* LVS简介及工作原理

LVS是Linux Virtual Server的简写，意即Linux虚拟服务器，是一个虚拟的服务器集群系统。本项目在1998年5月由[章文嵩](http://baike.baidu.com/view/3160926.htm" \t "_blank)博士成立，是中国国内最早出现的自由软件项目之一。

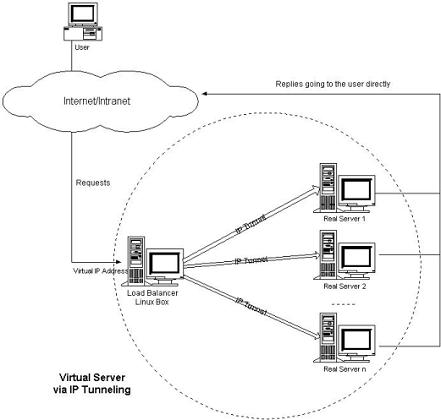
**LVS 简单工作原理：用户请求LVS VIP，LVS根据转发方式和算法，将请求转发给后端服务器，后端服务器接受到请求，返回给用户。对于用户来说，看不到WEB后端具体的应用。**

**LVS转发方式有三种，分别是NAT、DR、TUN模式，常用算法：RR、LC、WRR、WLC模式等（RR为轮询模式，LC为最少连接模式）**

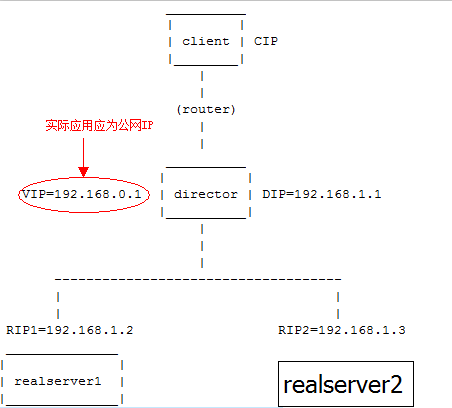
**LVS NAT原理：用户请求LVS到达director,director将请求的报文的目标地址改成后端的realserver地址，同时将报文的目标端口也改成后端选定的realserver相应端口，最后将报文发送到realserver，realserver将数据返给director，director再把数据发送给用户。（两次请求都经过director，所以访问大的话，director会成为瓶颈）**

LVS DR原理：用户请求LVS到达director,director将请求的报文的目标MAC地址改成后端的realserver MAC地址，目标IP为VIP（不变），源IP为用户IP地址（保持不变），然后Director将报文发送到realserver，realserver检测到目标为自己本地IP，如果在同一个网段，然后将请求直接返给用户。如果用户跟realserver不在一个网段，则通过网关返回用户。（此种转发效率最高）

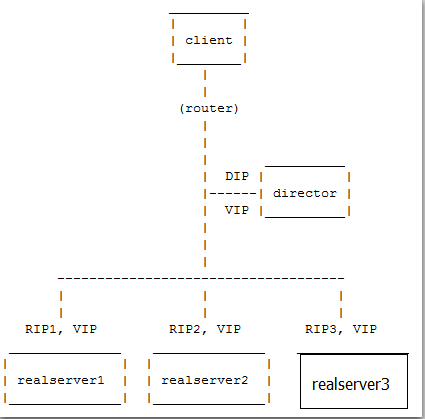
LVS TUN原理：跟LVS DR类似，也是改变封装MAC地址，多了一层隧道加密。实施环境复杂，比LVS DR模式效率略低。（图一为LVS负载均衡图）



