**集群&负载均衡**

## 一、集 群

一组计算机系统构成一个松耦合的多处理器系统，它们之间通过网络实现进程间的通信，实现分布式计算。在客户端看来就像是只有一个服务器。集群可以利用多个计算机进行并行计算从而获得很高的计算速度，也可以用多个计算机做备份，从而使得任何一个机器坏了整个系统还是能正常运行。

## 集群分类

**1高可用集群（HA）：**高可用的含义是最大限度地可以使用。从集群的名字上可以看出，此类集群实现的功能是保障用户的**应用程序持久、不间断地提供服务**。

**2负载均衡集群：**分为前端负载调度和后端服务两个部分。负载调度部分负责把客户端的请求按照不同的策略分配给后端服务节点，而后端节点是真正提供应用程序服务的部分。与高可用集群不同的是，负载均衡集群中，**所有的后端节点都处于活动动态，它们都对外提供服务，分摊系统的工作负载**。

**3 科学计算集群（HPC）：**高性能计算集群。这类集群致力于提供单个计算机所不能提供的强大计算能力，包括数值计算和数据处理，并且倾向于追求综合性能。最快的速度、最大的存储、最庞大的体积、最昂贵的价格代表了超级计算的特点。

## 三、负载均衡

**1 负载均衡建立在现有网络结构之上**，它提供了一种廉价、有效、透明的方法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**。

**2 负载均衡分类：**

**1） DNS 负载均衡：**最早的负载均衡技术，在 DNS 中为多个地址配置同一个名字，因而查询这个名字的客户机将得到其中某一个地址，从而使得不同的客户访问不同的服务器，达到负载均衡的目的。不能区分服务器的差异，也不能反映服务器的当前运行状态。

**2） 代理服务器负载均衡：**使用代理服务器将请求转发给内部的服务器，可以提升静态网页的访问速度。使用代理服务器将请求均匀转发给多台服务器，从而达到负载均衡的目的。

**3） 地址转换网关负载均衡：**地址转换网关将一个外部 IP 地址映射为多个内部 IP 地址，对每次 TCP 连接请求动态使用其中一个内部地址，达到负载均衡的目的。

**4） 反向代理负载均衡**：普通代理方式是代理内部网络用户访问 internet 上服务器的连接请求，客户端必须指定代理服务器，并将本来要直接发送到 internet 上服务器的连接请求发送给代理服务器处理。反向代理（Reverse Proxy）方式是指以代理服务器来接受 internet 上的连接请求，然后将请求转发给内部网络上的服务器，并将从服务器上得到的结果返回给 internet 上请求连接的客户端，此时代理服务器对外就表现为一个服务器。

**3 负载均衡的实例（ IPTV）**

华灯初上的时候，打开电视机，点播一个你爱看的电影，电影顷刻间就开演了！你可否想过，和你一样点播电视节目的有千家万户，为什么每家每户想看的节目都能及时送到眼前呢？这是怎么做到的呢？

在电视上点播节目是一种 IPTV 业务，被点播的电影、电视剧等节目我们称为内容。向众多用户提供内容是由一个专门的网络完成的，这个网络我们叫它内容分发网络（CDN），我们将这个分发网络上负责给电视传送内容的服务器叫做 **CDN 节点**（CDN node）。

在网络建设之初、IPTV 用户还较少的时候，这个网络并不庞大，可能一台设备就能轻松处理完全部用户的要求。然而随着业务的发展，用户量和业务量与日俱增，一台设备已经招架不住了。这时我们就需要给这台设备找些帮手。于是我们为这台设备“克隆”一些兄弟，然后将用户的要求分摊给所有兄弟，大家一起共同面对。**对于这个网络来说，大家要完成的任务即是负载，如此多队员的合作以及团队运作，即是负载均衡**。

大家合作，处理业务的能力是够了，但是任务来了后究竟由谁来处理呢？队员们会不会都不知道应该做什么？会不会都抢着做而打起来？因此必须要有一个领队（CDN Manager）来管理他们，指挥他们谁应该做什么、要达到什么样的目的。**在向各队员发出指示之前，领队需要首先统一接收任务，然后分析并分解任务、挑选合适的队员（CDN node）来执行任务，再将任务安排下去由合适的队员具体来执行。**在领队的合理安排下，队员们都有条不紊地执行任务，不会出现有的太闲而有的过忙的现象，“CDN 团队”融洽高效地完成了任务。

