haproxy 架构指南（中文翻译版）

版本 V1.0
时间 2012-07-18

作者 itnihao 邮箱 itnihao@qq.com
本文档来自 haproxy1.4.21 官方文档的翻译，如有不妥之处，请邮件联系我，谢谢！
注意：本文档 1-3.1 为此博客翻译 http://blog.chinaunix.net/uid-10249062-id-348470.html 3.1 章节以后为 itnihao 翻译
本文档不定期更新，地址 http://itnihao.blog.51cto.com，敬请关注

==========
1. 基于 cookie 插入的简单 HTTP 负载均衡
==========

由于包含脚本的原因，1个 web app 经常可以使得前段 server 有很高的 CPU loads。同样，web app 也依赖于没有太多负载性能的后端数据库。用户的上下文存放在 server 上，而不是在数据库中。因此，采用 TCP/IP 负载均衡的方简单增加另外的 server 并不能很好的工作。

```
                +--------+
                |clients|  clients and/or reverse-proxy
                +---+----+
                    |
                  -+------+--------+----
                         |        _|_db
                      +--+--+     (___)
                      | web |     (___)
                      +-----+     (___)
                   192.168.1.1    192.168.1.2
```

采用大型的 SMP 系统来替换普通 server 则会比增加低成本的 server 开销大很多。一个解决方案就是购买低成本的 server，并在其上安装 app。在其中 1 个旧的 server 上安装 haproxy，会把负载分散到新的 server 上。

```
 192.168.1.1      192.168.1.11-192.168.1.14     192.168.1.2
-------+-----------+-----+-----+-----+--------+----
       |           |     |     |     |         _|_db
    +--+--+      +-+-+ +-+-+ +-+-+ +-+-+      (___)
    |LB1 |      |A||B||C||D|                   (___)
    +-----+     +---+ +---+ +---+ +---+        (___)
    haproxy          4 cheap web servers
```

Config on haproxy (LB1) :

```
    listen webfarm 192.168.1.1:80
        mode http
        balance roundrobin
```

```
        cookie SERVERID insert indirect
        option httpchk HEAD /index.html HTTP/1.0
        server webA 192.168.1.11:80 cookie A check
        server webB 192.168.1.12:80 cookie B check
        server webC 192.168.1.13:80 cookie C check
        server webD 192.168.1.14:80 cookie D check
```

描述：

- LB1 接收 clients 的 requests
- 如果 1 个 request 不包含 cookie，则把这个 request 前传到分配的一个有效的 server
- 作为回报，1 个拥有 server 名称的 cookie "SERVERID"会被插入到 response 中
- 当 client 带有 cookie "SERVERID=A"再此访问时，LB1 就会知道这个 request 必须被前传到 server A. 同时删除这个 cookie 是的 server 不会看到它
- 当 server"webA"宕机时，request 会被前传至另外一个有效的 server，并且重新分配 cookie

2 带 cookie 前缀的 HTTP 负载均衡和高可用性

2.1 带 cookie 前缀的 HTTP 负载均衡和高可用性

现在你可以不用添加更多的 cookie，而是使用已有的 cookie。如果应用已经生成 J 足够跟踪 session 的 SESSIONIDcookie，我们会在看到该 cookie 时，在它前面加上 servername 前缀。由于 load-balancer 变得关键，因此，可以通过利用 keepalived 使得运行在 VRRP 模式下的第二台热备它。从网站上下载最新版本的 keepalived，并且安装在 load-balancer LB1 和 LB2 上。http://www.keepalived.org/在两个 load-balancer(我们仍然使用原始 IP)之间，我们使用一个共享 IP。在任何时刻只有 1 个 load-balancers 处于活跃状态。为了允许 proxy 在 Linux2.4 下绑定一个共享 IP，需要在/proc 中启用如下配置：

```
#echo1>/proc/sys/net/ipv4/ip_nonlocal_bind
```

Config on both proxies (LB1 and LB2) :

```
    listen webfarm 192.168.1.1:80
        mode http
        balance roundrobin
        cookie JSESSIONID prefix
        option httpclose
        option forwardfor
        option httpchk HEAD /index.html HTTP/1.0
        server webA 192.168.1.11:80 cookie A check
        server webB 192.168.1.12:80 cookie B check
```

```
        server webC 192.168.1.13:80 cookie C check
        server webD 192.168.1.14:80 cookie D check
```

提示：proxy 会修改 client 和 server 发出的每个 cookie，因此 proxy 能够访问每个 session 中的所有 request 的所有 cookie 是非常重要的。这意味着不是 keepalive(HTTP/1.1)，需要使用'httpclose'选项。只有你能确认 clients 不会使用 keepalive，才能删除这个选项。

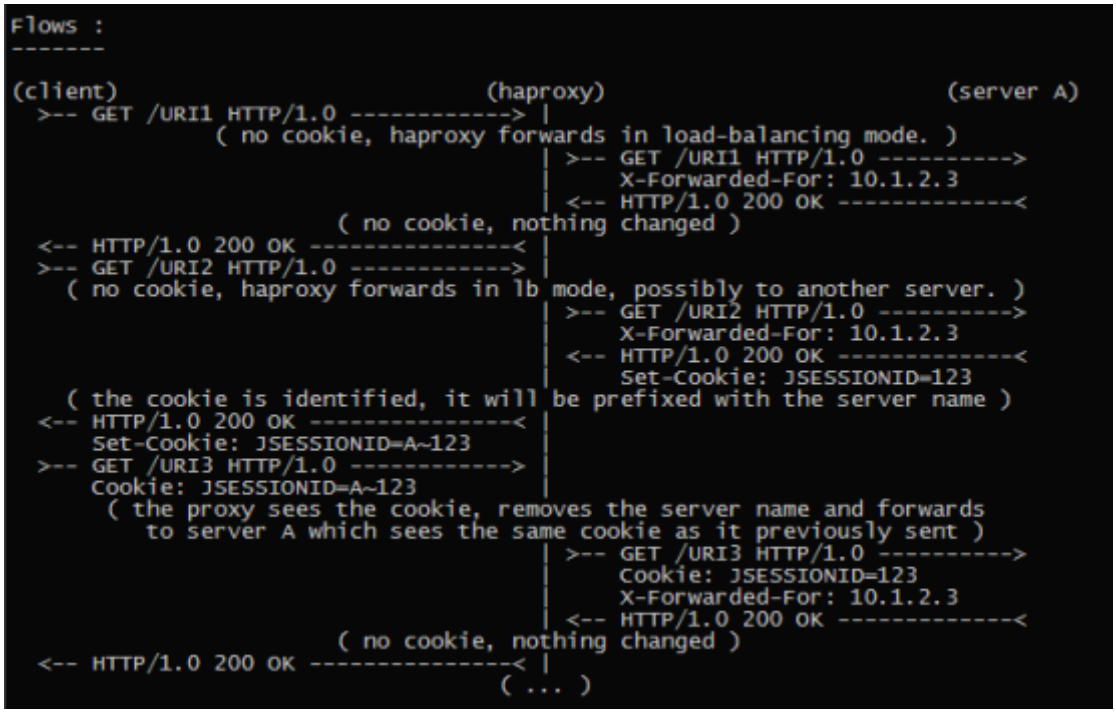
Configuration for keepalived on LB1/LB2：

```
    vrrp_script chk_haproxy {               # Requires keepalived-1.1.13
        script "killall -0 haproxy"         # cheaper than pidof
        interval 2                          # check every 2 seconds
        weight 2                            # add 2 points of prio if OK
    }

    vrrp_instance VI_1 {
        interface eth0
        state MASTER
        virtual_router_id 51
        priority 101                        # 101 on master, 100 on backup
        virtual_ipaddress {
            192.168.1.1
        }
        track_script {
            chk_haproxy
        }
    }
```

