负载均衡软件HAProxy

# 高性能负载均衡软件HAProxy介绍

基于硬件的负载均衡设备：f5,big-ip等

基于软件的负载均衡产品：HAProxy,lvs，nginx等

在软件的负载均衡产品中，分为基于系统的软负载实现和基于第三方软件的软负载实现，LVS是基于系统实现的一种软负载。HA proxy是基于第三方应用实现的软负载均衡HAProxy是一个开源的，高性能的，基于tcp第四层和http第七层应用的负载均衡软件。优点如下：

* 可靠性和稳定性非常好
* 最高可以同时维护40000-50000个并发连接。单位时间内处理最大的请求数为20000.最大数据处理能力可达10GBPS
* 支持多于8种负载均衡算法，同时也支持session保持
* 支持虚拟主机功能。
* 支持连接拒绝，全透明代理等功能
* HAProxy拥有一个功能强大的服务器状态监控页面
* HAProxy拥有强大的ACL支持

在业务系统方面。haproxy非常实用于那些并发量特别大而且需要持久连接或四层和七层处理机制的web系统，例如电商，另外haproxy也可用于mysql数据库（读操作）的负载均衡。

## 四层和七层负载均衡的区别

四层负载均衡器也称为四层交换机，它主要是通过分析ip层及tcp/udp层的流量实现的基于ip+端口。tcp应用实例：负载均衡器在接收到第一个来自SYN请求时，会通过设定的负载均衡算法选择一台最佳的后端服务器，同时将报文中的目标ip修改为后端服务器ip，然后直接转发给该服务器。负载均衡器在这个过程中相当于一个路由器的作用。例如：LVS，F5

七层负载均衡器也称为7层交换机，位于iso应用层，支持多种应用协议，常见的HTTP,FTP,SMTP等。七层负载均衡器可以根据报文内容，再配合负载均衡算法来选择后端服务器。因此也称为“内容交换机”。七层负载均衡器在这个情况下类似于一个代理服务器。负载均衡器在接收到获取的报文，然后分析该报文中的特定字段，在根据负载均衡中的特定算法来决定最终选择的内部服务器。例如HAProxy，Nginx

## haproxy与LVS的异同

|  |
| --- |
| 1. 两者都是软件负载均衡的产品。但是lvs基于系统，haproxy基于第三方应用 2. lvs是基于第四层的ip负载技术。haproxy是基于第四层和第七层，可提供tcp和http应用的负载均衡综合解决方案 3. LVS工作在iso模型第四层，状态监测功能单一。haproxy监测功能强大，可支持端口，url，脚本等多种检测方式 4. haproxy处理性能低于四层负载均衡模式的LVS   注意：选择那个作为负载均衡器，需要以实际环境来决定 |

# HAProxy基础配置与应用实例

下载地址：

## 安装HAProxy软件

|  |
| --- |
| #安装软件  tar -zxvf haproxy-1.5-1.5.15.tar.gz  cd haproxy-1.5-1.5.15  #TARGET指定内核版本，ARCH指定CPU架构  make TARGET=linux2628 ARCH=x86\_64  make install    #创建配置文件和启动文件  mkdir /etc/haproxy  cp /usr/local/src/haproxy-1.5-1.5.15/examples/haproxy.cfg /etc/haproxy/  cp /usr/local/src/haproxy-1.5-1.5.15/examples/haproxy.init /etc/init.d/haproxy  chmod u+x /etc/init.d/haproxy ln -s /usr/local/sbin/haproxy /usr/sbin/  mkdir /usr/share/haproxy  #编辑配置文件  # vi /etc/haproxy/haproxy.cfg  global  log 127.0.0.1 local0 info  maxconn 4096  chroot /usr/share/haproxy  user nobody  group nobody  daemon  nbproc 1    defaults  mode http  retries 3  timeout connect 10s  timeout client 20s  timeout server 30s  timeout check 5s    frontend www  bind \*:80  mode http  option httplog  option forwardfor  option httpclose  log global  default\_backend htmpool  backend htmpool  mode http  option redispatch  option abortonclose  balance roundrobin  cookie SERVERID  option httpchk GET /index.html  server web1 172.16.19.250:80 cookie server1 weight 6 check inter 2000 rise 2 fall 3  server web2 172.16.19.251:80 cookie server2 weight 6 check inter 2000 rise 2 fall 3  listen admin\_stats  bind 0.0.0.0:9188  mode http  log 127.0.0.1 local0 err  stats refresh 30s  stats uri /haproxy-status  stats realm welcome login\ Haproxy  stats auth admin:admin  stats hide-version  stats admin if TRUE |
| #服务相关操作  service haproxy start/stop/restart/status |

