第六章　负载均衡及服务器集群（lvs）

作者:田逸(sery@163.com)

在互连网应用技术里,负载均衡一直是热门话题,本书讨论的负载均衡技术,包括但不限于负载均衡本身。使用负载均衡技术主要的目的包括如下几点：

- 系统高可用性。组成系统的某些设备或部件失效，并不会影响正常的服务。
- 系统可扩展性。用户的增加，引起访问数乃至流量的增加，这种情形下，需要对系统进行扩容，以应对这种快速增长。对于提供高可用服务的互联网网站，其对可扩展的基本要求就是在保持系统服务不终止的情况下，透明的扩充容量，即用户不知道扩容的存在，或者说是扩容不对现有的服务产生任何负面作用。这些扩展主要包括：带宽扩展、服务器扩展、存储容量扩展、数据库扩展等，当然也包括主机增加内存等方面的扩展。
- 负载均衡能力。一个应用或服务由数个物理服务器提供,并且每个物理服务器运行的应用或服务是相同的,我们可以让用户的访问通过某种控制策略,把负载分摊到不同的物理服务器,从而保持每个物理服务器有比较合理的负载。当整个系统的负载趋于饱和时，通过增加物理服务器和扩充物理带宽来解决这个麻烦。增加物理服务器以后，系统的负载情况将重新在所有集群的物理服务器之间按照指定的算法重新达到新的均衡。

一个完整的负载均衡项目，一般由虚拟服务器、故障隔离及失败切换 3 个功能框架所组成。

虚拟服务器是负载均衡体系的基本架构，它分两层结构：转发器（Director）和真实服务器。图 6-1 为虚拟服务器的结构示意。

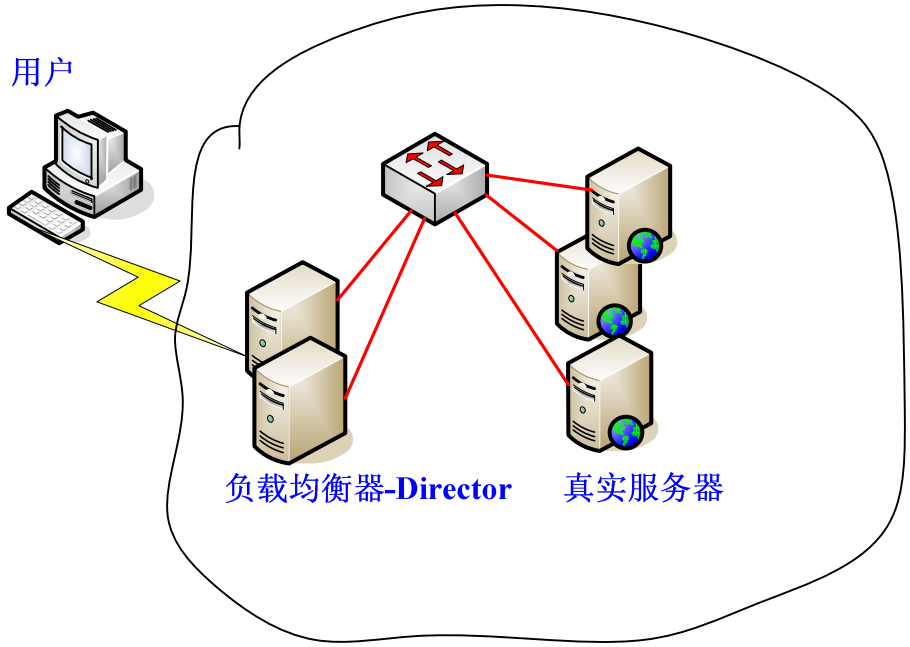

图 6-1 虚拟服务器结构

为什么称虚拟服务器？因为从用户的角度看来，似乎只是一个服务器在提供服务。虚拟服务器最主要的功能是提供包转发和负载均衡，这个功能可以通过撰写 ipvsadm 脚本具体实现。虚拟服务器项目由章文嵩博士所贡献，目前已被添加到各种 linux 发行版的内核。

故障隔离指虚拟服务器中的某个真实服务器（或某几个真实服务器）失效或发生故障，系统将自动把失效的服务器从转发队列中清理出去，从而保证用户访问的正确性；另一方面，当实效的服务器被修复以后，系统再自动地把它加入转发队列。

失败切换，这是针对负载均衡器 Director 采取的措施，在有两个负载均衡器 Director 的应用场景，当主负载均衡器（MASTER）失效或出现故障，备份负载均衡器（BACKUP）将自动接管主负载均衡器的工作；一旦主负载均衡器故障修复，两者将恢复到最初的角色。

要从技术上实现虚拟服务器、故障隔离及失败切换 3 个功能，需要两个工具：ipvsadm 和 keepalived。当然也有 heartbeat 这样的工具可以实现同样的功能，但相对于 keepalived，heartbeat 的实现要复杂得多（如撰写 ipvsadm 脚本，部署 ldirectord，编写资源文件等）。在采用 keepalived 的方案里，只要 ipvsadm 被正确的安装，简单的配置唯一的文件 keepalived 就行了。

6.1 lvs 核心 ipvs

Ipvs(IP Virtual Server)是整个负载均衡的基础，如果没有这个基础，故障隔离与失败切换就毫无意义了。在大部分 linux 发行版中，ipvs 被默认安装，而以本书前面介绍的方法定制安装系统，则 ipvs 没有被默认安装。

除大部分 linux 发行版支持 ipvs 外，FreeBSD 也可以支持 LVS，只不过实现起来要麻烦一些。

6.1.1 安装 ipvs

Ipvs 具体实现是由 ipvsadm 这个程序来完成，因此判断一个系统是否具备 ipvs 功能，只需要察看 ipvsadm 程序是否被安装。察看 ipvsadm 程序最简单的办法就是在任意路径执行命令 ipvsadm。表 6-1 为安装 ipvsadm 及未安装 ipvsadm 的输出对比。

	执行 ipvsadm 后的输出
未安装 ipvsadm	-bash: ipvsadm: command not foun
安装 ipvsadm	IP Virtual Server version 1.2.1 (size=4096) Prot LocalAddress:Port Scheduler Flags -> RemoteAddress:Port Forward Weight ActiveConn InActConn

表 6-1 ipvsadm 输出对比（样例来源 centos 5.2）

- Centos5.2 安装 ipvsadm(假定当前目录为/root)

1、 从官方网站下载 ipvsadm，目前最新的版本为 ipvsadm-1.25.tar.gz ，其发布时间是 2008 年 11 月 5 日。Wget http://www.linuxvirtualserver.org/software/kernel-2.6/ipvsadm-1.24.tar.gz 取得该版本[1]。

2、 创建一个连接文件，其命令为：ln –sv /usr/src/kernels/2.6.18-92.el5PAE-i686 /usr/src/linux . 注意一定要与当前的运行的内核相一致，因为/usr/src/kernels 目录下可多个目录。如果不创建这个连接文件，在编译时会出错，从而不能继续进行安装。

3、 解包。 tar zxvf ipvsadm-1.24

4、 编译并安装。 cd ipvsadm-1.24; make;make install 可执行文件被安装到/sbin/ipvsadm.

- 检验 ipvsadm 是否被正确安装

1、 执行 ipvsadm，看是否有表 6-1 第 2 栏的输出。

2、 检查当前加载的内核模块，看是否存在 ip_vs 模块。

```
[root@hd-4 ipvsadm-1.24]# lsmod|grep ip_vs
ip_vs                   77569  0
```

注 1、只有执行 ipvsadm 以后，才会在内核加载 ip_vs 模块。

注 2、不能以查进程的方式判断 ipvs 是否运行。

注[1]：如果下载最新的 ipvsadm-1.25.tar.gz 这个版本，在创建连接文件/usr/src/linux 后，执行编译时，可能需要修改/boot/grub/grub.conf 启动内核名称。一旦当前运行内核与连接文件所代表的内核名不一致时，将出现找不到*.h 这样的错误，从而导致安装不能正常进行。

6.1.2 lvs 客户端

Lvs 的客户端指负载均衡其/转发器（director）后面提供服务的真实机器。负载均衡类型（lb_kind）一般分直接路由模式 DR 、网络地址转换模式 NAT 以及隧道模式 TUN 三种。Lvs 客户端的配置是根据其所采用的负载均衡种类(lb_kind)来做相应操作的。在我们的应用环境里，为了获得最高的性能，采用的负载均衡种类(lb_kind)是直接路由模式 DR。

不管采取哪一种模式，lvs 客户端都不需安装额外的软件。

Lvs 可支持的客户端包括：各种 GNU/linux、大部分 unix 已经 windows。目前我已经成功运行的 lvs 客户端环境有 centos、redhat、freebsd、windows 等。需要注意的是，由于客户端操作系统的不同，lvs 客户端的配置也就各不相同了。本书中，将以 centos 及 freebsd 两种操作系统作为 lvs 的客户端，给出其直接路由模式 DR 的配置方法。

- lvs 客户端（真实服务器）操作系统是 centos 时的配置文件

```
[root@huludao-2 ~]# more /usr/local/bin/lvs_real
#!/bin/bash
#description : start realserver
VIP=61.135.20.16
/etc/rc.d/init.d/functions

case "$1" in
start)
echo " start LVS of REALServer"
/sbin/ifconfig lo:0 $VIP broadcast $VIP netmask 255.255.255.255 up
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
;;
stop)
```

```
/sbin/ifconfig lo:0 down
echo "close LVS Directorserver"
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 "Usage: $0 {start|stop}"
exit 1
esac
```

这里对配置文件里重要的一些项进行说明：

1、 vip(virtual ip)。直接路由模式的 vip 必须跟服务器对外提供服务的 ip 地址在同一个网段，并且 lvs 负载均衡器和其他所有提供相同功能的服务器都使用这个 vip.