**LVS-NAT模式图：**

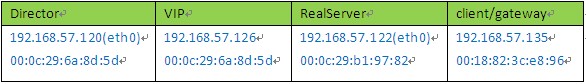


**LVS-DR模式图：**

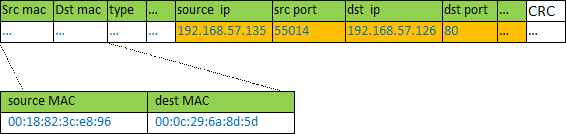


**LVS-DR模式原理详解图：**

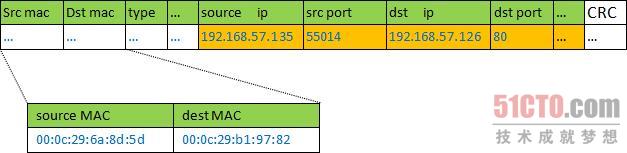
**原图：**



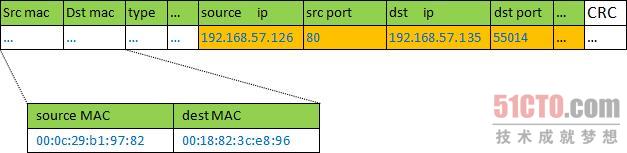
**第一步请求：**



**第二步传输：**



**第三步数据返回：**



* LVS环境安装配置

下载LVS所需软件ipvsadm-1.2.4.tar.gz软件，编译安装：

wget -c <http://www.linuxvirtualserver.org/software/kernel-2.6/ipvsadm-1.24.tar.gz>

ln -s /usr/src/kernels/2.6.\* /usr/src/linux //IPVS模块编译进内核里，需要做软连接

tar xzvf ipvsadm-1.24.tar.gz &&cd ipvsadm-1.24 && make && make install

LVS安装完毕之后，需要进行配置，配置的步骤有两步，第一步为定义端口服务，第二步为添加realserver后端服务。

ipvsadm -A -t 192.168.33.188:80 -s rr

ipvsadm -a -t 192.168.33.188:80 -r 192.168.33.12 -m -w 2

ipvsadm -a -t 192.168.33.188:80 -r 192.168.33.13 -m -w 2

参数说明：

-A 增加一台虚拟服务器地址。

-t [虚拟服务器](http://baike.baidu.com/view/319978.htm)提供的是tcp服务。

-s 使用的[调度算法](http://baike.baidu.com/view/2963962.htm" \t "_blank)。

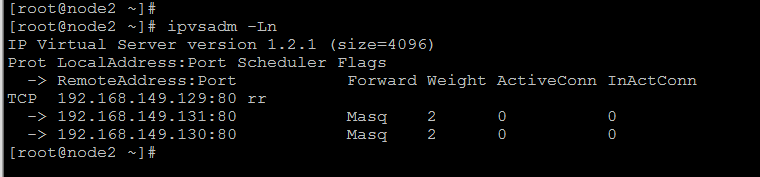
-a 在[虚拟服务器](http://baike.baidu.com/view/319978.htm" \t "_blank)中增加一台后端真实服务器。

-r 指定真实服务器地址。

-m 设置当前转发方式为NAT模式；-g为直接路由模式；-i 模式为[隧道模式](http://baike.baidu.com/view/3399426.htm" \t "_blank)。

-w 后端真实服务器的权重。

查看LVS转发列表命令为：ipvsadm –Ln



我们会发现，如果这台LVS发生突发情况，down机了，那后端所有的应用程序都访问不了。如何避免这种问题呢，这里需要用到故障切换，也就是如果有一台备用的LVS就好了，主down了，自动切换到从，怎么实现这个需求，接下来讲解的keepalived软件就是专门用来做故障检测及切换的。

Keepalived基于三层检测（IP层，TCP层，及应用层），主要用于检测web服务器的状态，如果有一台web服务器死机，或工作出现故障，Keepalived检测到并将有故障的web服务器从系统中剔除；

当web服务器工作正常后Keepalived自动将web服务器加入到服务器群中，这些工作全部自动完成，不需要人工干涉，需要人工做的只是修复故障的web服务器。

需要注意一点，如果使用了keepalived.conf配置，就不需要再执行ipvs –A命令去添加均衡的realserver命令了，所有的配置都会在keepalived.conf里面，一个配置文件搞定所有，即只需要安装ipvs模块。

* Keepalived安装配置

官方下载keepalived相应稳定版本：

cd /usr/src ;wget -c <http://www.keepalived.org/software/keepalived-1.1.15.tar.gz>

tar -xzvf keepalived-1.1.15.tar.gz &&cd keepalived-1.1.15 && ./configure && make && make install

安装完毕，配置keepalived服务为系统服务。

DIR=/usr/local/

cp $DIR/etc/rc.d/init.d/keepalived /etc/rc.d/init.d/ && cp $DIR/etc/sysconfig/keepalived /etc/sysconfig/ && mkdir -p /etc/keepalived && cp $DIR/sbin/keepalived /usr/sbin/

