计算机科学2002Vol. 29№. 4

Web服务器群集负载平衡技术与实现

Load Balance in Web Server Cluster and its Implementation

曾 洁 胡永培 卢显良

(电子科技大学计算机系8020教研室 成都610054)

Abstract In this paper we introduce various methods of load balancing in Web Server Cluster. In particular, we discuss the methods based on remapping requests and responses in the network. We also analyze the implementation of load balancer, a server in the front of the Web Server Cluster. We provide various implementation methods to the two techniques, redirection in the network layer and between the network and data link layer. Fault Tolerance and ASP sessions in load balance are also discussed.

Keywords Web server, Cluster, Load balance, Implementation

1. 引言

网络的飞速发展和网站访问量的急剧增加对Web服务器的性能提出了越来越高的要求。过去,服务器无法承受庞大的网络流量所带来的压力时,只能通过淘汰原服务器,代之以性能更高的新服务器来解决。然而随着网络的膨胀,这种淘汰将是迅速而且永无止境的,这造成了资源的极大浪费。群集技术(Cluster)为这一问题提供了较好的解决方案。将众多计算机协同起来完成同样的工作,使Web服务器具有了较好的可伸缩性,并且由于被群集在一起的计算机可以相互协作,系统的性能和健壮性也大大提高了。一旦某个结点出现故障或无法工作,群集中的其他机器都可以接替它完成工作。

对于要求同时处理上百万次请求的Web服务器,其负载平衡通常在硬件中实现。但通常情况下,软件方法便能符合实际应用的要求。本文系统地研究了与Web服务器群集负载平衡有关的负载平衡技术、负载平衡算法和实现方法。

2. 群集技术

计算机群集的提出和使用已有十多年的历史。作为群集技术的最初设计师之一G. Pfister将群集定义为"一个由一系列互联整机所组成的并行或分布式系统,而该系统被当作独立且统一的计算资源来使用"。

2.1 群集的分类

群集通常可以分为三类:服务器群(Server Farm)、双机热备份群集(FailoverCluster)和耦合群集(CoupledCluster)[1]。

服务器群是一种经典和简单的解决方法。它由一系列节点机组成,这些节点机从一个称为"管理器"的中心单元处获取任务。当网络中存在着大量的计算和处理需求而节点机之间的通讯量很低时,这种方法不失为一种简单有效且强大有力的技术。Pixar动画制作室使用一个服务器群来完成电影《玩具总动员》的动画设计工作。在这一过程中,总体的应用是由许多小的、精细复杂的子任务组成的。每一个子任务都由群集中的一台节点机单独完成。群集所具有的最基本的容错性能保证了,一旦对某台机器发出的请求失败,"管理器"可以将这一请求重新分配给其它机器。然而,服务器群技术是以"严格的平行"(服务器对等)而著称的。当遇到需要所有机器协同完成一样工作的情况时,这一模式便不再适用了。

双机热备份群集代表了另一种相对经典且简单的网络架构。它把精力放在了可用性而非可伸缩性上。它通常由两个节点组成:一个主节点和一个备份节点。两个节点共同来提供不间断的服务。如果一台机器出现了故障,另一台会马上接替它工作。

耦合群集中紧密协同在一起共同完成同一工作的多台机器不再是相对独立的,它需要进行大量的跨节点通讯。与双机热备份群集不同的是,可用性是由整个群体共同来提供的。这意味着耦合群集有能力从容地处理多点故障。

2.2 群集的特点

群集技术使多个计算机协同起来作为一个统一体以提高整个系统的性能。在Internet服务应用中,尤其显示出其优于单机系统的三大特点:增长的可伸缩性、高度的可达性和较好的性能/价格比[2]。

·可伸缩性 Web服务器往往需要同时处理来自不同用户的请求,这使得Web服务具有高度并行和服务时间短的特点。因此,群集技术非常适合于Web服务器之间的负载分担。利用群集技术高度伸缩的特点,我们淘汰了以前那种叉车式的推倒重来的升级方法,依据服务器负载的实际大小,动态调整Web服务器群集的尺寸和性能,从而更好地增强系统的可伸缩性。

·可达性 群集中各个服务器是相对独立的,对于群集系统,存在着一些自然的冗余。这种冗余使得我们可以对整个群集在在线状态下进行"热"更新,即我们可以交替地更新一部分服务器,对用户而言,任何时候系统都是可达的,这对Web服务来说是非常重要的。

·较好的性价比 群集技术可以将不同类型,不同性能的计算机组织起来协同工作。并且,这种协同通常能够达到某些大型机的性能。在硬件更换以及软件维护中,群集中普通计算机(例如PC机)的维护要比同等性能的大型机容易很多。