2、 vip 被绑定在环回接口 lo0:0 上，其广播地址是其本身，子网掩码是 255.255.255.255。这与标准的网络地址设置有很大的不同。采用这种可变长掩码方式把网段划分成只含一个主机地址的目的是避免 ip 地址冲突。

3、 echo “1”,echo “2” 这段的作用是抑制 arp 广播。如果不做 arp 抑制，将会有众多的机器向其他宣称：“嗨！我是奥巴马，我在这里呢！”，这样就乱套了。

- lvs 客户端（真实服务器）操作系统是 freebsd 时的配置文件

```
fav1# more /usr/local/bin/lvs_real_bsd
#!/usr/local/bin/bash
#description : start realserver
VIP=61.135.20.16

case "$1" in
start)
echo " start LVS of REALServer"
/sbin/ifconfig lo0 $VIP    netmask 255.255.255.255 alias -arp up
;;

stop)
/sbin/ifconfig lo0 alias down
echo "close LVS Directorserver"
/sbin/ifconfig lo0 127.0.0.1 arp up
;;
*)
echo "Usage: $0 {start|stop}"
exit 1
esac
```

在这里，我们同样对这个配置脚本的某些项进行说明：

1、vip 地址设置和 arp 抑制用 /sbin/ifconfig lo0 $VIP   netmask 255.255.255.255 alias -arp up 这么一行就实现了。

我们把这两种操作系统的 lvs 客户端做个比较，发现 freebsd 的配置书写上要简洁一些，是不是可以认为 freebsd 的网络功能比 linux 强大呢？

6.1.3 lvs 客户端验证

lvs 客户端不必依赖负载均衡器就可以独立运行，只不过这种运行对负载均衡没有任何作用，当然也没有任何副作用，所以我们把 lvs 客户端配置完成后（配置文件就是一个 shell 脚本），可以单独运行它，来检验配置是否正确。

● centos 脚本

配置脚本写好保存，给予脚本执行权限。脚本启停以 start 和 stop 这两个参数来控制。首先，我们来启动配置脚本，执行命令 /usr/local/bin/lvs_real start ,接着我们来检查网络的状态:

```
[root@huludao-2 ~]# ip add
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet    61.135.20.16/32 brd 61.135.20.16 scope global lo:0
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
qlen 100
    link/ether 00:15:17:60:21:a0 brd ff:ff:ff:ff:ff:ff
    inet    61.135.20.101/24 brd 125.38.38.255 scope global eth0
    inet6 fe80::215:17ff:fe60:21a0/64 scope link
        valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop qlen 1000
    link/ether 00:15:17:60:21:a1 brd ff:ff:ff:ff:ff:ff
4: sit0: <NOARP> mtu 1480 qdisc noop
    link/sit 0.0.0.0 brd 0.0.0.0
```

从输出可以看出，lo0:0 确实绑定了我们指定的 vip 地址。那么当我们执行 /usr/local/bin/lvs_real 时，vip 应当从 lo0:0 接口卸载。我们来看看输出是什么：

```
[root@huludao-2 ~]# lvs_real stop
close LVS Directorserver
[root@huludao-2 ~]# ip add
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
qlen 100
    link/ether 00:15:17:60:21:a0 brd ff:ff:ff:ff:ff:ff
```

```
    inet 125.38.38.101/28 brd 125.38.38.111 scope global eth0
    inet6 fe80::215:17ff:fe60:21a0/64 scope link
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop qlen 1000
    link/ether 00:15:17:60:21:a1 brd ff:ff:ff:ff:ff:ff
4: sit0: <NOARP> mtu 1480 qdisc noop
    link/sit 0.0.0.0 brd 0.0.0.0
```

噢，上帝！正是我们所期待的结果：vip 从 lo 上消失了。

●freebsd 脚本/usr/local/bin/lvs_real_bsd

启停所使用的方法和参数与 centos 那个配置脚本是一样的，但察看方法有所不同（freebsd 无 ip add 工具）。这里使用的方法是 ifconfig，启动时输出如下：

```
fav1# ifconfig
em0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500

options=19b<RXCSUM,TXCSUM,VLAN_MTU,VLAN_HWTAGGING,VLAN_HWCSUM,
TSO4>
        ether 00:15:17:6e:c8:46
        inet 61.135.20.69 netmask 0xffffffc0 broadcast 61.128.20.127
        media: Ethernet autoselect (100baseTX <full-duplex>)
        status: active
lo0: flags=80c9<UP,LOOPBACK,RUNNING,NOARP,MULTICAST> metric 0 mtu 16384
        inet6 fe80::1%lo0 prefixlen 64 scopeid 0x3
        inet6 ::1 prefixlen 128
        inet 127.0.0.1 netmask 0xff000000
        inet 61.135.20.16 netmask 0xffffffff
```

当执行/usr/local/bin/lvs_real_bsd stop 时，vip 被卸载。有时可能卸载不成功，可能需要多执行几次停止命令。

●arp 抑制验证

如果不配置负载均衡器的转发功能，那么在这个步骤所设置的 vip 将不会提供任何 ip 功能，除了能用 ifconfig 输出显示而外。

在没有配置或开启 lvs 负载均衡器的情况下，我们用其他机器 ping vip,应该是不通的。当然在网络前面启用了防火墙阻止 ping 不算。为了不影响测试，最好把你的防火墙打开。

Arp 抑制生效后，再也没有机器知道 vip 的存在，这是一个问题，谁来响应 vip 请求？这个问题交给负载均衡器吧，它知道这背后的一切。

6.2 故障隔离、失败切换框架 keepalived

Keepalived 是运行在 lvs 之上，它的主要功能是实现真实机的故障隔离及负载均衡器间的失败切换 FailOver.lvs 结合 keepalived，就实现了 3 层、4 层、5/7 层交换的功能，下面摘录来自官方网站 www.keepalived.org 的一段描述：

The main goal of the keepalived project is to add a strong & robust keepalive facility to the Linux Virtual Server project. This project is written in C with multilayer TCP/IP stack checks. Keepalived implements a framework based on three family checks : Layer3, Layer4 & Layer5/7. This framework gives the daemon the ability of checking a LVS server pool states. When one of the server of the LVS server pool is down, keepalived informs the linux kernel via a setsockopt call to remove this server entrie from the LVS topology. In addition keepalived implements an independent VRRPv2 stack to handle director failover. So in short keepalived is a userspace daemon for LVS cluster nodes healthchecks and LVS directors failover.

从这段描述中，我们可以得到几个有用的信息：

1、 keepalived 是 lvs 的扩展项目，因此它们之间具备良好的兼容性。这点应该是 keepalived 部署比其他类似工具能更简洁的原因吧！

2、 通过对服务器池对象的健康检查，实现对失效机器/服务的故障隔离。

3、 负载均衡器之间的失败切换 failover，是通过 VRRPv2（Virtual Router Redundancy Protocol）stack 实现的。

6.2.1 keepalived 体系结构

Keepalived 大致分两层结构：用户空间 user space 和内核空间 kernel space.图 6-2 是来自官方站点（http://www.keepalived.org/software_design.html）关于其结构的展示。

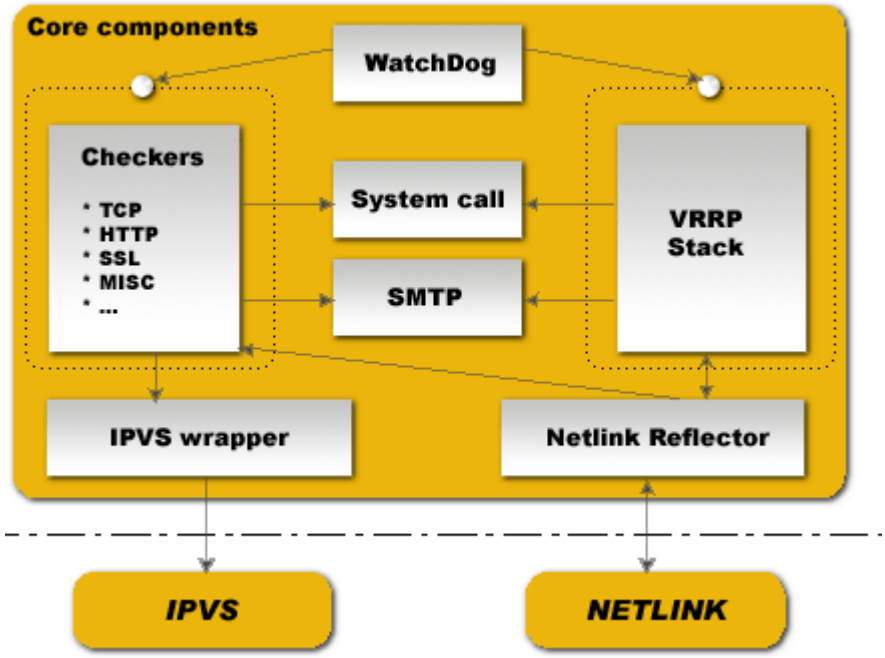

图 6-2 keepalived 内部结构图

在这个结构图里，处于下端的是内核空间，它包括 ipvs 和 NETLINK 两个部分。Ipvs 的作用在前面的章节已经做过描述，不再重复叙述；netlink 提供高级路由及其他相关的网络功能，如果我们在负载均衡器上启用 netfilter/iptable,将会直接影响它的性能。出于图形上方的组件为用户空间，由它来实现具体的功能，下面选取几个重要的来做说明：

1、 WatchDog 负责监控 checkers 和 VRRP 进程的状况。
2、 Checkers 负责真实服务器的健康检查 healthchecking，是 keepalived 最主要的功能。换句话说—可以没有 VRRP Stack,但健康检查 healthchecking 是一定要有的。
3、 VRRP Stack 负责负载均衡器之间的失败切换 FailOver.如果只用一个负载均衡器，则 VRRP 不是必须的。
4、 IPVS wrapper 用来发送设定的规则到内核 ipvs 代码。
5、 Netlink Reflector 用来设定 vrrp 的 vip 地址等。

Keepalived 各种功能的实现是通过设置其配置文件 keepalived.conf 来完成的，关于配置文件各项的用途，将在后面的章节进行描述。

6.2.2 安装 keepalived

安装 keepalived 非常的简单和容易，这跟安装其他 GNU 源码软件步骤是以模一样的。下面给出其安装过程。

1、 下载最新稳定版 wget http://www.keepalived.org/software/keepalived-1.1.17.tar.gz
2、 解包 tar zxvf keepalived-1.1.17.tar.gz
3、 切换目录 cd keepalived-1.1.17
4、 配置 ./configure –prefix=/usr/local/keepalive 因为 keepalived 运行在 ipvs 之上，因此这两个软件一定要安装在一个系统里面。如果 configure 操作能正常进行，运行完毕后将有如下的汇总输出：

```
Keepalived configuration
------------------------
Keepalived version           : 1.1.17
Compiler                     : gcc
Compiler flags               : -g -O2
Extra Lib                    : -lpopt -lssl -lcrypto
Use IPVS Framework           : Yes
IPVS sync daemon support : Yes
Use VRRP Framework           : Yes
Use LinkWatch                : No
Use Debug flags              : No
```

5、 编译和安装 make ; make install

6.2.3 keepalived 安装验证

Keepalived 安装完成后，会在安装目录/usr/local/keepalived 生成 bin,etc,man,sbin 这 4 个目录。其中 etc 为配置文件所在的目录，进入这个目录，看看里面都有些什么？

```
[root@ChinaTelecom-1 etc]# pwd
/usr/local/keepalived/etc

[root@ChinaTelecom-1 etc]# ll
```

```
total 12
drwxr-xr-x 3 root root 4096 Apr 23 10:23 keepalived
drwxr-xr-x 3 root root 4096 Apr 23 10:23 rc.d
drwxr-xr-x 2 root root 4096 Apr 23 10:23 sysconfig
```