那么领队如何决定把某个任务或子任务分给哪个队员呢？领队的决定可不是随意做出的，而是相当谨慎且科学的。领队是依据一些策略来做出决定的。**这些策略包括用户的网络地址、用户的分组、先到先得规则等。**根据这些策略，领队可以把一个大任务分解成多个子任务，让每个队员执行其中一个小任务，大家完成后由领队汇总，达到完成大任务的目的，这样这个任务就会执行得很快（如分布式计算）；领队也可以把一批任务中的每一个任务分散到每个队员去分头执行，由队员直接完成这些任务，这样这批任务也会执行得很快。大规模的 CDN 网络通常是一级一级、一层一层的，我们叫它分布式网络，相邻的层级之间形成依赖和备份关系。而你点播的节目最终通常由最靠近你的CDN 节点送达你的电视机，这就是为什么节目总是那么及时地开演。当最靠近你的 CDN 节点出现故障或者服务能力无法提供服务时，领队能够根据依赖和备份策略选取其他合适的节点提供服务。

**F5负载均衡器&Nginx负载均衡**

**1 硬负载均衡和软负载均衡**

**硬件服务器：**就是直接购买独立的服务器作为负载均衡服务器，硬件负载均衡效率高，但是价格贵，例如，阿里云已经提供。

**软件服务器：**使用具有代理功能的软件作为转发服务器，比如 Nginx，HAProxy，LVS 等等，软件负载均衡系统价格较低或者免费，效率较硬件负载均衡系统低。

**2 Nginx**

一个高性能的HTTP和反向代理服务器，也是一个IMAP/POP3/SMTP代理服务器。Nginx是目前使用最广泛的HTTP软负载均衡器。 Nginx的实现，采用的是反向代理技术。**代理服务器充当服务器集群的前置，负责接收所有来自用户的请求，然后代理服务器根据负载均衡算法，将请求分发给服务器集群。服务器集群处理完请求后，将响应信息先发送给反向代理服务器，反向代理服务器再返回给用户。**

\* Nginx对网络的依赖比较小；

\* Nginx安装和配置比较简单，测试方便；

\* Nginx也能够承担高负载压力，稳定，一般能支撑超过1万次的并发；

\* Nginx可以通过端口检测到服务器内部的故障；

\* Nginx对请求的异步处理可以帮助节点服务器减轻负载；

\* Nginx支持HTTP和Email；

\* Nginx不支持Session的保持，对Big request header的支持不好；

\* Nginx默认只支持两种负载均衡算法：Round-robin和P-hash；

**3 F5 BIG-IP LTM**

官方名称叫做本地流量管理器，可以做4-7层负载均衡。F5 BIG-IP LTM功能强大，具有负载均衡、应用交换、会话交换、状态监控、智能网络、地址转换、通用持续性、响应错误处理、IPv6网关、高级路由、智能端口镜像、SSL加速、智能HTTP压缩、TCP优化、第7层速率整形、内容缓冲、内容转换、连接加速、高速缓存、Cookie加密、选择性内容加密、应用攻击过滤、拒绝服务（DoS）攻击、SYN Flood保护、防火墙一包过滤、包消毒。

\*F5 BIG-IP提供12种灵活的算法，将所有流量均衡的分配到各个服务器，而面对用户，只是一台虚拟服务器。

\* F5 BIG-IP可以确认应用程序能否对请求返回对应的数据。假如F5 BIG-IP后面的某一台服务器发生服务停止、死机等故障，F5会检查出来并将

该服务器标识为宕机，从而不将用户的请求传送到该台发生故障的服务器上。

\* F5 BIG-IP具有动态Session的回话保持功能；

\* F5 BIG-IP的iRules功能可以做HTTP内容过滤，根据不同的域名、URL，将访问请求传送到不同的服务器。

**4 F5与Nginx的对比**

**1），F5的优点和缺点**

优点：可以直接通过智能交换机实现，处理能力更强，而且与系统无关，负载性能强更适用于一大堆设备、大访问量、简单应用。

缺点：成本高，除设备价格昂贵外，而且配置冗余。很难想象后面服务器做一个集群，但最关键的负载均衡设备却是单点配置。无法有效掌握服务器和应用的状态。一般来说，硬件负载均衡技术都不管实际系统和应用的状态，而只是从网络层来判断当前的负载。这样带来的问题就是，硬件负载均衡器不能判断服务器或者应用的状态，有时候服务器或者应用已经高负载了，但是网络还没有问题的话，就会造成误判。