在MASTER上/etc/keepalived/目录创建keepalived.conf配置文件，并写入如下内容：

! Configuration File for keepalived

global\_defs {

notification\_email {

wgkgood@163.com

}

notification\_email\_from wgkgood@163.com

smtp\_server 127.0.0.1

smtp\_connect\_timeout 30

router\_id LVS\_DEVEL

}

# VIP1

vrrp\_instance VI\_1 {

state BACKUP

interface eth0

lvs\_sync\_daemon\_inteface eth0

virtual\_router\_id 51

priority 100

advert\_int 5

nopreempt

authentication {

auth\_type PASS

auth\_pass 1111

}

virtual\_ipaddress {

192.168.33.188

}

}

virtual\_server 192.168.33.188 80 {

delay\_loop 6

lb\_algo wrr

lb\_kind DR

# persistence\_timeout 60

protocol TCP

real\_server 192.168.33.12 80 {

weight 100

TCP\_CHECK {

connect\_timeout 10

nb\_get\_retry 3

delay\_before\_retry 3

connect\_port 80

}

}

real\_server 192.168.33.13 80 {

weight 100

TCP\_CHECK {

connect\_timeout 10

nb\_get\_retry 3

delay\_before\_retry 3

connect\_port 80

}

}

}

如上配置文件，红色标记的地方需要注意，state状态主服务器设置MASTER，从设置为BACKUP，优先级备机设置比MASTER小，例如设置90，使用TCP端口检测。

在LVS BACKUP服务器写入如下配置，需要注意的是客户端的配置要修改优先级及状态：

! Configuration Filefor keepalived

global\_defs {

  notification\_email {

   wgkgood@163.com

  }

  notification\_email\_from wgkgood@163.com

  smtp\_server 127.0.0.1

  smtp\_connect\_timeout 30

  router\_id LVS\_DEVEL

}

# VIP1

vrrp\_instance VI\_1 {

  state BACKUP

  interface eth0

  lvs\_sync\_daemon\_inteface eth0

  virtual\_router\_id 51

  priority 90

  advert\_int 5

  authentication {

    auth\_type PASS

    auth\_pass 1111

  }

  virtual\_ipaddress {

    192.168.33.11

  }

}

#REAL\_SERVER\_1

virtual\_server 192.168.33.11 80 {

  delay\_loop 6

  lb\_algo wlc

  lb\_kind DR

  persistence\_timeout 60

  protocol TCP

  real\_server 192.168.33.130 80 {

    weight 100

    TCP\_CHECK {

    connect\_timeout 10

    nb\_get\_retry 3

    delay\_before\_retry 3

    connect\_port 80

    }

}

#REAL\_SERVER\_2

  real\_server 192.168.33.131 80 {

    weight 100

    TCP\_CHECK {

    connect\_timeout 10

    nb\_get\_retry 3

    delay\_before\_retry 3

    connect\_port 80

    }

  }

}

如上设置，LVS 主备配置完毕，接下来需要在realserver配置LVS VIP，为什么要在realserver绑定VIP呢？

客户端访问director的VIP，director接收请求，将通过相应的算法将请求转发给相应的realserver。在转发的过程中，会修改请求包的目的mac地址，目的ip地址不变。

Realserver接收请求，并直接响应客户端。这时便出现一个问题，director此时与realserver位于同一个网络中，当director直接将请求转发给realserver时，realserver检测到该请求包的目的ip是vip而并非自己，便会丢弃，而不会响应。为了解决这个问题，所以需要在所有Realserver上都配上VIP。

为什么一定要配置在lo接口上呢？

在realserver上的lo口配置VIP,这样限制了VIP不会在物理交换机上产生MAC地址表，从而避免IP冲突。

客户端启动Realserver.sh脚本内容：

#!/bin/sh

#LVS Client Server

VIP=192.168.33.188

case $1 in

start)

ifconfig lo:0 $VIP netmask 255.255.255.255 broadcast $VIP

/sbin/route add -host $VIP dev lo:0

echo "1" >/proc/sys/net/ipv4/conf/lo/arp\_ignore

echo "2" >/proc/sys/net/ipv4/conf/lo/arp\_announce

echo "1" >/proc/sys/net/ipv4/conf/all/arp\_ignore

echo "2" >/proc/sys/net/ipv4/conf/all/arp\_announce

sysctl -p >/dev/null 2>&1

echo "RealServer Start OK"

exit 0

;;

stop)

ifconfig lo:0 down

route del $VIP >/dev/null 2>&1

echo "0" >/proc/sys/net/ipv4/conf/lo/arp\_ignore

echo "0" >/proc/sys/net/ipv4/conf/lo/arp\_announce

echo "0" >/proc/sys/net/ipv4/conf/all/arp\_ignore

echo "0" >/proc/sys/net/ipv4/conf/all/arp\_announce

echo "RealServer Stoped OK"

exit 1

;;