还有子目录，这里着重关注一下 keepalived 目录，它的下面包含一个完整的配置文件 keepalived.conf（实际上是一个样例）以及一些单独的配置样例文件。

```
[root@ChinaTelecom-1 keepalived]# tree -l
.
|-- keepalived.conf
`-- samples
    |-- client.pem
    |-- dh1024.pem
    |-- keepalived.conf.HTTP_GET.port
    |-- keepalived.conf.SMTP_CHECK
    |-- keepalived.conf.SSL_GET
    |-- keepalived.conf.fwmark
    |-- keepalived.conf.inhibit
    |-- keepalived.conf.misc_check
    |-- keepalived.conf.misc_check_arg
    |-- keepalived.conf.sample
    |-- keepalived.conf.status_code
    |-- keepalived.conf.track_interface
    |-- keepalived.conf.virtual_server_group
    |-- keepalived.conf.virtualhost
    |-- keepalived.conf.vrrp
    |-- keepalived.conf.vrrp.localcheck
    |-- keepalived.conf.vrrp.lvs_syncd
    |-- keepalived.conf.vrrp.routes
    |-- keepalived.conf.vrrp.scripts
    |-- keepalived.conf.vrrp.static_ipaddress
    |-- keepalived.conf.vrrp.sync
    |-- root.pem
    `-- sample.misccheck.smbcheck.sh
```

值得注意的是，keepalived 的启动过程并不会对配置文件进行语法检查，就算没有配置文件，keepalived 的守护进程照样能够被运行起来。在默认状态下--即不指定配置文件的位置—keepalived 先查找文件 /etc/keepalived/keepalived.conf ，如果为了省事，可以手动创建这个文件，然后在这个文件里书写规则，来达到控制 keepalived 运行的目的。

这里我们先来试试默认情况，即没有配置文件下运行 keepalived.运行前先了解一下其语法：

[root@lvs-m keepalived]# **keepalived --help**

Keepalived v1.1.17 (06/23,2009)

Usage:

keepalived

keepalived -n

keepalived -f keepalived.conf

keepalived -d

keepalived -h

keepalived -v

Commands:

Either long or short options are allowed.

keepalived --vrrp -P Only run with VRRP subsystem.

keepalived --check -C Only run with Health-checker subsystem.

keepalived --dont-release-vrrp -V Dont remove VRRP VIPs & VROUTEs on daemon stop.

keepalived --dont-release-ipvs -I Dont remove IPVS topology on daemon stop.

keepalived --dont-fork -n Dont fork the daemon process.

keepalived --use-file -f Use the specified configuration file.

Default is /etc/keepalived/keepalived.conf.

keepalived --dump-conf -d Dump the configuration data.

keepalived --log-console -l Log message to local console.

keepalived --log-detail -D Detailed log messages.

keepalived --log-facility -S 0-7 Set syslog facility to LOG_LOCAL[0-7]. (default=LOG_DAEMON)

keepalived --help -h Display this short inlined help screen.

keepalived --version -v Display the version number

keepalived --pid -p pidfile

keepalived --checkers_pid -c checkers pidfile

keepalived --vrrp_pid -r vrrp pidfile

接下来我们参照这个帮助语法，执行命令 /usr/local/keepalived/sbin/keepalive –D ,然后来检查 keepalived 运行后的状况。

1、 查看进程 ps aux | grep keepalived ，其输出为：

```
[root@lvs-m ~]# ps aux| grep keepalived |grep -v grep
root     21786  0.0  0.0   4840   564 ?        Ss   15:39   0:00 keepalived -D
root     21787  4.8  0.0   4884  1336 ?        S    15:39  23:47 keepalived -D
root     21788  4.9  0.0   4884   904 ?        S    15:39  24:15 keepalived -D
```

Keepalived 正常运行时，共启动 3 个进程，其中一个进程是父进程，负责监控其子进程；一个是 vrrp 子进程；另外一个是 checkers 子进程。图 6-3 为 keepalived 3 个进程之间的关系。

```
[root@lvs-m ~]# pstree | grep keepalived
     |-keepalived---2*[keepalived]
```

图 6-3 keepalived 进程相关性

2、 查看内核模块，ip_vs 模块应该被加载到内核空间。 Lsmod | grep ip_vs .

3、 查看系统日志。因为我在启动 keepalived 是使用了选项 –D ,这将详细的打印日志消息。

```
[root@lvs-m ~]# tail -f /var/log/messages
Jun 27 00:58:05 lvs-m Keepalived: Starting VRRP child process, pid=22017
Jun 27 00:58:05 lvs-m Keepalived_healthcheckers: Netlink reflector reports IP 61.135.20.137 added
Jun 27 00:58:05 lvs-m Keepalived_healthcheckers: Registering Kernel netlink reflector
Jun 27 00:58:05 lvs-m Keepalived_vrrp: Netlink reflector reports IP 61.135.20.137 added
Jun 27 00:58:05 lvs-m Keepalived_healthcheckers: Registering Kernel netlink command channel
Jun 27 00:58:05 lvs-m Keepalived_vrrp: Registering Kernel netlink reflector
Jun 27 00:58:05 lvs-m Keepalived_vrrp: Registering Kernel netlink command channel
Jun 27 00:58:05 lvs-m Keepalived_vrrp: Registering gratutious ARP shared channel
Jun 27 00:58:05 lvs-m Keepalived_healthcheckers: Configuration is using : 2285 Bytes
Jun 27 00:58:05 lvs-m Keepalived_vrrp: Configuration is using : 28803 Bytes
```

逐项检查这个输出，可知图 6-2 所示的组件都可以在这里找到对应的纪录。从而进一步证实 keepalived 安装的正确性。

6.2.4 配置文件 keepalived.conf

一个功能比较完整的 keepalived 的配置文件，其配置文件 keepalived.conf 可以包含三个文本块：全局定义块、VRRP 实例定义块及虚拟服务器定义块。全局定义块和虚拟服务器定义块是必须的，如果在只有一个负载均衡器的场合，就不须 VRRP 实例定义块。

接下来，我们以一个配置文件模版为例，有选择的说明其中一些重要项的功能或作用。

```
#全局定义块
global_defs {
notification_email {
email
email
}
notification_email_from email
smtp_server host
smtp_connect_timeout num
lvs_id string
}
```

```
#VRRP 实例定义块
vrrp_sync_group string {
group {
string
string
}

vrrp_instance string {
state MASTER|BACKUP
interface string
mcast_src_ip @IP
lvs_sync_daemon_interface string
virtual_router_id num
priority num
advert_int num
smtp_alert
authentication {
auth_type PASS|AH
auth_pass string
}
virtual_ipaddress { # Block limited to 20 IP addresses
@IP
@IP
@IP
}
virtual_ipaddress_excluded { # Unlimited IP addresses number
@IP
@IP
@IP
}


#虚拟服务器定义块
virtual_server (@IP PORT)|(fwmark num) {
delay_loop num
lb_algo rr|wrr|lc|wlc|sh|dh|lblc
lb_kind NAT|DR|TUN
(nat_mask @IP)
persistence_timeout num
persistence_granularity @IP
virtualhost string
protocol TCP|UDP
sorry_server @IP PORT
```

```
real_server @IP PORT {
weight num
TCP_CHECK {
connect_port num
connect_timeout num
  }
}
real_server @IP PORT {
weight num
MISC_CHECK {
misc_path /path_to_script/script.sh
(or misc_path "/path_to_script/script.sh <arg_list>")
   }
  }

real_server @IP PORT {
weight num
HTTP_GET|SSL_GET {
url { # You can add multiple url block
path alphanum
digest alphanum
}
connect_port num
connect_timeout num
nb_get_retry num
delay_before_retry num
  }
 }
}
```

- 全局定义块

1、 email 通知。作用：有故障，发邮件报警。这是可选项目，建议不用，用 nagios 全面监控代替之。
2、 Lvs 负载均衡器标识（lvs_id）。在一个网络内，它应该是唯一的。
3、 花括号“{}”。用来分隔定义块，因此必须成对出现。如果写漏了，keepalived 运行时，不会得到预期的结果。由于定义块内存在嵌套关系，因此很容易遗漏结尾处的花括号，这点要特别注意。

- VRRP 定义块

1、 同步 vrrp 组 vrrp_sync_group。作用：确定失败切换（FailOver）包含的路由实例个数。即在有 2 个负载均衡器的场景，一旦某个负载均衡器失效，需要自动切换到另外一个负载均衡器的实例是哪些？
2、 实例组 group.至少包含一个 vrrp 实例。
3、 Vrrp 实例 vrrp_instance.实例名出自实例组 group 所包含的那些名字。

（1） 实例状态 state.只有 MASTER 和 BACKUP 两种状态，并且需要大写这些单词。其中 MASTER 为工作状态，BACKUP 为备用状态。当 MASTER 所在的服务器失效时，BACKUP 所在的系统会自动把它的状态有 BACKUP 变换成 MASTER；当失效的 MASTER 所在的系统恢复时，BACKUP 从 MASTER 恢复到 BACKUP 状态。

（2） 通信接口 interface。对外提供服务的网络接口，如 eth0,eth1.当前主流的服务器都有 2 个或 2 个以上的接口，在选择服务接口时，一定要核实清楚。

（3） lvs_sync_daemon_inteface。负载均衡器之间的监控接口，类似于 HA HeartBeat 的心跳线。但它的机制优于 Heartbeat，因为它没有“裂脑”这个问题，它是以优先级这个机制来规避这个麻烦的。在 DR 模式中，lvs_sync_daemon_inteface 与服务接口 interface 使用同一个网络接口。

（4） 虚拟路由标识 virtual_router_id.这个标识是一个数字，并且同一个 vrrp 实例使用唯一的标识。即同一个 vrrp_stance,MASTER 和 BACKUP 的 virtual_router_id 是一致的，同时在整个 vrrp 内是唯一的。

（5） 优先级 priority.这是一个数字，数值愈大，优先级越高。在同一个 vrrp_instance 里，MASTER 的优先级高于 BACKUP。若 MASTER 的 priority 值为 150，那么 BACKUP 的 priority 只能是 140 或更小的数值。

（6） 同步通知间隔 advert_int .MASTER 与 BACKUP 负载均衡器之间同步检查的时间间隔，单位为秒。

（7） 验证 authentication。包含验证类型和验证密码。类型主要有 PASS、AH 两种，通常使用的类型为 PASS，据说 AH 使用时有问题。验证密码为明文，同一 vrrp 实例 MASTER 与 BACKUP 使用相同的密码才能正常通信。

4、 虚拟 ip 地址 virtual_ipaddress . 可以有多个地址，每个地址占一行，不需要指定子网掩码。注意：这个 ip 必须与我们在 lvs 客户端设定的 vip 相一致！

● 虚拟服务器 **virtual_server** 定义块

虚拟服务器定义是 keepalived 框架最重要的项目了，是 keepalived.conf 必不可少的部分。

1、 虚拟服务器 virtual_server. 这个 ip 来自于 vrrp 定义块的第“4”步，后面一个空格，然后加上端口号。定义一个 vip，可以实现多个 tcp 端口的负载均衡功能。

（1） delay_loop。健康检查时间间隔，单位是秒。

（2） lb_algo. 负载均衡调度算法，互联网应用常使用 wlc 或 rr。

（3） lb_kind. 负载均衡转发规则。一般包括 DR,NAT,TUN3 种，在我的方案中，都使用 DR 的方式。

（4） persistence_timeout.会话保持时间，单位是秒。这个选项对动态网站很有用处：当用户从远程用帐号进行登陆网站时，有了这个会话保持功能，就能把用户的请求转发给同一个应用服务器。在这里，我们来做一个假设，假定现在有一个 lvs 环境，使用 DR 转发模式，真实服务器有 3 个，负载均衡器不启用会话保持功能。当用户第一次访问的时候，他的访问请求被负载均衡器转给某个真实服务器，这样他看到一个登陆页面，第一次访问完毕；接着他在登陆框填写用户名和密码，然后提交；这时候，问题就可能出现了—登陆不能成功。因为没有会话保持，负载均衡器可能会把第 2 次的请求转发到其他的服务器。

（5） 转发协议 protocol.一般有 tcp 和 udp 两种。实话说，我还没尝试过 udp 协议类的转发。

2、 真实服务器 real_server.也即服务器池。Real_server 的值包括 ip 地址和端口号。多个连续的真实 ip，转发的端口相同，是不是可以以范围表示？需要进一步实验。如写成 real_server 61.135.20.1-10 80 .