**2），Nginx的优点和缺点**

优点：Nginx是基于系统和应用的软件负载均衡技术，能够更好的根据系统或者应用的状态来分配负载。

缺点：负载能力受服务器本身性能的影响。服务器性能越好，负载能力越大。

**Haproxy**

**1 HAProxy是法国人Willy Tarreau开发的一个开源软件**，是一款应对客户端**10000**以上的同时连接的高性能的**TCP和 HTTP**负载均衡器。其功能是用来提供基于cookie的持久性， 基于内容的交换，过载保护的高级流量管制，自动故障切换 ，以正则表达式为基础的标题控制运行时间，基于Web的报表，高级日志记录以帮助排除故障的应用或网络及其他功能。

**2 相关概念**

**@四层负载均衡集群：**

lvs, nginx(stream)，haproxy(mode tcp)

**@七层负载均衡集群：**

http: nginx(http, ngx\_http\_upstream\_module), haproxy(mode http), httpd, ats, perlbal, pound...

**3 HAProxy功能**

**HAProxy是TCP / HTTP反向代理服务器，尤其适合于高可用性环境**

@可以针对HTTP请求添加cookie，进行路由后端服务器

@可平衡负载至后端服务器，并支持持久连接

@支持基于cookie进行调度

@支持所有主服务器故障切换至备用服务器

@支持专用端口实现监控服务

@支持不影响现有连接情况下停止接受新连接请求

@可以在双向添加，修改或删除HTTP报文首部

@支持基于pattern实现连接请求的访问控制

@通过特定的URI为授权用户提供详细的状态信息

**4 安装Haproxy**

# wget http://haproxy.1wt.eu/download/1.3/src/haproxy-1.3.20.tar.gz

# tar zcvf haproxy-1.3.20.tar.gz

**5 日志支持**

# vim /etc/syslog.conf

在最下边增加

local3.\* /var/log/haproxy.log

local0.\* /var/log/haproxy.log

#vim /etc/sysconfig/syslog

修改： SYSLOGD\_OPTIONS="-r -m 0"

重启日志服务service syslog restart

**6 启动服务**

启动服务：# /usr/local/haproxy/sbin/haproxy -f /usr/local/haproxy/haproxy.cfg

重启服务：# /usr/local/haproxy/sbin/haproxy -f /usr/local/haproxy/haproxy.cfg -st `cat /usr/local/haproxy/logs/haproxy.pid` (没有换行)

停止服务：# killall haproxy

在浏览器中输入haproxy监控地址:http://10.16.0.246:1080/haproxy-admin

在浏览器输入haproxy设置的对外访问地址：<http://10.16.0.244/，>

主程序：**/usr/sbin/haproxy**

配置文件：/etc/haproxy/**haproxy.cfg**

Unit file：/usr/lib/systemd/system/**haproxy.service**

haproxy.cfg主要有两部分组成：**global，和proxies**配置段

[root@CentOS6 ~]# vim /etc/haproxy/**haproxy.cfg**

frontend main \*:80 #设置监听ip：端口

default\_backend websrvs #调用后端RS组名

backend websrvs

balance roundrobin #轮询算法

server web1 192.168.45.11:80 check

server web2 192.168.45.12:80 check

**global：**全局配置段，进程及安全配置相关的参数，性能调整相关参数，Debug参数

global # 全局参数的设置

log 127.0.0.1 local2 # log语法：log <address\_1>[max\_level\_1] # 全局的日志配置，使用log关键字，指定使用127.0.0.1上的syslog服务中的local0日志设备，记录日志等级为info的日志