\*)

echo "Usage: $0 {start|stop}"

;;

esac

## LVS在企业中应用排错

当我们熟练构建LVS负载均衡配置了，在企业真实环境中，如何去排错呢，遇到问题我们该怎么办呢？

LVS使用过程中，我们都会遇到很多的问题，但是遇到问题后，我们需要该如何处理呢？那这里分享我的解决思路。

LVS+Keepalived+Nginx架构中，某天突然发现网站[www.jfedu.net](http://www.jfedu.net/) 部分用户访问巨慢，甚至无法访问，那这个问题我们该如何定位呢？分两种情况：如果有监控，如果有报警短信再好不过了。然后可以很快的定位到某一台机器。如果没有监控，或者其他的原因没法看监控，那我们该如何排查呢？思路是什么呢？

1. 首先我们想到ping www.jfedu.net ，通过ping返回数据部正常。
2. 登录LVS服务器，ipvsadm –Ln 查看当前后端web连接信息，显示如下：

[root@LVS-Master keepalived]# ipvsadm -Ln

IP Virtual Server version 1.2.1 (size=4096)

Prot LocalAddress:Port Scheduler Flags

-> RemoteAddress:Port Forward Weight ActiveConn InActConn

TCP 192.168.1.10:80 wlc

-> 192.168.1.6:80 Route 100 2 13

-> 192.168.1.5:80 Route 100 120 13

**-> 192.168.1.4:80 Route 100 1363 45**

通过LVS信息，我们看到LVS 选择的轮训方式为加权最少连接，而网站也是部分无法访问，我们可以猜测是其中一台web服务器无法访问或者访问巨慢导致，我们会想难道LVS不会自己判断吗？想法很好，那我们接下来查看keepalived.conf配置，部分截图如下：

real\_server 192.168.1.4 80 {

weight 100

TCP\_CHECK {

connect\_timeout 10

nb\_get\_retry 3

delay\_before\_retry 3

connect\_port 80

}

}

通过配置文件我们发现LVS默认用的是TCP检测方式，只要80端口能通，请求就会转发到后端服务器。紧接着在LVS /tmp目wget http://192.168.1.4/ 返回502超时，另外几台nginx返回正常，1.4服务器80端口对于LVS来说是打开的，所以LVS会把请求转发给给它。

这就造成了为什么部分用户可以访问，有的用户无法访问的问题。登录1.4 nginx服务器，pkill nginx ，临时停止nginx保证服务保证正常访问，然后再查看nginx日志发现是后端程序连接一台数据库出现的问题。

回过头来我们会发现，LVS不会检测你后端502超时错误，只关心80端口是否开启对于应用来说，这样检测明显不足，那我们需要如何处理呢？增加LVS对后端Nginx URL的检测，能访问URL则表示服务正常，直接看代码：

real\_server 192.168.1.4 80 {

weight 100

HTTP\_GET {

url {

path /monitor/warn.jsp

status\_code 200

}

connect\_timeout 10

nb\_get\_retry 3

delay\_before\_retry 3

}

}

我们对比之前的检测方式，从单纯的80端口到现在的URL检测，后端如果某台出现502超时错误，LVS会自动踢出，等后端恢复后自动添加。自此这个故障就解决完毕，当然还有很多故障，例如如下的问题：

某一天[www.jfedu.net](http://www.jfedu.net/)网站突然有用户反馈报404，而我在本地访问正常，这又是怎么回事呢？

………

**总结：LVS网站故障排查经验**

如果发现主网站无法访问，首先第一步ping网站域名是否能ping通，如果域名无法访问，试着使用IP能不能访问，如果IP能访问，首先排查到域名解析问题。

如果IP也无法访问，登录LVS服务器，使用命令ipvsadm –Ln 查看当前连接状态和查看/var/log/messages日志信息，可以在LVS上访问realserver ip，进行排查。

如果LVS服务正常，后端realserver服务异常，然后查看nginx日志信息，是否有大量恶意访问，临时重启看是否能访问。

如果有恶意ip访问，找出恶意ip，经确认可以关闭后，使用iptables防火墙临时关闭即可。