负载调度算法

负载均衡（Load Balance），又称为负载分担，就是将负载（工作任务）进行平衡、分摊到多个操作单元上进行执行，例如Web服务器、FTP服务器、企业关键应用服务器和其它关键任务服务器等，从而共同完成工作任务。负载均衡建立在现有网络结构之上，它提供了一种廉价又有效的方法来扩展网络设备和服务器的带宽、增加吞吐量、加强网络数据处理能力、提高网络的灵活性和可用性。

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，称之为VS/NAT技术。在分析VS/NAT的缺点和网络服务的非对称性的基础上，提出通过IP隧道实现虚拟服务器的方法VS/TUN，和通过直接路由实现虚拟服务器的方法VS/DR，它们可以极大地提高系统的伸缩性。

在内核中的连接调度算法上，IPVS实现了以下几种调度算法：

# 1 轮叫调度

## 1.1 轮叫调度含义

轮叫调度（Round Robin Scheduling）算法就是以轮叫的方式依次将请求调度不同的服务器，即每次调度执行i = (i + 1) mod n，并选出第i台服务器。算法的优点是其简洁性，它无需记录当前所有连接的状态，所以它是一种无状态调度。

轮叫是基站为终端分配带宽的一种处理流程，这种分配可以是针对单个终端或是一组终端的。为单个终端和一组终端连接分配带宽，实际上是定义带宽请求竞争机制，这种分配不是使用一个单独的消息，而是上行链路映射消息中包含的一系列分配机制。

## 1.2 轮叫调度算法流程

轮询调度算法的原理是每一次把来自用户的请求轮流分配给内部中的服务器，从1开始，直到N（内部服务器个数），然后重新开始循环。在系统实现时，我们引入了一个额外条件，即当服务器的权值为零时，表示该服务器不可用而不被调度。这样做的目的是将服务器切出服务（如屏蔽服务器故障和系统维护），同时与其他加权算法保持一致。所以，算法要作相应的改动，它的算法流程如下：

假设有一组服务器S = {S0, S1, …, Sn-1}，一个指示变量i表示上一次选择的服务器，W(Si)表示服务器Si的权值。变量i被初始化为n-1，其中n > 0。

　　j = i;

　　do {

　　j = (j + 1) mod n;

　　if (W(Sj) > 0) {

　　i = j;

　　return Si;

　　}

　　} while (j != i);

　　return [NULL](http://baike.soso.com/v126332.htm?ch=ch.bk.innerlink" \t "_blank);

轮叫调度算法假设所有服务器处理性能均相同，不管服务器的当前连接数和[响应速度](http://baike.soso.com/v324827.htm?ch=ch.bk.innerlink" \t "_blank)。该算法相对简单，不适用于服务器组中处理性能不一的情况，而且当请求服务时间变化比较大时，轮叫调度算法容易导致服务器间的负载不平衡。

## 1.3 轮叫调度逻辑实现

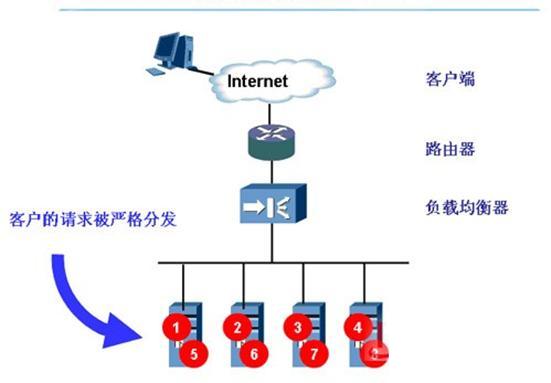


图1.3.1 轮叫调度逻辑实现图示

轮询调度算法假设所有的处理性能都相同，不关心每台服务器的当前连接数和响应速度。当请求服务间隔时间变化比较大时，轮询调度算法容易导致服务器见的负载不均衡。所以此种均衡算法适合于服务器组中的所有服务器都有相同的软硬件配置并且平均请求相对均衡的情况。

# 2 加权轮叫调度

## 2.1 加权轮叫调度含义

由于每台服务器的配置、安装的业务应用等不同，其处理能力会不一样。所以，我们根据服务器的不同处理能力，给每个服务器分配不同的权值，使其能够接受相应权值数的服务请求。

