'96 Forum on Research and Technology Advances in Digital Libraries, pages 91--100, Washington D.C., May 1996.
8. Cisco System. Scaling the Internet web servers, November 1997. White Paper.
9. Azer Bestavros, Mark Crovella, Jun Liu, and David Martin. Distributed packet rewriting and its application to scalable server architectures. Technical Report TR-98-003, Boston University, Boston, MA, February 1998.
10. Scott M. Baker and Bongki Moon.，《Distributed Cooperative Web Servers》. http://www.cs.arizona.edu/dcws
11. 张文嵩 linux 虚拟服务器项目 linux virtual sever project HTTP://WWW.LINUX-VS.ORG
12. 国家智能计算机研发中心，曙光信息产业有限公司《服务器操作系统—现状和选型》 1999.7
13. K. Egevang, P. Francish 《The Network Address Translator (NAT)》

RFC1631。

14. W. Simpson ，《 IP in IP Tunneling 》RFC 1853 October 1995
15. Eric Anderson, David Patterson, and Eric Brewer. "The MagicRouter: An application of fast packetinterposing." Available from http://HTTP.CS.Berkeley.EDU/~eanders/projects/magicrouter/osdi96-mr-submission.ps, May 1996.
16. Daniel M. Dias, William Kish, Rajat Mukherjee, and Renu Tewari, "A Scalable and Highly Available Web Server", Proceedings of IEEE COMPCON'96.
17. A. Bestavros, M. Crovella, J. Liu, and D. Martin "Distributed Packet Rewriting and its Application to Scalable Web Server Architectures," in Proceedings of ICNP'98: The 6th IEEE International Conference on Network Protocols, (Austin, TX), October 1998.
18. C. Perkins. "IETF RFC2003: IP Encapsulation within IP". Available from Http://ds.internic.net/rfc/rfc2003.txt
19. S. Deering. "IETF RFC1112: Host Extensions for IP Multicasting" . Available from Http://ds.internic.net/rfc/rfc1112.txt
20. Linux Virtual Server Project. How virtual server works? http://www.linux-vs.org/how.html