描述：

—LB1 是 VRRP 的主（keepalived），LB2 是备。他们都监控 haproxy 进程，并且如果 haproxyfailed 则降低他们的权重，迁移至另外的节点；
—LB1 将在 IP：192.168.1.1 上接收 clientrequest
—两个 LB 用他们的内部 IP 发送健康检测
—如果 request 不包含 cookie，request 会被前传至 1 个有效的 server
—在回复时，如果看到 JESSIONIDcookie，proxy 会在 cookie 加上以('~')为分隔符的 servername 前缀；
—如果 client 再次访问时带着"JSESSIONID=A~xxx" cookie，LB1 会知道这个 request 必须前传至 serverA。在把 cookie 发送给 server 之前，cookie 中的 servername 会先被剔除出来。
—如果 server"webA"宕机，requests 会被前传到另外的有效服务器，并且会重新分配 cookie；
数据流：



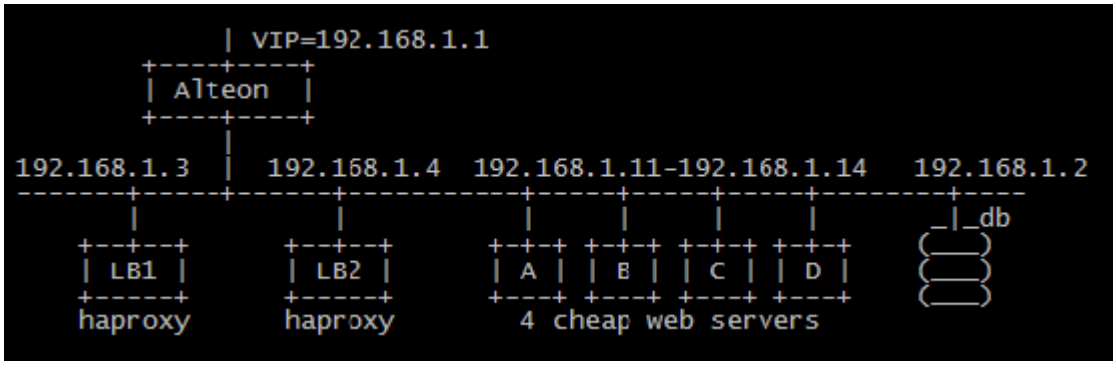
提示：

有时，在一个集群中有一些性能强劲的 server，也有一些性能差一些的 server。在这种情况下，很有必要告诉 haproxy 这些 server 在性能上的差异。假设 WebA 和 WebB 是两台老的 P3-1.2GHz，WebC 和 WebD 是最新的 Opteron-2.6GHz。如果你的 application 关注 CPU，则你可以假设这两台服务器之间的性能比为 2.6/1.2。你可以通过使用 1-256 之间数值标记的"weight"关键字，告诉 haproxy 这些性能差异。它可以用这些比例来平滑 load：

```
server webA 192.168.1.11:80 cookie A weight 12 check
server webB 192.168.1.12:80 cookie B weight 12 check
server webC 192.168.1.13:80 cookie C weight 26 check
server webD 192.168.1.14:80 cookie D weight 26 check
```

2.1 包含外部 L4load-balancers 的变化

作为基于 VRRP 主备方案的替代方案，他们也可以使用 L4load-balancer（如：Alteon）进行负载，这些 L4load-balancer 也可以检查这些运行在 proxy 上的服务：



Config on both proxies (LB1 and LB2) :

```
listen webfarm 0.0.0.0:80
    mode http
    balance roundrobin
    cookie JSESSIONID prefix
```

```
        option httpclose
        option forwardfor
        option httplog
        option dontlognull
        option httpchk HEAD /index.html HTTP/1.0
        server webA 192.168.1.11:80 cookie A check
        server webB 192.168.1.12:80 cookie B check
        server webC 192.168.1.13:80 cookie C check
        server webD 192.168.1.14:80 cookie D check
```

"dontlognull"选项用来防止记录 Alteon 发出的健康检测。如果一个 session 交互没有数据，这个 session 就不会被记录。

Config on the Alteon :

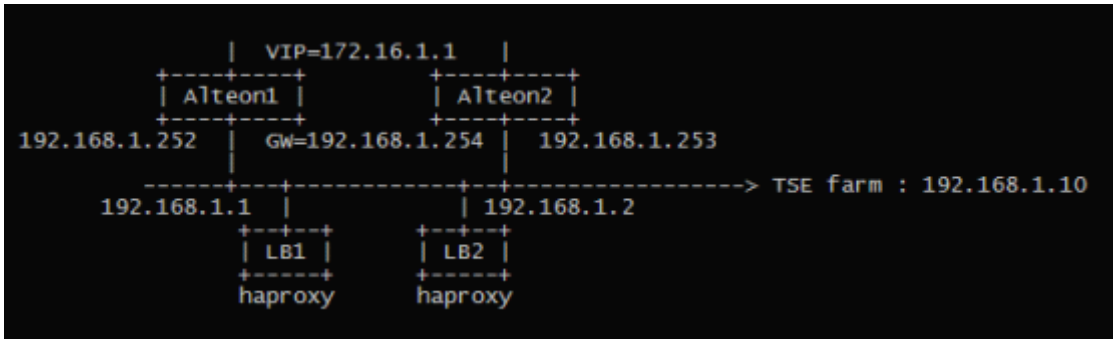
```
/c/slb/real    11
            ena
            name "LB1"
            rip 192.168.1.3
/c/slb/real    12
            ena
            name "LB2"
            rip 192.168.1.4
/c/slb/group 10
            name "LB1-2"
            metric roundrobin
            health tcp
            add 11
            add 12
/c/slb/virt 10
            ena
            vip 192.168.1.1
/c/slb/virt 10/service http
            group 10
```

提示：Alteon 上的健康检测被设置为'tcp'，用来防止 proxy 把连接前传。Alteon 也可以设置为'http'，但是这样 proxy 必须把配置 Alteon 的地址为"monitor-net"，那样 Alteon 可以真实的以 http 会话的方式检测 proxy 而不会把连接前传到后端的 servers。检查下一节可以看到怎样使用 monitor-net 的例子。