（1） 权重 weight.权重值是一个数字，数值越大，权重越高。使用不同的权重值的目的在于为不同性能的机器分配不同的负载，性能较好的机器，负载分担大些；反之，性能差的机器，则分担较少的负载，这样就可以合理的利用不同性能的机器资源。

（2） Tcp 检查 tcp_check.

关于配置文件的理论我们就先讲到这里。由于应用场景的不同，配置文件也会有很大的差异，在接下来的章节里，我将以两个具体的应用来展示 keepalived 神奇功效。

6.3 cdn 缓存服务器的负载均衡（lvs+keepalived）

在 cdn 的某个区域，需要部署多个缓存服务器，以集群方式来应对大量用户的访问请求，提高系统的高可用性。在机器数量较少的时候，我曾用 dns 轮训的方式把访问负载分担给服务器，效果不佳，因为 DNS 不会做健康检查。后来增加机器，做成 lvs 集群，就很安心了。

6.3.1 cdn 缓存服务器集群场景

6 个服务器，2 个服务器为 lvs 负载均衡器，4 个服务器为真实服务器。另外为了方便远程管理，还购置了一个 KVM OVER IP 设备。

申请到 8 个公网 ip 地址，6 个服务器用去 6 个，KVM OVER IP 设备用掉 1 个，剩下 1 个作为 VIP.

4 个真实服务器已经部署好应用（Varnish）,并且只要修改本地 windows 的 hosts 文件，把域名指向这些 ip，就可以通过这些缓存服务器访问到网站内容。关于缓存服务器的具体部署，请参照本书第 7 章“简单 cdn”。

为了方便阅读和理解，我把上述信息做成一个汇总表，如表 6-2。

角色	所需 ip	实现负载所需软件	操作
负载均衡器（MASTER）	接口 ip、VIP	ipvsadm，keepalived	安装 ipvsadm 及 keepalived
负载均衡器（BACKUP）	接口 ip、VIP	ipvsadm，keepalived	安装 ipvsadm 及 keepalived
真实服务器（server pool）	接口 ip、VIP	Lvs 客户端配置脚本	按“6.1.2”节内容编写配置脚本

表 6-2 缓存服务器集群信息汇总

6.3.2 cdn 缓存服务器集群部署

尽管部署 cdn 缓存服务器集群没有先后顺序之分，但为了部署顺利进行，我们可以先从简单的部分开始，即先从 lvs 客户端开始，然后再到负载均衡器这边。

- 真实服务器上的操作（每个服务器都是一样的操作）

1、照“6.1.2 节”的格式编写 lvs 客户端配置脚本，保存后给予执行权限。其完整的内容如下：

```
[root@huludao-2 ~]# more /usr/local/bin/lvs_real
#!/bin/bash
#description : start realserver
VIP=125.38.38.64
/etc/rc.d/init.d/functions

case "$1" in
start)
echo " start LVS of REALServer"
/sbin/ifconfig lo:0 $VIP broadcast $VIP netmask 255.255.255.255 up
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
;;
stop)
/sbin/ifconfig lo:0 down
echo "close LVS Directorserver"
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 "Usage: $0 {start|stop}"
exit 1
esac
```

2、运行和验证这个配置脚本，其具体方法如前所叙，不再赘述。

- 负载均衡器上的操作

MASTER 和 BACKUP 上安装 ipvsadm 及 keepalived 的方法都是一样的（具体步骤参见“6.1.1”及“6.2.2”两节的内容），两者之间的主要差异在于其配置文件 keepalived.conf 。

1、MASTER 的配置文件/etc/keepalived/keepalived.conf.

```
#writed by sery , contact sery@163.com
#guration File for keepalived
#global define
global_defs {
    router_id LVS_CNC_1
}
```

```
vrrp_sync_group VGM {
    group {
        VI_CACHE
    }
}

#########################################################################
#          vvrp_instance    define                                      #
#########################################################################
vrrp_instance VI_CACHE {
      state MASTER
      interface eth0
      lvs_sync_daemon_inteface eth0
      virtual_router_id 51
      priority 180
      advert_int 5
      authentication {
            auth_type PASS
            auth_pass 1111
      }
      virtual_ipaddress {
      125.38.38.64
      }
}

#########################################################################
#     virtual machine    setting                                        #
#########################################################################

# setting port 80 forward
virtual_server 125.38.38.64 80 {
      delay_loop 6
      lb_algo wlc
      lb_kind DR
#       persistence_timeout 20
      protocol TCP

      real_server 125.38.38.101 80 {
            weight 100
            TCP_CHECK {
                  connect_timeout 3
                  nb_get_retry 3
                  delay_before_retry 3
                  connect_port 80
```

```
        }
    }
    real_server 125.38.38.102 80 {
        weight 100
        TCP_CHECK {
            connect_timeout 3
            nb_get_retry 3
            delay_before_retry 3
            connect_port 80
        }
    }
    real_server 125.38.38.104 80 {
        weight 100
        TCP_CHECK {
            connect_timeout 3
            nb_get_retry 3
            delay_before_retry 3
            connect_port 80
        }
    }
    real_server 125.38.38.99 80 {
        weight 100
        TCP_CHECK {
            connect_timeout 3
            nb_get_retry 3
            delay_before_retry 3
            connect_port 80
        }
    }
}
```

2、 BACKUP 配置文件/etc/keepalived/keepalived.conf

```
#writed by sery , contact sery@163.com
#guration File for keepalived
#global define
global_defs {
    router_id LVS_CNC_2
}

vrrp_sync_group VGM {
    group {
        VI_CACHE
    }
}

#####################################################################
#       vrrp_instance setting                                       #
#####################################################################
vrrp_instance VI_CACHE {
       state BACKUP
       interface eth1
       lvs_sync_daemon_inteface eth1
       virtual_router_id 51
       priority 150
       advert_int 5
       authentication {
            auth_type PASS
            auth_pass 1111
       }
       virtual_ipaddress {
       125.38.38.64
       }
}

#####################################################################
#       virtual server    setting                                   #
#####################################################################

# setting port 80 forward
virtual_server 125.38.28.64 80 {
       delay_loop 6
       lb_algo wlc
       lb_kind DR
#       persistence_timeout 50
       protocol TCP
```

```
    real_server 125.38.38.101 80 {
        weight 100
        TCP_CHECK {
            connect_timeout 3
            nb_get_retry 3
            delay_before_retry 3
            connect_port 80
        }
    }
    real_server 125.38.38.102 80 {
        weight 100
        TCP_CHECK {
            connect_timeout 3
            nb_get_retry 3
            delay_before_retry 3
            connect_port 80
        }
    }
    real_server 125.38.38.104 80 {
        weight 100
        TCP_CHECK {
            connect_timeout 3
            nb_get_retry 3
            delay_before_retry 3
            connect_port 80
        }
    }
    real_server 125.38.38.99 80 {
        weight 100
        TCP_CHECK {
            connect_timeout 3
            nb_get_retry 3
            delay_before_retry 3
            connect_port 80
        }
    }
}
```

在这样只有一个 vrrp_instance 的环境里，主负载均衡器(MASTER)与备份负载均衡器（BACKUP）配置文件的差异一共只有 3 处: 全局定义的 route_id、vrrp_instance state 已经 vrrp_instance 的优先级 priority。

6.3.3 负载均衡服务的启用和验证

Lvs 客户端的启用和验证在前面的“6.1.3”一节有过详细的说明，此处略过。前面我们也提过，keepalived 启动过程不会检查配置文件的语法，因此在启动 keepalived 以前，需要人工对/etc/keepalived/keepalived.conf 文件做全面的语法检查。一个比较容易犯的错误就是把花括号“}”写漏了，不成对！

当 lvs 客户端都正常启动并且配置文件经检查无误后（当然有错误也无妨，随时可以修改嘛！），执行命令 /usr/local/keepalived/sbin/keepalived –D ,然后我们查看系统进程，看是否是 3 个 keepalived 进程。如果配置文件的路径不是/etc/keepalived/keepalived.conf 则需要在启动时用选项-f 指定。

最能反映 keepalived 启动情况的地方当属系统日志。手动执行启动操作后，使用命令 tail –f /var/log/messages 滚动查看输出，就能详细了解其运行情况。图 6-3 为某个 lvs 环境的 keepalived 启动输出：

```
Jul  2 07:05:41 ChinaTelecom-1 Keepalived_vrrp: Netlink reflector reports IP 211.18.89.229 added
Jul  2 07:05:41 ChinaTelecom-1 Keepalived_healthcheckers: Activating healthchecker for service [211.18.89.225:80]
Jul  2 07:05:41 ChinaTelecom-1 Keepalived_vrrp: Registering Kernel netlink reflector
Jul  2 07:05:41 ChinaTelecom-1 Keepalived_healthcheckers: Activating healthchecker for service [211.18.89.226:80]
Jul  2 07:05:41 ChinaTelecom-1 Keepalived_vrrp: Registering Kernel netlink command channel
Jul  2 07:05:41 ChinaTelecom-1 Keepalived_vrrp: Registering gratuticous ARP shared channel
Jul  2 07:05:41 ChinaTelecom-1 Keepalived_vrrp: Opening file '/etc/keepalived/keepalived.conf'.
Jul  2 07:05:41 ChinaTelecom-1 Keepalived_vrrp: Configuration is using : 37527 Bytes
Jul  2 07:05:41 ChinaTelecom-1 Keepalived_vrrp: VRRP sockpool: [ifindex(2), proto(112), fd(8,9)]
Jul  2 07:05:44 ChinaTelecom-1 Keepalived_vrrp: VRRP_Instance(VI_OUT1) Transition to MASTER STATE
Jul  2 07:05:49 ChinaTelecom-1 Keepalived_vrrp: VRRP_Instance(VI_OUT1) Entering MASTER STATE
Jul  2 07:05:49 ChinaTelecom-1 Keepalived_vrrp: VRRP_Instance(VI_OUT1) setting protocol VIPs.
Jul  2 07:05:49 ChinaTelecom-1 Keepalived_vrrp: Netlink: error: File exists, type=(20), seq=1246489542, pid=0
Jul  2 07:05:49 ChinaTelecom-1 Keepalived_vrrp: VRRP_Instance(VI_OUT1) Sending gratuitous ARPs on eth0 for 211.18.89.229
Jul  2 07:05:49 ChinaTelecom-1 Keepalived_vrrp: VRRP_Group(VGM) Syncing instances to MASTER state
Jul  2 07:05:54 ChinaTelecom-1 Keepalived_vrrp: VRRP_Instance(VI_OUT1) Sending gratuitous ARPs on eth0 for 211.18.89.229
```

