## Nginx基本知识点

nginx支持最大的并发量有5w，超过时，可以采用集群。

一般对负载均衡的使用是随着网站规模的提升根据不同的阶段来使用不同的技术。具体的应用需求还得具体分析，如果是中小型的Web应用，比如日PV小于1000万，用Nginx就完全可以了；如果机器不少，可以用DNS轮询，LVS所耗费的机器还是比较多的；大型网站或重要的服务，且服务器比较多时，可以考虑用LVS。

一种是通过硬件来进行进行，常见的硬件有比较昂贵的F5和Array等商用的负载均衡器，它的优点就是有专业的维护团队来对这些服务进行维护、缺点就是花销太大，所以对于规模较小的网络服务来说暂时还没有需要使用；另外一种就是类似于Nginx/LVS/HAProxy的基于Linux的开源免费的负载均衡软件，这些都是通过软件级别来实现，所以费用非常低廉。

目前关于网站架构一般比较合理流行的架构方案：Web前端采用Nginx/HAProxy+Keepalived作负载均衡器；后端采用MySQL数据库一主多从和读写分离，采用LVS+Keepalived的架构。当然要根据项目具体需求制定方案。

## Nginx负载均衡策略

### 轮询（默认）

每个请求按照时间顺序逐一分配到不同的后端服务器，如果后端服务器down掉，能自动剔除。

upstream backserver {   
server 192.168.0.14;   
server 192.168.0.15;   
}

### 指定权重

指定轮询机率，weight和访问机率成正比，用于后端服务器性能不均的情况。

upstream backserver {   
server 192.168.0.14 weight=10;   
server 192.168.0.15 weight=10;   
}

### IP绑定ip\_hash

每个请求按访问ip的hash结果分配，这样每个访客固定访问一个后端服务器，可以解决session的问题。

upstream backserver {   
ip\_hash;   
server 192.168.0.14:88;   
server 192.168.0.15:80;   
}

### fair（第三方）

按后端服务器的响应时间来分配请求，响应时间短的优先分配。

upstream backserver {   
server server1;   
server server2;   
fair;   
}

### url\_hash

按访问的url的hash结果来分配请求，使每个url定向到同一个后端服务器，后端服务器为缓存时比较有效。

upstream backserver {   
server squid1:3128;   
server squid2:3128;   
hash $request\_uri;   
hash\_method crc32;   
}

## 四层负载和七层负载

四层也就是基于IP地址和端口的负载均衡。七层是基于URL等应用层信息的负载均衡。

同理，还有基于MAC地址的二层负载均衡和基于IP地址的三层负载均衡。 换句换说，二层负载均衡会通过一个虚拟MAC地址接收请求，然后再分配到真实的MAC地址；三层负载均衡会通过一个虚拟IP地址接收请求，然后再分配到真实的IP地址；四层通过虚拟IP+端口接收请求，然后再分配到真实的服务器；七层通过虚拟的URL或主机名接收请求，然后再分配到真实的服务器。

所谓的四到七层负载均衡，就是在对后台的服务器进行负载均衡时，依据四层的信息或七层的信息来决定怎么样转发流量。 比如四层的负载均衡，就是通过发布三层的IP地址（VIP），然后加四层的端口号，来决定哪些流量需要做负载均衡，对需要处理的流量进行NAT处理，转发至后台服务器，并记录下这个TCP或者UDP的流量是由哪台服务器处理的，后续这个连接的所有流量都同样转发到同一台服务器处理。七层的负载均衡，就是在四层的基础上（没有四层是绝对不可能有七层的），再考虑应用层的特征，比如同一个Web服务器的负载均衡，除了根据VIP加80端口辨别是否需要处理的流量，还可根据七层的URL、浏览器类别、语言来决定是否要进行负载均衡。举个例子，如果你的Web服务器分成两组，一组是中文语言的，一组是英文语言的，那么七层负载均衡就可以当用户来访问你的域名时，自动辨别用户语言，然后选择对应的语言服务器组进行负载均衡处理。

负载均衡器通常称为四层交换机或七层交换机。四层交换机主要分析IP层及TCP/UDP层，实现四层流量负载均衡。七层交换机除了支持四层负载均衡以外，还有分析应用层的信息，如HTTP协议URI或Cookie信息。