## HAProxy基础配置文件详解

### 配置文件概述

global部分：用设定义全局参数，属于进程级的配置，通常和操作系统配置有关。

defaults部分：默认参数配置部分。在此部分中设置的参数值，默认会自动引用到下面的frontend、backend、listen部分中，因引，某些参数属于公用的配置，只需要在defaults部分添加一次即可。而如果frontend、backend、listen部分也配置了与defaults部分一样的参数，Defaults部分参数对应的值自动被覆盖。

frontend部分：用于设置接收用户请求的前端虚拟节点。frontend是在haproxy 1.3版本以后才引入的一个组件，同时引入的还有backend组件。通过引入这些组件，在很大程度上简化了haproxy配置文件的复杂性。forntend可以根据ACL规则直接指定要使用的后端backend。

backend部分：此部分用于设置集群后端服务器集群的配置，也就是用来添加一组真实的服务器，用于处理前端用户请求。

listen部分：是frontend和backend部分的结合体。在HAProxy1.3版本之前，HAProxy的所有配置选项都在这个部分中设置。为了保持兼容性，haproxy新的版本依然保留了listen组件配置试。两种配置方式任选一中。

注意时间格式：

* us: 微秒(microseconds)，即1/1000000秒；
* ms: 毫秒(milliseconds)，即1/1000秒；
* s: 秒(seconds)；
* m: 分钟(minutes)；
* h：小时(hours)；
* d: 天(days)；