图 6-3 启动 keepalived 时系统日志的输出（截取）

另外一个反映 keepalived 正常运行状态的地方是网络接口 vip 的启用。通过执行 ip add 即可看见 vip 已经被绑定在制定的网络接口（注意：ifconfig 不能显示 vip）。需要注意的是，BACKUP 的 vip 暂时不绑定。图 6-4 显示了这种比较。

MASTER	BACKUP
[root@ChinaTelecom-1 ~]# ip add list 1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00 inet 127.0.0.1/8 scope host lo inet6 ::1/128 scope host valid_lft forever preferred_lft forever 2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast qlen 100 link/ether 00:15:17:88:11:9c brd ff:ff:ff:ff:ff:ff inet 125.38.38.223/24 brd 125.38.38.255 scope global eth0 **inet 125.38.38.64/32 scope global eth0** inet6 fe80::215:17ff:fe88:119c/64 scope link valid_lft forever preferred_lft forever 3: eth1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop qlen 1000 link/ether 00:15:17:88:11:9d brd ff:ff:ff:ff:ff:ff 4: sit0: <NOARP> mtu 1480 qdisc noop link/sit 0.0.0.0 brd 0.0.0.0	[root@ChinaTelecom-5 ~]# ip add 1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00 inet 127.0.0.1/8 scope host lo inet6 ::1/128 scope host valid_lft forever preferred_lft forever 2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast qlen 100 link/ether 00:15:17:87:4e:08 brd ff:ff:ff:ff:ff:ff inet 125.38.38.227/24 brd 125.89.72.255 scope global eth0 inet6 fe80::215:17ff:fe87:4e08/64 scope link valid_lft forever preferred_lft forever 3: eth1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop qlen 1000 link/ether 00:15:17:87:4e:09 brd ff:ff:ff:ff:ff:ff 4: sit0: <NOARP> mtu 1480 qdisc noop link/sit 0.0.0.0 brd 0.0.0.0

图 6-4 主负载均衡器(MASTER) 与备份负载均衡器(BACKUP)输出对比

主负载均衡器（MASTER）和备份负载均衡器（BACKUP）的 keepalived 都把它运行起来，然后我们进行功能测试。

- 测试前的准备

1、 保证所有服务器的网络服务正常。这可以通过 ping 所有机器的 ip 地址已经 ping vip 来检查。

2、 修改本地计算机的 hosts 文件，把域名跟 vip 绑定起来。然后再 ping 一下域名，看是否正常。如 125.38.38.64 www.sery.com

- 转发功能测试

1、 在本地机器执行命令 telnet www.sery.com 80 ,检查访问是否正常。

2、 在本地计算机的浏览器地址栏输入 http://www.sery.com ，看网站默认页是否能正常访问。

3、 登录主负载均衡器，察看转发情况。Lvs 的转发情况，是不能通过 netstat –an 这样的方式来察看的。这时，前面我们安装的 ipvsadm 终于上场了。执行不带任何选项的 ipvsadm 指令，即可察看 tcp 连接情况。图 6-5 为主负载均衡器的转发情况：

```
[root@hld081028-mk ~]# ipvsadm
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  125.38.38.64:http wlc
  -> 125.38.38.99:http            Route   100    4898        12585
  -> 125.38.38.104:http           Route   100    4889        11991
  -> 125.38.38.102:http           Route   100    4890        11562
  -> 125.38.38.101:http           Route   100    4884        11658
```

图 6-5 负载均衡器转发状态

如果想知道当前测试机的访问请求被转发到那个服务器去了，可以在 ipvsadm 命令后带一个选项，其完整形式为：ipvsadm –lcn | grep 159.226.240.63 。

```
[root@hld081028-mk ~]# ipvsadm -lcn | grep 159.226.240.63
TCP    14:56      ESTABLISHED    159.226.240.63:39783    125.38.38.64:80
125.38.38.99:80
```

● 健康检查功能（故障隔离）测试

通过手工的方法，使真实服务器提供的服务实效，然后再从客户端发起访问请求，检验 lvs 提供的服务是否可用。这种测试，有两种方式：停止某个真实服务器的服务（本例为 varnish）、关闭网络服务。

1、 关闭真实服务器 125.38.38.99 的 varnish 服务。先 telnet 125.38.38.99 80 ,不能访问为正常。然后再从本地计算机执行 telnet 125.38.38.64 80,如访问正常，再检查负载均衡器的输出。这时停止服务的机器 125.38.38.99 将被 lvs 从转发列表中删除，系统日志也会有被删除的消息打印出来，如图 6-6 所示：

```
Jul  5 10:01:35 ChinaTelecom-1 Keepalived_healthcheckers: TCP connection to
[125.38.38.99:80] failed !!!
Jul  5 10:01:35 ChinaTelecom-1 Keepalived_healthcheckers: Removing service [125.38.38.99:80]
from VS [125.83.83.64:80]
```

图 6-6 服务停止，真实服务器从转发队列删除

2、 关闭主机的网络。可以关闭网络接口或关闭服务器，然后再按第“1”步的方法测试，检查客户端的访问请求是否不受影响；转发队列是否把关闭网络服务的对象删除。图 6-7 为某个真实服务器网络启停时 ipvsadm 输出对比。

未关闭 125.38.38.101 的网络服务	关闭 125.38.38.101 的网络服务
[root@hld081028-mk ~]# ipvsadm IP Virtual Server version 1.2.1 (size=4096) Prot LocalAddress:Port Scheduler Flags -> RemoteAddress:Port Forward Weight ActiveConn InActConn TCP 125.38.38.105:http wlc -> 125.38.38.99:http Route 100 4035 10736 -> 125.38.38.104:http Route 100 4027 10996 -> 125.38.38.102:http Route 100 4033 10552 -> 125.38.38.101:http Route 100 4025 10102	[root@hld081028-mk ~]# ipvsadm IP Virtual Server version 1.2.1 (size=4096) Prot LocalAddress:Port Scheduler Flags -> RemoteAddress:Port Forward Weight ActiveConn InActConn TCP 125.38.38.105:http wlc -> 125.38.38.99:http Route 100 3923 11139 -> 125.38.38.104:http Route 100 3919 10685 -> 125.38.38.102:http Route 100 3895 11788

图 6-7 关闭服务器前后 ipvsadm 输出对比

3、 关闭 lvs 的客户端配置脚本/usr/local/bin/lvs_real。这个测试不会成功，关闭真实机器的 vip 以后，负载均衡器依然会把用户请求转发过来，可是 tcp 连接却不能成功，部分用户的访问失败。因此在部署和运维 lvs 环境时，要特别注意这一点。

从上面的测试可知，关闭或停止服务，负载均衡器通过健康检查自动把实效的机器从转发队列删除掉，实现故障隔离，保证用户的访问不受影响。

● 失败切换（FailOver）测试

关闭主负载均衡器（MASTER）的 keepalived 进程，然后从客户端访问 vip 地址及本地绑定的域名。方法是 telnet 125.38.38.64 80 及用浏览器访问 http://www.sery.com . 检查访问是否正常。这是最直观的表现方法。

正常情况下，当主负载均衡器（MASTER）实效时，备份负载均衡器（BACKUP）能立即接替转发任务（接替时间由 keepalived.conf 文件的 advert_int 指定）。在确认主负载均衡器（MASTER）的 keepalived 进程关闭后，我们来看看备份负载均衡器的运行情况。这里我们观察两个地方：ipvsadm 的输出及系统日志的输出。

1、 ipvsadm 输出变化。未接替转发任务前，ipvsadm 的输出字段 ActionConn、InActionConn 对应的值皆为“0”（因为没有请求转发）；接替转发任务后，这两个字段的值立即发生变化。

2、 系统输出日志将记录备份负载均衡器从 BACKUP 向 MASTER 切换过程，下面截取系统日志关于 keepalived 输出的部分：

```
Jul 6 21:04:32 telcom-dl-1 Keepalived_vrrp: VRRP_Instance(VI_CACHE) Transition to MASTER STATE
Jul 6 21:04:32 telcom-dl-1 Keepalived_vrrp: VRRP_Group(VGM) Syncing instances to MASTER state
Jul 6 21:04:37 telcom-dl-1 Keepalived_vrrp: VRRP_Instance(VI_CACHE)
```

```
Entering MASTER STATE
Jul  6 21:04:37 telcom-dl-1 Keepalived_vrrp: VRRP_Instance(VI_CACHE) setting protocol VIPs.
Jul  6 21:04:37 telcom-dl-1 Keepalived_vrrp: VRRP_Instance(VI_CACHE) Sending gratuitous ARPs on eth1 for 218.24.35.105
Jul  6 21:04:37 telcom-dl-1 Keepalived_vrrp: Netlink reflector reports IP 125.38.38.64 added
Jul  6 21:04:37 telcom-dl-1 Keepalived_healthcheckers: Netlink reflector reports IP 125.38.38.64 added
Jul  6 21:04:42 telcom-dl-1 Keepalived_vrrp: VRRP_Instance(VI_CACHE) Sending gratuitous ARPs on eth1 for 125.38.36.64
```

现在再回来启动主负载均衡器（MASTER）的 keepalived 进程，接着察看辅助负载均衡器（BACKUP）的系统日志，截取一段输出如下：

```
Jul  6 21:18:12 telcom-dl-1 Keepalived_vrrp: VRRP_Instance(VI_CACHE) Received higher prio advert
Jul  6 21:18:12 telcom-dl-1 Keepalived_vrrp: VRRP_Instance(VI_CAHCE) Entering BACKUP STATE
Jul  6 21:18:12 telcom-dl-1 Keepalived_vrrp: VRRP_Instance(VI_CACHE) removing protocol VIPs.
Jul  6 21:18:12 telcom-dl-1 Keepalived_vrrp: VRRP_Group(VGM) Syncing instances to BACKUP state
Jul  6 21:18:12 telcom-dl-1 Keepalived_vrrp: Netlink reflector reports IP 125.38.38.64 removed
Jul  6 21:18:12 telcom-dl-1 Keepalived_healthcheckers: Netlink reflector reports IP 125.38.38.64 removed
```

