- 简介
- 整体架构图
- 实验环境简介
- 高可用负载均衡 lvs+keepalived 四层转发
  - lvs
  - keepalived
- nginx 负载均衡七层转发
  - 域名转发
  - URL/路径转发
  - 负载均衡
  - 常见 rewrite 规则应用
- web 层
  - jetty
  - nginx
  - apache
- 数据库层
  - mysql 基础
  - mysql 主从复制
  - mysql 双主高可用 lvs+keepalived
- 性能优化
  - 系统层面优化（系统服务、文件描述符、sysctl 等）
  - jetty 优化（配置文件、jvm）
  - nginx 优化
  - mysql 数据库优化
  - 数据库缓存（redis、memcached、内存数据库）
  - 静态资源缓存
- 自动化运维
  - Jenkins
  - ansible
  - saltstack
- 监控平台
  - Nagios
  - cacti
  - Ganglia

目录

简介

当前大多数的互联网系统都使用了服务器集群技术，集群即将相同服务部署在多台服务器上构成一个集群整体对外提供服务，这些集群可以是 Web 应用服务器集群，也可以是数据库服务器集群，还可以是分布式缓存服务器集群等等。

在实际应用中，在 Web 服务器集群之前总会有一台负载均衡服务器，负载均衡设备的任务就是作为 Web 服务器流量的入口，挑选最合适的一台 Web 服务器，将客户端的请求转发给它处理，实现客户端到真实服务端的透明转发。最近几年很火的「云计算」以及分布式架构，本质上也是将后端服务器作为计算资源、存储资源，由某台管理服务器封装成一个服务对外提供，客户端不需要关心真正提供服务的是哪台机器，在它看来，就好像

它面对的是一台拥有近乎无限能力的服务器，而本质上，真正提供服务的，是后端的集群。

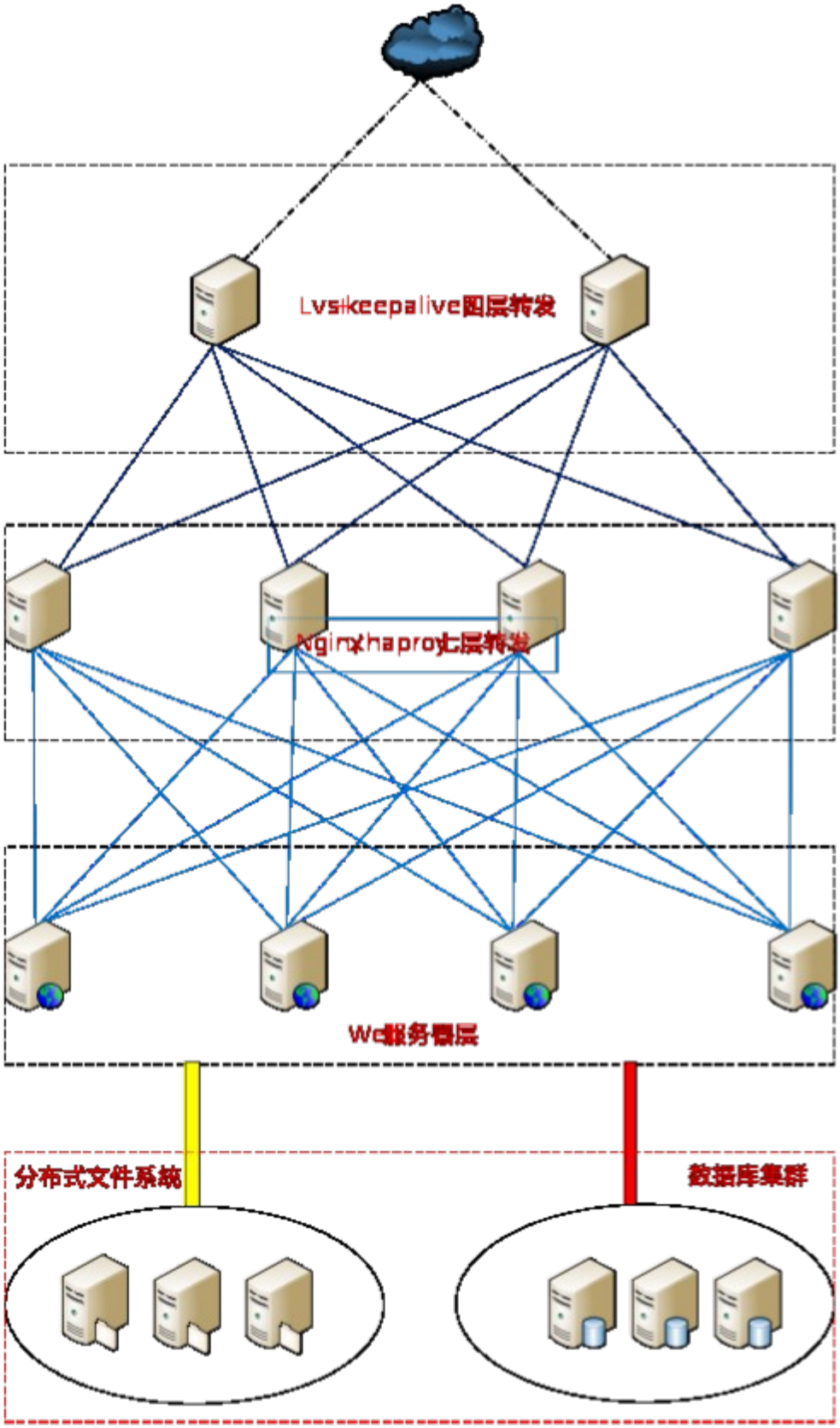
实现高可伸缩、高可用网络服务的需求可以归结为以下几点：

可伸缩性（Scalability），当服务的负载增长时，系统能被扩展来满足需求，且不降低服务质量。

高可用性（Availability），尽管部分硬件和软件会发生故障，整个系统的服务必须是每天 24 小时每星期 7 天可用的。

可管理性（Manageability），整个系统可能在物理上很大，但应该容易管理。价格有效性（Cost-effectiveness），整个系统实现是经济的、易支付的。

整体架构图


Lvs+keepalive四层转发
Nginx/haproxy七层转发
Web服务器层
分布式文件系统
数据库集群

实验环境简介

lvs 集群：

lvs1:192.168.1.225	lvs1
lvs1:192.168.1.226	lvs2
vip1:192.168.1.227	web vip
vip2:192.168.1.228	web vip
vip3:192.168.1.246	mysql vip
vip4:192.168.1.247	mysql vip

web 集群：

web1:192.168.1.232	nginx image jetty
web2:192.168.1.233	nginx image jetty

mysql 集群：

mysql1:192.168.1.248

mysql2:192.168.1.249

高可用负载均衡 lvs+keepalived 四层转发

LVS（Linux Virtual Server）

来源 LVS 官网：

简介：

LVS 是 Linux Virtual Server 的简称，也就是 Linux 虚拟服务器, 是一个由章文嵩博士发起的自由软件项目，它的官方站点是 www.linuxvirtualserver.org。现在 LVS 已经是 Linux 标准内核的一部分，在 Linux2.4 内核以前，使用 LVS 时必须要重新编译内核以支持 LVS 功能模块，

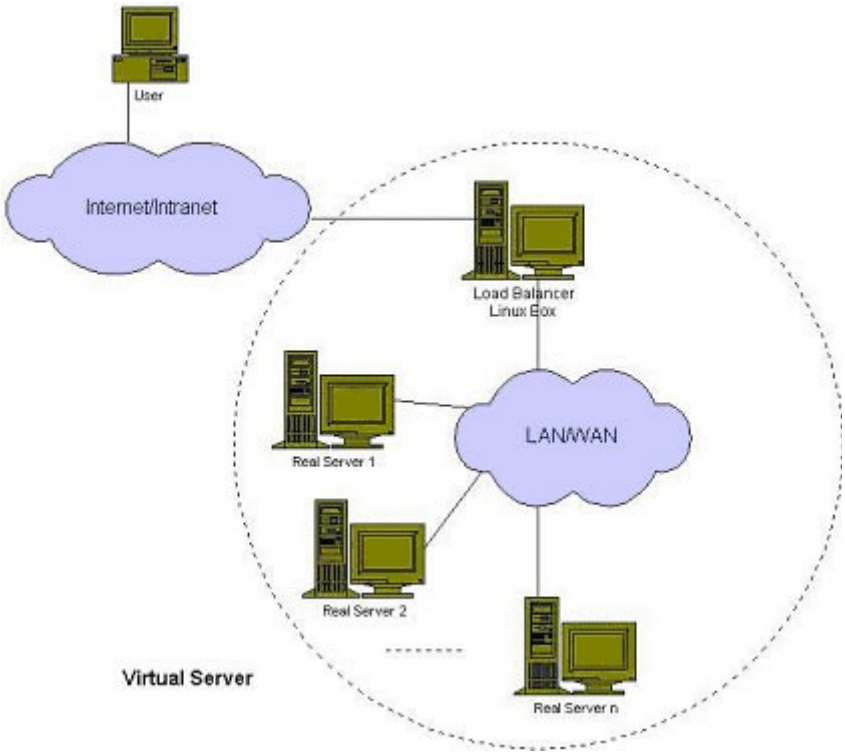
但是从 Linux2.4 内核以后，已经完全内置了 LVS 的各个功能模块，无需给内核打任何补丁，可以直接使用 LVS 提供的各种功能。

使用 LVS 技术要达到的目标是：通过 LVS 提供的负载均衡技术和 Linux 操作系统实现一个高性能、高可用的服务器群集，它具有良好可靠性、可扩展性和可操作性。从而以低廉的成本实现最优的服务性能。

Linux Virtual Server 项目：

针对高可伸缩、高可用网络服务的需求，**LVS** 给出了基于 **IP** 层和基于内容请求分发的负载平衡调度解决方法，并在 Linux 内核中实现了这些方法，将一组服务器构成一个实现可伸缩的、高可用网络服务的虚拟服务器。

虚拟服务器的体系结构如图所示，一组服务器通过高速的局域网或者地理分布的广域网相互连接，在它们的前端有一个负载调度器（Load Balancer）。负载调度器能无缝地将网络请求调度到真实服务器上，从而使得服务器集群的结构对客户是透明的，客户访问集群系统提供的网络服务就像访 问一台高性能、高可用的服务器一样。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机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置系统达到高可用性。由于我们的负载调度技术是在 Linux 内核中实现的，我们称之为 Linux 虚拟服务器（Linux Virtual Server）。

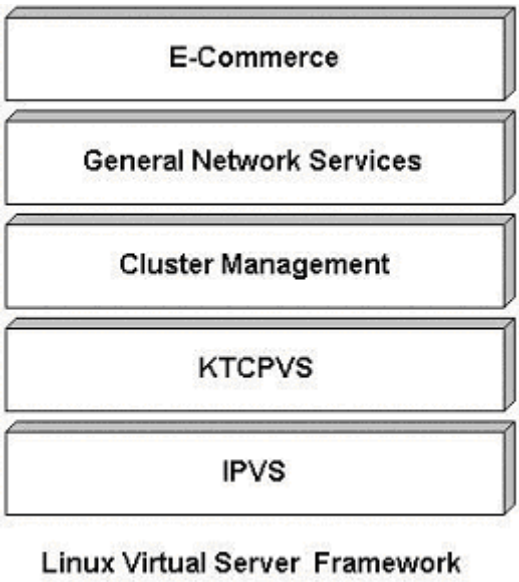


虚拟服务器的结构

在 1998 年 5 月，章文嵩博士成立了 Linux Virtual Server 的自由软件项目，进行 Linux 服务器集群的开发工作。同时，Linux Virtual Server 项目是国内最早出现的自由软件项目之一。

Linux Virtual Server 项目的目标 ：使用集群技术和 Linux 操作系统实现一个高性能、高可用的服务器，它具有很好的可伸缩性（Scalability）、可靠性（Reliability）和可管理性（Manageability）。