### HAProxy配置详解

|  |
| --- |
| **-- global部分**  global  #全局的日志配置，local0是日志设置，info表示日志级别。  #日志级别有：err、waning、info、debug四种可选。  #这个配置表示使用localhost上rsyslog服务中的local0日志设备，级别为info  log 127.0.0.1 local0 info  #设定每个HAProxy进程可接受的最大并发连接数，  #此选项等同于linux命令选项”ulimit -n”  maxconn 4096  #运行HAProxy进程的用户和组，也可以使用uid/gid  user nobody  group nobody  #让haproxy以守护进程的方式工作于后台，其等同于“-D”选项的功能，  #当然，也可以在命令行中以“-db”选项将其禁用；  daemon  #指定启动的haproxy进程个数，只能用于守护进程模式的haproxy；默认只启动  #一个进程，一般只在单进程仅能打开少数文件描述符的场景中才使用多进程模式；  nbproc 1  #指定haproxy进程的pid文件  pidfile /usr/local/haproxy/logs/haproxy.pid |
| **-- defaults部分**  defaults  #mode:设置haproxy实例默认的运行模式，有tcp，http，health3个模式  #tcp模式：客户端和服务端之间将建立一个全双工连接，不对七层报文做任何检查，默认为tcp模式，经常用于ssl，ssh，smtp等应用  #http模式：客户端请求在转发至后端服务器之前将会深度分析，所有与RFC格式不兼容的请求都会被拒绝  #health：已基本不用了。  mode http  #设置连接后端服务器的失败重试次数  retries 3  #连接到一台服务器的最长等待时间，默认毫秒  timeout connect 10s  #连接客户端发送数据最长等待时间，默认毫秒  timeout client 20s  #服务器回应客户端发送数据最长等待时间，默认毫秒  timeout server 30s  #对后端服务器的检测超时时间，默认毫秒  timeout check 5s |
| **-- frontend部分**  #定义前段虚拟节点'www'  frontend www  #监听套接字 bind addr:port\_range interface <interface>  bind \*:80  mode http  #开启日志记录http请求  option httplog  #如果后端服务器需要获取客户端的真实ip  option forwardfor  #在客户端端和服务器端完成一次连接请求时候，主动关闭tcp连接  option httpclose  #表示使用全局的日志配置，配置文件中global指定的log格式  log global  #指定默认后端服务器组（htmpool是真实服务器组名）  default\_backend htmpool |
| **-- backend部分**  backend htmpool  mode http  #用于cookie保持的环境中  option redispatch  #负载过高时，结束当前队列中处理时间较长的链接  option abortonclose  #用来定义负载均衡算法。  #roundrobin：基于权重进行的轮叫算法，在服务器的性能分布经较均匀时这是一种最公平的，最合量的算法。  #static-rr：也是基于权重时行轮叫的算法，不过此算法为静态方法，在运行时调整其服务权重不会生效。  #source：是基于请求源IP的算法，此算法对请求的源IP时行hash运算，然后将结果与后端服务器的权理总数相除后转发至某台匹配的后端服务器，这种方法可以使用一个客户端IP的请求始终转发到特定的后端服务器。  #leastconn：此算法会将新的连接请求转发到具有最少连接数目的后端服务器。在会话时间较长的场景中推荐使用此算法。例如数据库负载均衡等。此算法不适合会话较短的环境，如基于http的应用。  #uri：此算法会对部分或整个URI进行hash运算，再经过与服务器的总权重要除，最后转发到某台匹配的后端服务器上。  #uri\_param：此算法会椐据URL路径中的参数时行转发，这样可以保证在后端真实服务器数量不变时，同一个用户的请求始终分发到同一台机器上。  #hdr()：此算法根据httpd头时行转发，如果指定的httpd头名称不存在，则使用roundrobin算法进行策略转发。  balance source  #表示向cookie插入serverid  cookie SERVERID  #表示启用http服务状态检测功能，把前段请求分配到健康的backend节点上  #option httpchk <method> <uri> <version>  #method：http请求方式options,get,head几种方式。  #uri:表示要检测的url地址，正常情况将返回状态码200  #version:指定心跳检测时的HTTP的版本号  option httpchk GET /index.html  #server用来定义多台真实的后端服务器，不能用于defaults和frontend  #格式为：server name address:port param\*  #name：为后端真实服务器指定一个内部名称，随便这下义一个即可。  #address：后端真实服务器的iP地址或主机名。  #port：指定连接请求发往真实服务器时的目标端口，在未设定是地，将使用客户端请求时的同一端口。  #param\*：为后端服务器设定的一系列参数，可用参数非常多：   * check：表示启用对此后端服务器执行健康检查。 * inter：设置健康状态检查的时间间隔，单位为毫秒。 * rise：设置人故障状态转换至正常状态需要成功检查的次数，如 rise 2：表示2次检查正确就认为此服务器可用。 * fall：设置后端服务器从正常状态转换为不可用状态需要检查的次数，如 fall 3表示3 次检查失败就认为此服务器不可用。 * cookie：为指定的后端服务器设定cookie值，此外指定的值将在请求入站时被检查，第一次为此值挑选的后端服务器将在后续的请求中一直被选中，其目的在于实现持久连接的功能。 cookie server1：表示web1的serverid为server1。 * weigth：设置后端真实服务器的权重，默认为1，最大值为256，设置为0表示不参与负载均衡。 * backup：设置后端真实服务器的备份服器，仅仅在后端所有真实服务器均不可用的情况下才启用。   server web1 192.168.32.129:80 cookie server1 weight 6 check inter 2000 rise 2 fall 3  server web2 192.168.32.202:80 cookie server2 weight 6 check inter 2000 rise 2 fall 3 |
| **--** **listen部分**  #定义一个名为admin\_stats的监控页面  listen admin\_stats  bind 0.0.0.0:9188  mode http  log 127.0.0.1 local0 err  #设置HAProxy监控统计页面自动刷新的时间。  stats refresh 30s  #置HAProxy监控统计页百的RUL路径，可随意指定  stats uri /haproxy-status  #设置登录HAProxy统计页面时密码框上的文本提示信息。  stats realm welcome login\ Haproxy  #设置登录HAProxy统计页面的用户名和密码。用户名与密码用冒号分割。  #可设多个用户，每行一个。  stats auth admin:admin  #用来隐藏统计页面上的HAProxy的版本信息  stats hide-version  #通过设置此选项，可以在监控页面上手一启用或禁用后端真实服器。  stats admin if TRUE |

## HAProxy日志配置策略

|  |
| --- |
| #定义两种日志类型，并制定了日志的输出路径  vim /etc/rsyslog.d/haproxy.conf |
| $ModLoad imudp  $UDPServerRun 514  local3.\* /usr/local/haproxy/logs/haproxy.log  local0.\* /usr/local/haproxy/logs/haproxy.log |
| #修改/etc/sysconfig/rsyslog  vim /etc/sysconfig/rsyslog |
| #-r表示接受远程日志  SYSLOGD\_OPTIONS="-c 2 -r -m 0"  #重启rsyslog服务  service rsyslog restart |