1、负载均衡分为L4 switch（四层交换），即在OSI第4层工作，就是TCP层啦。此种Load Balance不理解应用协议（如HTTP/FTP/MySQL等等）。例子：LVS，F5。

2、另一种叫做L7 switch（七层交换），OSI的最高层，应用层。此时，该Load Balancer能理解应用协议。例子：  haproxy，MySQL Proxy。

注意：上面的很多Load Balancer既可以做四层交换，也可以做七层交换。

## 负载均衡算法

考虑到服务请求的不同类型、服务器的不同处理能力以及随机选择造成的负载分配不均匀等问题，为了更加合理的把负载分配给内部的多个服务器，就需要应用相应的能够正确反映各个服务器处理能力及网络状态的**负载均衡算法**：

　　轮循均衡（Round Robin）：每一次来自网络的请求轮流分配给内部中的服务器，从1至N然后重新开始。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服务请求相对均衡的情况。

　　权重轮循均衡（Weighted Round Robin）：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。例如：服务器A的权值被设计成1，B的权值是 3，C的权值是6，则服务器A、B、C将分别接受到10%、30％、60％的服务请求。此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。

　　随机均衡（Random）：把来自网络的请求随机分配给内部中的多个服务器。

　　权重随机均衡（Weighted Random）：此种均衡算法类似于权重轮循算法，不过在处理请求分担时是个随机选择的过程。

　　响应速度均衡（Response Time）：负载均衡设备对内部各服务器发出一个探测请求（例如Ping），然后根据内部中各服务器对探测请求的最快响应时间来决定哪一台服务器来响应客户端的服务请求。此种均衡算法能较好的反映服务器的当前运行状态，但这最快响应时间仅仅指的是负载均衡设备与服务器间的最快响应时间，而不是客户端与服务器间的最快响应时间。

　　最少连接数均衡（Least Connection）：客户端的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，随着工作时间加长，如果采用简单的轮循或随机均衡算法，每一台服务器上的连接进程可能会产生极大的不同，并没有达到真正的负载均衡。最少连接数均衡算法对内部中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。此种均衡算法适合长时处理的请求服务，如FTP。

　　处理能力均衡：此种均衡算法将把服务请求分配给内部中处理负荷（根据服务器CPU型号、CPU数量、内存大小及当前连接数等换算而成）最轻的服务器，由于考虑到了内部服务器的处理能力及当前网络运行状况，所以此种均衡算法相对来说更加精确，尤其适合运用到第七层（应用层）负载均衡的情况下。

　　DNS响应均衡（Flash DNS）：在Internet上，无论是HTTP、FTP或是其它的服务请求，客户端一般都是通过域名解析来找到服务器确切的IP地址的。在此均衡算法下，分处在不同地理位置的负载均衡设备收到同一个客户端的域名解析请求，并在同一时间内把此域名解析成各自相对应服务器的IP地址（即与此负载均衡设备在同一位地理位置的服务器的IP地址）并返回给客户端，则客户端将以最先收到的域名解析IP地址来继续请求服务，而忽略其它的IP地址响应。在种均衡策略适合应用在全局负载均衡的情况下，对本地负载均衡是没有意义的。

　　尽管有多种的负载均衡算法可以较好的把数据流量分配给服务器去负载，但如果负载均衡策略没有对网络系统状况的检测方式和能力，一旦在某台服务器或某段负载均衡设备与服务器网络间出现故障的情况下，负载均衡设备依然把一部分数据流量引向那台服务器，这势必造成大量的服务请求被丢失，达不到不间断可用性的要求。所以良好的负载均衡策略应有对网络故障、服务器系统故障、应用服务故障的检**测方式和能力**：

　　Ping侦测：通过ping的方式检测服务器及网络系统状况，此种方式简单快速，但只能大致检测出网络及服务器上的操作系统是否正常，对服务器上的应用服务检测就无能为力了。