3. Web服务器群集负载平衡技术

HTTP协议是无状态的,所以对Web页面上任何对象(例如,一个图标)的请求都必须单独建立一个TCP连接来完成。因此,每个HTTP请求都可以被独立寻址。HTTP的这种特性使得可以通过将发送到同一逻辑服务器的请求转发到多个具有相同内容的物理服务器的方法来实现Web服务器群集的网络负载平衡。

Web 页面地址 URLs(Universal Resource Locators)通常包含一个规范名字 CNAME(Canonical Name)而不是一个 IP 地址。当请求通过域名服务器 DNS(Domain Name System)时,CNAME 被映射成相应的 IP 地址。而对具有相同内容的、对等的服务器群集,负载平衡要求能将一个逻辑地址映射到多个不同的物理服务器。这种映射通常可以在三个逻辑层次上进行:客户、服务器和网络。换句话说,就是在获取 Web 对象的过程中,URLs 将依次被映射成 IP 地址、MAC 地址和目标定位符(object locator),最终在某台服务器上找到所要获取的对象,如图1所示。URLs 的每次映射都给我们提供了一次将请求转发到对负载平衡而言最佳的服务器的机会。而事实上,负载平衡就是在某个逻辑层次进行请求转发。因此,依据负载平衡进行的不同逻辑层次,负载平衡技术可以分为三大类:客户端、服务器端和网络端[3]。

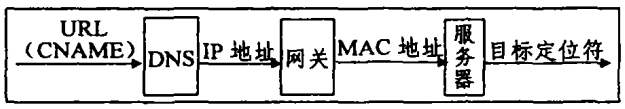

图1 URL 映射流程图

3.1 在客户端进行的负载平衡技术

客户端的负载平衡是指在 URLs 第一次映射时实现负载平衡,常见的有两类技术:对客户端不透明的智能 client 技术和对客户端透明的 DNS 负载平衡技术。

3.1.1 *智能客户端* 由客户端来完成 URLs 到具体服务器地址的映射以进行负载平衡。客户端缓存所有服务器的负载信息,当需要发送 HTTP 请求时,客户端根据缓存的信息自主决定把请求发给哪台服务器。这种技术要求客户端同服务器之间通过通信更新服务器负载信息。传统的方法是采用轮询或 Lazy Update。轮询就是客户端开一个进程,自始至终不断的向 Server Manager(服务器管理程序)发请求,而 Server Manager 不断将各服务器的最新预测负载,机器加入和退出信息会发给客户端,这无疑增加了不必要的网络传输开销。客户端的请求间隔越小,所缓存的服务器负载信息越精确,网络传输也越大;客户端的请求间隔越大,网络传输变小,但又不能及时反映服务器的信息变更,可能导致客户端负载平衡决策的偏差。Lazy Update 方法,是在客户端初起时,Server Manager 一次性将预测负载传给它,而后客户端选择具体的服务器为其服务。当服务器没有负载变动时,服务器就正常响应客户端的请求;当服务器发生负载变动时,就将变动后的信息附在正常的对客户端请求的回答上,节省了网络开销。但该方法有一个明显的缺点,那就是只有被客户端请求的服务器的信息会得到更新,其它服务器的信息得不到更新。这可能会严重影响负载平衡效果。另外,一些改进的方法通过特定算法在客户端预测负载来适当减少客户同服务器之间的通信量。

智能客户端的优点在于将负载平衡工作分散到各个客户,大大减轻了服务器的负担,但客户端所缓存的负载信息的滞后性以及客户/服务器会话所带来的额外的网络传输又降低了负载平衡的准确性和效率。Netscape 就曾在其浏览器中加入智能客户端功能以负载平衡对 Netscape 站点的访问[4]。

3.1.2 *基于 DNS 的负载平衡* 最早的负载平衡技术是通过 DNS 服务中的随机名字解析来实现的。在 DNS 服务器中,可以为多个不同的地址配置同一个名字,DNS 解析这个名字时返回给客户机其中的一个地址。因此,对于同一个名字,不同的客户机会得到不同的地址,访问不同地址上的 Web 服务器,从而达到负载平衡的目的。

Round-Robin DNS 是现在被广泛使用的一种方法,它使用 Round-Robin 算法来调度系统,进行负载平衡。例如如果希望使用三个 Web 服务器来回应对 www.uestc.edu.cn 的 HTTP 请求,就可以设置该域的 DNS 服务器中关于该域的数据包括有与下面例子类似的结果:

```
www1    IN    A      202.112.14.167
www2    IN    A      202.112.14.168
www3    IN    A      202.112.14.169
www           IN     CNAME    www1
www           IN     CNAME    www2
www           IN     CNAME    www3
```

此后外部的客户机就可能随机地得到对应 www 的不同地址,那么随后的 HTTP 请求也就发送给不同地址了。

DNS 负载平衡的优点是简单、易行,并且服务器可以位于互联网的任意位置上,当前使用在包括 Yahoo 在内的 Web 站点上。然而,由于它忽视单个服务器的实际负载和可达性,因而存在不少缺点[5]:

第一,DNS 负载平衡的结果可能造成负载向个别 Server 倾斜。为减少冗余和增强 DNS 的性能,Internet 上大量 DNS 对通过它的域名到 IP 的映射做了缓存。于是,从 Round-Robin DNS 得到的映射信息将被客户端本地 DNS 缓存。所有被该本地 DNS 服务的请求都会到达同一个 IP 地址(即同一个服务器),从而导致当大量用户使用同一个 DNS 时,负载分配将倾斜到个别服务器上。最终造成 Web 服务器群集内部不均衡、高度可变和不可预测的负载分配。例如,在一个拥有通常网络内容和流量的4个服务器组成的系统中,其中的一个服务器可能会收到75%的负载,从而导致该服务器的崩溃。

第二,DNS 的缓存策略也可能导致客户继续访问已崩溃的服务器。当某个服务器崩溃后,DNS 缓存并不能被及时更新。这样,客户请求将被继续送往已崩溃的服务器。虽然可以通过减小缓存的 TTL(time to live)值来提高缓存的更新率,但这样会大幅增大 DNS 自身的负载和不必要的网络传输。

第三,DNS 无法探测 TCP 连接状态、服务器状态和应用信息,当某个服务器崩溃后,Round-Robin DNS 无法知道,并将继续把客户请求分配到该服务器。

虽然如此,由于其简单和易操作性,基于 DNS 的负载平衡在实际中仍然被广泛使用。

3.2 在服务器端进行的负载平衡技术

服务器端负载平衡是指在服务器端应用层上,通过修改 Web 服务器,用 http 重定向或代理机制实现负载分担,即在 URLs 第三次映射时实现负载平衡。常见的技术有 HTTP 重定向技术和反向代理技术。

3.2.1 *HTTP 重定向技术* 用 HTTP 返回 URL Redirection 码来实现负载平衡。当服务器自身负载非常重时,就不对接收到的请求进行服务,而是将请求发给另一个服务器。由于发送 Location 指令比起执行服务请求,对 Web 服务器的负载要小得多,因此可以防止单个服务器负载过大。

这种方法要求必须修改服务器支持这种功能,实现起来比较麻烦,而且没有机制保证重定向的服务器是空闲的,可能造成死循环,因此这种方法在实际应用中并不多见。有些特定情况下可以使用 CGI(包括使用 FastCGI 或 mod_perl 扩展来改善性能)来模拟这种方式去分担负载,而 Web 服务器仍然保持简洁、高效的特性,此时避免 Location 循环的任务将由

用户的CGI程序来承担。

3.2.2 *反向代理负载平衡* 用代理服务器将请求均匀转发给多台内部Web服务器，从而达到负载平衡的目的。这种代理方式与普通的代理方式有所不同，标准代理方式是客户使用代理访问多个外部Web服务器，而这种代理方式是多个客户使用它访问内部Web服务器，因此也被称为反向代理模式。

因为地址映射在应用层进行，每针对一次代理，代理服务器就必须打开两个连接，一个为对外的连接、一个为对内的连接，因此对于连接请求数量非常大的时候，代理服务器的负载也就非常之大了，最终反向代理服务器将成为系统的瓶颈。例如，使用Apache的mod_rproxy模块来实现负载平衡功能时，提供的并发连接数量受Apache本身的并发连接数量的限制。一般来讲，可以使用它来对连接数量不是特别大，但每次连接都需要消耗大量处理资源（即请求处理时间是影响服务响应时间的主要因素）的站点进行负载平衡，例如搜寻，或是注册等需要同数据库打交道的操作。

反向代理的优点是，可以将负载平衡和代理服务器的高速缓存技术结合在一起，提供有益的性能；并且外部客户不能直接访问真实服务器，可以提高系统的安全性；而且可以探测各个服务器的真实负载，以实现较好的负载平衡策略，让负载可以非常均衡地分给内部服务器，不会出现负载集中到某个服务器的偶然现象。

UCSB设计和实现的可升级Web服务器SWEB[6]（a Scalable Web Server）便是基于HTTP重定向的反向代理负载平衡。

3.3 在网络端进行地址映射的负载平衡技术

网络端负载平衡是指在URLs第二次映射时实现负载平衡，也就是在真实服务器与网络接口的前端放置一个负载平衡器（load balancer）拦截所有请求服务的IP包，在负载平衡器的网络协议栈中插入实现负载平衡的代码，在协议栈底层将负载分流到各个真实服务器上。网络端的地址映射可以做在两个层次上：网络层或网络层与链路层之间。

