Implementacion de Linux Virtual Server con maquinas virtuales.

Cristhoper Jaña, *Alumno Ingenieria Civil Electronica, UTFSM*

***Abstract*— Dada la creciente explosion de Internet, es necesario contar con nuevas herramientas para soportar este crecimiento. Es por ello que es necesario conocer nuevas formas de aumentar la disponibilidad y una rapida respuesta de servicios hacia los usuarios finales. El siguiente documento se presenta como un explicativo para el desarrollo de Linux Virtual Server sobre maquinas reales, utilizando maquinas virtuales que poseen un contacto directo con una red real.**

***Index Terms*— LVS, VM, balance de carga, servidor real, servidor virtual.**

I. Introduccion

UNA forma de demostrar el funcionamiento de Linux Virtual Server, utilizando bajos recursos, es utilizando maquinas virtuales. Una desventaja de este tipo de implementacion es la necesidad de que cualquier tipo de red creada con estas maquinas, debe ser realizada fuera de la maquina real, para asi no aumentar el tiempo de procesamiento en la CPU. Es de esta forma, que nuestra maquina real posee tantas interfaces de red como maquinas virtuales. Esto limita la cantidad de maquinas virtuales, pero disminuye la carga del procesador, por lo cual es necesario utilizar un switch o hub para diseñar nuestra granga de servidores.

II. Linux Virtual Server

Linux Virtual Server (LVS) es un servidor altamente escalable y de alta disponibilidad, construido en base a un cluster de servidores reales. La arquitectura es totalmente transparente para los usuarios finales y estos pueden interactuar con la granja de servidores como si solo existiera un solo servidor de alto rendimiento.

Los servidores reales y el balanceador de carga (director) pueden estar conectados por una red LAN de alta velocidad o por una red WAN dispersa (VPN) . El director, puede despachar solicitudes de diferentes servidores y crear servicios paralelos dentro del cluster de manera que sean servicios virtuales asociados a uns sola direccion IP, y la respuesta a las solicitudes pueden usar tecnologias de balance de carga a nivel de IP o nivel de aplicacion.

La escalabilidad del sistema se alcanza agregando o removiendo nodos en el cluster. La alta disponibilidad es proporcionada detectando nodos o fallas de *demonios* y reconfigurando el sistema apropiadamente.

La configuracion para LVS, puede ser variada. Existen tres formas de implementacion; utilizando NAT, IP *tunneling* y Ruteo Directo. Cada uno de ellos posee diversas ventajas y desventajas.

	LVS/NAT	**LVS/TUN**	**LVS/DR**
Servidor	Cualquiera	Tunneling	NIC no-arp
Red	Privada	WAN/LAN	LAN
Numero	20 aprox.	sobre 100	sobre 100
Gateway	Balanceador	Propio router	Propio Router

Para nuestro caso, dada la baja cantidad de maquinas, se considero la configuracion NAT.

III. Servidor Virtual via NAT

La ventaja de esta implementacion, es que los servidores reales pueden utilizar cualquier sistema operativo que soporte el protocolo TCP/IP: los servidores pertenecientes al *cluster* pueden usar IP privadas, mientras que se necesita solo una IP publica para el balanceador de carga, siendo esta ultima la IP que observara el usuario final.

La estructura de VS-NAT implementado, consta de lo siguiente:

- Servidores reales: Servidores que contiene la informacion, ya sea distribuida o redundante.
- Director: Host encargado del balance de carga y del ruteo basado en NAT, para los servidores reales.
- Cliente: Usuario Final. No observara la estructura interna de la red, solo podra acceder al servidor virtual.

Esta configuracion permite atender una mayor cantidad de solicitudes, respecto de una respuesta normal de un servidor *standalone*.

Para demostrar esto, se utilizo la misma implementacion de un servidor web, en este caso *Apache*, para todas las maquinas virtuales, obteniendose la respuesta en la tabla I y II.

La maquina virtual utilizada cumple las siguientes caracteristicas:

- Procesador Pentium 4 1.7 GHz.
- Memoria Ram 64 Mb
- Tarjeta de Red 10 Mbps

La maquina real posee:

- Procesador Pentium 4 1.7 GHz.
- Memoria Ram 512 Mb
- 6 Tarjetas de Red 10 Mbps

La idea de tener 6 tarjetas de red, es permitir a cada maquina virtual que pueda administrar una interfaz de red independiente y ademas no usar un switch por software, para asi disminuir la carga de la CPU real; de esta forma se puede.