目前，LVS 项目已提供了一个实现可伸缩网络服务的 Linux Virtual Server 框架，如图所示。在 LVS 框架中，提供了含有三种 IP 负载均衡技术的 IP 虚拟服务器软件 IPVS、基于内容请求分发的内核 Layer-7 交 换机 KTCPVS 和集群管理软件。可以利用 LVS 框架实现高可伸缩的、高可用的 Web、Cache、Mail 和 Media 等网络服务；在此基础上，可以开 发支持庞大用户数的、高可伸缩的、高可用的电子商务应用。



Linux 虚拟服务器框架

IP 虚拟服务器软件 IPVS：

在调度器的实现技术中，IP 负载均衡技术是效率最高的。在已有的 IP 负载均衡技术中有通过网络地址转换（Network Address Translation）将一组服务器构成一个高性能的、高可用的虚拟服务器，我们称之为 VS/NAT 技术（Virtual Server via Network Address Translation），大多数商品化的 IP 负载均衡调度器产品都是使用此方法，如 Cisco 的 LocalDirector、F5 的 Big/IP 和 Alteon 的 ACEDirector。在分析 VS/NAT 的缺点和网络服务的非对称性的基础上，我们提出通过 IP 隧道实现虚拟服务器的方法 VS/TUN（Virtual Server via IP Tunneling），和通过直接路由实现虚拟服务器的方法 VS/DR（Virtual Server via Direct Routing），它们可以极大地提高系统的伸缩性。所以，IPVS 软件实现了这三种 IP 负载均衡技术，它们的大致原理如下（我们将在其他章节对其工作原 理进行详细描述），

1. **Virtual Server via Network Address Translation（VS/NAT）**
   通过网络地址转换，调度器重写请求报文的目标地址，根据预设的调度算法，将请求分派给后端的真实服务器；真实服务器的响应报文通过调度器时，报文的源地址被重写，再返回给客户，完成整个负载调度过程。
2. **Virtual Server via IP Tunneling（VS/TUN）**
   采用 NAT 技术时，由于请求和响应报文都必须经过调度器地址重写，当客户请求越来越多时，调度器的处理能力将成为瓶颈。为了解决这个问题，调度器把请求报文通过 IP 隧道转发至真实服务器，而真实服务器将响应直接返回给客户，所以调度器只处理请求报文。由于一般网络服务应答比请求报文大许多，采用 VS/TUN 技术后，集群系统的最大吞吐量可以提高 10 倍。
3. **Virtual Server via Direct Routing（VS/DR）**

VS/DR 通过改写请求报文的 MAC 地址，将请求发送到真实服务器，而真实服务器将响应直接返回给客户。同 VS/TUN 技术一样，VS/DR 技术可极大地 提高集群系统的伸缩性。这种方法没有 IP 隧道的开销，对集群中的真实服务器也没有必须支持 IP 隧道协议的要求，但是要求调度器与真实服务器都有一块网卡连 在同一物理网段上。

lvs 调度算法：

针对不同的网络服务需求和服务器配置，IPVS 调度器实现了如下八种负载调度算法：

1. 轮叫（**Round Robin**）
   调度器通过"轮叫"调度算法将外部请求按顺序轮流分配到集群中的真实服务器上，它均等地对待每一台服务器，而不管服务器上实际的连接数和系统负载。
2. 加权轮叫（**Weighted Round Robin**）
   调度器通过"加权轮叫"调度算法根据真实服务器的不同处理能力来调度访问请求。这样可以保证处理能力强的服务器处理更多的访问流量。调度器可以自动问询真实服务器的负载情况，并动态地调整其权值。
3. 最少链接（**Least Connections**）
   调度器通过"最少连接"调度算法动态地将网络请求调度到已建立的链接数最少的服务器上。如果集群系统的真实服务器具有相近的系统性能，采用"最小连接"调度算法可以较好地均衡负载。
4. 加权最少链接（**Weighted Least Connections**）
   在集群系统中的服务器性能差异较大的情况下，调度器采用"加权最少链接"调度算法优化负载均衡性能，具有较高权值的服务器将承受较大比例的活动连接负载。调度器可以自动问询真实服务器的负载情况，并动态地调整其权值。
5. 基于局部性的最少链接（**Locality-Based Least Connections**）
   "基于局部性的最少链接" 调度算法是针对目标 IP 地址的负载均衡，目前主要用于 Cache 集群系统。该算法根据请求的目标 IP 地址找出该目标 IP 地址最近使用的服务器，若该服务器 是可用的且没有超载，将请求发送到该服务器；若服务器不存在，或者该服务器超载且有服务器处于一半的工作负载，则用"最少链接"的原则选出一个可用的服务 器，将请求发送到该服务器。
6. 带复制的基于局部性最少链接（**Locality-Based Least Connections with Replication**）
   "带复制的基于局部性最少链接"调度算法也是针对目标 IP 地址的负载均衡，目前主要用于 Cache 集群系统。它与 LBLC 算法的不同之处是它要维护从一个 目标 IP 地址到一组服务器的映射，而 LBLC 算法维护从一个目标 IP 地址到一台服务器的映射。该算法根据请求的目标 IP 地址找出该目标 IP 地址对应的服务 器组，按"最小连接"原则从服务器组中选出一台服务器，若服务器没有超载，将请求发送到该服务器，若服务器超载；则按"最小连接"原则从这个集群中选出一 台服务器，将该服务器加入到服务器组中，将请求发送到该服务器。同时，当该服务器组有一段时间没

有被修改，将最忙的服务器从服务器组中删除，以降低复制的 程度。

7. 目 标 地 址 散 列 （ **Destination Hashing** ）
"目标地址散列"调度算法根据请求的目标 IP 地址，作为散列键（Hash Key）从静态分配的散列表找出对应的服务器，若该服务器是可用的且未超载，将请求发送到该服务器，否则返回空。
8. 源 地 址 散 列 （ **Source Hashing** ）
"源地址散列"调度算法根据请求的源 IP 地址，作为散列键（Hash Key）从静态分配的散列表找出对应的服务器，若该服务器是可用的且未超载，将请求发送到该服务器，否则返回空。

内核 Layer-7 交换机 KTCPVS

在基于 IP 负载调度技术中，当一个 TCP 连接的初始 SYN 报文到达时，调度器就选择一台服务器，将报文转发给它。此后通过查发报文的 IP 和 TCP 报文头地 址，保证此连接的后继报文被转发到该服务器。这样，IPVS 无法检查到请求的内容再选择服务器，这就要求后端服务器组提供相同的服务，不管请求被发送到哪 一台服务器，返回结果都是一样的。但是，在有些应用中后端服务器功能不一，有的提供 HTML 文档，有的提供图片，有的提供 CGI，这就需要基于内容的调度 (Content-Based Scheduling)。
由于用户空间 TCP Gateway 的开销太大，我们提出在操作系统的内核中实现 Layer-7 交换方法，来避免用户空间与核心空间的切换和内存复制的开销。在 Linux 操作系统的内核中，我们实现了 Layer-7 交换，称之为 KTCPVS（Kernel TCP Virtual Server）。目前，KTCPVS 已经能对 HTTP 请求进行基于内容的调度，但它还不很成熟，在其调度算法和各种协议的功能支持等方面，有大量的工作需要做。
虽然应用层交换处理复杂，它的伸缩性有限，但应用层交换带来以下好处：

- 相同页面的请求被发送到同一服务器，可以提高单台服务器的 Cache 命中率。
- 一些研究[5]表明 WEB 访问流中存在局部性。Layer-7 交换可以充分利用访问的局部性，将相同类型的请求发送到同一台服务器，使得每台服务器收到的请求具有更好的相似性，可进一步提高单台服务器的 Cache 命中率。
- 后端服务器可运行不同类型的服务，如文档服务，图片服务，CGI 服务和数据库服务等。

LVS 集群的特点：

LVS 集群的特点可以归结如下：

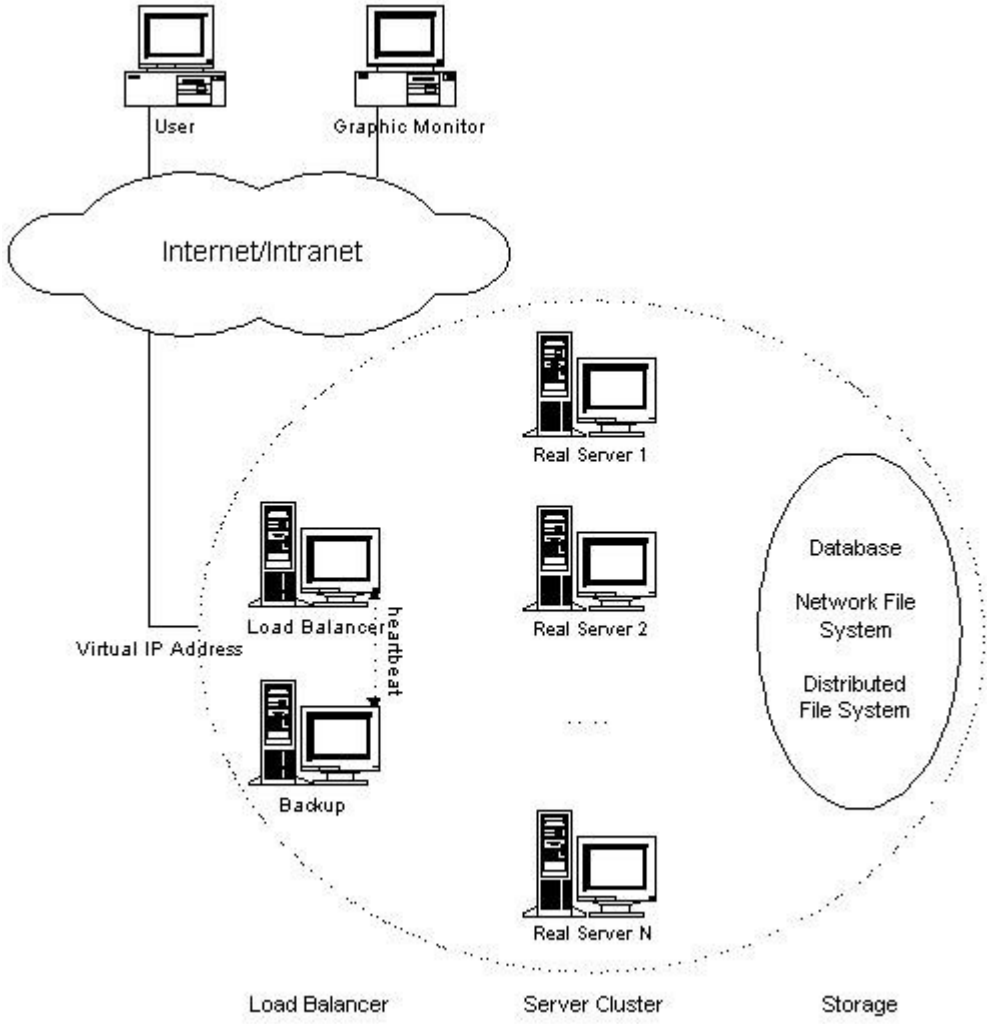
1. 功能
   有实现三种 IP 负载均衡技术和八种连接调度算法的 IPVS 软件。在 IPVS 内部实现上，采用了高效的 Hash 函数和垃圾回收机制，能正确处理所调度报文相 关的 ICMP 消息（有些商品化的系统反而不能）。虚拟服务的设置数目没有限制，每个虚拟服务有自己的服务器集。它支持持久的虚拟服务（如 HTTP Cookie 和 HTTPS 等需要该功能的支持），并提供详尽的统计数据，如连接的处理速率和报文的流量等。针对大规模拒绝服务（Deny of Service）攻击，实现了三种防卫策略。有基于内容请求分发的应用层交换软件 KTCPVS，它也是在 Linux 内核中实现。有相关的集群管理软件对资源进行监测，能及时将故障屏蔽，实现系统的高可用性。主从调度器能周期性地进行状态同步，从而实现更高的可用性。
2. 适用性
   后端服务器可运行任何支持 TCP/IP 的操作系统，包括 Linux，各种 Unix（如 FreeBSD、Sun Solaris、HP Unix 等），Mac/OS 和 Windows NT/2000 等。负载调度器能够支持绝大多数的 TCP 和 UDP 协议：

