最强lvs总结

LVS的全称Linux vitual system, linux vitual server
是由目前阿里巴巴的著名工程师章文嵩博士开发的一款开源软件。LVS工作在一台server上提供Directory(负载均衡器)的功能，本身并不提供服务，只是把特定的请求转发给对应的realserver(真正提供服务的主机)，从而实现集群环境中的负载均衡。
LVS的核心组件ipvs工作在kernel中，是真正的用于实现根据定义的集群转发规则把客户端的请求转发到特定的realserver。而另一个组件ipvsadm是工作在用户空间的一个让用户定义ipvs规则的工具。故我们只要在server上装了ipvsadm软件包就可以定义ipvs规则，
而在linux kernel的2.6版本之后kernel是直接支持ipvs的。
注：由于ipvs是接受netfilter五个钩子函数的中的local_in函数控制的。故ipvs不能和netfilter的一些控制规则同时使用。

三种模式
DR直接路由模式、NAT转发模式、IP隧道模式
十几种算法
轮训算法 加权轮训算法 最小连接算法 加权最下连接算法 .....
LVS Scheduling Method LVS的调度方法:
1.Fixed Scheduling Method 静态调服方法
(1).RR 轮询
(2).WRR 加权轮询
(3).DH 目标地址hash
(4).SH 源地址hash
2.Dynamic Scheduling Method 动态调服方法
(1).LC 最少连接
(2).WLC 加权最少连接
(3).SED 最少期望延迟
(4).NQ 从不排队调度方法
(5).LBLC 基于本地的最少连接
(6).LBLCR 带复制的基于本地的最少连接

模式介绍
基于DR直接路由来实现。当用户请求到达director之后，director将请求报文的目标地址（即VIP）改成选定的realserver地址，还要改写请求报文的mac地址，将请求发送到指定mac的realserver，而realserver将响应直接返回给客户端，不经过director。这个方式是三种调度中性能最好的，也是我们生产环境中使用最多的。

当一个client发送一个WEB请求到VIP，LVS服务器会根据算法让这次请求的VIP选择对应的real-server，然后LVS服务器会在hash表中记录该次连接，并基于mac地址进行转发请求数据包，将client的请求包发给选择的Real-server，最后选择的Real- server把应答包直接传给client；当client继续发包过来时，LVS根据更才记录的hash表的信息，将属于此次连接的请求直接发到刚才选 择的Real-server上；当连接中止或者超时，hash表中的记录将被删除。

由于DR模型在内网中基于mac地址进行转发请求数据包，并且我们的

director和realserver都配有一个VIP地址，故我们要限制realserver的arp通告和arp响应级别，以保证我们数据包能到达director指定要发送的realserver。而我们的linux系统提供了这样的功能，通过修改kernel的两个参数来控制arp的级别。
arp_announce = 2 表示只宣告arp所请求的那个地址和本机接受arp请求的那个网卡在同一个子网内的那个地址
arp_ignore = 1 表示只响应arp的目标地址和接收的网卡的地址相同的arp请求

配置lo别名，并且定义lo:0的广播域为本网卡，使VIP不能向网络内发送广播，以防止网络出现混乱：
ifconfig lo:0 192.168.1.1 broadcast 192.168.1.1 netmask 255.255.255.255
route add -host 192.168.1.1 dev lo:0

特点：
1，集群节点和director必须在一个物理网络内
2，RIP可以使用公网地址或私有地址
3，director仅处理入站请求
4，集群节点网关不指向director，故出站不经过director
5，不支持端口映射
6，大多数操作系统可以作为realserver，要支持隔离arp广播
7，director服务器的压力比较小

NAT模式工作原理：
1> client发送request到LVS的VIP上，VIP选择一个Real-server，并记录连接信息到hash表中，然后修改client的request的目的IP地址为Real-server的地址，将请求发给Real-server;
2> Real-server收到request包后，发现目的IP是自己的IP，于是处理请求，然