3.3.1 *在网络层进行的负载平衡技术* 在网络层进行负载分流要求负载平衡器拥有Web站点对外的IP地址，所有的真实服务器各自拥有不同的IP地址。所有访问站点的IP包必须首先到达负载平衡器，负载平衡器依据一定算法在网络层将请求分配到各个真实服务器。其拓扑结构如图2。网络层地址映射负载平衡系统中http请求的处理流程如下[7]：

·客户发送一个目的地址为A的http包；

·路由器把这个包发送到负载平衡器（IP address＝A）；

·负载平衡器根据连接状态表和负载平衡算法确定这个包应该被哪台真实服务器处理，例如，真实服务器2。然后，负载平衡器将该包的目的IP地址由A改为B2（真实服务器2的IP地址），重算该包的TCP和IP校验和，最后，把该包发送到真实服务器2；

·真实服务器2收到该包并处理后，将通过负载平衡器把应答包送回给客户，因为所有真实服务器都把负载平衡器作为自己的网关；

·负载平衡器收到来自真实服务器2的应答包后，将该包的源IP地址改为A（负载平衡器的IP地址），重算TCP和IP校验和，然后将该应答包发送给客户。

在网络层实现地址映射的优点在于，负载平衡同时对服务器和客户透明，而且真实服务器可以不受空间限制，分布在Internet的任何位置。但是，这种技术要求进出Web Cluster的网络流量都必须通过负载平衡器，对每个请求，负载平衡器都需要维护和查询其连接信息，并在转发时修改IP包和重算校验和，这样便增加了负载平衡器自身的负载，在网络流量太大的情况下（例如处理大尺寸静态页面），负载平衡器本身可能成为系统的瓶颈。

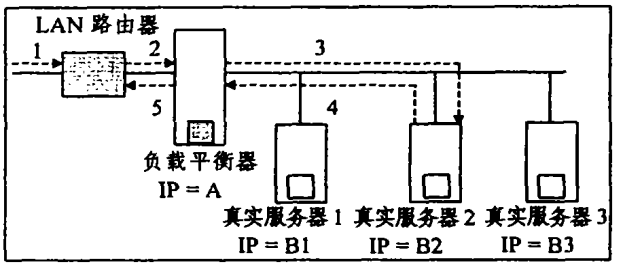

图2 在网络层进行的负载平衡流程图

Cisco的LocalDirector和UNL（University of Nebraska-Lincoln）的LSNAT[8]便采用了在网络层进行负载平衡。

3.3.2 *在网络层和链路层之间进行的负载平衡技术* 在网络层和链路层之间进行负载分流要求负载平衡器同所有的真实服务器共享同一个IP地址。所有访问站点的IP包必须首先到达负载平衡器，负载平衡器依据一定算法在网络层和链路层之间将请求分配到各个真实服务器。其拓扑结构如图3。网络层和链路层之间地址映射的负载平衡系统中http请求的处理流程[7]：

·客户发送一个目的地址为A的http包；

·路由器把这个包发送到负载平衡器（IP address＝A）；

·负载平衡器根据连接状态表和负载平衡算法确定这个包应该被哪台真实服务器处理，例如，真实服务器2。负载平衡器将该包的目的MAC地址由负载平衡器的MAC地址改为真实服务器2的MAC地址，最后，将该包发送到真实服务器2；

·真实服务器2收到该包后进行处理，并直接将应答包发送给客户，不再经过负载平衡器转发。

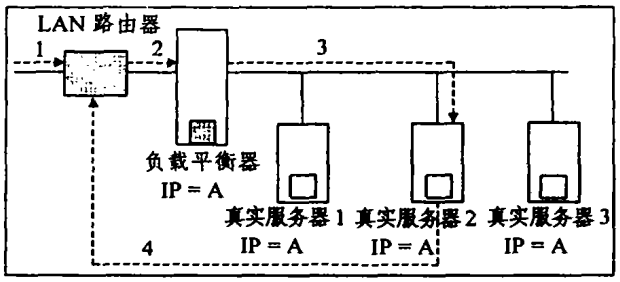

图3 在网络层和链路层之间进行的负载平衡的流程图

同在网络层进行的负载平衡相比，在网络层和链路层之间进行的优点在于，负载平衡器自身的负载较小，不易造成瓶颈。由于负载平衡器和所有真实服务器共享同一个IP地址，因此真实服务器无需做任何额外工作就可以直接将应答包发送回客户，而无需经过负载平衡器转发，这样便减少了负载平衡器一般的网络流量。同时，负载平衡器在分配负载时，只需修改MAC地址，不用对IP包做改动。这种方法的缺点在于，需要一定的策略来实现多机共享IP，并且，由于转发是MAC地址寻址，所以要求平衡器同所有的真实服务器必须有点到点的直接的物理连接。

IBM 的 eNetwork Dispatcher 和 UNL(University of Nebraska-Lincoln)的 LSMAC[6]便采用了在网络层和链路层之间进行负载平衡。