2.2 通用的 TCP 中继和外部 L4load-balancers

有时，能够中继通用的 TCP 协议(如：SMTP,TSE,VNC 等)是非常有用的，如访问内部稀有网络。当你在使用外部 load-balancers 需要发送周期行的健康检测至 proxy 时问题就会出现，这是因为健康检测会被前传至后端 servers。对这个问题的 1 个解决方案是通过使用"monitor-net"用一个专用的网络来监控系统，并且必须不会前传连接和记录。

提示：这个特性只有在 1.1.32 或者 1.2.6 版本以上才支持。



"monitor-net" 选项指示 haproxy 对来自 192.168.1.252 和 192.168.1.253 的请求不记录、不前传并且马上关闭连接。Alteonload-balancers 将可以检测到 proxy 是活跃的而不会扰动正常服务。

Config on the Alteon :

```
/c/l3/if 1
        ena
        addr 192.168.1.252
        mask 255.255.255.0
/c/slb/real   11
        ena
        name "LB1"
        rip 192.168.1.1
/c/slb/real   12
        ena
        name "LB2"
        rip 192.168.1.2
/c/slb/group 10
        name "LB1-2"
        metric roundrobin
        health tcp
        add 11
        add 12
/c/slb/virt 10
        ena
        vip 172.16.1.1
/c/slb/virt 10/service 1494
        group 10
/c/slb/virt 10/service 3389
        group 10
/c/slb/virt 10/service 5900
        group 10
```

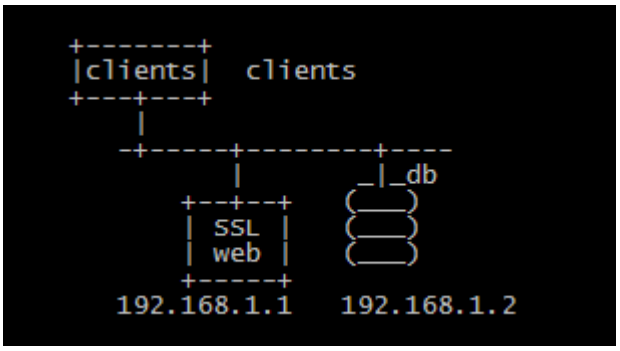
SSL 的特殊处理：

有时，即使在 TCP 模式下，你需要给远程系统发送健康检测以便能够在 server 发生故障时能够切换到备份服务器。当然，你可以简单地启用 TCP 健康检查，但是在 proxy 和远程 server 之间的 firewalls 会自己确认 TCP 连接，从而显示为 1 个始终活跃的 server。由于这是在使用

SSL 的长途通信中普遍遇到的问题，一个 SSL 健康检查已实现来要解决这个问题。它发出的 SSLmessages 消息到远程 server，server 回复 Hellomessages。
可以很容易的进行配置：

```
listen tcp-syslog-proxy
    bind :1514          # listen to TCP syslog traffic on this port (SSL)
    mode tcp
    balance roundrobin
    option ssl-hello-chk
    server syslog-prod-site 192.168.1.10 check
    server syslog-back-site 192.168.2.10 check backup
```

3 带 cookie 插入的简单 HTTP/HTTPS 负载均衡
这是跟示例 1 相同的情况下，但是 Webserver 使用的是 https。



由于 haproxy 不能处理 SSL，SSL 这一部分将不得不从 server 中提取出来（腾出更多的资源），而安装在 load-balancer 上。在一个 box 上安装 haproxy 和 apache+mod_ssl，把该 box 的负载分散至新的 box 上。Apache 运行在 SSL 的 reverse-proxy-cache 模式下。如果应用程序设计的合理，甚至可能会降低其负载。不过，由于现在是一个与客户端和缓存的 haproxy，一些安全必须采取措施，以确保插入的 cookies 不被缓存。

Config on haproxy (LB1) :

```
listen 127.0.0.1:8000
    mode http
    balance roundrobin
    cookie SERVERID insert indirect nocache
    option httpchk HEAD /index.html HTTP/1.0
    server webA 192.168.1.11:80 cookie A check
    server webB 192.168.1.12:80 cookie B check
    server webC 192.168.1.13:80 cookie C check
```

```
server webD 192.168.1.14:80 cookie D check
```

描述：

----------LB1 上的 apache 在 443 端口上接收 clients 的请求

-apache 前传请求至绑定在 127.0.0.1:8000 上的 haproxy

-如果 request 不带 cookie，则被前传至一台有效的 server

-在回复时，haproxy 会在回复中插入一个包含 server 名称（如：A）的"SERVERID"cookie，和一个"Cache-control:private"header。那样 apache 就不会 cache 带这样 cookie 的 page；

- 如果 client 再次访问时带了"SERVERID=A"cookie，则 LB1 会知道必须把该 request 前传至 serverA。haproxy 会删除该 cookie，而 server 不会看到它；

- 如果 server"webA"宕机，request 会被前传至另外一个有效的 server，而 cookie 会被重置；

提示：

----------如果 cookie 工作在"prefix"模式下，haproxy 就不需要配置"nocache"选项。因为，这个 applicationcookie 会被修改，并且 applicationflags 会被保留；

- 如果 haproxy 的前段使用了 apache1.3，则它会一直禁用后端 HTTP keep-alive，因此可以不必在 haproxy 上配置"httpclose"；

-如果 application 需要知道 client'sIP，则在 apache 上配置 X-Forwarded-For header，而不要在 haproxy 上配置；

```
Flows :
-------

(apache)                              (haproxy)                              (server A)
  >-- GET /URI1 HTTP/1.0 ------------> |
                ( no cookie, haproxy forwards in load-balancing mode. )
                                       | >-- GET /URI1 HTTP/1.0 ---------->
                                       | <-- HTTP/1.0 200 OK --------------<
                ( the proxy now adds the server cookie in return )
  <-- HTTP/1.0 200 OK ----------------< |
      Set-Cookie: SERVERID=A           |
      Cache-Control: private           |
  >-- GET /URI2 HTTP/1.0 ------------> |
      Cookie: SERVERID=A               |
      ( the proxy sees the cookie. it forwards to server A and deletes it )
                                       | >-- GET /URI2 HTTP/1.0 ---------->
                                       | <-- HTTP/1.0 200 OK --------------<
    ( the proxy does not add the cookie in return because the client knows it )
  <-- HTTP/1.0 200 OK ----------------< |
  >-- GET /URI3 HTTP/1.0 ------------> |
      Cookie: SERVERID=A               |
                                     ( ... )
```