后发送reply给LVS;
3> LVS收到reply包后，修改reply包的的源地址为VIP，发送给client;
4> 从client来的属于本次连接的包，查hash表，然后发给对应的Real-server。
5> 当client发送完毕，此次连接结束或者连接超时，那么LVS自动从hash表中删除此条记录

基于NAT机制实现。当用户请求到达director之后，director将请求报文的目标地址（即VIP）改成选定的realserver地址，同时将报文的目标端口也改成选定的realserver的相应端口，最后将报文请求发送到指定的realserver。在服务器端得到数据后，realserver将数据返给director，而director将报文的源地址和源端口改成VIP和相应端口，然后把数据发送给用户，完成整个负载调度过程。
特点:
1，所有的realserver和director要在同一个网段内
2，RIP是私有地址，仅用于集群节点之间进行通信
3，director同时处理请求和应答数据包
4，realserver的网关要指向DIP
5，可以实现端口映射
6，readlserver可以是任意操作系统
缺点:
director很可能成为系统性能瓶颈，所有的请求director都需要处理应答

IP隧道模式:
1) 工作过程:
1> client 发送request包到LVS服务器的VIP上。
2> VIP按照算法选择后端的一个Real-server，并将记录一条消息到hash表

中，然后将client的request包封装到一个新的IP包里，新IP包的目的IP是Real-server的IP，然后转发给Real-server。
3> Real-server收到包后，解封装，取出client的request包，发现他的目的地址是VIP，而Real-server发现在自己的 lo:0口上有这个IP地址，于是处理client的请求，然后将relpy这个request包直接发给client。
4> 该client的后面的request包，LVS直接按照hash表中的记录直接转发给Real-server，当传输完毕或者连接超时，那么将删除hash表中的记录。

工作原理：这种方法通过ip隧道技术实现虚拟服务器。当用户请求到达director之后，director将请求报文的目标地址（即VIP）改成选定的realserver地址.然后，调度器采用ip隧道技术将用户请求发送到某个realserver，而这个realserver将直接相应用户的请求，不再经过director。此外，对realserver的地域位置没有要求。和director在不在同一网段都可以。
特点：
1，realserver和director可以不在一个物理网络中，可以跨越互联网
2，RIP一定不能是私有地址（因为要用到隧道传输）
3，director仅处理入站请求
4，realserver的网关不能指向DIP
5，不支持端口映射
6，支持ip隧道功能的操作系统才能作为realserver

lvs + keepalived参数优化
IPVS connection hash table 20 默认是12
好多人误认为哈希表的大小=这台设备连接数的大小吗?
实际不是的，哈希表是记录了每个进来的连接及路由取向信息的
哈希表是用来解救哈希冲突的，当大于配置的哈希表大小的时候
只会降低性能，并不会去功能产生影响。
keepalived优化参数：
lvs_sync_daemon_interface bond0 实时同步主上的会话到备节点，配合抢占延迟会大幅度降低主备宕切所产生的会话重置。
preempt_delay 200   master启动多久之后进行接管资源（VIP/Route信息

等），给备机同步会话时间，保证能把所有当前的会话接管过去。
inhibit_on_failure
当检测RS1失效后将权重标记为0，并且后面的请求不会调度到挂掉的RS1上面。
当RS1在哈希表内存储的会话过期期间 恢复的时候，那些没有过期的会话还会继续使用起来，同时后续的会话会继续调度到这台RS上使用。
应用场景，在高峰期间发现有大量的RS会被踢出在加上，大大影响了服务。
网络优化：

irqbalance 分为两种模式 一个是性能模式 一个是省电模式
性能模式会将中断尽可能均匀地分发给各个CPU核心
省电模式时irqbalance会将中断集中分配给第一个CPU
但是一般呢 我们的专用的应用程序通常是绑定在特定的CPU上的 所以我们这里要把他关闭掉。