4. 负载平衡算法

服务器性能主要受三个因素的影响:网络带宽、服务器硬件性能(内存、CPU、I/O)和任务大小。根据评定负载的不同粒度,我们可以以请求为粒度计算负载,也可以以上述影响服务器性能的因素为粒度计算负载。通常有以下几种负载平衡算法。

4.1 Round-Robin 方法

Round-Robin 方法不考虑各个服务器上的实际负载情况,同等对待各个服务器,依次轮流向每个服务器分派客户请求。该方法简单,易实现,但是可能造成将很多工作量较大的请求分配到同一台服务器,而造成用户等待时间过长甚至服务器崩溃的情况。

4.2 最少活动连接数方法

最少活动连接数方法要求负载平衡器维护每台服务器的活动连接数,依次权衡各个服务器的负载,将最新的请求分配到具有最少活动连接数的服务器。这是一种动态的调度方法。当多个服务器具有相同的活动连接数时,最新请求被分配到序号(静态定义的服务器序列)最小的服务器。该方法避免了因"长"请求过于集中而造成个别服务器超负荷,但维护每个服务器的活动连接数将增大负载平衡器自身的负载。

4.3 最短服务时间方法

最短服务时间方法要求负载平衡器维护每个 TCP 连接的连接状态,对每个连接,计算它的服务响应时间(即向服务器传送请求包的第一个字节起到向客户传送应答包的最后一个字节止之间的时间),并为每个服务器计算在统一时间段内(average window)所有连接的平均服务响应时间。最新请求被分配到具有最短平均服务响应时间的服务器上。缺省的统一时间段通常设置为1秒。在每个统一时间段的末尾,将各个服务器的最短平均服务响应时间复位。当多个服务器具有相同的平均最短服务响应时间时,将最新请求分配给具有最少活动连接数的服务器。这是一种比较精确衡量负载的方法,但过多的计算加大了负载平衡器自身的负载。

4.4 最少网络流量方法

最少网络流量方法与最短服务时间方法类似,也设置一个统一时间段,负载平衡器记录每个服务器在当前统一时间段内所传输的总的字节数,据此对请求进行负载平衡。当多个服务器在当前统一时间段内传输了同样多的字节数时,同样使用活动连接数来进一步进行负载分配。其优缺点也与最小服务响应时间类似。

4.5 区别不同服务类型的负载平衡算法

不同的服务器类型对服务器资源的利用率不同,例如,处理静态文档占用更多的网络带宽资源,而处理动态文档占用更多的 CPU 资源。上述负载平衡算法不同服务类型带来的差异,这在大多数条件下是适合的。然而,更精确的负载平衡算法,可以区分不同的服务类型的负载,从 CPU、disk I/O 及网络 I/O 等多方面分别监视服务器负载,使负载平衡更趋于均衡。这种算法的不足在于,为了区分不同的服务,要求负载平衡器对每一请求都要探测上层协议(如 HTTP 协议)的内容,这样势必加重负载平衡器自身的负载。

5. Web 服务器群集负载平衡的实现方法

对于第3节中研究的各种负载平衡技术,智能客户端、HTTP 重定向和反向代理技术在实际应用中用得较少。DNS 负载平衡被广泛使用,但由于其嵌入在 DNS 域名服务器中,很难适应特定应用的灵活多变。目前常用的方法是在网络端进行负载平衡。这种负载平衡技术的拓扑结构决定了外部网络无法直接访问 Web 服务器,在 Web 站点的管理过程中,可以把负载平衡、网络安全等一系列问题考虑进去;同时,这种灵活多变的网络结构易于将其移植到各种具体应用中去,以更好地符合特定应用的要求。因此,本部分重点分析在网络端进行负载平衡的各种实现方法。

在网络端进行的负载平衡技术,将整个负载平衡的功能集中到 Web 服务器群集前端的负载平衡器上,因此,以下的方法便是围绕对负载平衡器的设计进行的。我们实现的负载平衡器分为三个模块:核心请求分流模块、虚拟核心设备驱动管理模块和系统监控模块。

核心请求分流模块是系统中最主要的模块,完成对用户请求的分流与应答,即实现负载平衡的核心功能;虚拟核心设备驱动管理模块,通过虚拟设备驱动程序来对分流模块进行配置和管理;系统监控模块,对系统各服务器进行监控,以处理突发的错误,维护系统的稳定性。根据进行负载平衡的不同层次,核心请求分流模块的实现可以分为两类:在网络层进行和在网络层和链路层之间进行。

5.1 在网络层进行的负载平衡实现

在网络层进行的负载平衡通常采用 NAT(Network Address Translation)技术,在负载平衡器中维护一个连接状态表,对所有进入系统的包进行规则匹配。连接状态表包含以下一些信息:源 IP、源端口号、目的 IP、目的端口号、协议类型、服务器 IP、服务器端口号。主要步骤如下:

·负载平衡器收到客户的 IP 包,首先查询连接状态表:(1)若已有该连接的表项,则用表项中保存的服务器 IP 和服务器端口号替换 IP 包的目的 IP 和目的端口号;(2)若无该连接的表项,便为该连接分配一个新的连接状态表项(调用负载平衡算法,为该请求分配一个合适的真实服务器,获取该 IP 包的源 IP、源端口号、目的 IP、目的端口号、协议类型和被选中的真实服务器的 IP 和端口号,将上述信息填入新表项),用选中服务器的 IP 和端口号替换 IP 包的目的 IP 和目的端口号。然后,重算校验和,转发包。

·负载平衡器收到真实服务器的 IP 包,查询连接状态表,用保存的目的 IP 和目的端口号替换 IP 包的源 IP 和源端口号,重算校验和,转发包。

在实现过程中还存在两个难点:(1)为连接状态表设计合理的数据结构和查询方法,例如高效的 hash 算法,这是决定此类负载平衡系统性能的重要因素之一;(2)确定连接状态表项过期的方法,维护 TCP 连接状态是较精确的方法,但繁琐、复杂,对 CPU 资源的消耗大;更好的做法是在简化的连接状态中辅以定时器,过期即废弃表项,但对阈值的选择必须合理。

5.2 在网络层和链路层之间进行的负载平衡实现

这种技术的实现通常是在协议栈中网络层和链路层之间插入一个模块,完成负载平衡功能。同前面的实现一样,需要维护一张连接状态表,表项内容包括:源 IP、源端口号、协议类型、服务器 MAC 地址。不同的是,在转发包之前,负载平衡

器要做的是修改包的目的 MAC 地址，而不对 IP 包做任何修改。

在具体实现中，需要考虑怎样实现多个服务器共享一个 IP 地址而又不产生 IP 冲突。一种方法是通过配置网络接口实现：

·将负载平衡器的第一 IP(primary IP)配置为 Web 站点的 IP；

·将真实服务器的第二 IP(secondary IP)配置为 Web 站点的 IP，并配置使服务器发包时都使用第二 IP；

·用静态 ARP 缓存入口配置最近的网关，使得所有访问 Web 站点的包都首先被发送到负载平衡器。

另一种方法是禁用 ARP 协议：

·禁用真实服务器的 ARP 协议，并在真实服务器 ARP 缓存中静态固化最近网关的 MAC 地址；

·对负载平衡器的 ARP 协议不做任何的修改。

5.3 系统健壮性问题

在负载平衡系统的实现中还应该考虑 Web 服务器和负载平衡器的容错问题。对于 Web 服务器的容错，可以通过在负载平衡器上运行一个监控程序，周期性向所有服务器发询问请求探测服务器是否活跃，当发现某服务器不再活跃时，立即将它从群集中删去，调整负载平衡器的负载策略，并通知系统管理员处理。

对负载平衡器的容错通常采用热备份的方式，在负载平衡器同备份机之间采用缓存一致性策略，如果备份机未能收到负载平衡器的定时讯息，则可认为负载平衡器出现故障，便自动取代负载平衡器完成负载平衡任务。

5.4 负载平衡下的 ASP 会话

在交互式 Web 页面服务中，ASP 越来越成为企业级网络应用程序的选择。ASP 要求服务器维护 session 状态，这同群集技术的要求相违背。因为对群集下真正的负载平衡来说，每当浏览一个新页面时，服务器都潜在地丢失用户的 session 信息。为了解决这个问题，可以采用完全不使用 session，使用临时 cookies，购买第三方组件来处理 session 管理或仅对 Web 范围内的第一次点击进行负载平衡等方法来解决。

结束语 近年来，Web 服务器群集技术在实现高性能 Web 服务器方面得到了广泛的关注。作为其实现中的关键部分，负载平衡得到了广泛的研究。本文在介绍各种负载平衡技术的基础上，着重讨论了在网络端进行地址映射的负载平衡技术和该类负载平衡器的实现方法。我们看到，对于不同的应

(下转第92页)

(上接第141页)

根结点到每个目录项的路径称为 DN(Distinguished Name)，如：uid=jbxu,ou=People,o=xtpu.org.cn。在 LDAP 的 C/S 模式中，一个 LDAP 客户可以方便地完成诸如在目录树中查找、添加、修改和删除目录项等操作，比如要完成修改目录树中的某个目录项，LDAP 客户先与 LDAP 服务器通过端口(如389)建立连接，然后向 LDAP 服务器提交要修改的目录项属性值的 DN，LDAP 服务器先使用 DN 在目录树中查找到该目录项，然后完成修改其属性值的操作。