3.1 使用 Stunnel 的替代方案

如果只需要 SSL 而不需要 cache，则 stunnel 是一个比 Apache+mod_ssl 更廉价的解决方案。stunnel 默认不出 HTTP 并且不能增加 X-Forwarded-For header，但是在 haproxy 的官方网站上有针对最新 stunnelversions 版本支持该特性的 patch。这时，stunnel 只是处理 HTTPS 而不处理 HTTP。这也意味着 haproxy 会接收所有的 HTTP 访问。因此，haproxy 需要在 HTTP 访问中增加 X-Forwarded-Forheader，而不用对 HTTPS 访问做处理。因为，stunnel 已经做过处理了。我们可以使用"except"关键字告诉 haproxy，来自本地的连接已经有有效的 header 了。

stunnel(LB1)上的配置：

```
cert=/etc/stunnel/stunnel.pem
setuid=stunnel
setgid=proxy
socket=l:TCP_NODELAY=1
socket=r:TCP_NODELAY=1
[https]
accept=192.168.1.1:443
connect=192.168.1.1:80
xforwardedfor=yes
```

haproxy(LB1)上的配置：

```
listen 192.168.1.1:80
    mode http
    balance roundrobin
    option forwardfor except 192.168.1.1
    cookie SERVERID insert indirect nocache
    option httpchk HEAD /index.html HTTP/1.0
    server webA 192.168.1.11:80 cookie A check
    server webB 192.168.1.12:80 cookie B check
    server webC 192.168.1.13:80 cookie C check
    server webD 192.168.1.14:80 cookie D check
```

描述：

----------LB1 上的 stunnel 会在 443 上接收 client 的 requests；

-stunnel 前传 request 至绑定在 80 的 haproxy；

-haproxy 会在 80 端口接收 HTTPclientrequest，在同一个端口（80）上解析来自 stunnel 的 SSLrequests；

-stunnel 增加 X-Forwarded-Forheader（SSL 请求）

- haproxy 在除了来自本地的地址(stunnel)的每个 request 中增加 X-Forwarded-Forheader

==========

4.1 软停止----使用服务器上的文件

==========

这一个方法比较常用，而且实现简单，上传一文件到服务器交由代理进行检查，当你想要停止服务的时候，要先移除这个文件。代理发现这个服务器已经失效了，将不会把新的会话转发给服务器，如果使用了 persist 选项，仅仅是已经建立的会话在保持连接，过一会儿，在停止服务时候，将不会收到任何新的连接了。

```
listen 192.168.1.1:80
    mode http
    balance roundrobin
    cookie SERVERID insert indirect
    option httpchk HEAD /running HTTP/1.0
    server webA 192.168.1.11:80 cookie A check inter 2000 rise 2 fall 2
    server webB 192.168.1.12:80 cookie B check inter 2000 rise 2 fall 2
    server webC 192.168.1.13:80 cookie C check inter 2000 rise 2 fall 2
    server webD 192.168.1.14:80 cookie D check inter 2000 rise 2 fall 2
    option persist
    redispatch
```

```
        contimeout 5000
```

描述

--每隔 2 秒钟，代理将会访问服务上的 /running 文件，在检查服务器 2 次后不成功认为该服务器已经失效(即 4 秒钟)

-只有服务器响应 200 或 3XX 响应将才被使用

-如果请求不包含一个 cookie，它会被转发到一个有效的 服务器

-如果一个请求包含一个发生故障的服务器的 cookie，haproxy 将试图尝试连接到达该服务器，让用户完成他的请求。("persist" 选项)

-如果请求的那台服务器服务完全停止，连接将失败，那么代理 会将客户分配到另一台健康的服务器（“redispatch”选项）

在服务器上面使用的操作命令 ：

```
- to start the server :        开启服务
      # /etc/init.d/httpd start
      # touch /home/httpd/www/running
- to soft-stop the server    软停止服务
      # rm -f /home/httpd/www/running
- to completely stop the server : 等没有客户端连接的时候停止服务
      # /etc/init.d/httpd stop
```

不足之处

如果服务器已经断电，代理将会尝试为有此服务器 cookie 的客户端转发请求到此服务器，连接将试图尝试 5 秒钟（由 ”contimeout“选项值决定的），在此尝试操作过程之后，会把客户端请求转发到其他健康的服务器，所以这种模式仅仅用于软件的升级，在升级过程中，服务的进程将停止，会导致客户端的连接无效。同样的问题是在服务器宕机情况下，将会对已经建立会话请求的用户造成波动。

4.2 使用备份服务器软停止

一个比较好的解决方法是为每一种情况都使用备用服务器，1.1.30 版本已经修正了这个 bug，即无法使备份服务器和正常服务器共享相同的 cookie。

```
    listen 192.168.1.1:80
        mode http
        balance roundrobin
        redispatch
        cookie SERVERID insert indirect
        option httpchk HEAD / HTTP/1.0
        server webA 192.168.1.11:80 cookie A check port 81 inter 2000
        server webB 192.168.1.12:80 cookie B check port 81 inter 2000
        server webC 192.168.1.13:80 cookie C check port 81 inter 2000
        server webD 192.168.1.14:80 cookie D check port 81 inter 2000

        server bkpA 192.168.1.11:80 cookie A check port 80 inter 2000 backup
        server bkpB 192.168.1.12:80 cookie B check port 80 inter 2000 backup
        server bkpC 192.168.1.13:80 cookie C check port 80 inter 2000 backup
```

```
server bkpD 192.168.1.14:80 cookie D check port 80 inter 2000 backup
```

描述

4 台服务器 webA..D 将每隔 2 秒检测一次 81 端口，与此同名的 bkpA..D 将检测的是 80 端口，切共享同一个 cookie。在没有其他服务器可用的情况下，备份服务器将使用相同的 cookie。

当 web 服务启动的时候，备份服务器也立即可用了。在服务器上面，你需要将 81 端口转发到 80 端口，或者与本地代理（例如：一个简单的 haproxy TCP 的实例），或使用 iptables（Linux）或 PF（OpenBSD 的）。这是因为我们希望真正的 Web 服务（检测）在此端口上面，而不是假的。

例如，用 iptables 的：

```
# /etc/init.d/httpd start
# iptables -t nat -A PREROUTING -p tcp --dport 81 -j REDIRECT --to-port 80
```

几秒钟之后，被认为是正常的服务器将会生效，且 haproxy 开始发送新的请求到
其真正的 80 端口（仅在新用户没有 cookie 的时候）

如果一台服务器完全宕机（即使在 IP 层不响应），正常和备份服务器都将失效，所以客户端请求到到这个服务器的会话将重新分派到其他存活的服务器，会话将不会连接到宕机服务器。

假如你想对服务器进行维护操作，只需停止从 81 端口的响应，其正常的实例将被视为失败，但备份仍然可以工作。将不会对已连接的用户服务产生任何影响：

```
# iptables -t nat -D PREROUTING -p tcp --dport 81 -j REDIRECT --to-port 80
```

这个服务器 81 端口的健康检查，很快就会失效，并且正常服务器将被视为失效。没有新的会话将被发送到这个服务器，一个有效的 cookie 的现有客户仍将达到，因为备份服务器将仍然存在。