chroot /var/lib/haproxy #改变当前工作目录

pidfile /var/run/haproxy.pid #当前进程id文件

maxconn 4000 #最大连接数

user haproxy #所属用户group haproxy #所属组

daemon #以守护进程方式运行haproxy

stats socket /var/lib/haproxy/stats #基于本地的文件传输

**proxies：**代理配置段，defaults为frontend, backend, listen提供默认配置。fronted：前端，相当于nginx, server {}。backend：后端，相当于nginx, upstream {}。listen：同时拥有前端和后端,适用于一对一环境

mode http #默认的模式mode { tcp|http|health }，tcp是4层，http是7层，health只会返回OK

timeout http-request 10s #http请求超时时间

timeout queue 1m #一个请求在队列里的超时时间

timeout connect 10s #连接超时

timeout client 1m #客户端超时

timeout server 1m #服务器端超时

timeout http-keep-alive 10s #设置http-keep-alive的超时时间

timeout check 10s #检测超时

maxconn 3000 #每个进程可用的最大连接数

frontend main \*:80 #监听地址为80

acl url\_static path\_beg -i /static /images /javascript /stylesheets

acl url\_static path\_end -i .jpg .gif .png .css .js

use\_backend static if url\_static

default\_backend my\_webserver #定义一个名为my\_app前端部分。此处将对应的请求转发给后端

backend static #使用了静态动态分离（如果url\_path匹配 .jpg .gif .png .css .js静态文件则访问此后端）

balance roundrobin #负载均衡算法（#banlance roundrobin 轮询，balance source 保存session值，支持static-rr，leastconn，first，uri等参数）

server static 127.0.0.1:80 check #静态文件部署在本机（也可以部署在其他机器或者squid缓存服务器）

backend my\_webserver #定义一个名为my\_webserver后端部分。PS：此处my\_webserver只是一个自定义名字而已，但是需要与frontend里面配置项default\_backend 值相一致

balance roundrobin #负载均衡算法

server web01 172.31.2.33:80 check inter 2000 fall 3 weight 30 #定义的多个后端

server web02 172.31.2.34:80 check inter 2000 fall 3 weight 30 #定义的多个后端

server web03 172.31.2.35:80 check inter 2000 fall 3 weight 30 #定义的多个后端

### **7 Balance配置** **roundrobin：基于权重轮询**，动态算法，支持权重的运行时调整，支持慢启动；每个后端backend中最多支持4095个

**static-rr：基于权重轮询，静态算法**，不支持权重的运行时调整及慢启动；后端主机数量无上限

**leastconn：加权最少连接**，动态算法，最少连接的后端服务器优先分配接收新连接，相同连接时轮询，适用于长连接场景，例如 MySQL、LDAP等，不适合http

**first：根据服务器在列表中的位置**，自上而下进行调度；前面服务器的连接数达到上限，新请求才会分配给下一台服务

**source：源地址hash**，新连接先按权重分配，后续连接按source分配请求

### **8 支持https协议** **1 支持ssl会话；** bind \*:*443* ssl crt /PATH/TO/SOME\_PEM\_FILE crt 后证书文件为PEM格式，且同时包含证书和所有私钥 cat demo.crt demo.key > demo.pem

**2 把80端口的请求重向定443**  
bind \*:80  
redirect scheme https if !{ ssl\_fc }

**3 向后端传递用户请求的协议和端口（frontend或backend）**  
http\_request set-header X-Forwarded-Port %[dst\_port]  
http\_request add-header X-Forwared-Proto https if { ssl\_fc }