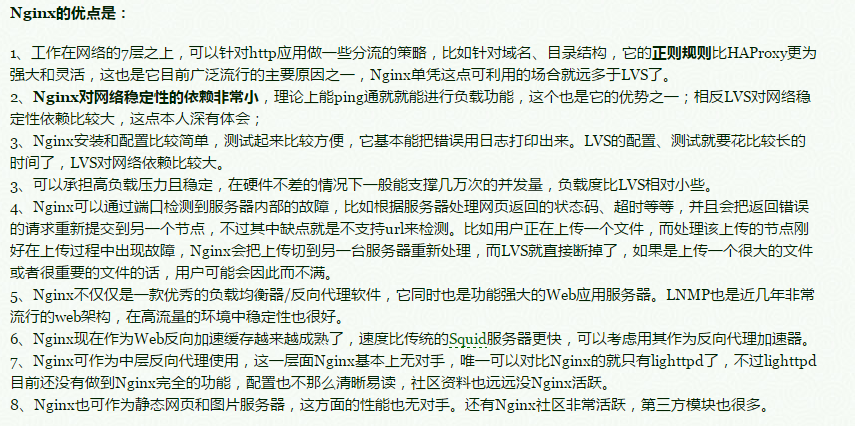
　　TCP Open侦测：每个服务都会开放某个通过TCP连接，检测服务器上某个TCP端口（如Telnet的23口，HTTP的80口等）是否开放来判断服务是否正常。

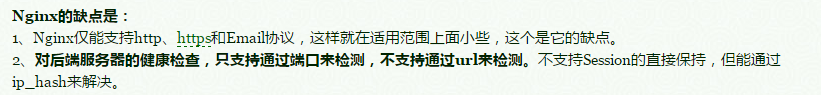
　　HTTP URL侦测：比如向HTTP服务器发出一个对main.html文件的访问请求，如果收到错误信息，则认为服务器出现故障。

　　负载均衡策略的优劣除受上面所讲的两个因素影响外，在有些应用情况下，我们需要将来自同一客户端的所有请求都分配给同一台服务器去负担，例如服务器将客户端注册、购物等服务请求信息保存的本地数据库的情况下，把客户端的子请求分配给同一台服务器来处理就显的至关重要了。有两种方式可以解决此问题，一是根据IP地址把来自同一客户端的多次请求分配给同一台服务器处理，客户端IP地址与服务器的对应信息是保存在负载均衡设备上的；二是在客户端浏览器 cookie内做独一无二的标识来把多次请求分配给同一台服务器处理，适合通过代理服务器上网的客户端。

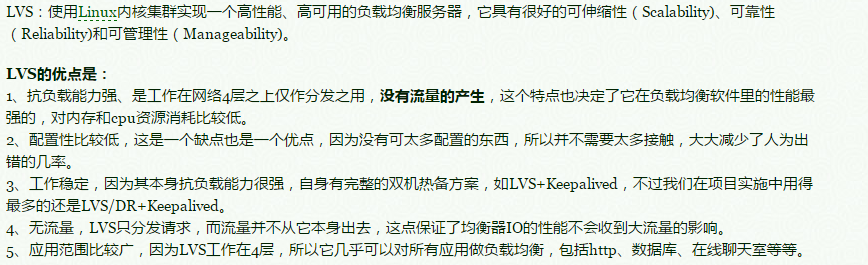
　　还有一种路径外返回模式（Out of Path Return），当客户端连接请求发送给负载均衡设备的时候，中心负载均衡设备将请求引向某个服务器，服务器的回应请求不再返回给中心负载均衡设备，即绕过流量分配器，直接返回给客户端，因此中心负载均衡设备只负责接受并转发请求，其网络负担就减少了很多，并且给客户端提供了更快的响应时间。此种模式一般用于HTTP服务器群，在各服务器上要安装一块虚拟网络适配器，并将其IP地址设为服务器群的VIP，这样才能在服务器直接回应客户端请求时顺利的达成三次握手。

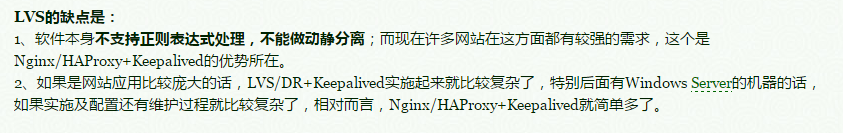
## Nginx 七层负载均衡





## LVS 四层负载均衡





## HAProxy 四层负载均衡