4 目录检索服务应用

Netscape Directory Server 是一个基于 LDAP 的目录服务器，可以运行在 Solaris、HP-UX、Linux、NT 操作系统上。LDAP V3的 API 功能在 RFC2251中有详细的定义，Netscape 公司的 LDAP SDK for C 和 LDAP SDK for Java 很好地实现了 RFC2251的功能。使用 LDAP SDK 提供的 API 函数，我们能开发自己的 Client 程序，完成诸如检索目录数据库的目录项，向目录数据库中添加、修改、删除和重命名目录项等操作，当然也能根据检索到目录项的属性值完成诸如身份认证这样的应用。由于 Netscape LDAP SDK for C 和 LDAP SDK for Java 完全建构在 TCP/IP 上，这样可以在支持 TCP/IP 的 UNIX、Linux、Window 等各种平台下开发客户端应用程序。下面应用 LDAP SDK for C 的几个重要的 API 函数，给出一个简单的目录项检索实例：

```
#include "ldap.h"
#define HOSTNAME "www.xtpu.edu.cn"
#define PORT 389
#define FIND_DN "uid=jbxu,ou=People,o=xtpu.org.cn"
int main( int argc,char * *argv )
{ LDAP *ld;
  LDAPMessage *result, *e;
  BerElement *ber;
  char *a, * *vals;
  ld=ldap_init( HOSTNAME,PORT)
    /* 获取一个到 LDAP Server 连接的句柄 */;
  ldap_simple_bind_s( ld,NULL,NULL );
    /* 绑定一个匿名到 LDAP server */
    /* 查找 LDAP Server 中的数据 */
  if ((ldap_search_ext_s( ld,FIND_DN,LDAP_SCOPE_BASE,
    "(objectclass = *)", NULL, 0, NULL, NULL, LDAP_NO_
      LIMIT,
    LDAP_NO_LIMIT,&result ))!= LDAP_SUCCESS ){
    perror("查找失败"); return( 1 ); }
  if (ldap_first_entry( ld,result )!= NULL ){
    printf( "找到%s:\n",FIND_DN );
    /* 在检索出的目录项中循环输出属性名和值 */
    for (a = ldap_first_attribute( ld,e,&ber ); a != NULL;
      a= ldap_next_attribute( ld,e,ber )){
     if ((vals = ldap_get_values( ld,e,a))!= NULL ){
       for (int i = 0; vals[i] != NULL; i ++ ){
         printf( "%s: %s\n",a,vals[i] );
     }ldap_value_free( vals );
    }ldap_memfree( a );
   }if ( ber != NULL ){ ber_free( ber,0 ); }
  }ldap_msgfree( result ); ldap_unbind( ld ); return( 0 );
}
```

如果在 UNIX 平台上编译和连接上述客户程序，请确保设置 LD_LIBRARY_PATH 路径变量指向库文件 libldap41.so 的位置。如果在 NT 平台连接客户程序，请将动态链接库 nsldap32v41.dll 拷到 winnt\system32目录中。

参考文献

1 Authentication Methods for LDAP. URL: ftp://ftp.isi.edu/in-notes/rfc2829.txt

2 Lightweight Directory Access Protocol (v3). URL: ftp://ftp.isi.edu/in-notes/rfc2830.txt

3 OpenLDAP Root Service. URL: ftp://ftp.isi.edu/in-notes/rfc3088.txt

4 Storing Vendor Information in the LDAP root DSE. URL: ftp://ftp.isi.edu/in-notes/rfc3045.txt

5 LDAP Authentication Password Schema. URL: ftp://ftp.isi.edu/in-notes/rfc3112.txt

6 Use of Language Codes in LDAP. URL: ftp://ftp.isi.edu/in-notes/rfc2596.txt

链路时就更新 Hash 表的内容。也可通过手工设置路由表来改变网络中节点路由配置。

为了说明路由表的结构，我们采用图10所求的拓扑结构，节点之间的链路用 VLINK 标记。表1中显示了节点1和节点4的路由表结构。在表中节点的 IP 地址和链路的端口分配是任意的。图11是节点1上的用户界面，节点上使用 Windows NT 作为操作系统，采用 Java 语言编程实现。

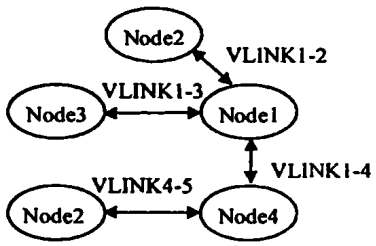

图10 模拟实验网络拓扑

表1 模拟实验网络系统中节点1和节点4的路由表

NODE1		NODE2	
Key	Vlaue	Key	Vlaue
Node2 128.6.43.52:2000	VLINK1-2	Node1 128.6.43.20:6000	VLINK1-4
Node3 128.6.30.3:4000	VLINK1-3	Node2 128.6.43.52:2000	VLINK2-4
Node4 128.6.21.18:3000	VLINK1-4	Node3 128.6.30.3:4000	VLINK3-4
Node5 128.6.21.19:3000	VLINK1-5	Node5 128.6.21.19:3000	VLINK5-4