在等待一段 时间后，已经建立连接的用户不在对此服务器发送数据，观察到网卡没有接收到新的数据包请求的时候，即可停止服务，命令如下：

```
# /etc/init.d/httpd stop
```

相对应的备份服务器将会失效，如果任何客户端仍试图访问这个特定的服务器，他会被发运到其它有效的服务器，因为使用了“redispatch”的选项。

这种方法有一个优点：你不用接触到 proxy 服务器（haproxy）的维护。管理服务器的人，（只需在 web 服务器上面操作），即可使故障转移顺利地通过。

4.2.1 不使用防火墙的操作变化

缺点是，你只需要在服务器对重定向（端口）做健康检查。如果服务器操作系统不支持任何防火墙软件，那么，这种重定向，可以由简单的 haproxy 在 TCP 模式处理

```
global
        daemon
        quiet
        pidfile /var/run/haproxy-checks.pid
 listen 0.0.0.0:81
        mode tcp
        dispatch 127.0.0.1:80
        contimeout 1000
        clitimeout 10000
        srvtimeout 10000
```

To start the web service： 开启 web 服务

```
# /etc/init.d/httpd start
# haproxy -f /etc/haproxy/haproxy-checks.cfg
```

To soft-stop the service :停止 web 服务

```
# kill $(</var/run/haproxy-checks.pid)
```

81 端口将停止响应时在负载平衡器此服务器将会失效

4.2.2 集中服务器管理

一个认为比较好的管理方式是对 haproxy 服务器本身做管理，那么，端口重定向可以被安装在 haprxy 负载均衡器本身上。例如下面的 iptables

Make the servers appear as operational： 使服务生效的操作

```
# iptables -t nat -A OUTPUT -d 192.168.1.11 -p tcp --dport 81 -j DNAT --to-dest :80
# iptables -t nat -A OUTPUT -d 192.168.1.12 -p tcp --dport 81 -j DNAT --to-dest :80
# iptables -t nat -A OUTPUT -d 192.168.1.13 -p tcp --dport 81 -j DNAT --to-dest :80
# iptables -t nat -A OUTPUT -d 192.168.1.14 -p tcp --dport 81 -j DNAT --to-dest :80
```

Soft stop one server： 软停止服务

```
# iptables -t nat -D OUTPUT -d 192.168.1.12 -p tcp --dport 81 -j DNAT --to-dest :80
```

另一种方案是使用“COMAFILE”补丁由亚历山大 Lazic 提供，可以在这里下载：http://w.ods.org/tools/haproxy/contrib/

4.2.3 注意事项

千万不要把健康检查对一个不是真实服务的 81 端口做检查，只要假的服务在运行，那么一个真正的 web 在时效后不会被检查到，你需要将做健康检查的端口（81）转发到真实存在的 web 服务的应用上面

- 健康检查，将持续发送两次，一次为每个正常服务器，一次每个备份服务器。这一切都将通过 multiplicated

如果您使用的是多进程模式。你必须确保 发送到服务器的所有检查之前不重新加载配置。

4.3 热加载配置文件

使用 haproxy 的用户有两种

- those who can never do anything in production out of maintenance periods ;

一类用户是除基本的维护之外无法对配置进行任何改变的

- those who can do anything at any time provided that the consequences are limited.

一类用户是在有限的操作权限内，可以对配置进行任何的操作

第一类的用户，在维护期间，他们能做任何的操作，如可以先停止服务再来重载配置文件，因此，他们可以做一个停止/启动的操作，将在下一次启动的时候重载配置文件，这些代表了大多数的 haproxy 使用情况，下面试图改善这一情况。

然而，第二类用户就和第一类用户有所不同了，他们在修正配置文件的时候，不会引起任何的察觉，对第一类用户，他们的做法不是稳妥的方法，因为有时会引起用户服务的波动。

正是这个原因，在 1.1.34 版本中，引入了一个热加载配置文件的方法。使用起来比较简单，即将运行环境设定在 chroot 里面并赋予较低的权限。根据 SIGTTOU 信号的接收，代理将停止监听到所有的端口。这将释放（监听）的端口，这样，就可以开始一个新的实例。现有的所有连接将不会被中断。如果新的实例无法启动，然后回到原来的进程发送 SIGTTIN 信号将恢复监听端口。没有任何特殊的权限，因为 sockets 连接并没有被关闭，所以在 bind（）仍然是有效的。否则，如果新的进程启动成功，然后发送一个 SIGUSR1 信号，旧的信号进程，将在最后的会话连接后立即结束。

热重载脚本如下：

```
# save previous state
mv /etc/haproxy/config /etc/haproxy/config.old
mv /var/run/haproxy.pid /var/run/haproxy.pid.old

mv /etc/haproxy/config.new /etc/haproxy/config
```

```
        kill -TTOU $(cat /var/run/haproxy.pid.old)
        if haproxy -p /var/run/haproxy.pid -f /etc/haproxy/config; then
           echo "New instance successfully loaded, stopping previous one."
           kill -USR1 $(cat /var/run/haproxy.pid.old)
           rm -f /var/run/haproxy.pid.old
           exit 1
        else
           echo "New instance failed to start, resuming previous one."
           kill -TTIN $(cat /var/run/haproxy.pid.old)
           rm -f /var/run/haproxy.pid
           mv /var/run/haproxy.pid.old /var/run/haproxy.pid
           mv /etc/haproxy/config /etc/haproxy/config.new
           mv /etc/haproxy/config.old /etc/haproxy/config
           exit 0
     fi
```

在此之后，你可以发送一个 SIGTERM 信号结束一个老的连接。如下：

```
    kill $(cat /var/run/haproxy.pid.old)
    rm -f /var/run/haproxy.pid.old
```

在多进程的模式下，需要注意，一些 pids 可能被重新分配到完全不同的进程了。

5.本地优先的多站点负载均衡

5.1 问题描述

设想一下一个公司的站点分布在世界各地，有两个主要应用站点 Site1 和 Site2 的主机具有相同的应用程序。他们在世界各地有设置的有办事处。出于通信速度和成本的原因，每个办事处将默认访问最近的站点，但在网站或者应用程序发生故障的时候，可以切换到备份站点。对各办事处的用户，默认是访问本地的站点，但当本地站点或者应用程序发生故障的时候，将切换到其他地点正常的站点访问。

主要制约因素是：

- 应用程序的持久性：虽然是两个站点是相同的应用程序 ，但两个站点的会话并不同步。失败一台服务器或一个网站，可以导致用户切换到另一台服务器 或网站，但服务器或网站恢复的时候，用户将不能保持原先得会话。
- 通信费用：站点间的通信应减少到 最低限度。具体来说，在本地应用程序发生故障的情况下，每个办公区应该可以切换到其他网站，不继续使用默认站点。