协议	内容
TCP	HTTP，FTP，PROXY，SMTP，POP3，IMAP4，DNS，LDAP，HTTPS，SSMTP 等
UDP	DNS，NTP，ICP，视频、音频流播放协议等

3. 无需对客户机和服务器作任何修改，可适用大多数 Internet 服务。
4. 性能
   LVS 服务器集群系统具有良好的伸缩性，可支持几百万个并发连接。配置 100M 网卡，采用 VS/TUN 或 VS/DR 调度技术，集群系统的吞吐量可高达 1Gbits/s；如配置千兆网卡，则系统的最大吞吐量可接近 10Gbits/s。
5. 可靠性
   LVS 服务器集群软件已经在很多大型的、关键性的站点得到很好的应用，所以它的可靠性在真实应用得到很好的证实。有很多调度器运行一年多，未作一次重启动。
6. 软件许可证
   LVS 集群软件是按 GPL（GNU Public License）许可证发行的自由软件，这意味着你可以得到软件的源代码，有权对其进行修改，但必须保证你的修改也是以 GPL 方式发行。

LVS 集群的体系结构

LVS 集群采用 IP 负载均衡技术和基于内容请求分发技术。调度器具有很好的吞吐率，将请求均衡地转移到不同的服务器上执行，且调度器自动屏蔽掉服 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。整个服务器集群的结构对客户是透明的，而且无需修改客户端和服务器端的程序。



LVS 集群的体系结构

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。一般来说，LVS 集群采用三层结构，其体系结构如图 1 所示，三层主要组成部分为：

- 负载调度器（**load balancer**），它是整个集群对外面的前端机，负责将客户的请求发送到一组服务器上执行，而客户认为服务是来自一个 IP 地址（我们可称之为虚拟 IP 地址）上的。
- 服务器池（**server pool**），是一组真正执行客户请求的服务器，执行的服务有 WEB、MAIL、FTP 和 DNS 等。
- 共享存储（**shared 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。

调度器是服务器集群系统的唯一入口点（Single Entry Point），它可以采用 IP 负载均衡技术、基于内容请求分发技术或者两者相结合。在 IP 负载均衡技术中，需要服务器池拥有相同的内容提供相同的服务。当 客户请求到达时，调度器只根据服务器负载情况和设定的调度算法从服务器池中选出一个服务器，将该请求转发到选出的服务器，并记录这个调度当这个请求的其 他报文到达，也会被转发到前面选出的服务器。在基于内容请求分发技术中，服务器可以提供不同的服务，当客户请求到达时，调度器可根据请求的内容选择服务

器 执行请求。因为所有的操作都是在 Linux 操作系统核心空间中将完成的，它的调度开销很小，所以它具有很高的吞吐率。

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，请求间不存在很强的相关性，请求可以在不同的结点上并行执行，所以整个系统的性能基本上可以随着服务器池的结点数目增加而线性增长。

共享存储通常是数据库、网络文件系统或者分布式文件系统。服务器结点需要动态更新的数据一般存储在数据库系统中，同时数据库会保证并发 访问时数据的一致性。静态的数据可以存储在网络文件系统（如 NFS/CIFS）中，但网络文件系统的伸缩能力有限，一般来说，NFS/CIFS 服务器只能 支持 3~6 个繁忙的服务器结点。对于规模较大的集群系统，可以考虑用分布式文件系统，如 AFS[1]、GFS[2.3]、Coda[4]和 Intermezzo[5]等。分布式文件系统可为各服务器提供共享的存储区，它们访问分布式文件系统就像访问本地文件系统一样同时分布式文件系统可提 供良好的伸缩性和可用性。此外，当不同服务器上的应用程序同时读写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。开发者在写应用程序时，可以使用分布式锁管理器来保证应用程序在不同结点上并发访 问的一致性。

负载调度器、服务器池和共享存储系统通过高速网络相连接，如 100Mbps 交换网络、Myrinet 和 Gigabit 网络等。使用高速的网络，主要为避免当系统规模扩大时互联网络成为整个系统的瓶颈。

Graphic Monitor 是为系统管理员提供整个集群系统的监视器，它可以监视系统的状态。Graphic Monitor 是基于浏览器的，所以无论管理员在本地还是异地都可以监测系统的状况。为了安全的原因，浏览器要通过 HTTPS（Secure HTTP）协议和身份认证后，才能进行系统监测，并进行系统的配置和管理。

为什么使用层次的体系结构：

层次的体系结构可以使得层与层之间相互独立，每一个层次提供不同的功能，在一个层次可以重用不同的已有软件。例如，调度器层提供了负载平衡、可伸缩 性和高可用性等在服务器层可以运行不同的网络服务，如 Web、Cache、Mail 和 Media 等，来提供不同的可伸缩网络服务。明确的功能划分和清晰 的层次结构使得系统容易建设，以后整个系统容易维护，而且系统的性能容易被扩展。

为什么是共享存储：

共享存储如分布式文件系统在这个 LVS 集群系统是可选项。当网络服务需要有相同的内容，共享存储是很好的选择，否则每台服务器需要将相同的 内容复制到本地硬盘上。当系统存储的内容越多，这种无共享结构（Shared-nothing Structure）的代价越大，因为每台服务器需要一样大的存储空间，任何的更新需要涉及到每台服务器，系统的维护代价会非常高。

共享存储为服务器组提供统一的存储空间，这使得系统的内容维护工作比较轻松，如 Webmaster 只需要更新共享存储中的页面，对所有 的服务器都有效。分布式文件系统提供良好的伸缩性和可用性，当分布式文件系统的存储空间增加时，所有服务器的存储空间也

随之增大。对于大多数 Internet 服务来说，它们都是读密集型（Read-intensive）的应用，分布式文件系统在每台服务器使用本地硬盘作 Cache（如 2Gbytes 的空间），可以使得访问分布式文件系统本地的速度接近于访问本地硬盘。

此外，存储硬件技术的发展也促使从无共享的集群向共享存储的集群迁移。存储区域网（Storage Area Networks）技术解决了集群的每个结点可以直接连接/共享一个庞大的硬盘阵列，硬件厂商也提供多种硬盘共享技术，如光纤通道（Fiber Channel）、共享 SCSI（Shared SCSI）。InfiniBand 是一个通用的高性能 I/O 规范，使得存储区域网中以更低的延时传输 I/O 消息和集群通讯消息，并且提供很好的伸缩性。 InfiniBand 得到绝大多数的大厂商的支持，如 Compaq、 Dell、 Hewlett-Packard、 IBM、 Intel、 Microsoft 和 SUN Microsystems 等，它正在成为一个业界的标准。这些技术的发展使得共享存储变得容易，规模生产也会使得成本逐步降低。

高可用性：

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置系统来实现，使得系统收到的请求能被存活的结点处理。

通常，我们在调度器上有资源监测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 ping 不可达时或者探测她的网络服务在指定的时间没有响应时，资源监测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 不会被调度到坏的结点。资源监测进程能通过电子邮件或传呼机向管理员报告故障。一旦监测进程到服务器恢复工作，通知调度器将其加入调度列表进行调度。另 外，通过系统提供的管理程序，管理员可发命令随时可以将新机器加入服务来提高系统的处理性能，也可以将已有的服务器切出服务，以便对服务器进行系统维护。

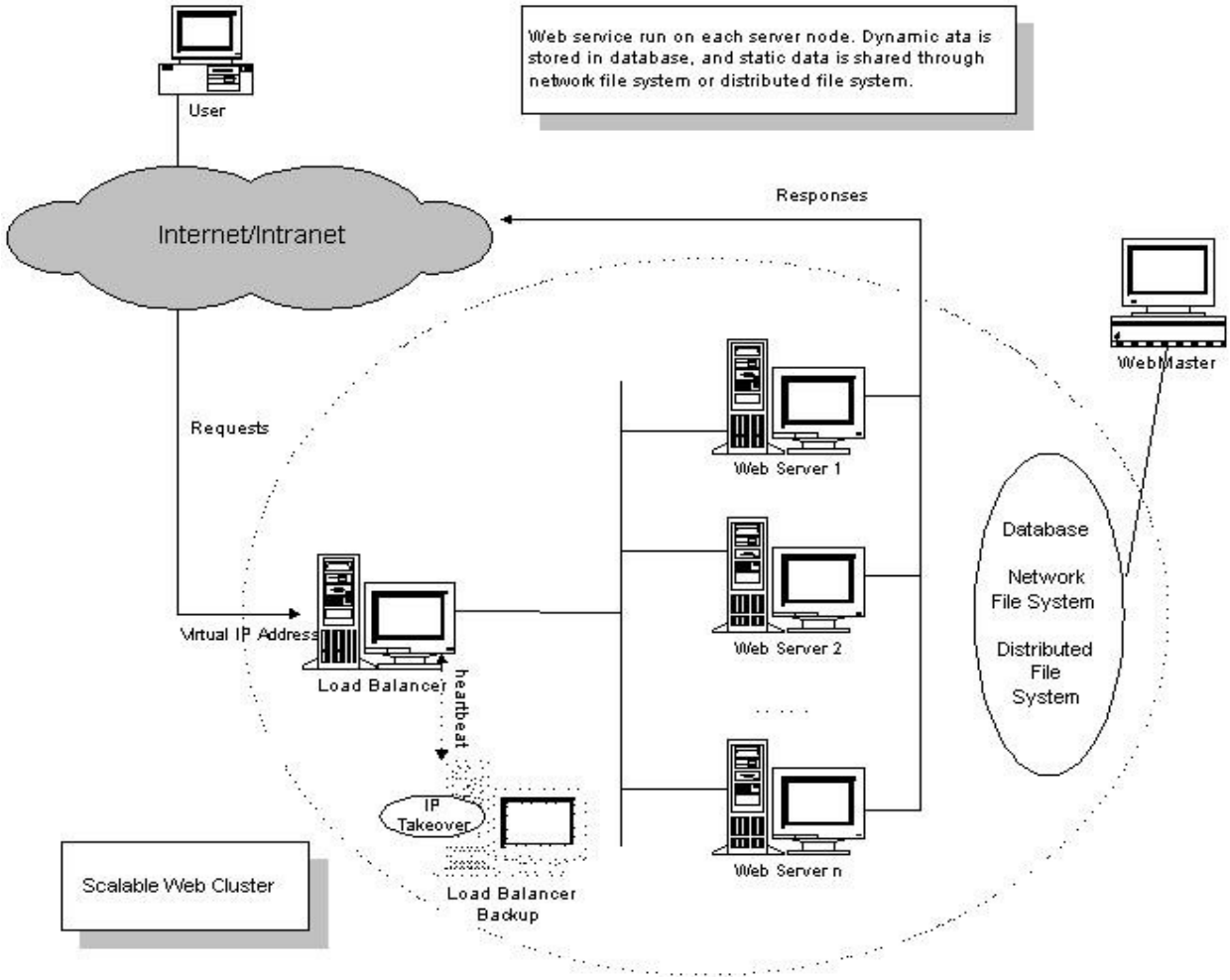
现在前端的调度器有可能成为系统的单一失效点（Single Point of Failure）。一般来说，调度器的可靠性较高，因为调度器上运行的程序较少而且大部分程序早已经遍历过，但我们不能排除硬件老化、网络线路或者人为误 操作等主要故障。为了避免调度器失效而导致整个系统不能工作，我们需要设立一个从调度器作为主调度器的备份。两个心跳（Heartbeat）进程[6]分 别在主、从调度器上运行，它们通过串口线和 UDP 等心跳线来相互定时地汇报各自的健康状况。当从调度器不能听得主调度器的心跳时，从调度器通过 ARP 欺骗 （Gratuitous ARP）来接管集群对外的 Virtual IP Address，同时接管主调度器的工作来提供负载调度服务。当主调度器恢复时，这里有两种方法，一是主调度器自动变成从调度器，二是从调度器释放 Virtual IP Address，主调度器收回 Virtual IP Address 并提供负载调度服务。这里，多条心跳线可以使得因心跳线故障导致误判（即从调度器认为主调度器已经失效，其实主调度器还在正常工作）的概论 降到最低。