加权轮叫调度（Weighted Round-Robin Scheduling）算法可以解决服务器间性能不一的情况，它用相应的权值表示服务器的处理性能，服务器的缺省权值为1。假设服务器A的权值为1，B的权值为2，则表示服务器B的处理性能是A的两倍。加权轮叫调度算法是按权值的高低和轮叫方式分配请求到各服务器。权值高的服务器先收到的连接，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。

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。

## 2.2 加权轮叫调度算法流程

　假设有一组服务器S = {S0, S1, …, Sn-1}，W(Si)表示服务器Si的权值，一个指示变量i表示上一次选择的服务器，指示变量cw表示当前调度的权值，max(S)　　表示集合S中所有服务器的最大权值，gcd(S)表示集合S中所有服务器权值的最大　　公约数。变量i和cw最初都被初始化为零。

　　while (true) {

　　 if (i == 0) {

　　 cw = cw - gcd(S);

　　 if (cw <= 0) {

　　 cw = max(S);

　　 if (cw == 0)

　　 return NULL;

　　 }

　　 } else i = (i + 1) mod n;

　　 if (W(Si) >= cw)

　　 return Si;

　　}

例如，有三个服务器A、B和C分别有权值4、3和2，则在一个调度周期内(mod sum(W(Si)))调度序列为AABABCABC。加权轮叫调度算法还是比较简单和高效。当请求的服务时间变化很大，单独的加权轮叫调度算法依然会导致服务器间的负载不平衡。

从上面的算法流程中，我们可以看出当服务器的权值为零时，该服务器不被被调度；当所有服务器的权值为零，即对于任意i有W(Si)=0，则没有任何服务器可用，算法返回NULL，所有的新连接都会被丢掉。加权轮叫调度也无需记录当前所有连接的状态，所以它也是一种无状态调度。

## 2.3 加权轮叫调度逻辑实现

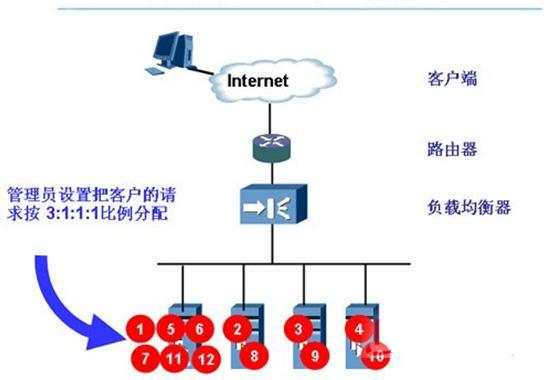
这种算法的逻辑实现如图2.3.2所示，图中我们假定四台服务器的处理能力为3:1:1:1。

图2.3.1 加权轮叫调度实现逻辑图示

由于权重轮询调度算法考虑到了不同服务器的处理能力，所以这种均衡算法能确保高性能的服务器得到更多的使用率，避免低性能的服务器负载过重。所以，在实际应用中比较常见。轮询调度算法以及权重轮询调度算法的特点是实现起来比较简洁，并且实用。目前几乎所有的负载均衡设备均提供这种功能。

# 3 最少连接调度

## 3.1 最少连接调度含义

在实际情况中，客户端每一次请求服务在服务器停留的时间可能会有较大的差异。随着工作时间的延伸，如果采用简单的轮叫或随机均衡算法，每一台服务器上的连接进程数目可能会产生极大的不同，这样实际上并没有达到真正的负载均衡。而最少连接数均衡算法对内部需负载的每一台服务器都有一个数据记录，记录当前该服务器正在处理的连接数量。当有新的服务连接请求时，将把当前请求分配给连接数最少的服务器，使均衡更加符合实际情况，负载更加均衡。

最少连接调度（Least-Connection Scheduling）算法是把新的连接请求分配到当前连接数最小的服务器。最小连接调度是一种动态调度算法，它通过服务器当前所活跃的连接数来估计服务器的负载情况。

与轮叫调度算法想反，最小连接调度是一种动态调度算法，它通过服务器当前所活跃的连接数来估计服务器的负载情况。调度器需要记录各个服务器已建立连接的数目，当一个请求被调度到某台服务器，其连接数加1;当连接中止或超时，其连接数减一。此种均衡算法适合长时处理的请求服务。