RANI 主动网络基于菜单方式实现网络的编程，为用户提供一个通信应用程序和包处理的统一环境，信任操作可以允许用户调用新的服务程序或更新一个已存在的服务程序。通过图形界面可以使用户方便地进行设置和操作，可以通过重新启动服务来适应节点的拓扑结构改变。但由于 RANI 将主动节点建立在 TCP/IP 协议栈之上，因此使得对主动包的处理速度较慢。此外，由于假定网络可靠、链路稳定、路由表的静态性，而这些情况都与实际有差别的。

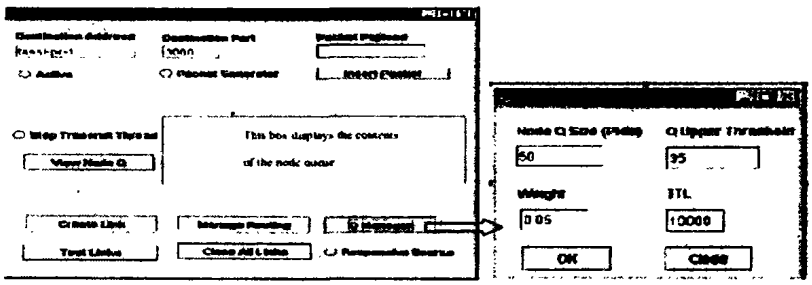

图11 在节点1上对 RANI 测试的用户界面

主动网络是一种崭新的网络结构，它采用的主动技术涉及到编译技术、操作系统、网络技术等各个方面。它使得网络可以动态地配置和动态控制，极大地提高了网络的性能并增加了网络的灵活性和扩展性，并为宽带网络的发展提供了广阔的前景。目前虽然还没有实用的产品推出，但是它得到了广泛的关注，现在正在对其关键性的技术如路由、资源分配、安全性、开发语言和平台等进行研究，可以相信，主动网络已经开始改变了传统网络的概念，它对未来技术的影响将起非常重要的推动作用。

参考文献

1 Hicks M, et al. PLAN Programmer's Guide. University of Pennsylvania, May 2000

2 Wetherall D, et al. Introducing New Internet Services: Why and How. IEEE Networks Magazine, May/June 1998

3 Moore O T, Nettles S M. Towards Practical Programmable Packets. Technical Report MS-CIS-00-12, University of Pennsylvania, May 2000

4 Schwartz B, et al. Smart Packets: Applying Active Networks to Network Management. ACM Trans. on Computer Systems, 2000, 18(1): 67～88

5 Alexander D S, et al. The SwitchWare Active Network Architecture. IEEE Network Magazine, 1998, 12(3): 29～36

6 Braden B, et al. Introduction to the ASP execution environment: [Technical report, USC/Information Science Institute]. Feb. 2000

7 Scott Alexander D, et al. The SwitchWare Active Network Architecture. IEEE Network Magazine, 1998, 12(3): 29～36

(上接第110页)

用类型，所采取的负载平衡策略可能会有很大的差异，因此在具体实现中，应充分考虑应用的特殊性，只有选择和设计最适合特定应用的方案，才能获得负载平衡的最佳效果。从这个角度来说，并不存在一种统一的最佳方案。随着实际需求中对负载平衡效率和精确度的更高要求，负载平衡硬件化的趋势也会越来越大。

参考文献

1 Brewer E. Clusters: Multiply and Conquer. Data Communications. Jul. 1997

2 Fox A, Gribble S, Chawathe Y, Brewer E A. Cluster-Based Scalable Network Services. In: Proc. of SOSP '97, St. Malo, France, Oct. 1997

3 Bryhni H, Klovning E, Kure Q. A Comparison of Load Balance Techniques for Scalable Web Servers. IEEE Network, Jul.-Aug. 2000. 58～64

4 Mosedale D, Foss W, McCool R. Lessons Learned Administering Netscape's Internet Site. IEEE Internet Comp., Mar.-Apr. 1997. 28～35

5 Cisco-Scaling the Internet Web Server. http://www.cisco.com. Jun. 2000

6 Andresen D, et al. SWEB: Towards a Scalable WWW Server on MultiComputers. In: Proc. 10th Int'l. Parallel Processing Symp., HI, Apr. 1996

7 Schroeder T, Goddard S, Ramamurthy B. Scalable Web Server Clustering Technologies. IEEE Network, May-Jun. 2000. 38～45

8 Gan X, Schroeder T, Goddard S, Ramamurthy B. LSMAC vs. LSNAT: Scalable Cluster-based Web Servers. Cluster Computing: the Journal of Networks, Software Tools and Applications, 2000, 3(3): 175～185