通常，当主调度器失效时，主调度器上所有已建立连接的状态信息将丢失，已有的连接会中断。客户需要向重新连接，从调度器才会将新连接调 度到各个服务器上，这对客户会造成一定的不便。为此，IPVS 调度器在 Linux 内核中实现一种高效状态同步机制，将主调度器的状态信息及时地同步到从调度器。当从调度器接管时，绝大部分已建立的连接会持续下去。

可伸缩的服务架构实例

可伸缩 Web 服务

基于 LVS 的 Web 集群的体系结构如图所示：第一层是负载调度器，一般采用 IP 负载均衡技术，可以使得整个系统有较高的吞吐率；第二层是 Web 服务器池，在每个结点上可以分别运行 HTTP 服务或 HTTPS 服务、或者两者都运行；第三层是共享存储，它可以是数据库，可以是网络文件系统或分布 式文件系统，或者是三者的混合。集群中各结点是通过高速网络相连接的。



基于 **LVS** 的 **Web** 集群

对于动态页面（如 PHP、JSP 和 ASP 等），需要访问的动态数据一般存储在数据库服务器中。数据库服务运行在独立的服务器上，为所有 Web 服务器 共享。无论同一 Web 服务器上多个动态页面访问同一数据，还是不同 Web 服务器上多个动态页面访问同一数据，数据库服务器有锁机制使得这些访问有序地进 行，从而保证数据的一致性。

对于静态的页面和文件（如 HTML 文档和图片等），可以存储在网络文件系统或者分布式文件系统中。至于选择哪一种，看系统的规模和需求 而定。通过共享的网络文件系统

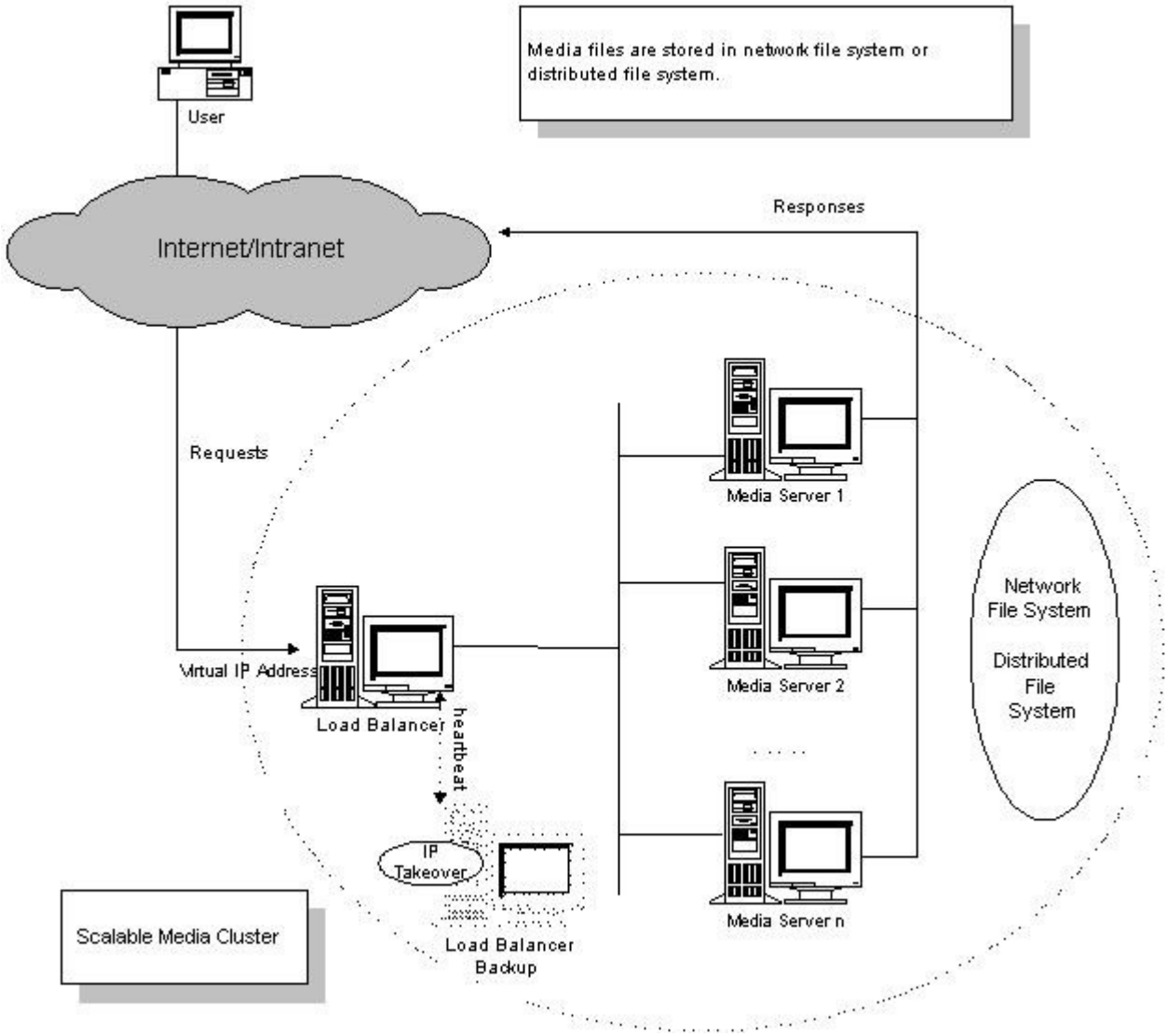
或者分布式文件系统，Webmaster 可以看到统一的文档存储空间，维护和更新页面比较方便，对共享存储中页面的修改对所有的服务器都有效。

在这种结构下，当所有服务器结点超载时，管理员可以很快地加入新的服务器结点来处理请求，而无需将 Web 文档等复制到结点的本地硬盘上。

有些 Web 服务可能用到 HTTP Cookie，它是将数据存储在客户的浏览器来追踪和标识客户的机制。使用 HTTP Cookie 后，来同一客户的不同连接存在相关性，这些连接必须被发送到同一 Web 服务器。一些 Web 服务使用安全的 HTTPS 协议，它是 HTTP 协议加 SSL（Secure Socket Layer）协议。另有些 Web 服务可能使用安全的 HTTPS 协议，它是 HTTP 协议加 SSL 协议。当客户访问 HTTPS 服务（HTTPS 的缺省端口为 443）时，会先建立一个 SSL 连接，来交换对称公钥加密的证书并协商一个 SSL Key，来加密以后的会话。在 SSL Key 的生命周期内，后续的所有 HTTPS 连接都使用这个 SSL Key，所以同一客户的不同 HTTPS 连接也存在相关性。针对这些需要，IPVS 调度器提供了持久服务的功能，它可以使得在设定的时间内，来自同一 IP 地 址的不同连接会被发送到集群中同一个服务器结点，可以很好地解决客户连接的相关性问题。

可伸缩媒体服务

基于 LVS 的媒体集群的体系结构如图所示：第一层是负载调度器，一般采用 IP 负载均衡技术，可以使得整个系统有较高的吞吐率；第二层是 Web 服务器池，在每个结点上可以运行相应的媒体服务；第三层是共享存储，通过网络文件系统/分布式文件系统存储媒体节目。集群中各结点是通过高速网络相 连接。



基于 **LVS** 的媒体集群

IPVS 负载调度器一般使用直接路由方法（即 VS/DR 方法），来架构媒体集群系统。调度器将媒体服务请求较均衡地分发到 各个服务器上，而媒体服务器将响应数据直接返回给客户，这样可以使得整个媒体集群系统具有很好的伸缩性。

媒体服务器可以运行各种媒体服务软件。目前，LVS 集群对于 Real Media、 Windows Media 和 Apple Quicktime 媒体服务都有很好的支持，都有真实的系统在运行。一般来说，流媒体服务都会使用一个 TCP 连接（如 RTSP 协议：Real-Time Streaming Protocol）进行带宽的协商和流速的控制，通过 UDP 将流数据返回客户。这里，IPVS 调度器提供功能将 TCP 和 UDP 集中考虑，保证来自同一客户 的媒体 TCP 和 UDP 连接会被转发到集群中同一台媒体服务器，使得媒体服务准确无误地进行。

共享存储是媒体集群系统中最关键的问题，因为媒体文件往往非常大（一部片子需要几百兆到几千兆的存储空间），这对存储的容量和读的速度 有较高的要求。对于规模较小的媒体集群系统，例如有 3 至 6 个媒体服务器结点，存储系统可以考虑用带千兆网卡的 Linux 服务器，使用软件 RAID 和日志型 文件系统，再运行内核的 NFS 服务，会有不错的效果。对于规模较大的媒体集群系统，最好选择对文件分段（File Stripping）存储和文件缓存有较好支持的分布式文件系统；媒体文件分段存储在分布式文件系统的多个存储结点上，

可以提高文件系统的性能和存储结点 间的负载均衡；媒体文件在媒体服务器上自动地被缓存，可提高文件的访问速度。否则，可以考虑自己在媒体服务器上开发相应的工具，如缓存工具能定时地统计出 最近的热点媒体文件，将热点文件复制到本地硬盘上，并替换缓存中的非热点文件，最后通知其他媒体服务器结点它所缓存的媒体文件以及负载情况；在媒体服务器 上有应用层调度工具，它收到客户的媒体服务请求，若所请求的媒体文件缓存在本地硬盘上，则直接转给本地媒体服务进程服务，否则先考虑该文件是否被其他媒体 服务器缓存；如该文件被其他服务器缓存并且该服务器不忙，则将请求转给该服务器上的媒体服务进程处理，否则直接转给本地媒体服务进程，从后端的共享存储中 读出媒体文件。