如果集群系统的真实服务器具有相近的系统性能，采用最少连接调度算法可以较好地均衡负载。

## 3.2 最少连接调度算法流程

在系统实现时，我们也引入当服务器的权值为零时，表示该服务器不可用而不被调度，它的算法流程如下：

假设有一组服务器S = {S0, S1, ..., Sn-1}，W(Si)表示服务器Si的权值，C(Si)表示服务器Si的当前连接数。

for (m = 0; m < n; m++) {

if (W(Sm) > 0) {

for (i = m+1; i < n; i++) {

if (W(Si) <= 0)

continue;

if (C(Si) < C(Sm))

m = i;

}

return Sm;

}

}

return NULL;

当各个服务器有相同的处理性能时，最小连接调度算法能把负载变化大的请求分布平滑到各个服务器上，所有处理时间比较长的请求不可能被发送到同一台服务器上。但是，当各个服务器的处理能力不同时，该算法并不理想，因为TCP连接处理请求后会进入TIME\_WAIT状态，TCP的TIME\_WAIT一般为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，而性能低的服务器已经忙于处理所收到的连接，还不断地收到新的连接请求。

# 4 加权最少连接调度

## 4.1 加权最少连接调度含义

加权最小连接调度（Weighted Least-Connection Scheduling）算法是在做少连接数调度算法的基础上，根据服务器的不同处理能力，给每个服务器分配不同的权值，使其能够接受相应权值数的服务请求，是在最少连接数调度算法的基础上的改进。

它是最小连接调度的超集，各个服务器用相应的权值表示其处理性能。服务器的缺省权值为1，系统管理员可以动态地设置服务器的权值。加权最小连接调度在调度新连接时尽可能使服务器的已建立连接数和其权值成比例。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。

## 4.2 加权最少连接调度算法流程

假设有一组服务器S = {S0, S1, ..., Sn-1}，W(Si)表示服务器Si的权值，C(Si)表示服务器Si的当前连接数。所有服务器当前连接数的总和为

CSUM = ΣC(Si) (i=0, 1, .. , n-1)

当前的新连接请求会被发送服务器Sm，当且仅当服务器Sm满足以下条件

(C(Sm) / CSUM)/ W(Sm) = min { (C(Si) / CSUM) / W(Si)} (i=0, 1, . , n-1)，其中W(Si)不为零

因为CSUM在这一轮查找中是个常数，所以判断条件可以简化为

C(Sm) / W(Sm) = min { C(Si) / W(Si)} (i=0, 1, . , n-1) 其中W(Si)不为零

因为除法所需的CPU周期比乘法多，且在Linux内核中不允许浮点除法，服务器的权值都大于零，所以判断条件C(Sm) / W(Sm) > C(Si) / W(Si) 可以进一步优化为C(Sm)\*W(Si) > C(Si)\* W(Sm)。同时保证服务器的权值为零时，服务器不被调度。

wlc算法以下流程：

for (m = 0; m < n; m++) {

if (W(Sm) > 0) {

for (i = m+1; i < n; i++) {

if (C(Sm)\*W(Si) > C(Si)\*W(Sm))

m = i;

}

return Sm;

}

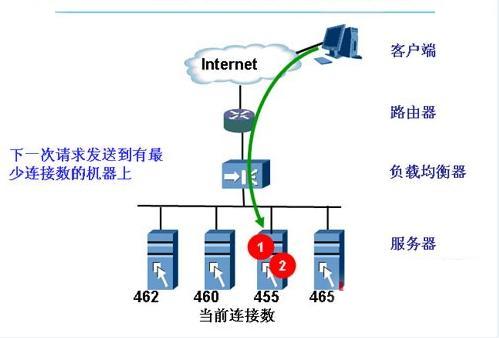
}

return NULL;

## 4.3 加权最少连接调度逻辑实现

这种算法的逻辑实现如图4.3.1和图4.3.2所示：

图4.3.1显示在在T时刻，一个新请求到来时，负载均衡设备的响应情况：

图4.3.1 (加权)最少连接调度逻辑示意图

当经过t时间后，新的请求到来时，负载均衡设备的调度情况变化结果如下

图4.3.2 经过t时间后(加权)最少连接调度逻辑示意图

权重最少连接数调度算法特点是实现起来比较简洁，在大多数情况下非常有效，在很多产品，如ArrayNetWorks等公司的产品中均有实现。

# 5 基于局部的最少连接

## 5.1 基于局部的最少连接调度含义

基于局部的最少连接调度（Locality-Based Least Connections Scheduling）算法先根据请求的目标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其一半的工作负载，则用“最少连接”的原则选出一个可用的服务器，将请求发送到该服务器。