5.2 解决办法

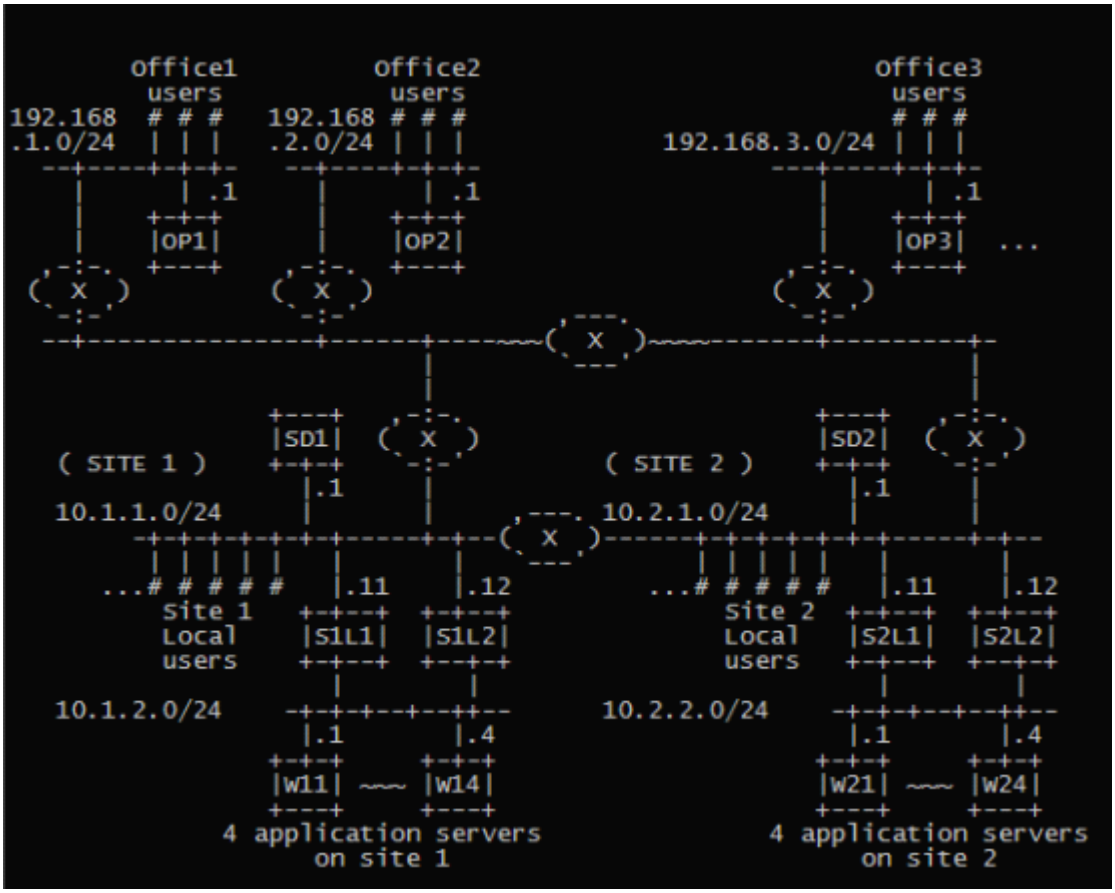
- 每个办事点在应用程序的前面放 2 个 haproxy 负载平衡器提供负载并提供高可用我们把他们叫做“S1L1”和“S1L2”在站点 1，“S2L1”和的“S2L2"站点 2。这些代理将扩展应用程序把 JSESSIONID cookie 加在服务器的名称作为前缀。

- 每个办事点将有一个前端的 haproxy director 为本地用户或者远程办公的用户提供服务。它将负载平衡横跨两个本地负载平衡器，并使用其他站点负载均衡器的作为备份服务器。它会插入本地站点标识符在网站的 cookie，为本地负载平衡器和远程站点标识符远程负载平衡器。这些 front-end 会被称为“SD1”和“SD2”为“site Director”。
- 每个办事点在靠近边界网关的地方有一个 haproxy，默认情况下，将本地用户转发到就近访问，本地失

效的时候转发到备份站点访问， 它还将分析站点的 cookie，用户直接在 cookie 中引用的网站。因此，首选网站将被宣布为一个正常的服务器，备份站点将 声明仅作为备份服务器，这将只用于当主 网站无法访问，或当主站点的 director 已转发 交通的第二个站点。这些代理将被称为“OP1”。“OPXX”，“Office Proxy # XX”

5.3 网络拓扑

注意：Office1 和 2 在相同的连接上面 site1，office3 连接在 site3 上面，每个区域可以通过第二个广域网的专用链接连接



5.4 描述

5.4.1 本地用户

-Office 1 的用户连接到 OP1=192.168.1.1

-Office 2 的用户连接到 OP2=192.168.2.1

-Office 3 的用户连接到 OP3=192.168.3.1

- Site 1 的用户连接到 SD1 = 10.1.1.1

- Site 2 的用户连接到 SD2 = 10.2.1.1

5.4.2 办公代理

- Office 1 connects to site 1 by default and uses site 2 as a backup.
- Office 2 connects to site 1 by default and uses site 2 as a backup.
- Office 3 connects to site 2 by default and uses site 1 as a backup.

办公点每隔 30 秒检测一次本地站点代理 SD，每隔 60 秒检测一次远程代理

Configuration for Office Proxy OP1

```
listen 192.168.1.1:80
```

```
        mode http
        balance roundrobin
        redispatch
        cookie SITE
        option httpchk HEAD / HTTP/1.0
        server SD1 10.1.1.1:80 cookie SITE1 check inter 30000
        server SD2 10.2.1.1:80 cookie SITE2 check inter 60000 backup
```

Configuration for Office Proxy OP2

```
    listen 192.168.2.1:80
        mode http
        balance roundrobin
        redispatch
        cookie SITE
        option httpchk HEAD / HTTP/1.0
        server SD1 10.1.1.1:80 cookie SITE1 check inter 30000
        server SD2 10.2.1.1:80 cookie SITE2 check inter 60000 backup
```

Configuration for Office Proxy OP3

```
    listen 192.168.3.1:80
        mode http
        balance roundrobin
        redispatch
        cookie SITE
        option httpchk HEAD / HTTP/1.0
        server SD2 10.2.1.1:80 cookie SITE2 check inter 30000
        server SD1 10.1.1.1:80 cookie SITE1 check inter 60000 backup
```

5.4.3 Site directors (SD1 and SD2)

该站点的调度器将流量转发到本地的负载均衡器，并设置 cookie 来识别站点，如果本地的服务不可用，或者是本地应用服务失效，将会把流量转发到远程，并且告诉此网站的 cookie。为了不白白加载每个站点的广域网链路，每个 SD 在检查其他站点的健康情况时使用一个较少的开销速率，站点的调度器会插入客户端的来源 ip 地址，以确保应用服务知道是哪个远端用户来访问。