## 通过HAProxy的ACL规则实现智能负载均衡

HAProxy通过ACL规则完成两种主要的功能，分别是：

* 通过设置acl规则检查客户端请求是否合法，如果符合要求，那么放行。如果不符合规则，就直接中断请求。
* 符合acl规则要求的请求将被提交到后端的backend服务器集群，进而实现基于acl规则的负载均衡

acl规则经常使用在frontend字段中，使用方法：

acl 自定义acl名称 acl方法 -i(不区分大小写) 匹配的路径或者文件

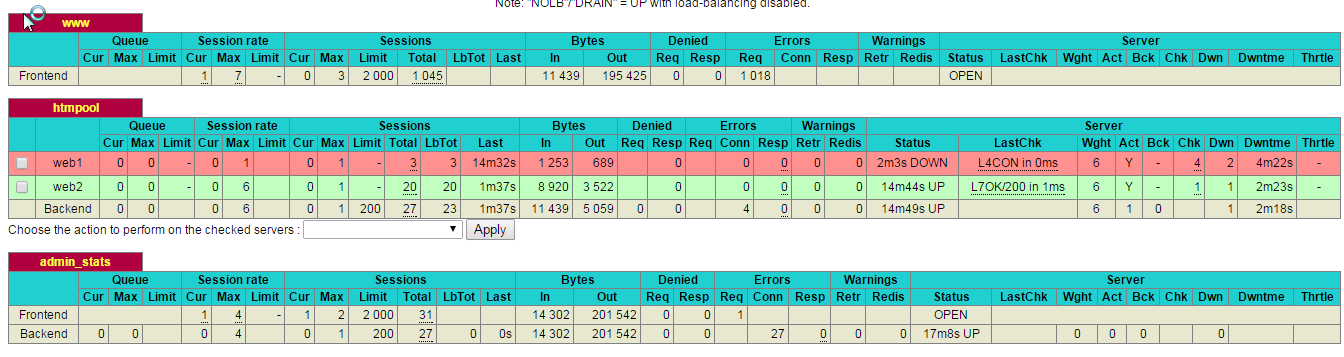
常见的acl方法：

* hdr\_reg(host)：主机地址通过正则匹配
* hdr\_dom(host)：主机地址的域名匹配
* hdr\_beg(host)：主机地址的开始字符
* url\_sub(host)：请求中url包含字符
* url\_dir(host)：
* path\_beg(host)：
* path\_end(host)：url以给定字符串结尾

|  |
| --- |
| acl bbs hdr\_reg(host) -i ^(bbs.test.com|forum.test.com) #使用正则匹配  acl bbs\_path path\_beg -i /bbs #url 目录  acl youxi path\_beg -i /youxi  acl static path\_end -i .html .css .js #url 结尾文件  acl php path\_end -i .php  acl jsp path\_end -i .jsp .do  use\_backend bbs\_pool if bbs or bbs\_path #注意 "or"  use\_backend youxi\_pool if youxi  use\_backend static\_pool if static  use\_backend php\_pool if php  use\_backend jsp\_pool if jsp  default\_backend www.test.com |

## 使用HAProxy的Web监控平台

url：http://172.16.19.249:9188/haproxy-status



# 使用HAProxy实现MongoDB集群

MongoDB分片集群的入口mongos自身没有failover机制。官方建议是将mongos和应用服务器部署在一起，多个应用服务器就要部署mongos实例，这样很是不方便。还可以使用LVS或者HAProxy来实现多个mongos的failover机制，但是一定要注意使用client affinity即客户端关联特性。

配置如下：

|  |
| --- |
| global  log 127.0.0.1 local0 info  maxconn 65536  chroot /usr/share/haproxy  user nobody  group nobody  daemon  nbproc 1  pidfile /var/run/haproxy.pid  stats socket /tmp/haproxy level admin  stats maxconn 20  node master\_loadbalance1  description lb1  nosplice  spread-checks 3    defaults  log global  mode tcp  option abortonclose  option allbackups  option tcpka  option redispatch  retries 3  timeout check 60s  timeout connect 600s  timeout queue 600s  timeout server 600s  timeout tarpit 60s  timeout client 600s        frontend mongos\_pool 0.0.0.0:27017  mode tcp  maxconn 32768  no option dontlognull  option tcplog  log global  option log-separate-errors  default\_backend mongos\_pool    backend mongos\_pool  mode tcp  balance roundrobin  default-server inter 2s fastinter 1s downinter 5s slowstart 60s rise 2 fall 5 weight 30  server mongo\_1 10.161.219.27:27017 check maxconn 2000  server mongo\_2 10.161.220.172:27017 check maxconn 2000  server mongo\_3 10.162.57.241:27017 check maxconn 2000 |