LBLC调度算法是针对请求报文的目标IP地址的负载均衡调度，目前主要用于Cache集群系统，因为在Cache集群中客户请求报文的目标IP地址是变化的。这里假设任何后端服务器都可以处理任一请求，算法的设计目标是在服务器的负载基本平衡情况下，将相同目标IP地址的请求调度到同一台服务器，来提高各台服务器的访问局部性和主存Cache命中率，从而整个集群系统的处理能力。

## 5.2 基于局部的最少连接调度算法流程

假设有一组服务器S = {S0, S1, ..., Sn-1}，W(Si)表示服务器Si的权值，C(Si)表示服务器Si的当前连接数。ServerNode[dest\_ip]是一个关联变量，表示目标IP地址所对应的服务器结点，一般来说它是通过Hash表实现的。WLC(S)表示在集合S中的加权最小连接服务器，即前面的加权最小连接调度。Now为当前系统时间。

if (ServerNode[dest\_ip] is NULL) then {

n = WLC(S);

if (n is NULL) then return NULL;

ServerNode[dest\_ip].server = n;

} else {

n = ServerNode[dest\_ip].server;

if ((n is dead) OR

(C(n) > W(n) AND

there is a node m with C(m) < W(m)/2))) then {

n = WLC(S);

if (n is NULL) then return NULL;

ServerNode[dest\_ip].server = n;

}

}

ServerNode[dest\_ip].lastuse = Now;

return n;

此外，对关联变量ServerNode[dest\_ip]要进行周期性的垃圾回收（Garbage Collection），将过期的目标IP地址到服务器关联项进行回收。过期的关联项是指哪些当前时间（实现时采用系统时钟节拍数jiffies）减去最近使用时间超过设定过期时间的关联项，系统缺省的设定过期时间为24小时。

# 6 带复制的基于局部性最少连接调度

## 6.1 带复制的基于局部性最少连接含义

带复制的基于局部性最少连接调度（Locality-Based Least Connections with Replication Scheduling，以下简称为LBLCR）算法先根据请求的目标IP地址找出该目标IP地址对应的服务器组；按“最小连接”原则从该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”原则从整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。

LBLCR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。对于一个“热门”站点的服务请求，一台Cache 服务器可能会忙不过来处理这些请求。这时，LBLC调度算法会从所有的Cache服务器中按“最小连接”原则选出一台Cache服务器，映射该“热门”站点到这台Cache服务器，很快这台Cache服务器也会超载，就会重复上述过程选出新的Cache服务器。这样，可能会导致该“热门”站点的映像会出现在所有的Cache服务器上，降低了Cache服务器的使用效率。LBLCR调度算法将“热门”站点映射到一组Cache服务器（服务器集合），当该“热门”站点的请求负载增加时，会增加集合里的Cache服务器，来处理不断增长的负载；当该“热门”站点的请求负载降低时，会减少集合里的Cache服务器数目。这样，该“热门”站点的映像不太可能出现在所有的Cache服务器上，从而提供Cache集群系统的使用效率。

## 6.2 带复制的基于局部性最少连接算法流程

假设有一组服务器S = {S0, S1, ..., Sn-1}，W(Si)表示服务器Si的权值，C(Si)表示服务器Si的当前连接数。ServerSet[dest\_ip]是一个关联变量，表示目标IP地址所对应的服务器集合，一般来说它是通过Hash表实现的。WLC(S)表示在集合S中的加权最小连接服务器，即前面的加权最小连接调度；WGC(S)表示在集合S中的加权最大连接服务器。Now为当前系统时间，lastmod表示集合的最近

修改时间，T为对集合进行调整的设定时间。

if (ServerSet[dest\_ip] is NULL) then {

n = WLC(S);

if (n is NULL) then return NULL;

add n into ServerSet[dest\_ip];

} else {

n = WLC(ServerSet[dest\_ip]);

if ((n is NULL) OR

(n is dead) OR

(C(n) > W(n) AND

there is a node m with C(m) < W(m)/2))) then {

n = WLC(S);

if (n is NULL) then return NULL;

add n into ServerSet[dest\_ip];