站点的 cookie 是前端调度设置的，office 的代理器也可以识别到。这一点非常重要，例如 SD1 决定将请求转发给 site2，将会在”SITE“的 cookie 插入”SITE2“的标记，在下一次请求的时候，假如 SITE2 可达，office 的代理器将请求自动转发给 SITE2。如果它不能，它仍然会发送流量到 SITE1，其中 SD1 的将打开力争达到 SITE2 中的。

负载均衡器将检查 81 端口，正如我们将进一步看到，负载平衡器提供了健康监测端口 81 转发到 80 端口，但允许他们告诉 SD，他们会下降很快，那么，SD，将不再使用它们了。

Configuration for SD1

```
    listen 10.1.1.1:80
        mode http
        balance roundrobin
        redispatch
```

```
        cookie SITE insert indirect
        option httpchk HEAD / HTTP/1.0
        option forwardfor
        server S1L1 10.1.1.11:80 cookie SITE1 check port 81 inter 4000
        server S1L2 10.1.1.12:80 cookie SITE1 check port 81 inter 4000
        server S2L1 10.2.1.11:80 cookie SITE2 check port 81 inter 8000 backup
        server S2L2 10.2.1.12:80 cookie SITE2 check port 81 inter 8000 backup
```

Configuration for SD2

```
    listen 10.2.1.1:80
        mode http
        balance roundrobin
        redispatch
        cookie SITE insert indirect
        option httpchk HEAD / HTTP/1.0
        option forwardfor
        server S2L1 10.2.1.11:80 cookie SITE2 check port 81 inter 4000
        server S2L2 10.2.1.12:80 cookie SITE2 check port 81 inter 4000
        server S1L1 10.1.1.11:80 cookie SITE1 check port 81 inter 8000 backup
        server S1L2 10.1.1.12:80 cookie SITE1 check port 81 inter 8000 backup
```

5.4.4 Local load-balancers S1L1, S1L2, S2L1, S2L2

本地负载均衡 S1L1, S1L2, S2L1, S2L2

首先请注意 SD1 的和 SD2 因为都使用相同的 cookie 在同一个网站的服务器，每个站点的第二负载平衡器接收负载平衡的请求，站点的 cookie 被设置，只有第一 LB 将收到的请求，因为它会第一个匹配的 cookie。

负载平衡器将遍布 4 个本地 Web 服务器的负载，使用应用程序提供的 JSESSIONID 提供服务器的持久性使用新的“前缀”的方法。所述，也将实行软停止在上文第 4 条。此外，这些代理会提供自己的维修软停止。 80 端口将用于应用程序的流量，而 81 端口仅可用于健康检查和本地改端口 80。宽限时间将指定的服务端口 80，但不能在 81 端口。这样，软杀（杀 USR1 信号）代理只会杀死健康检查转发现场导演知道它不能再使用此负载平衡器。但该服务将仍然工作 20 秒，只要有建立会议。

这些代理也将是唯一的，禁用 HTTP 保持活动链，因为它是足够做它在一个地方，有必要做它与'前缀'饼干。

Configuration for S1L1/S1L2

```
    listen 10.1.1.11:80 # 10.1.1.12:80 for S1L2
        grace 20000    # don't kill us until 20 seconds have elapsed
        mode http
        balance roundrobin
        cookie JSESSIONID prefix
        option httpclose
```

```
        option forwardfor
        option httpchk HEAD / HTTP/1.0
        server W11 10.1.2.1:80 cookie W11 check port 81 inter 2000
        server W12 10.1.2.2:80 cookie W12 check port 81 inter 2000
        server W13 10.1.2.3:80 cookie W13 check port 81 inter 2000
        server W14 10.1.2.4:80 cookie W14 check port 81 inter 2000

        server B11 10.1.2.1:80 cookie W11 check port 80 inter 4000 backup
        server B12 10.1.2.2:80 cookie W12 check port 80 inter 4000 backup
        server B13 10.1.2.3:80 cookie W13 check port 80 inter 4000 backup
        server B14 10.1.2.4:80 cookie W14 check port 80 inter 4000 backup
listen 10.1.1.11:81 # 10.1.1.12:81 for S1L2
        mode tcp
        dispatch 10.1.1.11:80    # 10.1.1.12:80 for S1L2
```

Configuration for S2L1/S2L2

```
    listen 10.2.1.11:80 # 10.2.1.12:80 for S2L2
        grace 20000    # don't kill us until 20 seconds have elapsed
        mode http
        balance roundrobin
        cookie JSESSIONID prefix
        option httpclose
        option forwardfor
        option httpchk HEAD / HTTP/1.0
        server W21 10.2.2.1:80 cookie W21 check port 81 inter 2000
        server W22 10.2.2.2:80 cookie W22 check port 81 inter 2000
        server W23 10.2.2.3:80 cookie W23 check port 81 inter 2000
        server W24 10.2.2.4:80 cookie W24 check port 81 inter 2000

        server B21 10.2.2.1:80 cookie W21 check port 80 inter 4000 backup
        server B22 10.2.2.2:80 cookie W22 check port 80 inter 4000 backup
        server B23 10.2.2.3:80 cookie W23 check port 80 inter 4000 backup
        server B24 10.2.2.4:80 cookie W24 check port 80 inter 4000 backup

    listen 10.2.1.11:81 # 10.2.1.12:81 for S2L2
        mode tcp
        dispatch 10.2.1.11:80    # 10.2.1.12:80 for S2L2
```

5.5

因为每个站点的 director 都会为每个站点设定一个 cookie，远端 office 的用户使用它们自己的代理服务器将请求转发到正确的站点，并且在应用程序和站点可用的情况下会一直保持连接，用户将在访问应用程序站点的时候，站点的 director 将根据站点的 cookie 转发请求到

响相应的站点。

如果应用程序站点的外网(WAN)连接不通，远程 office 的用户将不会访问此站点了，所以用户的请求流量将被转发到令一个(监控状态的)站点.如果上图所示的中间站点可用，第二个 SD 发现 cookie 存在，仍然会访问原来的站点，例如：

Office 1 发送下面的请求到应用 OP1 :

GET / HTTP/1.0

Cookie: SITE=SITE1; JSESSIONID=W14~123

当外部路由不可达时，OP1 不能到达 site 1，所以 SD1 server 被视为失效，这个时候 OP1 将会转发请求到 SD2 的站点 stie2，而不管网站的 cookie。

SD2 的 site2 收到一个包含“SITE1”的 cookie，幸运的是，可以达到网站 1 的负载平衡器 S1L1 的和 S1L2。因此，将请求转发到 S1L1（与第一个相同的 cookie）

S1L1（在 site1）发现在 JSESSIONID cookie 中有“W14”，因此它可以将请求转发到正确的服务器，用户会话还会继续保持。一旦 site1 的广域网链路恢复正常，OP1 就能连到到 SD1，并不会再路由通过站点 2 了。