Ahora que conocemos los tiempo de respuesta promedio por maquina, observamos que no existe mucha diferencia entre

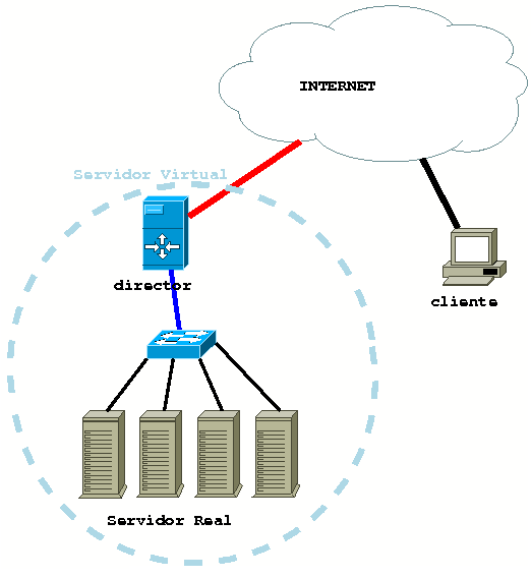

Fig. 1. Implementacion de LVS-NAT

Velocidad de transferencia	315.783$[kBytes/s]$
Solicitudes por segundo	71.24 $[1/s]$
Tiempo por solicitud	13.97 $[ms]$
Bytes solicitados	4100$[Bytes]$

TABLE I
RESPUESTA A 10.000 SOLICITUDES.

Velocidad de transferencia	238.043$[kBytes/s]$
Solicitudes por segundo	55.24 $[1/s]$
Tiempo por solicitud	18.10 $[ms]$
Bytes solicitados	4100$[Bytes]$

TABLE II
RESPUESTA A 100.000 SOLICITUDES.

10.000 y 100.000 solicitudes. Eto indica claramente que un sistema es capaz de atender a 60 clientes por segundo, lo que para un sistema de alto rendimiento es bastante malo, por ello implementamos el Servidor virtual.

Considerando la estructura de la figura **??**, la puerta de enlace para nuestros servidores debe ser nuestro balanceador de carga; por ende, debe tener previamente instalado en el kernel el modulo Virtual Server, `ipvsadm` e `iptables`. A continuacion se muestra la configuracion del kernel 2.4 para esta implementacion

```
Networking options  --->
    [*] Network packet filtering (replaces ipchains)
    [*] TCP/IP networking
      IP: Netfilter Configuration  --->
        <M> IP tables support
            (required for filtering/masq/NAT)
        <M> Full NAT
        <M> MASQUERADE target support
      IP: Virtual Server Configuration  --->
        <M> virtual server support (EXPERIMENTAL)
        [*]   IP virtual server debugging
        (12)   IPVS connection table size
        --- IPVS scheduler
        <M>   round-robin scheduling
        <M>   weighted round-robin scheduling
        <M>   least-connection scheduling scheduling
        <M>   weighted least-connection scheduling
        <M>   locality-based least-connection scheduling
        <M>   locality-based least-connection with
              replication scheduling
        <M>   destination hashing scheduling
        <M>   source hashing scheduling
        --- IPVS application helper
        <M>   FTP protocol helper

  <M> IP tables support (required for filtering/masq/NAT)
  <M>   limit match support
  <M>   MAC address match support
  <M>   netfilter MARK match support
  <M>   Multiple port match support
  <M>   TOS match support
  <M>   Connection state match support
  <M>   Unclean match support (EXPERIMENTAL)
  <M>   Owner match support (EXPERIMENTAL)
  <M>   Packet filtering
  <M>     REJECT target support
  <M>     MIRROR target support (EXPERIMENTAL)
  <M>   Full NAT
  <M>     MASQUERADE target support
  <M>     REDIRECT target support
  <M>   Packet mangling
  <M>     TOS target support
  <M>     MARK target support
  <M>   LOG target support
  < > ipchains (2.2-style) support
  < > ipfwadm (2.0-style) support
```

Para no tener problemas con la configuracion de las IP de cada servidores real, es recomendable tener un servidor DHCP para estos; de esta forma, logramos un mayor control sobre los servidores desde el director (balanceador de carga).

Lo primero ha hacer, luego de compilar nuestro kernel, es utilizar NAT para enrutar nuestros servidores reales, es por ello que ejecutamos:

```
# iptables -t nat -A POSTROUTING -s <ip_real> -j MASQUERADE
```

Ahora que tenemos los servidores escondidos, debemos colocar reglas para la atencion a los requerimientos, es decir, como administrar la cola de requerimientos.

En LVS existen distintos tipos de administracion para colas, dependiendo de la estructura de los servidores reales. Dentro de ellas estan:

- round-robin
- least-connection
- weighted round-robin
- weighted least-connection
- locality-based least-connection
- locality-based least-connection with replication
- destination hashing
- source hashing

Para nuestro caso, usaremos *weighted round-robin* pues nos permite indicar cuanto podremos cargar nuestros servidores con solicitudes, dado que no necesitamos configurar estos.

Necesitamos que los servidores, puedan acceder a la red externa, para ello se debe tener como puerta de enlace la iIP privada del director.

```
rserver# route add default gw 10.0.100.1
```

O simplemente se puede utilizar un servidor de DHCP para configurar la puerta de enlace y la direccion de cada servidor

Ahora que el trafico es permitido, desde el servidor real hasta el cliente y del cliente hasta el director, debemos realizar el balance de carga, para ello usa administracion de colas. El *scheduler* seleccionado es round robin con pesos, es decir, que

a cada cola se le asigna un nuero, y mientras mayor sea ese numero, menos solicitudes es capaz de procesar.