这段输出显示，备份服务器尽管曾经行使了一段 MASTER 的职权，一旦原来的 MASTER 复活，它就得把控制权乖乖地交给原 MASTER，自己打回原形 BACKUP。是什么东西在起作用呢？是配置文件里设置的那个优先级“priority”。为了保证 FailOver 正常发挥作用，应确保主负载均衡器的“priority”值大于备份负载均衡器的“priority”值。同样，辅助负载均衡器的 ipvsadm 的输出也会发生变化，这里不再多做说明。

6.4 多 vrrp_instance 负载均衡应用

在“CDN 缓存服务器的负载均衡”一节中，我们详细的介绍了单实例（vrrp_instance）、单虚拟地址（vip）实现负载均衡的方方面面，在这个应用场景中，它最主要的特征就是：主负载均衡器负责转发，而备份负载均衡器则处于等待状态，只有主负载均衡器失效，备份负载均衡器才承担用户请求转发任务。在多 vrrp_instance 负载均衡应用场景下，我将让两个负载均衡器都转发用户请求，其主要用意在于提高资源的利用率。

6.4.1 多 vrrp_instance 负载均衡需求描述

本方案要实现一个 web 及自定义的 tcp 服务的负载均衡.其中 web 为 3 个站点,运行在同一个

服务器上,以虚拟机的方式实现;自定义的 tcp 服务,使用两个端口号,运行在不同的服务器上。

在这个环境中，共有 14 个服务器：2 个负载均衡器（分别命名为 lvs-1、lvs-2）,4 个 web 服务器，4 个运行自定义端口为 3000 tcp 服务的服务器，以及 4 个运行自定义端口为 4000 tcp 服务的服务器。本方案仍然使用负载均衡的 DR 模式，为了有效地使用紧缺的 ip 地址资源，我们只需要使用 2 个 vip 地址就可达到目的----web 服务使用一个 vip, 后面 2 个服务共用一个 vip。为了更清楚地理解前面的描述，表 6-4、6-5 对整个需求进行了汇总。

负载均衡器	Vip	Vrrp_instance	角色
Lvs-1	61.135.93.99 60.135.93.100	VI_WEB VI_CTCP	VI_WEB -→ MASTER VI_CTCP-→ BACKUP
Lvs-2	61.135.93.99 60.135.93.100	VI_WEB VI_CTCP	VI_WEB--→ BACKUP VI_CTCP-→ MASTER

表 6-4 负载均衡器需求汇总

项目	Vip	转发端口(TCP)	转发规则
Web（bbs、blog、www）	61.135.93.99	80	61.135.93.99:80→ 61.135.93.x:80
自定义服务 1	61.135.99.100	3000	61.135.93.100:3000→61.135.93.y:3000
自定义服务 2	61.135.93.100	4000	61.135.93.100:4000->61.135.93.z:4000

表 6-5 应用服务需求汇总

Web 服务支持的 3 个站点均为动态网站，其运行环境为 apache 加 php,因为涉及帐号登录，因此负载均衡过程必须启用会话保持。这里把 3 个站点整合到一个物理服务器上，既能保持服务配置的一致性，又能最大限度的利用资源。关于动态站点及 apache 虚拟的配置，请参看其他章节的内容。

6.4.2 多 vrrp 负载集群部署

参照“6.3.2 cdn 缓存服务器集群部署”操作步骤，多 vrrp_instance 集群的部署也按真实服务器和负载均衡器 2 个环节来处理。

- 真实服务器上进行的操作

1、 编写负载均衡客户端配置脚本。本例中有 3 组真实服务器，每组服务器使用相同的 lvs 客户端配置脚本。配置脚本除了 vip 而外，其余的部分与本章其他部分所列的 lvs 客户端配置脚本完全相同。关于 3 个组 vip 地址使用情况，请看参表 6-4，表 6-5。

2、 检验 lvs 客户端配置脚本的正确性。

- 负载均衡器上的操作

1、 负载均衡器 lvs_1

（1） 安装 ipvsadm。方法如前文所叙。

（2） 安装 keepalived。方法如前文所叙。

（3） 新增配置文件 /etc/keepalived/keepalived.conf。为了跟第二个 lvs 负载均衡器做比较，我在后面把 2 个配置文件放在一个表格里（表 6-6），方便查看。

2、 负载均衡器 lvs_2

（1）安装 ipvsadm。方法如前文所叙。

（2）安装 keepalived。方法如前文所叙。

（3）新增配置文件 /etc/keepalived/keepalived.conf。

负载均衡器 lvs_1 配置文件 /etc/keepalived/keepalived.conf	负载均衡器 lvs_2 配置文件 /etc/keepalived/keepalived.conf

#guration File for keepalived,writed by sery
#global define
global_defs {
router_id lvs_1
}
 	
vrrp_sync_group VGM {
group {
VI_WEB
}
}
vrrp_sync_group VGB {
group {
VI_CTCP
}
}
 	
vrrp_instance VI_WEB{
state MASTER
interface eth0
lvs_sync_daemon_inteface eth0
virtual_router_id 51
priority 180
advert_int 5
authentication {
auth_type PASS
auth_pass 1111
}
virtual_ipaddress {
61.135.93.99
}
}
# setting port 80 forward 	
virtual_server 61.135.93.99 80 {
	#guration File for keepalived,writed by sery
	#global define
	global_defs {
	router_id lvs_2
	}
	
	vrrp_sync_group VGM {
	group {
	VI_CTCP
	}
	}
	vrrp_sync_group VGB {
	group {
	VI_WEB
	}
	}
	
	vrrp_instance VI_WEB{
	state BACKUP
	interface eth0
	lvs_sync_daemon_inteface eth0
	virtual_router_id 51
	priority 150
	advert_int 5
	authentication {
	auth_type PASS
	auth_pass 1111
	}
	virtual_ipaddress {
	61.135.93.99
	}
	}
	# setting port 80 forward
	virtual_server 61.135.93.99 80 {

```
    delay_loop 6
    lb_algo wlc
    lb_kind DR
    persistence_timeout 10
    protocol TCP

    real_server 61.135.99.80 80 {
        weight 100
        TCP_CHECK {
            connect_timeout 3
            nb_get_retry 3
            delay_before_retry 3
            connect_port 80
        }
    }
  real_server 61.135.93.81 80 {
        weight 100
        TCP_CHECK {
            connect_timeout 3
            nb_get_retry 3
            delay_before_retry 3
            connect_port 80
        }
    }
  real_server 61.135.93.82 80 {
        weight 90
        TCP_CHECK {
            connect_timeout 3
            nb_get_retry 3
            delay_before_retry 3
            connect_port 80
        }
    }
real_server 61.135.93.83 80 {
        weight 90
        TCP_CHECK {
            connect_timeout 3
            nb_get_retry 3
            delay_before_retry 3
            connect_port 80
        }
    }
}
```

```
    delay_loop 6
    lb_algo wlc
    lb_kind DR
    persistence_timeout 10
    protocol TCP

    real_server 61.135.99.80 80 {
        weight 100
        TCP_CHECK {
            connect_timeout 3
            nb_get_retry 3
            delay_before_retry 3
            connect_port 80
        }
    }
  real_server 61.135.93.81 80 {
        weight 100
        TCP_CHECK {
            connect_timeout 3
            nb_get_retry 3
            delay_before_retry 3
            connect_port 80
        }
    }
  real_server 61.135.93.82 80 {
        weight 90
        TCP_CHECK {
            connect_timeout 3
            nb_get_retry 3
            delay_before_retry 3
            connect_port 80
        }
    }
real_server 61.135.93.83 80 {
        weight 90
        TCP_CHECK {
            connect_timeout 3
            nb_get_retry 3
            delay_before_retry 3
            connect_port 80
        }
    }
}
```

```
vrrp_instance VI_CTCP {
    state BACKUP
    interface eth0
    lvs_sync_daemon_inteface eth0
    virtual_router_id 52
    priority 150
    advert_int 5
    authentication {
        auth_type PASS
        auth_pass 2222
    }
    virtual_ipaddress {
        61.135.93.100
    }
}

virtual_server 61.135.93.100 3000 {
    delay_loop 6
    lb_algo wlc
    lb_kind DR
    persistence_timeout 50
    protocol TCP

    real_server 61.135.93.84    3000{
        weight 100
        TCP_CHECK {
            connect_timeout 3
            nb_get_retry 3
            delay_before_retry 3
            connect_port 3000
        }
    }
    real_server 61.135.93.85    3000{
        weight 100
        TCP_CHECK {
            connect_timeout 3
            nb_get_retry 3
            delay_before_retry 3
            connect_port 3000
        }
    }
    real_server 61.135.93.86    3000{
        weight 100
        TCP_CHECK {
```

```
vrrp_instance VI_CTCP {
    state MASTER
    interface eth0
    lvs_sync_daemon_inteface eth0
    virtual_router_id 52
    priority 180
    advert_int 5
    authentication {
        auth_type PASS
        auth_pass 2222
    }
    virtual_ipaddress {
        61.135.93.100
    }
}

virtual_server 61.135.93.100 3000 {
    delay_loop 6
    lb_algo wlc
    lb_kind DR
    persistence_timeout 50
    protocol TCP

    real_server 61.135.93.84    3000{
        weight 100
        TCP_CHECK {
            connect_timeout 3
            nb_get_retry 3
            delay_before_retry 3
            connect_port 3000
        }
    }
    real_server 61.135.93.85    3000{
        weight 100
        TCP_CHECK {
            connect_timeout 3
            nb_get_retry 3
            delay_before_retry 3
            connect_port 3000
        }
    }
    real_server 61.135.93.86    3000{
        weight 100
        TCP_CHECK {
```

```
                connect_timeout 3
                nb_get_retry 3
                delay_before_retry 3
                connect_port 3000
            }
        }
        real_server 61.135.93.87 3000{
            weight 100
            TCP_CHECK {
                connect_timeout 3
                nb_get_retry 3
                delay_before_retry 3
                connect_port 3000
            }
        }
}

virtual_server 61.135.93.100 4000 {
        delay_loop 6
        lb_algo wlc
        lb_kind DR
        persistence_timeout 50
        protocol TCP

        real_server 61.135.93.88    4000{
            weight 100
            TCP_CHECK {
                connect_timeout 3
                nb_get_retry 3
                delay_before_retry 3
                connect_port 4000
            }
        }
        real_server 61.135.93.89    4000{
            weight 100
            TCP_CHECK {
                connect_timeout 3
                nb_get_retry 3
                delay_before_retry 3
                connect_port 4000
            }
        }
        real_server 61.135.93.90    4000{
            weight 100
```

```
                connect_timeout 3
                nb_get_retry 3
                delay_before_retry 3
                connect_port 3000
            }
        }
        real_server 61.135.93.87 3000{
            weight 100
            TCP_CHECK {
                connect_timeout 3
                nb_get_retry 3
                delay_before_retry 3
                connect_port 3000
            }
        }
}

virtual_server 61.135.93.100 4000 {
        delay_loop 6
        lb_algo wlc
        lb_kind DR
        persistence_timeout 50
        protocol TCP

        real_server 61.135.93.88    4000{
            weight 100
            TCP_CHECK {
                connect_timeout 3
                nb_get_retry 3
                delay_before_retry 3
                connect_port 4000
            }
        }
        real_server 61.135.93.89    4000{
            weight 100
            TCP_CHECK {
                connect_timeout 3
                nb_get_retry 3
                delay_before_retry 3
                connect_port 4000
            }
        }
        real_server 61.135.93.90    4000{
            weight 100
```