ARP问题：
通常，DR模式需要在Real-server上配置VIP，配置的方式为：
/sbin/ifconfig lo:0 inet VIP netmask 255.255.255.255

i) 原因在于，当LVS把client的包转发给Real-server时，因为包的目的IP地址是VIP，那么如果Real-server收到这个包后，发现包的目的IP不是自己的系统IP，那么就会认为这个包不是发给自己的，就会丢弃这个包，所以需要将这个IP地址绑到网卡上；当发送应答包给client时，Real-server就会把包的源和目的地址调换，直接回复给client。

ii) 关于ARP广播：
* 上面绑定VIP的掩码是”255.255.255.255″ ，说明广播地址是其本身，那么他就不会将ARP发送到实际的自己该属于的广播域了，这样防止与LVS上VIP

冲突，而导致IP冲突。
* 另外在Linux的Real-server上，需要设置ARP的sysctl选项:（下面是举例说明设置项的）

lvs监控方面
监控keepalived服务进程
监控keepalived日志获取主备宕切 及 RS踢出告警
监控lvs设备的流量 最近发现lvs设备流入达到了700+M
LVS 的 bonding 是否有实际作用?

解释:
地址解析协议，即ARP（Address Resolution Protocol），是根据IP地址获取物理地址的一个TCP/IP协议。主机发送信息时将包含目标IP地址的ARP请求广播到网络上的所有主机，并接收返回消息，以此确定目标的物理地址。
报文(message)是网络中交换与传输的数据单元，即站点一次性要发送的数据块。报文包含了将要发送的完整的数据信息，其长短很不一致，长度不限且可变。

问题:
lvs可以实现三线吗？可以实现 使用第三种IP隧道实现 我们为什么没有做？
效率低，特殊设备支持IP隧道
RS被踢出前有大量的session那么，被踢出后当前在这台RS的session会如何处理？处理吗？会过期吗？在过期前恢复会被重新使用吗?
主备都挂了咋整？主挂了备没接管咋整?

坑：
经常碰到的坑rs之间不再同一个vlan 或者 lvs和rs不再同一个vlan子网里，导致realserver收不到lvs机做的转发
新员工处理RS 重启网卡后 没有执行lo:0 路由 虚拟vip
RS经常被踢出

为什么到现在还没有做？自身问题，当运维不合格！

自我反省 改正！

一、LVS1和LVS2均正常工作时

LVS1先启动keepalived后，角色自动为Master，通过ipvsadm -lcn看到如下信息：

```
[root@LVS1 ~]# ipvsadm -lcn
IPVS connection entries
pro expire state       source             virtual            destination
UDP 04:59  UDP         192.168.1.152:42433 192.168.1.160:5000 192.168.1.142:5000
UDP 04:59  UDP         192.168.1.151:34127 192.168.1.160:5000 192.168.1.143:5000
UDP 05:56  UDP         192.168.1.152:0     192.168.1.160:5000 192.168.1.142:5000
UDP 05:52  UDP         192.168.1.151:0     192.168.1.160:5000 192.168.1.143:5000
```

主机的连接列表中的expire列在非0端口行（上面两行）中为5分钟，此时间应该对应“ipvsadm -l --timeout”中的udp时间；0端口（下面两行）中为6分钟，此时间应该对应keepalived配置文件中的“persistence_timeout 360”，但此6分钟在逐渐减少后又会自动恢复为1分钟（此1分钟时间不知哪里可以设置？），因此这些0端口的连接项在主机中一直存在；

然后启动LVS2的keepalived后，角色自动为Backup，看到如下信息：

```
[root@LVS2 ~]# ipvsadm -lcn
IPVS connection entries
pro expire state       source             virtual            destination
UDP 04:38  UDP        192.168.1.151:34127 192.168.1.160:5000 192.168.1.143:5000
UDP 04:42  UDP        192.168.1.152:42433 192.168.1.160:5000 192.168.1.142:5000
UDP 02:38  UDP        192.168.1.151:0     192.168.1.160:5000 192.168.1.143:5000
UDP 02:42  UDP        192.168.1.152:0     192.168.1.160:5000 192.168.1.142:5000
```