# HAProxy+Keepalived实现高可用

## HAProxy+Keepalived主从配置

|  |
| --- |
| #master  ! Configuration File for keepalived  global\_defs {  notification\_email {  xiaoxiazhang1@hengtiansoft.com  }    notification\_email\_from mail@example.org  smtp\_server 127.0.0.1  smtp\_connect\_timeout 30  router\_id LVS\_DEVEL  }  vrrp\_script chk\_haproxy {  script "</dev/tcp/127.0.0.1/80"  interval 1 #检测脚本执行的间隔  weight -2 #权重  }    vrrp\_instance VI\_1 {  state MASTER  interface eth0  virtual\_router\_id 51  priority 101  advert\_int 1  mcast\_src\_ip 172.16.19.249    authentication {  auth\_type PASS  auth\_pass p@$$w0rd  }    virtual\_ipaddress {  172.16.19.147  }    track\_script {  chk\_haproxy #调用检测脚本  }  } |
| #backup  ! Configuration File for keepalived  global\_defs {  notification\_email {  xiaoxiazhang1@hengtiansoft.com  }    notification\_email\_from mail@example.org  smtp\_server 127.0.0.1  smtp\_connect\_timeout 30  router\_id LVS\_DEVEL  }  vrrp\_script chk\_haproxy {  script "</dev/tcp/127.0.0.1/80"  interval 1 #检测脚本执行的间隔  weight -2 #权重  }      vrrp\_instance VI\_1 {  state BAKEUP  interface eth0  virtual\_router\_id 51  priority 100  advert\_int 1  mcast\_src\_ip 172.16.19.250    authentication {  auth\_type PASS  auth\_pass p@$$w0rd  }    virtual\_ipaddress {  172.16.19.147  }    track\_script {  chk\_haproxy #调用检测脚本  }  } |

## HAProxy+Keepalived双主配置

|  |
| --- |
| #master\_1  ! Configuration File for keepalived  global\_defs {  notification\_email {  xiaoxiazhang1@hengtiansoft.com  }    notification\_email\_from mail@example.org  smtp\_server 127.0.0.1  smtp\_connect\_timeout 30  router\_id LVS\_DEVEL  }  vrrp\_script chk\_haproxy {  script "</dev/tcp/127.0.0.1/80"  interval 1 #检测脚本执行的间隔  weight -2 #权重  }    vrrp\_instance VI\_1 {  state MASTER  interface eth0  virtual\_router\_id 51  priority 101  advert\_int 1  mcast\_src\_ip 172.16.19.249    authentication {  auth\_type PASS  auth\_pass p@$$w0rd  }    virtual\_ipaddress {  172.16.19.147  }    track\_script {  chk\_haproxy #调用检测脚本  }  }  vrrp\_instance VI\_2 {  state BACKUP  interface eth0  virtual\_router\_id 52  priority 99  advert\_int 1  mcast\_src\_ip 172.16.19.250    authentication {  auth\_type PASS  auth\_pass p@$$w0rd  }    virtual\_ipaddress {  172.16.19.148  }    track\_script {  chk\_haproxy #调用检测脚本  }  } |
| #master\_2  ! Configuration File for keepalived  global\_defs {  notification\_email {  xiaoxiazhang1@hengtiansoft.com  }    notification\_email\_from mail@example.org  smtp\_server 127.0.0.1  smtp\_connect\_timeout 30  router\_id LVS\_DEVEL  }  vrrp\_script chk\_haproxy {  script "</dev/tcp/127.0.0.1/80"  interval 1 #检测脚本执行的间隔  weight -2 #权重  }    vrrp\_instance VI\_1 {  state BACKUP  interface eth0  virtual\_router\_id 51  priority 99  advert\_int 1  mcast\_src\_ip 172.16.19.249    authentication {  auth\_type PASS  auth\_pass p@$$w0rd  }    virtual\_ipaddress {  172.16.19.147  }    track\_script {  chk\_haproxy #调用检测脚本  }  }  vrrp\_instance VI\_2 {  state MASTER  interface eth0  virtual\_router\_id 52  priority 100  advert\_int 1  mcast\_src\_ip 172.16.19.250    authentication {  auth\_type PASS  auth\_pass p@$$w0rd  }    virtual\_ipaddress {  172.16.19.148  }    track\_script {  chk\_haproxy #调用检测脚本  }  } |