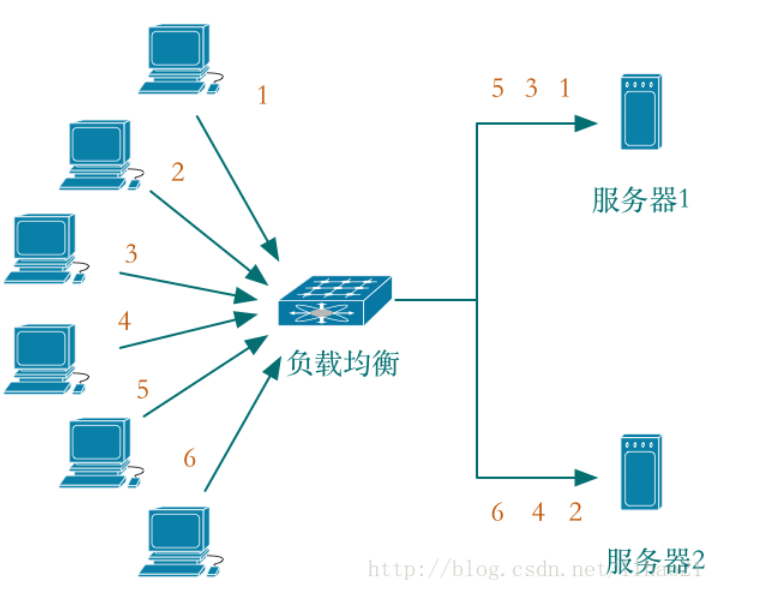
**负载均衡算法**

**1 轮询法（Round Robin）**

轮询法是负载均衡中最常用的算法，它容易理解也容易实现。

**轮询法是指负载均衡服务器（load balancer）将客户端请求按顺序轮流分配到后端服务器上，以达到负载均衡的目的**。

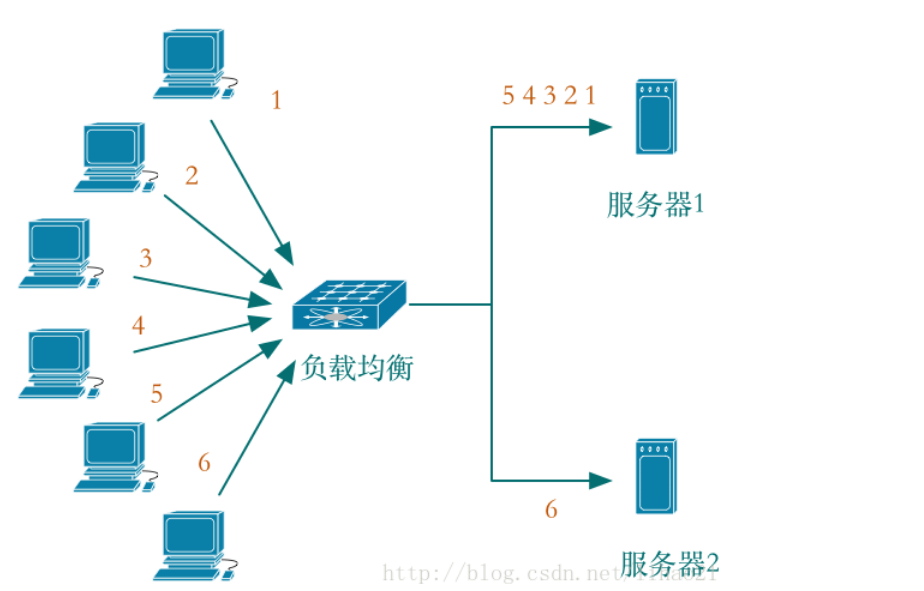
假设现在有6个客户端请求，2台后端服务器。当第一个请求到达负载均衡服务器时，负载均衡服务器会将这个请求分派到后端服务器1；当第二个请求到害时，负载均衡服务器会将这个请求分派到后端服务器2。然后第三个请求到达，由于只有两台后端服务器，故请求3会被分派到后端服务器1。依次类推，其示意图如图1所示。