} else

if (|ServerSet[dest\_ip]| > 1 AND

Now - ServerSet[dest\_ip].lastmod > T) then {

m = WGC(ServerSet[dest\_ip]);

remove m from ServerSet[dest\_ip];

}

}

ServerSet[dest\_ip].lastuse = Now;

if (ServerSet[dest\_ip] changed) then

ServerSet[dest\_ip].lastmod = Now;

return n;

此外，对关联变量Se verSet[dest\_ip]也要进行周期性的垃圾回收（Garbage Collection），将过期的目标IP地址到服务器关联项进行回收。过期的关联项是指哪些当前时间（实现时采用系统时钟节拍数jiffies）减去最近使用时间（lastuse）超过设定过期时间的关联项，系统缺省的设定过期时间为24小时。

# 7 目标地址散列调度

## 7.1 目标地址散列调度含义

目标地址散列调度（Destination Hashing Scheduling）算法也是针对目标IP地址的负载均衡。但它是一种静态映射算法，即通过一个散列（Hash）函数将一个目标IP地址映射到一台服务器。

目标地址散列调度算法先根据请求的目标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。

## 7.1 目标地址散列调度算法流程

假设有一组服务器S = {S0, S1, ..., Sn-1}，W(Si)表示服务器Si的权值，C(Si)表示服务器Si的当前连接数。ServerNode[]是一个有256个桶（Bucket）的Hash表，一般来说服务器的数目会运小于256，当然表的大小也是可以调整的。算法的初始化是将所有服务器顺序、循环地放置到ServerNode表中。若服务器的连接数目大于2倍的权值，则表示服务器已超载。

n = ServerNode[hashkey(dest\_ip)];

if ((n is dead) OR

(W(n) == 0) OR

(C(n) > 2\*W(n))) then

return NULL;

return n;

在实现时，我们采用素数乘法Hash函数，通过乘以素数使得散列键值尽可能地达到较均匀的分布。所采用的素数乘法Hash函数如下：

static inline unsigned hashkey(unsigned int dest\_ip)

{

return (dest\_ip\* 2654435761UL) & HASH\_TAB\_MASK;

}

其中，2654435761UL是2到2^32 (4294967296)间接近于黄金分割的素数， (sqrt(5) - 1) / 2 = 0.618033989

2654435761 / 4294967296 = 0.618033987

# 8 源地址散列调度

## 8.1 源地址散列调度算法简介

源地址散列调度（Source Hashing Scheduling）算法正好与目标地址散列调度算法相反，它根据请求的源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。这里我们设定某个服务器的连接数目大于2倍的权值，则表示此服务器已超载。它采用的散列函数与目标地址散列调度算法的相同。可以看出，这种方式和目标地址散列调度方法是类似的，唯一的区别是以源地址作为散列键。

在实际应用中，源地址散列调度和目标地址散列调度可以结合使用在防火墙集群中，它们可以保证整个系统的唯一出入口。

# 9 最短期望延迟调度

## 9.1 最短期望延迟调度含义

最短期望延迟调度(Shortest Expected Delay Scheduling)算法为服务器分配预计最短延迟的网络连接。SED算法主要是对处理时间相对比较长的网络服务，不再考虑非活动连接数。

SED调度和无需等待调度是最新版IPVS软件包中新增的调度算法。

## 9.1 最短期望延迟调度算法流程

Sed算法基于WLC算法。举例来说：ABC三台机器分别权重123 ，连接数也分别是123。那么如果使用WLC算法的话一个新请求进入时它可能会分给ABC中的任意一个。使用sed算法后会进行这样一个运算：

A:(1+1)/1

B:(1+2)/2

C:(1+3)/3

根据运算结果，把连接交给C。

# 10无需等待调度

## 10.1无需等待调度含义

无须等待调度（Never Queue Scheduling）采取双速模式：

1. 当有空闲服务器可用时，作业将被发送到空闲服务器，而不是等待速度较快的那一台。
2. 若没有空闲服务器可用时，作业将被发送到可用最小化预期延迟的那台服务器。

## 10.2 无需等待调度算法流程

NQ是对SED的改进，当新请求过来的时候不仅要取决于SED算法所得到的值，还要取决于Real Server上是否有活动连接。如果有台 realserver的连接数＝0就直接分配过去，不需要在进行sed运算。主要是对处理时间相对比较长的网络服务。