然而，当 office 的新用户在一个连接到应用程序 site1 失败时，它不包含任何 cookie。由于网络故障，OP1 是连接不到 SD1 的，它将请求直接转发到 SD2 的 site2，在默认情况下，请求流量将转发到本地负载平衡器 S2L1 和 S2L2。因此，只有最初的用户连接到中间站点，后面新请求的用户则不会连接到中间站点。

6.源地址的负载均衡

有时，它可能会发现用户访问服务器的 IP 地址池而不是只有一个或两个。有些设备（NAT 防火墙，负载均衡器）的源地址明确，往往需要许多来源分发其中包括其内部的 hash bucket 的负载均匀。

要做到这一点，你只需使用几次同一个服务器不同的来源。例如：

```
listen 0.0.0.0:80
    mode tcp
    balance roundrobin
    server from1to1 10.1.1.1:80 source 10.1.2.1
    server from2to1 10.1.1.1:80 source 10.1.2.2
    server from3to1 10.1.1.1:80 source 10.1.2.3
    server from4to1 10.1.1.1:80 source 10.1.2.4
    server from5to1 10.1.1.1:80 source 10.1.2.5
    server from6to1 10.1.1.1:80 source 10.1.2.6
    server from7to1 10.1.1.1:80 source 10.1.2.7
    server from8to1 10.1.1.1:80 source 10.1.2.8
```

7.对高负载的应用服务进行管理

负载平衡器经常扮演的一个角色是，在高流量的时候减轻单台服务器的负载。在很多的时候，我们看到沉重的框架来提供灵活多变的网页设计。有时，在高负载的服务器上，并发却比较低。而且响应时间也相当长的。人们开发的这种网站往往依靠这种框架寻找一个合适的负载平衡器，能够将负载均匀分配在服务器上面，让其更好的工作。

haproxy 有一个强大的功能完全满足这个需求：请求排队相关的并发连接限制。

比方说，你有一个应用程序服务器支持最多 20 个并发请求。你有 3 台服务器，所以你可以接受多达 60 并发 HTTP 连接，这通常意味着在 30 个并发用户的情况下保持活动（每个用户的持续 2 个连接）。

即使你禁用 keep-alive，如果服务器需要很长的时间来响应，你仍然在同一时间多个用户连接的高风险，因为服务器达到极限，无法响应他们的请求。要解决这个问题，你增加服务器上的并发连接的限制，但他们的表现在更高负载降低。

解决的办法是限制客户端到服务器的连接数。您设置的 haproxy 到每一个服务器的连接数限制，对每个服务器的基本连接数进行限制。然后，它会达到每个服务器连接限制的数量，将会把剩余的连接防止队列中，等待一个连接从服务器上释放。

这将确保五项基本原则：

- 所有的客户可以正常访问，在服务器不崩溃时不考虑其连接数量，唯一的影响，响应时间需要延迟。

- 服务器可以达到连接的上限，没有延迟， 通过参数的调整让其达到最佳性能。

- 响应时间，可以减少后端服务器的连接阻塞，有效地转发，使后端响应时间更短，达到一个合适的系统负载。

- 没有多米诺骨牌效应，当一台服务器停机或启动。请求将 排队或多或少，始终保持服务器的限制。

- 这是容易实现的，即使在内存等有限的硬件条件下仍然能达到高性能。 实际上，沉重的框架通常会消耗大量的 RAM，而且 CPU 不总是够用。在一定的故障系数内，减少并发连接的数量 可以防止内存用完，同时还 能确保最佳的 CPU 使用率

举例：

安装在应用服务器前端的 Haproxy，将每个服务器的并发连接数限制为 4（每个 CPU 一个线程），从而保证快速的响应时间

Haproxy LB1 的配置文件

```
listen appfarm 192.168.1.1:80
    mode http
    maxconn 10000
    option httpclose
    option forwardfor
    balance roundrobin
    cookie SERVERID insert indirect
```

```
        option httpchk HEAD /index.html HTTP/1.0
        server railsA 192.168.1.11:80 cookie A maxconn 4 check
        server railsB 192.168.1.12:80 cookie B maxconn 4 check
        server railsC 192.168.1.13:80 cookie C maxconn 4 check
        contimeout 60000
```

说明：

代理监听的 IP 192.168.1.1，端口 80，将接受 HTTP 请求。它可以接受此套接字上到 10000 个并发连接。它遵循 roundrobin 算法分配，只要服务器连接服务器不饱和。

它允许最多每个服务器的 4 个并发连接，高于此值将放于等待队列中。 “contimeout”参数用来设置 establish 连接状态的最大时间，但在这里它被设置用来作为连接处于等待队列中保持的最大时间（1 分钟）。

如果服务器每次可以 4 个并发请求，耗时平均为 10 毫秒，那么并发 3000 的连接，响应时间将被推迟至多：
3000/3 服务器/ 4 连接数* 10 毫秒= 2.5 秒
这对庞大的用户和如此少的服务器来说，似乎是比较荒诞的。

当连接队列填满时，应用程序服务器就会处于等待状态，响应时间会变长长，用户可能通过点击“停止”按钮中止网站访问请求。这是中止请求发送到服务器，是非常无益的，因为它们会消耗 CPU 的利用。

已加入一个选项来处理这一具体情况：选项 abortonclose，通过指定它，你能告诉的 haproxy，如果一个输入通道上的处于 closed 状态请求的客户端的仍然在队列中等待，在用户可能已经停止请求的时候，我们可以从等待队列中将其删除，之前的请求仍然会被送达。

不公平的响应时间管理

有时，应用程序服务器对一些请求的响应会非常缓慢（如：登录页），但对其他的也没请求会快一些。这可能会导致请求队列会比预计的时候要慢，服务器上的所有线程将被阻塞。那么唯一的解决办法是增加的并发连接数，使服务器可以处理大量的能快速处理的连接线程，二不被处理速较慢的连接线程锁影响。

但是，正如我们看到的，服务器上不断增长的并发连接数，将会是一个不好的表现（例如：Apache 进程的达到上限（lock）。为了避免这种状况，可以应用“minconn”参数。当它被设置，将服务器上的最大连接并发约束此值，将增加与客户的等待人数和限制在队列中，直到客户端连接到的 haproxy 达到代理的 maxconn，这种情况下，将达到每服务器连接服务器的 maxconn。这意味着在低到中等负载，将应用于 minconn，在高负载重 maxconn 将被应用。它确保两个最佳的响应时间正常负载和非常高的负载下的可用性。

举例

```
    listen appfarm 192.168.1.1:80
        mode http
```

```
    maxconn 10000
    option httpclose
    option abortonclose
    option forwardfor
    balance roundrobin
    # The servers will get 4 concurrent connections under low
    # loads, and 12 when there will be 10000 clients.
    server railsA 192.168.1.11:80 minconn 4 maxconn 12 check
    server railsB 192.168.1.12:80 minconn 4 maxconn 12 check
    server railsC 192.168.1.13:80 minconn 4 maxconn 12 check
  contimeout 6000
```