TCP_CHECK {
connect_timeout 3
nb_get_retry 3
delay_before_retry 3
connect_port 4000
}
}
real_server 61.135.93.91 4000{
weight 100
TCP_CHECK {
connect_timeout 3
nb_get_retry 3
delay_before_retry 3
connect_port 4000
}
}
}
	TCP_CHECK {
	connect_timeout 3
	nb_get_retry 3
	delay_before_retry 3
	connect_port 4000
	}
	}
	real_server 61.135.93.91 4000{
	weight 100
	TCP_CHECK {
	connect_timeout 3
	nb_get_retry 3
	delay_before_retry 3
	connect_port 4000
	}
	}
	}

表 6-6 lvs 负载均衡器配置文件 keepalived.conf

6.4.3 多 vrrp_instance 负载均衡集群功能测试

在进行测试之前,需要把 lvs 客户端、负载均衡器（keepalived）、各真实服务器上的服务都启动，并逐个检验其正确性，具体的检查方法可参照本章前面的内容，不再赘述。

● 健康检查功能（故障隔离）测试

多 vrrp_instance 健康检查功能（故障隔离）测试的方法与单 vrrp_instance 相同。因多 vrrp_instance 所涉及的服务较多（本案 3 个），因此需要多花些时间逐个测试。如果多 vrrp_instance 涉及的服务很多，则可采用随机抽样的方式进行测试，通过观察 lvs 负载均衡器的转发状态(ipvsadm)及 lvs 负载均衡器的系统日志了解健康检查测试的有效性。

● 负载均衡器失败切换

本案的多 vrrp_instance 负载均衡环境正常运行后，每个 lvs 负载均衡器都有 MASTER 和 BACKUP 两种状态，每一个 lvs 负载均衡器都承担转发任务。当我们在每个 lvs 负载均衡器上执行 ipvsadm 时，其输出结果正好交错的，即 lvs-1 上如果有非 0 输出时，则 lvs_2 相对应的项为 0；反之亦然。表 6-7 为某运行环境 lvs 负载均衡器的输出截取对比。

Lvs_1 的 ipvsadm 输出（部分）	Lvs_2 的 ipvsadm 输出（部分）
IP Virtual Server version 1.2.1 (size=4096) Prot LocalAddress:Port Scheduler Flags -> RemoteAddress:Port Forward Weight ActiveConn InActConn TCP 61.128.10.4:9000 wlc persistent 50 -> 61.128.10.16:9000 Route 200 4607 139 -> 61.128.10.17:9000 Route 200 4611 154	IP Virtual Server version 1.2.1 (size=4096) Prot LocalAddress:Port Scheduler Flags -> RemoteAddress:Port Forward Weight ActiveConn InActConn TCP 61.128.10.4:9000 wlc persistent 50 -> 61.128.10.16:9000 Route 100 0 0 -> 61.128.10.17:9000 Route 100 0 0

-> 61.128.10.5:9000	Route	100	2306	65	-> 61.128.10.5:9000	Route	100	0	0
-> 61.128.10.8:9000	Route	100	2307	89	-> 61.128.10.8:9000	Route	100	0	0
……					……				
TCP 61.128.20.124:http wlc persistent 30					TCP 61.128.20.124:http wlc persistent 20				
-> 61.128.20.98:http	Route	100	0	0	-> 61.128.20.98:http	Route	100	821	1437
-> 61.128.20.93:http	Route	100	0	0	-> 61.128.20.93:http	Route	100	823	1562
-> 61.128.20.81:http	Route	100	0	0	-> 61.128.20.82:http	Route	100	823	1336
-> 61.128.20.82:http	Route	100	0	0	-> 61.128.20.81:http	Route	100	825	1033

表 6-7 多 vrrp_instance 负载均衡器转发输出对比（注意观察 ActiveConn InActConn 的值）

现在,我们来停掉 lvs_1 的 keepalived 进程,观察 lvs_2 的系统日志，将会发现 vrrp_instance(VI_WEB)从 BACKUP 状态变成 MASTER.于是 lvs_2 负载均衡器有了两个 MASTER 状态；另外 lvs_2 的 ipvsadm 的输出字段 “ActiveConn ”的值将全部变成非零。同样，当我们启动 lvs_1 的 keepalived 进程、关闭 lvs_2 的 keepalived 进程，则 lvs_1 的状态都是 MASTER,所有的转发请求都被转移到 lvs_1。

在上述 lvs_1 和 lvs_2 手动切换过程中,我们可以通过在客户端浏览器地址栏输入站点名(dns 解析域名到 lvs 的 vip）,来验证 lvs 的可用性。自定义服务可以通过在客户端执行 telnet 61.135.93.100 3000 及 telnet 61.135.93.100 4000 来初步验证。

6.5 lvs 负载均衡集群运行维护

当我们把所有的配置做好并通过各项功能测试后，就可以把这个集群环境正式运行起来（这里忽略其它处理过程），但这并不意味着万事大吉。要使负载均衡环境真正的高可用并符合业务需求,还有事情需要去做的。这些事情包括：负载均衡环境中对象新增、变更及删除，状态监控，故障的排查处理等。

6.5.1 对象新增、变更及删除

这里的对象包括 vip、真实服务器、服务（vip+端口号）等。从前面的事例可以得知，一对负载均衡器，可以承担多个服务的转发任务，因此在运行过程中，很可能因为业务本身的变化而新增、变更或删除对象。比如：某个服务负载趋于饱和，需新加服务器；有些业务下线了，需要从转发队列中把服务删除掉。

对象增加、变更及删除的操作，涉及负载均衡器和真实服务器。在负载均衡器方面，主要的操作就是修改 keepalived 的配置文件 keepalived.conf；在真实服务器上，进行的操作主要是编写 lvs 配置脚本、运行或者关闭这个配置脚本。

在有 2 个负载均衡器的 lvs 环境，所作的配置文件 keepalived.conf 变更操作要在这两个服务器上都进行一遍，以保持配置和服务的一致性。

当我们进行对象增加、变更或删除的操作时，只要注意好执行的先后顺序，就能保证提供的服务不中断，用户的正常访问不受影响。

● 对象新增

假定在负载均衡环境新增一个 web 服务器，其操作顺序是：

1、 启用新增服务器的 web 服务。
2、 启用新增服务器的 lvs 客户端配置脚本。
3、 检验“1”和“2”两步的正确性。
4、 修改负载均衡器的配置文件 keepalived.conf。
5、 关闭第一个 lvs 负载均衡器，所有的转发服务将切换到另外一个负载均衡器上。
6、 启用“5”关闭的那个负载均衡器，然后关闭“5”中还在运行的那个负载均衡器。
7、 重新启动“6”所关闭的负载均衡器。

● 删除对象

假定在负载均衡环境删除一个 web 服务器，其操作顺序是：

1、 关闭欲下线服务器的 web 服务。这样负载均衡器的健康检查会自动把该 web 服务从转发队列删除掉。
2、 卸载欲下线服务器的 vip 地址。即执行/usr/local/bin/lvs_real stop 操作。
3、 修改负载均衡器的配置文件 keepalived.conf。
4、 关闭第一个 lvs 负载均衡器，所有的转发服务将切换到另外一个负载均衡器上。
5、 启用“4”关闭的那个负载均衡器，然后关闭“4”中还在运行的那个负载均衡器。
6、 重新启动“6”所关闭的负载均衡器。

● 变更对象

与前两种方式的操作步骤基本相似，不再赘述。

这里再强调一下，如果真实服务器上的服务没关闭而把其上的 vip 卸载的话，某些用户的请求仍然会被负载均衡器转发过来，导致请求失败。因此，要记住，无论如何，请先关服务！

6.5.2 状态监控

为了随时随地了解整个 lvs 负载均衡环境的运行情况，我们必须对其进行有效的监控。当出现异常或故障时，监控系统能及时有效的通知维护人员，以便问题得以及时地处理。这也是提高可靠性的一个保障措施。

有很多开源的或商业类型的监控系统可供选择，本书选定开源的 nagios 作为监控平台，关于 nagios 的相关细节，请参照“网络服务及主机资源监控-nagios”一章。

可供 nagios 监控的对象很多，对 lvs 负载均衡环境而言，怎么选定对象才是最有效的呢？这里我们先来回顾一下 lvs 负载均衡环境运行时，其存在的表现形式有哪些？

1、 负载均衡器及真实服务器。
2、 各真实服务器上运行的服务。
3、 Lvs 公用的 vip。

根据这些表现形式，我们选取存活检查及服务状态作为监控对象，就可以清晰地了解 lvs 负载均衡环境的运行状况。把它具体化，可分为：

1、 负载均衡器及真实服务器的存活检查。只有这些服务器运行正常，才可能有其他依赖服务。

2、 Vip 的存活检查。一般情况下，启用了 lvs 环境后，是可以用 ping 的方式检查 vip 的。
3、 真实服务器服务状态检查。
4、 Vip 对应的服务状态检查。一般通过 check_tcp 加端口号的形式实现。如果 web 集群，可以以 check_http!url 的方式更精确的检查。

6.5.3 故障处理

在对 lvs 运行环境进行有效的监控后，一旦有故障或异常发生，系统管理人员将会得到及时的通知。并且这些报警信息往往包含故障的基本情况，如负载过高、主机 down 了、服务严重不可用（critical）、磁盘空间快满了等等，这些信息非常有利于系统管理员定位故障点。如果没有一个有效的监控系统，故障的报告往往来自用户的报告。这些报告笼统而模糊，可能包含“你们的网站不能访问了”之类的字眼，要定位故障点，可能会花费更多的时间。

在知晓和定位故障以后，结下来就是分析和处理故障。Lvs 负载均衡的故障点可分为：负载均衡器故障、真实服务器故障、vip 故障、服务故障这几个部分。这些故障出现后，怎么着手处理？下面分别论述之。