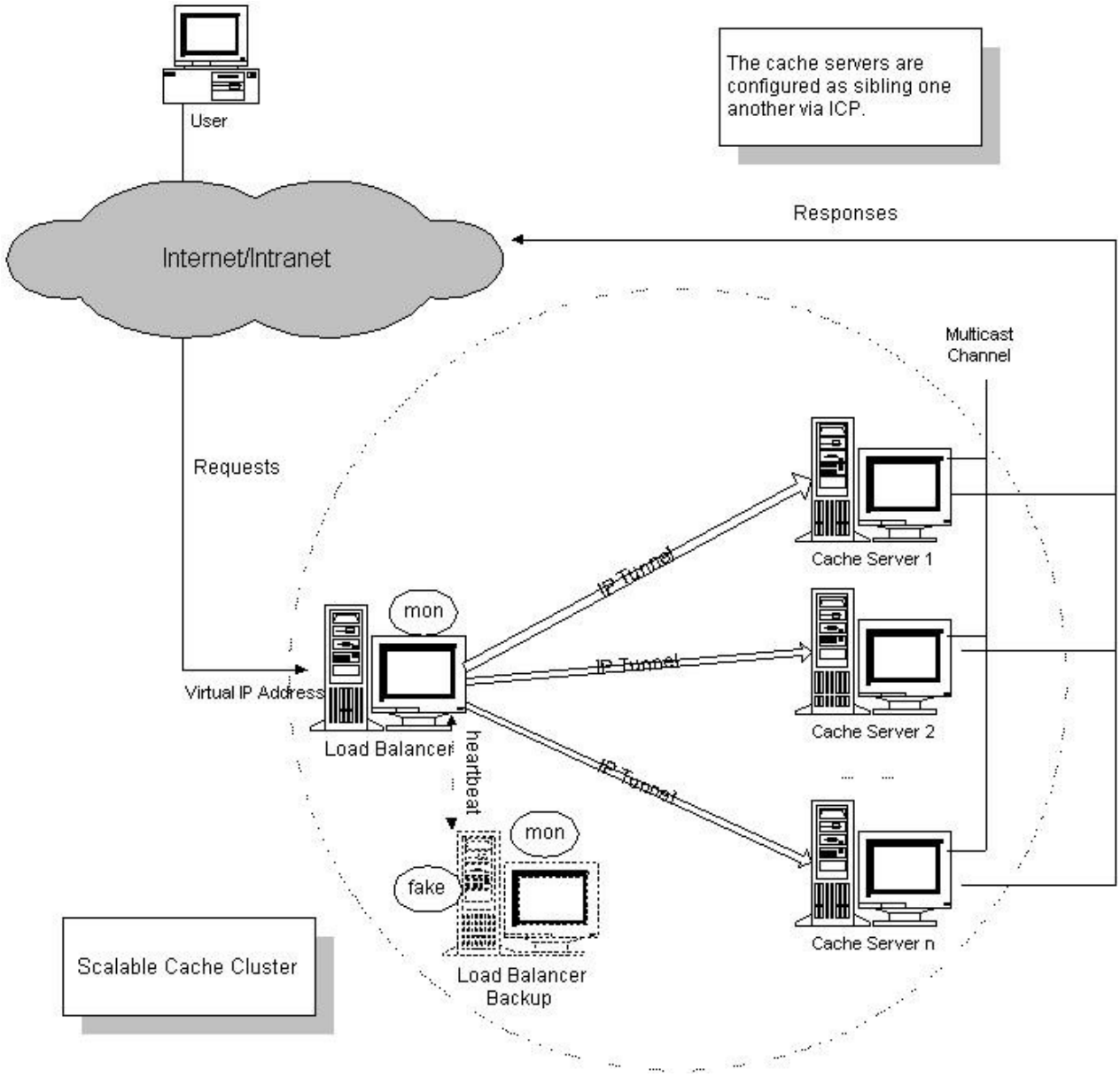
共享存储的好处是媒体文件的管理人员看到统一的存储空间，使得媒体文件维护工作比较方便。当客户访问不断增加使得整个系统超载时，管理员可以很快地加入新的媒体服务器结点来处理请求。

Real 公司以其高压缩比的音频视频格式、 Real 媒体服务器和媒体播放器 RealPlayer 而闻名。Real 公司正在使用以上结构将由 20 多台服务器组成的 LVS 可伸缩 Web 和媒体集群，为其全球用户提供 Web 和音频视频服务。Real 公司的高级技术主管声称 LVS 击败所有他们尝试过的 商品化负载均衡产品。

可伸缩 **Cache** 服务

有效的网络 Cache 系统可以大大地减少网络流量、降低响应延时以及服务器的负载。但是，若 Cache 服务器超载而不能及时地处理请求，反而会增加响应延时。所以，Cache 服务的可伸缩性很重要，当系统负载不断增长时，整个系统能被扩展来提高 Cache 服务的处理能力。尤其，在主干网上的 Cache 服务可能需要几个 Gbps 的吞吐率，单台服务器远不能达到这个吞吐率。可见，通过 PC 服务器集群实现可伸缩 Cache 服务是很有效的方法，也是性能价格比最高的方法。

基于 LVS 的 Cache 集群的体系结构如图所示：第一层是负载调度器，一般采用 IP 负载均衡技术，可以使得整个系统有较高的吞吐率； 第二层是 Cache 服务器池，一般 Cache 服务器放置在接近主干 Internet 连接处，它们可以分布在不同的网络中。调度器可以有多个，放在离客户接 近的地方。



基于 **LVS** 的 **Cache** 集群

IPVS 负载调度器一般使用 IP 隧道方法（即 VS/TUN 方法），来架构 Cache 集群系统，因为 Cache 服务器可能被 放置不同的地方（例如在接近主干 Internet 连接处），而调度器与 Cache 服务器池可能不在同一个物理网络中。 采用 VS/TUN 方法，调度器只调度 Web Cache 请求，而 Cache 服务器将响应数据直接返回给客户。在请求对象不能在本地命中的情况下，Cache 服务器要向源服务器发请求，将结果取回，最 后将结果返回给客户；若采用 NAT 技术的商品化调度器，需要四次进出调度器，完成这个请求。而用 VS/TUN 方法（或者 VS/DR 方法），调度器只调度一 次请求，其他三次都由 Cache 服务器直接访问 Internet 完成。所以，这种方法对 Cache 集群系统特别有效。

Cache 服务器采用本地硬盘来存储可缓存的对象，因为存储可缓存的对象是写操作，且占有一定的比例，通过本地硬盘可以提高 I/O 的访 问速度。Cache 服务器间有专用的多播通道（Multicast Channel），通过 ICP 协议（Internet Cache Protocol）来交互信息。当一台 Cache 服务器在本地硬盘中未命中当前请求时，它可以通过 ICP 查询其他 Cache 服务器是

否有请求对象的副本， 若存在，则从邻近的 Cache 服务器取该对象的副本，这样可以进一步提高 Cache 服务的命中率。

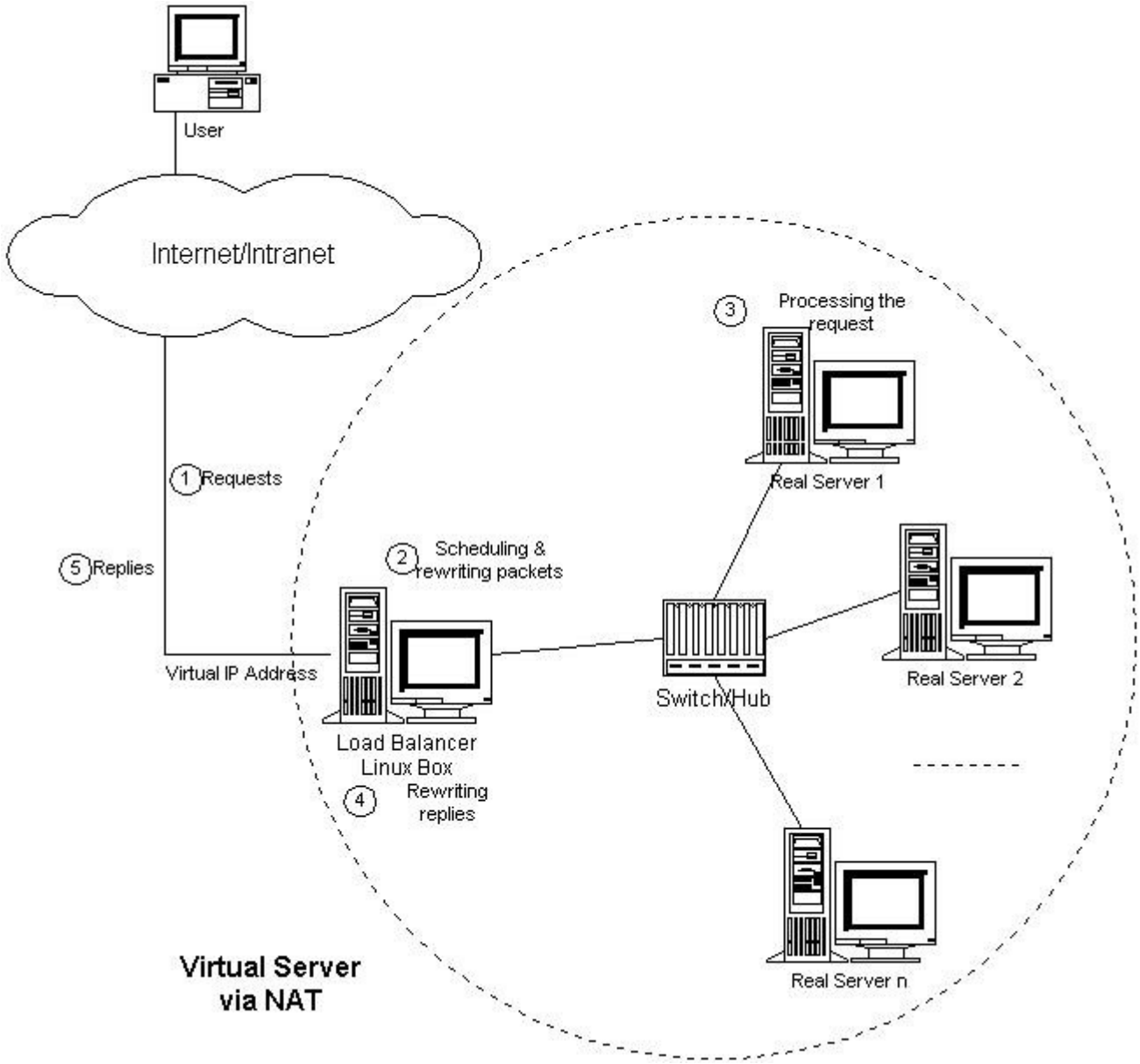
为 150 多所大学和地区服务的英国国家 JANET Web Cache 网在 1999 年 11 月用以上 LVS 结构实现可伸缩的 Cache 集群，只用了原有 50 多台相互独立 Cache 服务器的一半，用户反映网络速 度跟夏天一样快（学生放暑假）。可见，通过负载调度可以摸平单台服务器访问的毛刺（Burst），提高整个系统的资源利用率。

LVS 的三种调度方式

VS/NAT

由 IPv4 中 IP 地址空间的日益紧张和安全方面的原因，很多网络使用保留 IP 地址（10.0.0.0/255.0.0.0、 172.16.0.0/255.128.0.0 和 192.168.0.0/255.255.0.0）[64, 65, 66]。这些地址不在 Internet 上使用，而是专门为内部网络预留的。当内部网络中的主机要访问 Internet 或被 Internet 访问时，就需要 采用网络地址转换（Network Address Translation, 以下简称 NAT），将内部地址转化为 Internets 上可用的外部地址。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。由此，可以用 NAT 方法将不同 IP 地址的并行网络服务变成在一个 IP 地址 上的一个虚拟服务。

VS/NAT 的体系结构如图 2 所示。在一组服务器前有一个调度器，它们是通过 Switch/HUB 相连接的。这些服务器 提供相同的网络服务、相同的内容，即不管请求被发送到哪一台服务器，执行结果是一样的。服务的内容可以复制到每台服务器的本地硬盘上，可以通过网络文件系 统（如 NFS）共享，也可以通过一个分布式文件系统来提供。