**2 加权轮询法（Weighted Round Robin）**

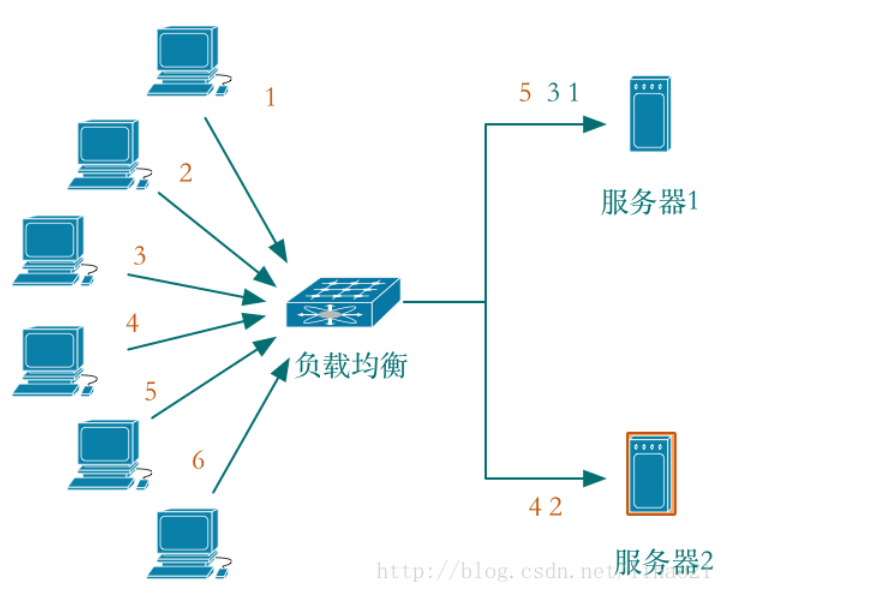
简单的轮询法并不考虑后端机器的性能和负载差异。**给性能高、负载低的机器配置较高的权重，让其处理较多的请求；而性能低、负载高的机器，配置较低的权重，让其处理较少的请求。**加权轮询法可以很好地处理这一问题，它将请求顺序且按照权重分派到后端服务器。

假设有6个客户端请求，2台后端服务器。后端服务器1被赋予权值5，后端服务器2被赋予赋予权值1。这样一来，客户端请求1，2，3，4，5都被分派到服务器1处理；客户端请求6被分派到服务器2处理。接下来，请求7，8，9，10，11被分派到服务器1，请求12被分派到服务器2，依次类推。这个请求分派的过程可以用图2来表示。



**3 最小连接数法（Least Connections）**

即使后端机器的性能和负载一样，不同客户端请求复杂度不一样导致处理时间也不一样。最小连接数法根据后端服务器当前的连接数情况，**动态地选取其中积压连接数最小的一台服务器来处理当前的请求，尽可能提高后端服务器的利用效率，合理地将请求分流到每一台服务器。**假设客户端请求1，2，3，4，5已被分派给服务器1和服务器2，其分派的情况如图3所示：



**4 随机法（Random）**

随机法也很简单，就是**随机选择一台后端服务器进行请求的处理**。由于每次服务器被挑中的概率都一样，客户端的请求可以被均匀地分派到所有的后端服务器上。

**5 源地址哈希法（Source Hashing）**

源地址哈希的思想是根据获取客户端的IP地址，**通过哈希函数计算得到的一个数值，用该数值对服务器列表的大小进行取模运算，得到的结果便是客服端要访问服务器的序号**。采用源地址哈希法进行负载均衡，同一IP地址的客户端，当后端服务器列表不变时，它每次都会映射到同一台后端服务器进行访问。 如果后端服务器是一缓存系统，当后端服务器增加或者减少时，采用简单的哈希取模的方法，会使得命中率大大降低，这个问题可以采用一致性哈希的方法来解决

**集群和分布式**

**1 分布式是个工作方式，一个业务分拆多个子业务**，部署在多个服务器上，不同的服务器节点完成不同的任务；分布式系统可能运行在一个集群上，也可能运行在不属于一个集群的多台（2台也算多台）机器上。集群是个物理形态。

**2 分布式是指将不同的业务分布在不同的地方**。 而集群指的是将几台服务器集中在一起，实现同一业务。 分布式中的每一个节点，都可以做集群。 而集群并不一定就是分布式的。

**3 举例：就比如新浪网**，访问的人多了，他可以做一个群集，前面放一个响应服务器，后面几台服务器完成同一业务，如果有业务访问的时候，响应服务器看哪台服务器的负载不是很重，就将给哪一台去完成。 而分布式，从窄意上理解，也跟集群差不多， 但是它的组织比较松散，不像集群，有一个组织性，一台服务器垮了，其它的服务器可以顶上来。

分布式的每一个节点，都完成不同的业务，一个节点垮了，哪这个业务就不可访问了。