- 负载均衡器发生故障的检查点

1、 查看系统日志 /var/log/messages ,了解内核是否有报错信息。因为 keepalived 的日志也被追加到系统日子，因此通过系统日志，也能了解 keepalived 的运行情况。
2、 检查负载均衡器的网络连通状况。这包括 ip 地址的设置是否正确，是否能远程访问（如 ping 、tracert 等）。
3、 检查 keepalived 的运行情况。这包括进程是否处于运行中，ipvs 模块是否被加载到系统的内核，vip 是否被绑定到网络接口，ipvsadm 是否有输出。
4、 检查负载均衡器的系统负载。
5、 检查 keepalived 的配置文件书写是否正确。因为 keepalived 启动过程不对配置文件做语法检查，因此在运行前，必须按需求表逐项检查配置文件 keepalived.conf 的内容。有时，就可能就是因为漏写了一个“}”符号而导致意外的结果。配置文件的内容检查还包括主从优先级 priority、虚拟路由标识 virtual_router_id、路由标识 router_id 等几个部分，这些值有些是必须相同的、有些则必须不同。
6、 检查负载均衡器是否启用防火墙规则。

- 真实服务器发生故障的检查点

1、 查看系统日志 /var/log/messages ,了解内核是否有报错信息。
2、 检查服务器的网络连通状况。
3、 检查服务是否正常运行。可以结合察看进程、模拟用户访问来确定。
4、 检查服务器的负载情况，看哪些进程占用较高的资源。如果暂停占资源高的进程，情况会怎么样？
5、 检查 vip 是否被绑定。Linux 只能通过 ip add 指令察看，freebsd 用 ifconfig 就可以了。
6、 检查主机防火墙是否被启用。如果需要启用主机防火墙，则应设置好过滤规则。
7、 从客户端直接访问服务器的服务，看是否能正常访问。这是 dr 模式的一个优点。

- vip 发生故障的检查点

1、 检查负载均衡器的 vip 是否被绑定。

2、 检查负载均衡器 ipvsadm 的输出，察看输出的 vip 项是否与我们的设定相一致。
3、 检查各真实服务器的 vip 是否被绑定。
4、 从客户端测试一下 vip 的连通情况，如 ping vip。
5、 检查 vip 地址是否与其它服务器的地址相冲突。

● 服务发生故障检查点
1、 检查服务是否正常运行。如查进程、模拟用户访问等。
2、 检查系统的负载情况。
3、 检查是否启用主机防火墙。

一旦知道问题的所在，解决问题本身就不再是什么困难的事情了，因此这里不再一一列举对应的解决方法。

6.5.4 数据备份

Lvs 负载均衡环境需要备份的数据包括 keepalived 配置文件和 lvs 客户端配置脚本。因为这两个文件都是文本文件，并且尺寸小（几 k 而已），因此可以以复制的方式进行备份。如果把备份放在 windows 环境的话，尽量不要用 word、写字板一类的工具修改它。如果用 windows 的 word 之类的工具编辑这两个文件，上传到 linux 服务器时，会出现格式问题，而导致运行错误。当然，可以用 dos2unix 修正格式，或者在 vi 的命令模式下，以“:set ff=unix”方式转换格式。

6.6 杂项

一、lvs 负载均衡转发模式及调度算法

1、负载均衡转发模式包括直接路由模式 DR、网络地址转换模式 NAT 以及隧道模式 TUN 三种。在一般的互联网应用环境，选择直接路由模式是比较有利的，原因有：
（1）DR 利用大多数 Internet 服务的非对称特点，负载调度器中只负责调度请求，而服务器直接将响应返回给客户，可以极大地提高整个集群系统的吞吐量—引用自 http://www.linuxvirtualserver.org/zh/lvs3.html。从原理上可以知道，DR 模式下，负载均衡器的输出和输出流量应该是基本一致的，图 6-8 证实了这个事实。

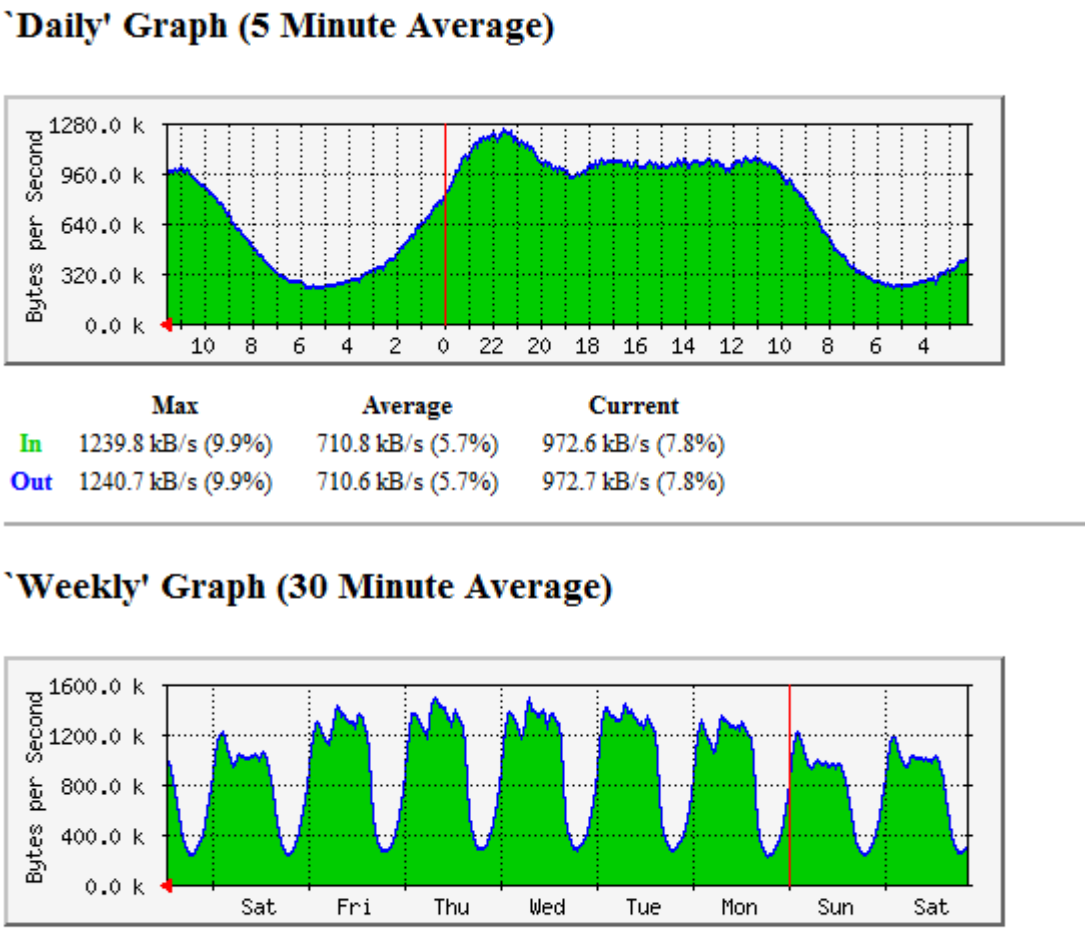

图 6-8 负载均衡器流量图

（2）排错方便迅速。如果通过 vip 访问不到服务，则可以直接通过访问真实服务器的方式直接定位问题的所在。

（3）当负载均衡器都停止工作时，DR 模式易于应急处理。通过修改 dns 的 A 记录，把先前主机名对应的 vip 改成真实服务器的 ip 地址，使服务迅速恢复起来，从而赢得时间处理负载均衡器的故障。

2、负载均衡器的调度算法

Lvs 负载均衡器的调度算法有 8 种，详情请访问 http://www.linuxvirtualserver.org/zh/lvs4.html 。一般的互联网应用，多采用轮叫调度 rr（Round-Robin Scheduling）及加权最小连接调度 wlc（Weighted Least-Connection Scheduling）。

二、lvs 负载均衡环境安全问题

前面我们讲过，在负载均衡器上，为了获得更好的转发性能，尽量不要使用主机防火墙。那么，这怎样保证系统的安全呢？个人觉得，还是采购硬件防火墙放在负载均衡器的前面比较

可靠。如果资金充裕，购买具备防 ddos 的硬件防火墙则更胜一筹。

三、同义词

网上有些词语有不同的说法，为了便于理解，这里给出相同意义的一些说法。
1、负载均衡器与 Director 为同义词。
2、真实服务器 realserver 与 pool server 是同义词。
3、ip 负载均衡技术与负载均衡模式是同义词。

四、关于负载均衡器后面真实服务器的数量

有人曾经问我：“sina 是不是做了负载均衡？是不是一个负载均衡器背后放几百个服务器？”。理论上，放几百个服务器没有问题（章文嵩博士给出的数值是 100），技术上也可能实现。但这不是一个好的策略。最佳的策略可能是：根据访问情况对大的应用进行分割，弄成一个个小的集群。具体到一个网站，可以把访问量大的频道独立出来，一个频道或几个频道组成一个 lvs 负载均衡集群。

五、话题讨论
近来 nginx 很受追捧，于是就有人使用了图 6-9 的架构来实现负载均衡。这种 3 层结构的负载均衡相对于两层的负载均衡架构，谁优谁劣？欢迎讨论。

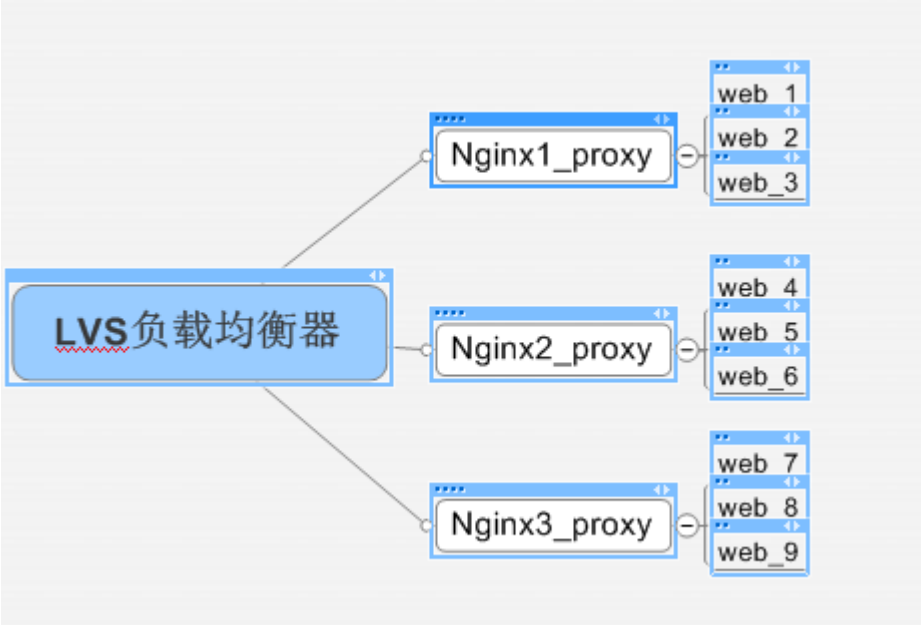

图 6-9 3 层结构负载均衡