VS/NAT 的体系结构

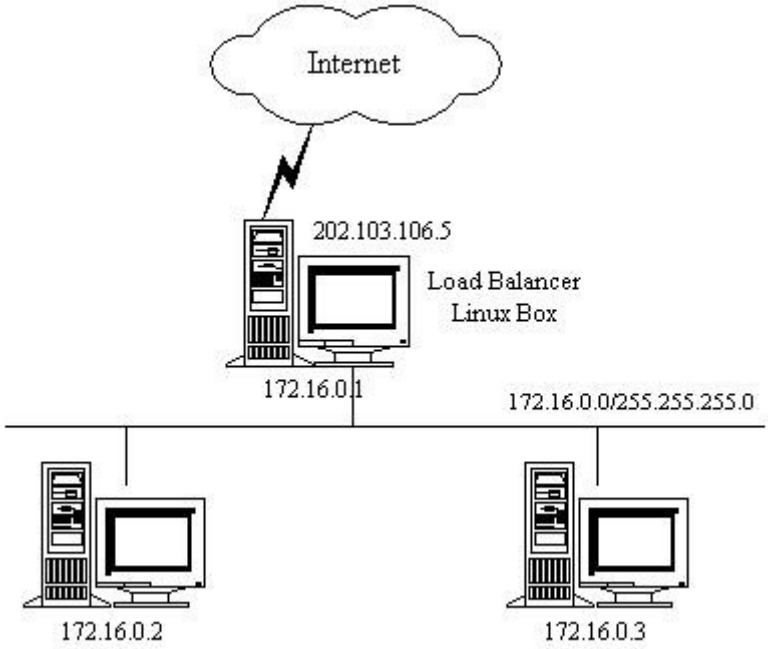
客户通过 Virtual IP Address（虚拟服务的 IP 地址）访问网络服务时，请求报文到达调度器，调度器根据连接调度算法从一组真实服务器中选出一台服务器，将报文的目标地址 Virtual IP Address 改写成选定服务器的地址，报文的目标端口改写成选定服务器的相应端口，最后将修改后的报文发送给选出的服务器。同时，调度器在连接 Hash 表中记录这个连接，当这个连接的下一个报文到达时，从连接 Hash 表中可以得到原选定服务器的地址和端口，进行同样的改写操作，并将报文传给原选定的服务 器。当来自真实服务器的响应报文经过调度器时，调度器将报文的源地址和源端口改为 Virtual IP Address 和相应的端口，再把报文发给用户。我们在连接上引入一个状态机，不同的报文会使得连接处于不同的状态，不同的状态有不同的超时值。在 TCP 连接中，根据标准的 TCP 有限状态机进行状态迁移，这里我们不一一叙述，请参见 W. Richard Stevens 的《TCP/IP Illustrated Volume I》；在 UDP 中，我们只设置一个 UDP 状态。不同状态的超时值是可以设置的，在缺省情况下，SYN 状态的超时为 1 分钟，ESTABLISHED 状态的超 时为 15 分钟，FIN 状态的超时为 1 分钟；UDP 状态

的超时为 5 分钟。当连接终止或超时，调度器将这个连接从连接 Hash 表中删除。

这样，客户所看到的只是在 Virtual IP Address 上提供的服务，而服务器集群的结构对用户是透明的。对改写后的报文，应用增量调整 Checksum 的算法调整 TCP Checksum 的值，避免了扫描整个报文来计算 Checksum 的开销。

在 一些网络服务中，它们将 IP 地址或者端口号在报文的数据中传送，若我们只对报文头的 IP 地址和端口号作转换，这样就会出现不一致性，服务会中断。所以，针 对这些服务，需要编写相应的应用模块来转换报文数据中的 IP 地址或者端口号。我们所知道有这个问题的网络服务有 FTP、 IRC、 H.323、 CUSeeMe、 Real Audio、 Real Video、 Vxtreme / Vosiac 、 VDOLive 、 VIVOActive 、 True Speech 、 RSTP 、 PPTP 、 StreamWorks 、 NTT AudioLink、NTT SoftwareVision、Yamaha MIDPlug、iChat Pager、Quake 和 Diablo。

下面，举个例子来进一步说明 VS/NAT，如图所示：



VS/NAT 的例子

VS/NAT 的配置如下表所示，所有到 IP 地址为 202.103.106.5 和端口为 80 的流量都被负载均衡地调度的真实服务器 172.16.0.2:80 和 172.16.0.3:8000 上。目标地址为 202.103.106.5:21 的报文被转移到 172.16.0.3:21 上。而到其他端口的报文将被拒 绝。

Protocol	Virtual IP Address	Port	Real IP Address	Port	Weight
TCP	202.103.106.5	80	172.16.0.2	80	1
			172.16.0.3	8000	2
TCP	202.103.106.5	21	172.16.0.3	21	1

从以下的例子中，我们可以更详细地了解报文改写的流程。

访问 Web 服务的报文可能有以下的源地址和目标地址：

SOURCE	202.100.1.2:3456	DEST	202.103.106.5:80

调度器从调度列表中选出一台服务器，例如是 172.16.0.3:8000。该报文会被改写为如下地址，并将它发送给选出的服务器。

SOURCE	202.100.1.2:3456	DEST	172.16.0.3:8000

从服务器返回到调度器的响应报文如下：

SOURCE	172.16.0.3:8000	DEST	202.100.1.2:3456

响应报文的源地址会被改写为虚拟服务的地址，再将报文发送给客户：

SOURCE	202.103.106.5:80	DEST	202.100.1.2:3456

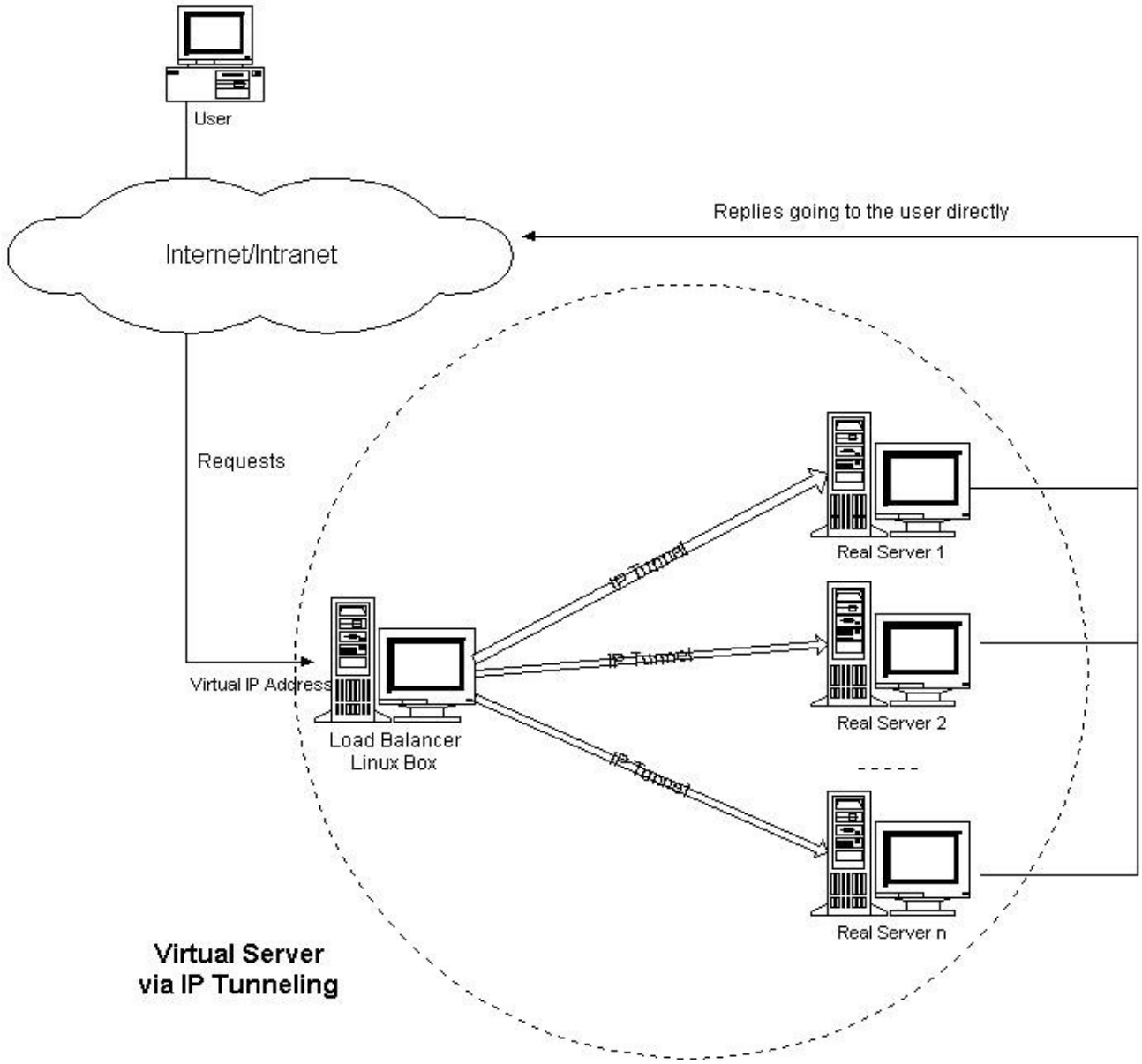
这样，客户认为是从 202.103.106.5:80 服务得到正确的响应，而不会知道该请求是服务器 172.16.0.2 还是服务器 172.16.0.3 处理的。

VS/TUN

在 VS/NAT 的集群系统中，请求和响应的数据报文都需要通过负载调度器，当真实服务器的数目在 10 台和 20 台之间时，负载调度器将成为整个集群系统的新瓶颈。大多数 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，即在负载调度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。

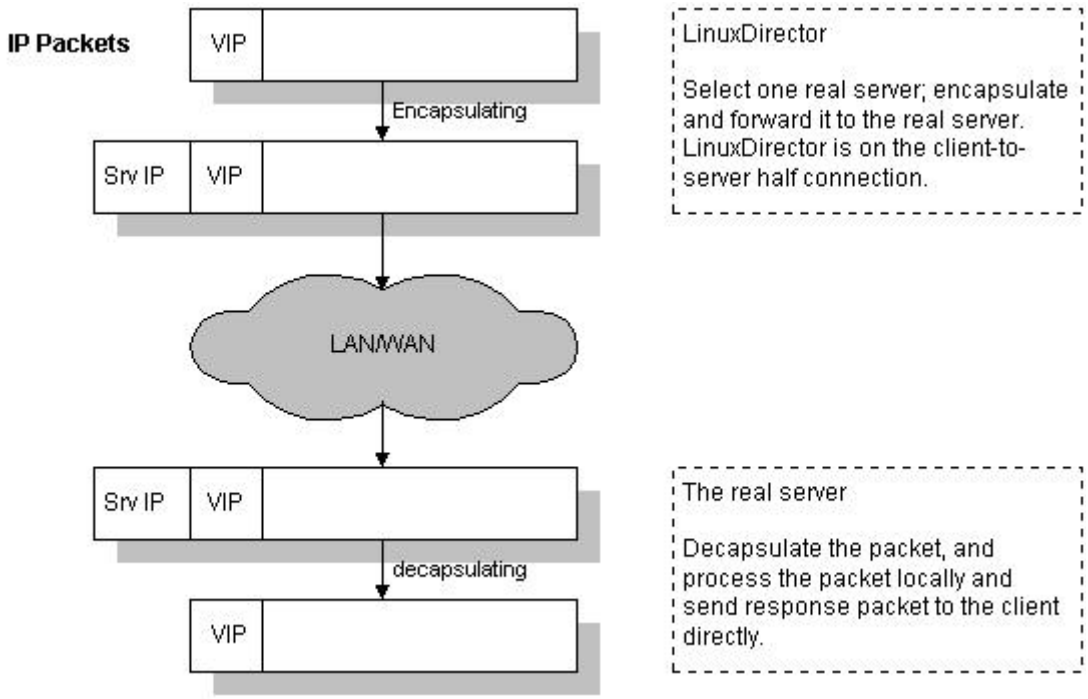
IP 隧道（IP tunneling）是将一个 IP 报文封装在另一个 IP 报文的技术，这可以使得目标为一个 IP 地址的数据报文能被封装和转发到另一个 IP 地址。IP 隧道技 术亦称为 IP 封装技术（IP encapsulation）。IP 隧道主要用于移动主机和虚拟私有网络（Virtual Private Network），在其中隧道都是静态建立的，隧道一端有一个 IP 地址，另一端也有唯一的 IP 地址。

我们利用 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，所以我们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 4 所示，各个服务器将 VIP 地址配置在自己的 IP 隧道设备上。