备机的连接列表中的expire列在非0端口行（上面两行）中为5分钟，此时间应该对应“ipvsadm -l --timeout”中的udp时间；0端口（下面两行）中为3分钟（此3分钟时间不知哪里可以设置？），但此3分钟在逐渐减少后又会自动恢复，因此这些0端口的连接项在备机中一直存在；

二、当LVS1宕机后（如模拟直接关机），LVS2自动转换为新的Master

由于两边之前的连接列表内容同步，因此在切换LVS后，Client依然保持和先前的RealServer的连接通路；但此时在LVS2上查看“ipvsadm -lcn”如下：

```
[root@LVS2 ~]# ipvsadm -lcn
IPVS connection entries
pro expire state       source             virtual            destination
UDP 04:59  UDP        192.168.1.151:34127 192.168.1.160:5000 192.168.1.143:5000
```

UDP 04:59 UDP   192.168.1.152:42433 192.168.1.160:5000 192.168.1.142:5000

原先0端口的两行记录在3分钟倒计时逐渐减少后将不再恢复直至消除。是否有方法在不断开Client的连接情况下可以生成同样链路的0端口记录？
此时Client与RealServer之间的连接通路还会继续保持。

vrrp_instance VI_1 {   //定义vrrp实例
state MASTER   //主LVS是MASTER,从的BACKUP
interface eth0   //LVS监控的网络接口
virtual_router_id 51 //同一实例下virtual_router_id必须相同   tcpdump -i eth0 -nn vrrp 抓包看不能相同
priority 100   //定义优先级，数字越大，优先级越高
advert_int 5   //MASTER与BACKUP负载均衡器之间同步检查的时间间隔，单位是秒
authentication {   //验证类型和密码
auth_type PASS
auth_pass 1111
}

virtual_server 192.168.1.8 80 {   //定义虚拟服务器
delay_loop 6   //健康检查时间，单位是秒
lb_algo rr   //负载调度算法，这里设置为rr，即轮询算法
lb_kind DR   //LVS实现负载均衡的机制，可以有NAT、TUN和DR三个模式可选
persistence_timeout 50   //会话保持时间，单位是秒（可以适当延长时间

以保持session）
protocol TCP　　//转发协议类型，有tcp和udp两种
sorry_server 127.0.0.1 80　　//web服务器全部失败，vip指向本机80端口

real_server 192.168.1.16 80 { //定义WEB服务器
weight 1　　//权重
TCP_CHECK {　　//通过tcpcheck判断RealServer的健康状态
connect_timeout 5　　//连接超时时间
nb_get_retry 3　　//重连次数
delay_before_retry 3 //重连间隔时间
connect_port 80　　//检测端口
}
}

virtual_server 60.55.33.81 80 {　　# hntv
persistence_timeout 600
lb_kind DR
lb_algo wrr
protocol TCP
delay_loop 3

vrrp_instance VI_1 {
state BACKUP
interface em1
virtual_router_id 14
priority 150
advert_int 1
authentication {
auth_type PASS

auth_pass fastweb678
}

网卡中断

在网卡多队列开启的情况下，可以通过cat /proc/interrupts来看看eth的那些，通常会看到eth0-TxRx-0等，这表明是打开了的，网卡的多队列个数是固定的，因此可能会出现网卡的队列个数和cpu核数不对应的现象，网卡驱动在初始化的时候会根据cpu core数来决定绑定（有些网卡驱动不一定是这样，所以最好确认下），也可以自己手工修改/proc/irq/[中断号]/smp_affinity来绑定。

net.core.netdev_max_backlog = 30000 每个网络接口接收数据包的速率比内核处理这些包的速率快时,允许送到队列的数据包的最大数目

分析rate
ipvsadm -ln --rate | grep TCP | sort -k6rn | head; ipvsadm -ln --rate | grep TCP | sort -k6rn | awk '{sum+=$6} END {print sum}' ; ipvsadm -ln --rate | grep TCP | sort -k6rn | head > bps_top; for IP in $(cat bps_top | cut -d' ' -f3 | cut -d':' -f1); do ip -4 addr | grep --color=auto $IP/; done