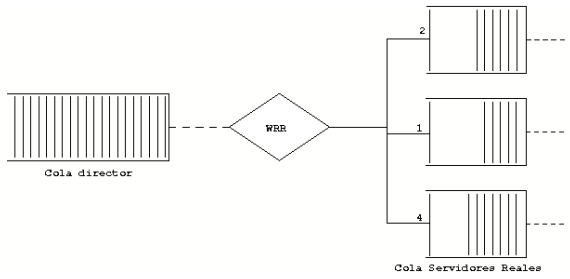

Fig. 2. Colas de peso.

Para realizar esto usamos `ipvsadm`, aplicacion que administra el servidor virtual, tanto el tipo de colas como el tipo de arquitectura

```
# ipvsadm -A -t 192.168.28.167:80 -s rr
# ipvsadm -a -t 192.168.28.167:80 -r 10.0.100.11:80 -m -w 1
# ipvsadm -a -t 192.168.28.167:80 -r 10.0.100.12:80 -m -w 1
# ipvsadm -a -t 192.168.28.167:80 -r 10.0.100.13:80 -m -w 1
# ipvsadm -a -t 192.168.28.167:80 -r 10.0.100.15:80 -m -w 1
```

Posterior a esto, habilitamos el redireccionamiento de IP

```
dir# echo 1 >> /proc/sys/net/ipv4/ip_forward
```

Al realizar mediciones sobre el cluster, obtenemos:

Velocidad de transferencia	580.75$[kBytes/s]$
Solicitudes por segundo	131.60 $[1/s]$
Tiempo por solicitud	7.60 $[ms]$
Bytes solicitados	4100$[Bytes]$

TABLE III
RESPUESTA A 100.000 SOLICITUDES.

Al realizar una comparacion para distinta cantidad de solicitudes, obtenemos

Se observa un aumento de un 154% respecto de la cantidad de solicitudes atendidas por segundo. Esto demuestra que una buena forma de responder al aumento de requerimientos sin actualizar la maquina real.

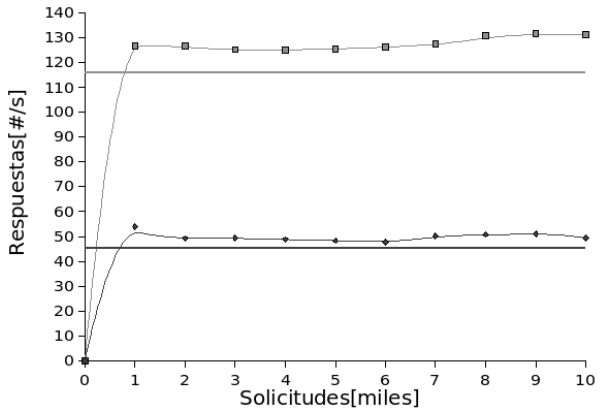

Fig. 3. Comparativa entre VS y RS

El tiempo de respuesta, de este servidor, esta limitado a la cantidad de servidores reales de la red interna. Considerando que el ancho de banda real utilizado por cada servidor real, de acuerdo con el procesamiento de los datos, es promedio $300[kB/s]$, y considerando paquetes de $1500[Bytes]$, nuestro cuello de botella se forma acuerdo al tiempo de procesamiento en la traduccion de paquetes, para el caso de este el tiempo de procesamiento para paquetes es de aproximadamente $100[us]$ por lo que el ancho de banda es de $14.3[MB/s]$ lo que permite hasta 48 servidores reales.

IV. CONCLUSION

Esta implementacion, utilizando maquinas virtuales, no pretende demostrar una mayor eficiencia frente a maquinas reales. Pero, si se pudiese virtualizar la CPU y puentear (modo *bridge*) las interfaces de red, se lograria una mejor eficiencia, pues VMware necesita un servidor X corriendo para funcionar, lo que reduce a un 95% el uso de CPU para el resto de los procesos. Ademas, la maquina en estado *stand-by*, consume un 3% de tiempo de CPU, por lo que la CPU disponible se reduce a 92% y ese tiempo debe ser repartido entre los servidores virtuales, que dejan disponible un 5% aproximadamente, restante para la maquina director.

REFERENCES

[1] Wensong Zhang *Linux Virtual Server for Scalable Network Services*, 2002
[2] Wensong Zhang, Shiyao Jin, Quanyuan Wu, Joseph Mack, Lockheed Martin *Creating Linux Virtual Servers*, `http://www.austintek.com/LVS/linuxexpo99/`
[3] Joseph Mack *LVS-HOWTO*, `http://www.austintek.com/LVS/LVS-HOWTO/`
[4] *The Linux Virtual Server Project*, `http://www.linuxvirtualserver.org/`
[5] *The User-Mode-Linux Kernel*, `http://user-mode-linux.sourceforge.net/`
[6] *VMware*, `http://www.vmware.com/`
[7] *The Linux Kernel Archives*, `http://www.kernel.org/`