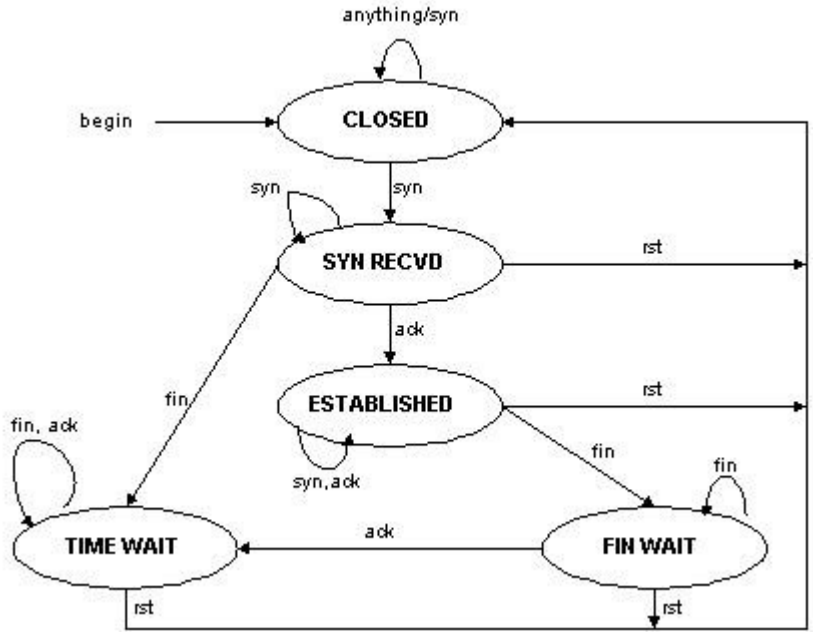
VS/TUN 的体系结构

VS/TUN 的工作流程如图 5 所示：它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。调度器根据各个服务器的负载情况，动态地选择一台服务器， 将请求报文封装在另一个 IP 报文中，再将封装后的 IP 报文转发给选出的服务器；服务器收到报文后，先将报文解封获得原来目标地址为 VIP 的报文，服务器发 现 VIP 地址被配置在本地的 IP 隧道设备上，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。



VS/TUN 的工作流程

在这里需要指出，根据缺省的 TCP/IP 协议栈处理，请求报文的目标地址为 VIP，响应报文的源地址肯定也为 VIP，所以响应报文不需要作任何修改，可以直接返回给客户，客户认为得到正常的服务，而不会知道究竟是哪一台服务器处理的。

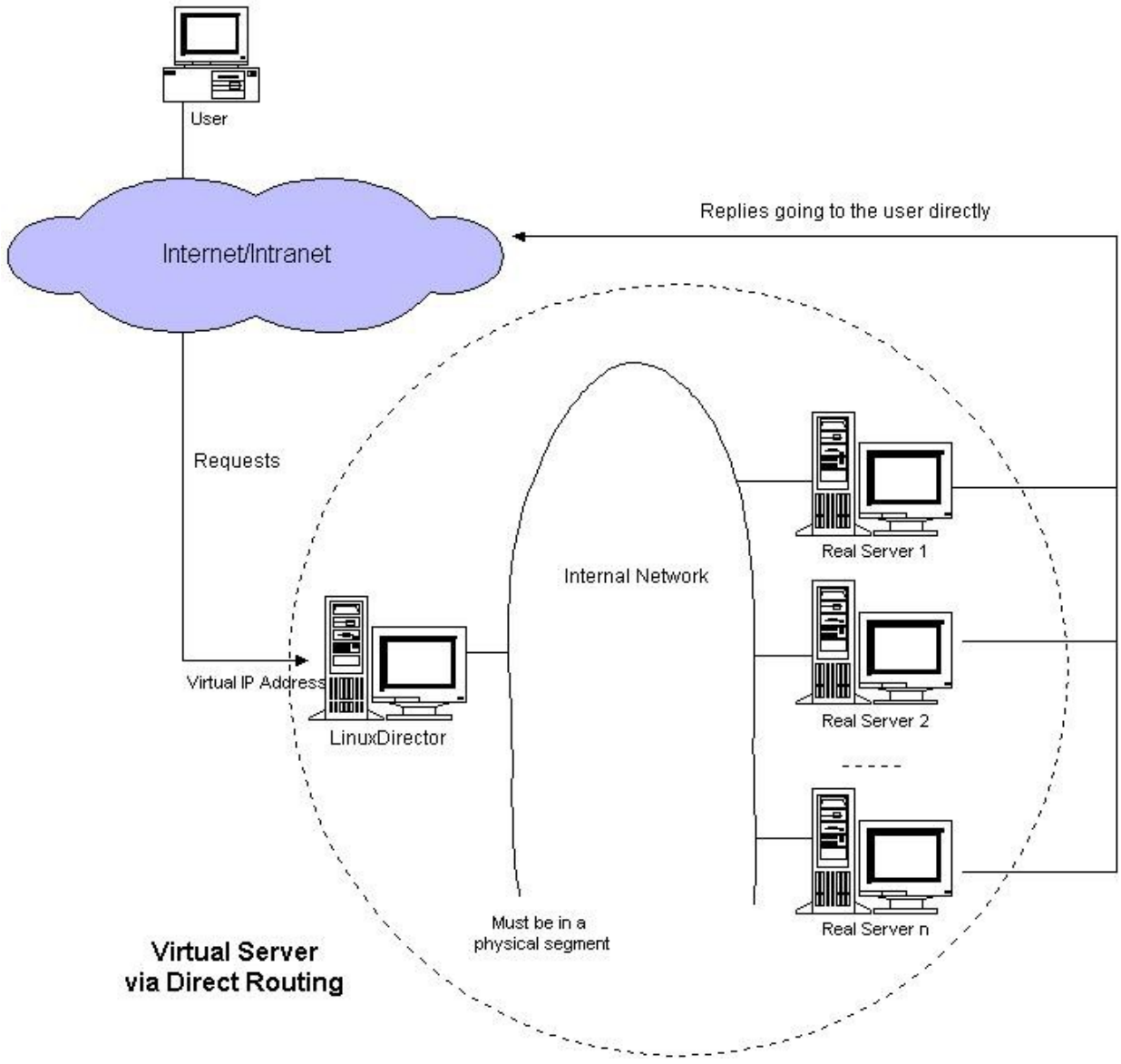


半连接的 **TCP** 有限状态机

VS/DR

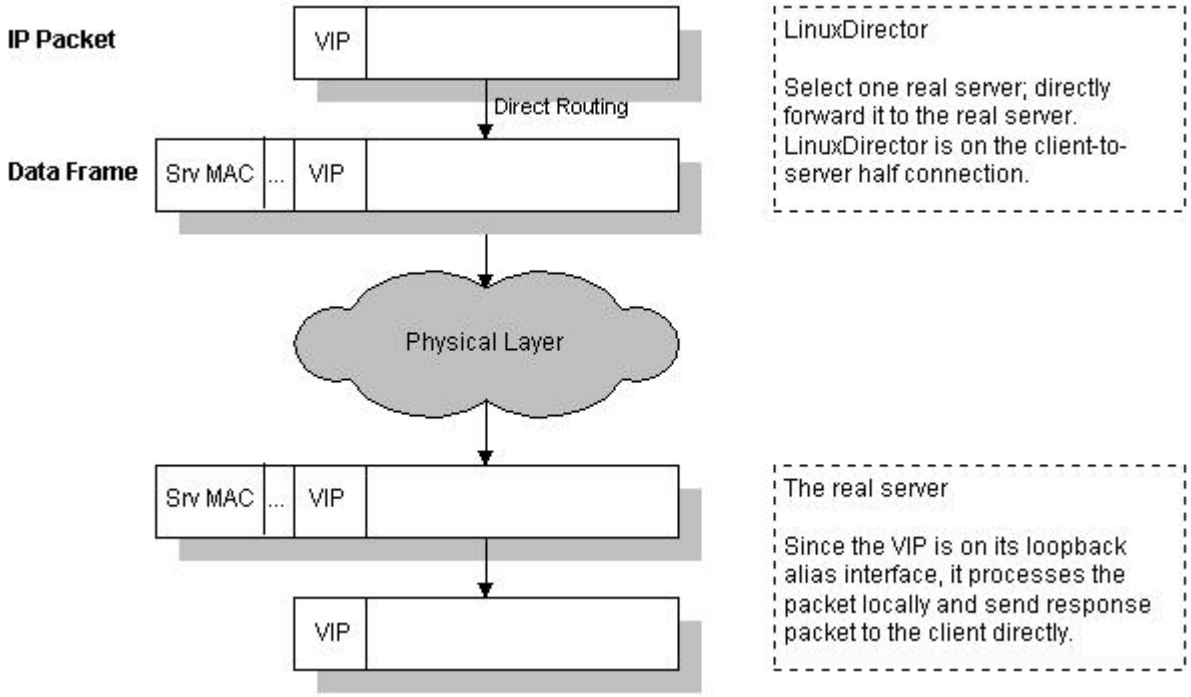
跟 VS/TUN 方法相同，VS/DR 利用大多数 Internet 服务的非对称特点，负载调度器中只负责调度请求，而服务器直接将响应返回给客户，可以极大地提高整个集群 系统的吞吐量该方法与 IBM 的 NetDispatcher 产品中使用的方法类似（其中服务器上的 IP 地址配置方法是相似的），但 IBM 的 NetDispatcher 是非常昂贵的商品化产品，我们也不知道它内部所使用的机制，其中有些是 IBM 的专利。

VS/DR 的体系结构如图 7 所示：调度器和服务器组都必须在物理上有一个网卡通过不分断的局域网相连，如通过高速的交换机或者 HUB 相连。VIP 地址为调度器和服务器组共享，调度 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。



VS/DR 的体系结构

VS/DR 的工作流程如图所示：它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，调度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这个数据帧，从中可以获得该 IP 报文。当服务器发现 报文的目标地址 VIP 是在本地的网络设备上，服务器处理这个报文，然后根据路由表将响应报文直接返回给客户。



VS/DR 的工作流程

在 VS/DR 中，根据缺省的 TCP/IP 协议栈处理，请求报文的目标地址为 VIP，响应报文的源地址肯定也为 VIP，所以响应报文不需要作任何修改，可以直接返回给客户，客户认为得到正常的服务，而不会知道是哪一台服务器处理的。

VS/DR 负载调度器跟 VS/TUN 一样只处于从客户到服务器的半连接中，按照半连接的 TCP 有限状态机进行状态迁移。

三种方法的优缺点比较

三种 IP 负载均衡技术的优缺点归纳在下表中：

_	VS/NAT	VS/TUN	VS/DR
Server	any	Tunneling	Non-arp device
server network	private	LAN/WAN	LAN
server number	low (10~20)	High (100)	High (100)
server gateway	load balancer	own router	Own router

注： 以上三种方法所能支持最大服务器数目的估计是假设调度器使用 100M 网卡，调度器的硬件配置与后端服务器的硬件配置相同，而且是对一般 Web 服务。使用更 高的硬件配置（如千兆网卡和更快的处理器）作为调度器，调度器所能调度的服务器数量会相应增加。当应用不同时，服务器的数目也会相应地改变。所以，以上数 据估计主要是为三种方法的伸缩性进行量化比较。

Virtual Server via 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 地址配置在调度器上，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，调度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调度器。 我们在 Pentium 166 处理器的主机上测得重写报文的平均延时为 60us，性能更高的处理器上延时会短一些。假设 TCP 报文的平均长度为 536 Bytes，则调度器的最大吞吐量为 8.93 MBytes/s. 我们再假设每台服务器的吞吐量为 800KBytes/s，这样一个调度器可以带动 10 台服务器。（注：这是很早以前测得的数据）

基于 VS/NAT 的的集群系统可以适合许多服务器的性能要求。如果负载调度器成为系统新的瓶颈，可以有三种方法解决这个问题：混合方法、VS/TUN 和 VS/DR。在 DNS 混合集群系统中，有若干个 VS/NAT 负载调度器，每个负载调度器带自己的服务器集群，同时这些负载调度器又通过 RR-DNS 组成简 单的域名。但 VS/TUN 和 VS/DR 是提高系统吞吐量的更好方法。

对于那些将 IP 地址或者端口号在报文数据中传送的网络服务，需要编写相应的应用模块来转换报文数据中的 IP 地址或者端口号。这会带来实现的工作量，同时应用模块检查报文的开销会降低系统的吞吐率。

Virtual Server via IP Tunneling

在 VS/TUN 的集群系统中，负载调度器只将请求调度到不同的后端服务器，后端服务器将应答的数据直接返回给用户。这样，负载调度器就可以处理大量的请求，它甚至可以调 度百台以上的服务器（同等规模的服务器），而它不会成为系统的瓶颈。即使负载调度器只有 100Mbps 的全双工网卡，整个系统的最大吞吐量可超过 1Gbps。所以，VS/TUN 可以极大地增加负载调度器调度的服务器数量。VS/TUN 调度器可以调度上百台服务器，而它本身不会成为系统的瓶颈，可以 用来构建高性能的超级服务器。

VS/TUN 技术对服务器有要求，即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，我们没对其他操作系统进行测试。因为“IP Tunneling”正成为各个操作系统的标准协议，所以 VS/TUN 应该会适用运行其他操作系统的后端服务器。

Virtual Server via Direct Routing

跟 VS/TUN 方法一样，VS/DR 调度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。

跟 VS/TUN 相比，这种方法没有 IP 隧道的开销，但是要求负载调度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，或者能将报文重定向（Redirect）到本地的 Socket 端口上。

keepalived

简介

Keepalived 是一个基于 VRRP 协议来实现的 LVS 服务高可用方案，可以利用其来避免单点故障。一个 LVS 服务会有 2 台服务器运行 Keepalived，一台为主服务器（MASTER），一台为备份服务器（BACKUP），但是对外表现为一个虚拟 IP，主服务器会发送特定的消息给备份服务器，当备份服务器收不到这个消息的时候，即主服务器宕机的时候， 备份服务器就会接管虚拟 IP，继续提供服务，从而保证了高可用性。Keepalived 是 VRRP 的完美实现。

详细原理参考 keepalived 权威指南

实际环境配置

lo：

```
[root@vm1 shell]# cat lo.sh
#!/bin/bash
VIP1=192.168.1.227
VIP2=192.168.1.228
case "$1" in
start)
        ifconfig lo:0 $VIP1 netmask 255.255.255.255 broadcast $VIP1 up
        /sbin/route add -host $VIP1 dev lo:0
        echo "1" >/proc/sys/net/ipv4/conf/lo/arp_ignore
        echo "2" >/proc/sys/net/ipv4/conf/lo/arp_announce
        echo "1" >/proc/sys/net/ipv4/conf/all/arp_ignore
        echo "2" >/proc/sys/net/ipv4/conf/all/arp_announce

        ifconfig lo:1 $VIP2 netmask 255.255.255.255 broadcast $VIP2 up
        /sbin/route add -host $VIP2 dev lo:1
        echo "1" >/proc/sys/net/ipv4/conf/lo/arp_ignore
        echo "2" >/proc/sys/net/ipv4/conf/lo/arp_announce
        echo "1" >/proc/sys/net/ipv4/conf/all/arp_ignore
        echo "2" >/proc/sys/net/ipv4/conf/all/arp_announce
```

```
        sysctl -p >/dev/null
        echo "RealServer Start OK"
        ;;
stop)
        ifconfig lo:0 down
        route del $VIP1 2>/dev/null
        echo "1" >/proc/sys/net/ipv4/conf/lo/arp_ignore
        echo "2" >/proc/sys/net/ipv4/conf/lo/arp_announce
        echo "1" >/proc/sys/net/ipv4/conf/all/arp_ignore
        echo "2" >/proc/sys/net/ipv4/conf/all/arp_announce

         ifconfig lo:1 down
        route del $VIP2 2>/dev/null
        echo "1" >/proc/sys/net/ipv4/conf/lo/arp_ignore
        echo "2" >/proc/sys/net/ipv4/conf/lo/arp_announce
        echo "1" >/proc/sys/net/ipv4/conf/all/arp_ignore
        echo "2" >/proc/sys/net/ipv4/conf/all/arp_announce
        echo "RealServer Stoped"
        ;;
*)
        echo "Usage: $0 {start|stop}"
        exit 1
esac
exit 0
```

arp:

```
echo "1"> /proc/sys/net/ipv4/conf/lo/arp_ignore

echo "2"> /proc/sys/net/ipv4/conf/lo/arp_announce

echo "1"> /proc/sys/net/ipv4/conf/all/arp_ignore

echo "2"> /proc/sys/net/ipv4/conf/all/arp_announce
```

lvs1：

```
[root@lvs1 ~]# cat /etc/keepalived/keepalived.conf
! Configuration File for keepalived

global_defs {
#   notification_email {
#     acassen@firewall.loc
```

```
#    failover@firewall.loc
#    sysadmin@firewall.loc
#  }
#  notification_email_from Alexandre.Cassen@firewall.loc
#  smtp_server 192.168.200.1
#  smtp_connect_timeout 30
#  router_id LVS_DEVEL
}

vrrp_instance VI_1 {
   state MASTER
   interface eth0
   virtual_router_id 201
   priority 150
   advert_int 1
   authentication {
      auth_type PASS
      auth_pass 1111
   }
   virtual_ipaddress {
      192.168.1.227
      192.168.1.247
   }
}


vrrp_instance VI_2 {
   state SLAVE
   interface eth0
   virtual_router_id 93
   priority 110
   advert_int 1
   authentication {
      auth_type PASS
      auth_pass 1111
   }
   virtual_ipaddress {
      192.168.1.228
      192.168.1.246
   }
}
```

```
virtual_server_group www {
        192.168.1.227 80
        192.168.1.228 80
}
virtual_server_group mysql {
        192.168.1.247 3306
        192.168.1.246 3306
}

virtual_server group www {
    delay_loop 1
    lb_algo wlc
    lb_kind DR
    persistence_timeout 300
    protocol TCP

    real_server 192.168.1.232 80 {
        weight 100
        TCP_CHECK {
                connect_timeout 5
           }
        }

    real_server 192.168.1.233 80 {
        weight 100
        TCP_CHECK {
                connect_timeout 5
           }
        }

}
virtual_server group mysql {
    delay_loop 1
    lb_algo wlc
    lb_kind DR
    persistence_timeout 300
    protocol TCP

    real_server 192.168.1.248 3306 {
        weight 100
        TCP_CHECK {
                connect_timeout 5
```

```
        }
    }
  real_server 192.168.1.249 3306 {
    weight 100
    TCP_CHECK {
        connect_timeout 5
      }
    }
}
```

lvs2:

```
[root@lvs2 ~]# cat /etc/keepalived/keepalived.conf
! Configuration File for keepalived

global_defs {
#   notification_email {
#     acassen@firewall.loc
#     failover@firewall.loc
#     sysadmin@firewall.loc
#   }
#   notification_email_from Alexandre.Cassen@firewall.loc
#   smtp_server 192.168.200.1
#   smtp_connect_timeout 30
#   router_id LVS_DEVEL
}

vrrp_instance VI_1 {
    state SLAVE
    interface eth0
    virtual_router_id 201
    priority 110
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        192.168.1.227
        192.168.1.247
    }
}
```

```
vrrp_instance VI_2 {
    state MASTER
    interface eth0
    virtual_router_id 93
    priority 150
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        192.168.1.228
        192.168.1.246
    }
}

virtual_server_group www {
        192.168.1.227 80
        192.168.1.228 80
}
virtual_server_group mysql {
        192.168.1.247 3306
        192.168.1.246 3306
}

virtual_server group www {
    delay_loop 7
    lb_algo wlc
    lb_kind DR
    persistence_timeout 300
    protocol TCP

    real_server 192.168.1.232 80 {
        weight 100
        TCP_CHECK {
                connect_timeout 5
           }
        }

    real_server 192.168.1.233 80 {
```

```
    weight 100
    TCP_CHECK {
        connect_timeout 5
      }
    }

}
virtual_server group mysql {
  delay_loop 7
  lb_algo wlc
  lb_kind DR
  persistence_timeout 300
  protocol TCP

  real_server 192.168.1.248 3306 {
    weight 100
    TCP_CHECK {
        connect_timeout 5
      }
    }

  real_server 192.168.1.249 3306 {
    weight 100
    TCP_CHECK {
        connect_timeout 5
      }
    }

}
```

WEB 层的配置

缓存暂时略过

nginx

```
[root@iZ23jo5tyxxZ conf]#  cat nginx.conf
user  goujia goujia;
worker_processes  4;
worker_rlimit_nofile 100000;

error_log  logs/error.log warn;
pid        logs/nginx.pid;

events {
        worker_connections 2048;
        multi_accept on;
        use epoll;
}


http {
    include       mime.types;
    default_type  application/octet-stream;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';
    access_log  logs/access.log  main;
    #access_log  off;

    server_tokens off;
    sendfile        on;
    tcp_nopush     on;

   # keepalive_timeout  65;
    keepalive_timeout 20;
    client_header_timeout 20;
    client_body_timeout 20;
    reset_timedout_connection on;
    send_timeout 20;

    gzip  on;
    charset UTF-8;
    gzip_disable "msie6";
```

```
    gzip_proxied any;
    gzip_min_length 1000;
    gzip_comp_level 6;
    gzip_types text/plain text/css application/json application/x-javascript text/xml application/xml
application/xml+rss text/javascript;

    #open_file_cache max=100000 inactive=20s;
    #open_file_cache_valid 30s;
    #open_file_cache_min_uses 2;
    #open_file_cache_errors on;

    #cache
    #client_max_body_size 10m;
    #client_body_buffer_size 328k;
    #proxy_connect_timeout 90;
    #proxy_send_timeout 90;
    #proxy_read_timeout 90;
    #proxy_buffer_size 40k;
    #proxy_buffers 4 320k;
    #proxy_busy_buffers_size 640k;
    #proxy_temp_file_write_size 640k;
    #proxy_temp_path /home/goujia/project/nginxcache/temp_dir;
                        #proxy_cache_path       /home/goujia/project/nginxcache/cache       levels=1:2
keys_zone=cache_one:200m inactive=30d max_size=30g;

    #include
    #include /usr/local/jimmy/conf/v_host/test_ges.conf;
    #include /usr/local/jimmy/conf/v_host/test_bbs.conf;
    #include /usr/local/jimmy/conf/v_host/test_trade.conf;
    #include /usr/local/jimmy/conf/v_host/test_api.conf;
    #include /usr/local/jimmy/conf/v_host/web.conf;
    #include /usr/local/jimmy/conf/v_host/img.conf;
    #include
    include /usr/local/jimmy/conf/v_host/uber.conf;
    include /usr/local/jimmy/conf/v_host/test_ges.conf;
    include /usr/local/jimmy/conf/v_host/test_bbs.conf;
}
```

```
[root@iZ23jo5tyxxZ v_host]# cat ges.conf
upstream  ges {
  server   127.0.0.1:9997;
}
```

```
server {
        listen       80;
        server_name     ges.goujiawang.com;
        access_log     /usr/local/jimmy/logs/ges.log;
        index index.html;

         limit_rate_after 100M;
    limit_rate 200k;
        client_max_body_size    100m;
        gzip on;

        error_page  500 502 503 504  /50x.html;



        proxy_redirect off;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

        #client_max_body_size 10m;
        #client_body_buffer_size 128k;
        proxy_connect_timeout 900;
        proxy_send_timeout 900;
        proxy_read_timeout 900;
        proxy_buffer_size 4k;
        proxy_buffers 32 4k;
        proxy_busy_buffers_size 64k;

         #location ~* ^.+.gzjs$ {
         #    root /usr/local/apache2/htdocs;
         #    add_header  Content-Encoding  gzip;
         #}

        location / {
            proxy_pass  http://ges;
        }

        location ~* \.(gif|jpg|jpeg|png|bmp|ico|rar|css|js|zip|swf)$ {
          proxy_pass  http://ges;
          expires 24h;
        }
```

```
    }
```

一个配置示例文件;