Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Servidor de Internet de alta disponibilidad con equilibrado de carga

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Marcos Ramos Montes

Tutor: Pedro Juan López Rodríguez

Curso 2017 - 2018

Resumen

En los últimos años, el crecimiento en la demanda de los servicios de Internet ha provocado que, en muchos casos, un único servidor sea incapaz de atender el volumen de peticiones de los clientes. Esto ha motivado un cambio en la arquitectura de los servidores, que han pasado a implementarse mediante clusters, ya que ofrecen una excelente relación coste-prestaciones. El acceso al clúster suele realizarse a través de un nodo director, que se encarga de repartir las peticiones de los clientes entre un conjunto de servidores que son los que realmente implementan el servicio demandado. En este Trabajo Fin de Grado se plantea instalar, configurar y evaluar un prototipo servidor de Internet basado en un clúster de computadores. El sistema incorporará dos nodos directores para garantizar un funcionamiento continuo, así como mecanismos para repartir la carga entre los nodos servidores. Con el objeto de facilitar el desarrollo del trabajo, el prototipo estará basado en el uso de un entorno de virtualización, de manera que el clúster estará formado por varias máquinas virtuales. Se utilizarán herramientas y paquetes informáticos utilizadas en instalaciones reales.

El plan de trabajo previsto es el siguiente:

- ✓ Preparación e instalación del sistema en los nodos.
- ✓ Instalación y configuración de/los sistemas de equilibrado de carga con alta disponibilidad.
- ✓ Instalación y configuración de otros sistemas.
- ✓ Verificación del funcionamiento del sistema configurado.
- ✓ Evaluación del sistema configurado.

Palabras clave: cluster, equilibrado, carga, servidor, Internet, HAProxy, ipvs, keepalived

Tabla de contenidos

1.Objetivos

El objetivo de este proyecto es implementar, en un entorno virtualizado, un servidor de alta disponibilidad con equilibrado de carga todo ello en entornos Linux utilizando software disponible comercialmente.

Pretendemos realizar una demostración práctica de los principales conceptos asociados al reparto de carga y la alta disponibilidad, de manera que el sistema desplegado podría con no muchos ajustes, funcionar en un entorno productivo del mundo real, de manera que los conceptos y procedimientos que aquí se exponen son fácilmente escalables y adaptables a otros entornos. La calidad de las herramientas de virtualización disponibles permite la construcción de prototipos fácilmente adaptables que permiten la fácil realización de pruebas del tipo “Que pasaría si…”.

Dadas las limitaciones de tiempo asociadas a la realización de un trabajo fin de grado, hemos dejado fuera del proyecto el siguiente paso natural en el mismo, que es el de llevar nuestro cluster virtual a un cluster real.

Por último, comentar que si bien es fácil encontrar trabajos que abordan el reparto de carga para sitios web, no ocurre lo mismo si pretendemos repartir otro tipo de tráfico, en este sentido, el hecho de tener que repartir tanto tráfico TCP como UDP en el mismo cluster y con distintos niveles de abstracción dentro del modelo de referencia OSI, nos ha llevado a combinar dos de las principales herramientas para el reparto de carga y la alta disponibilidad en Linux como son HAProxy y keepalived.

2. Introducción

El equilibrado de carga consiste en repartir de una manera equitativa la carga de trabajo entre todos los procesadores disponibles y con ello obtener la máxima velocidad de ejecución.

Partimos de una cantidad fija de procesos que pueden ser ejecutados en paralelo. Suponiendo, además, que los procesos se distribuyen uniformemente entre los nodos disponibles y que tienen asignada una cantidad preestablecida de trabajo.

No obstante, puede darse el caso de que algunos nodos finalicen sus tareas antes que otros y queden ociosos como consecuencia de que el trabajo no se haya repartido de un modo equitativo, o bien, al hecho de que algunos nodos tengan mayor potencia de cálculo que otros. Lo que se pretende es mantener a todos los nodos lo más ocupados posibles evitando la ociosidad, consiguiendo de esta manera la mayor productividad posible. A todo el conjunto de propiedades descritas es lo que se conoce como equilibrado de carga.

La Figura 1 muestra como el reparto de carga produce el mínimo tiempo de ejecución. En la Figura 1.a, el procesador P_1 está calculando durante un periodo de tiempo mayor que el resto y el P_4 completa su trabajo en menos tiempo que los demás. El tiempo de ejecución final viene dado por el tiempo del procesador P_1. Lo ideal sería que parte del trabajo del procesador P_1 lo hiciera P_4 para igualar las cargas de trabajo. En la Figura 1.b, todos los nodos están trabajando durante todo el tiempo, consiguiéndose un reparto de carga perfecto, con lo que se disminuye el tiempo de ejecución. Otra forma de abordar esta problemática es la siguiente: para ejecutar la aplicación con un único procesador se requieren k ciclos de reloj con p procesadores. Sin considerar sobrecargas adicionales (tiempo de comunicación) para la implementación paralela, el tiempo de ejecución se reduciría a $\frac{k}{p}$ ciclos de reloj.

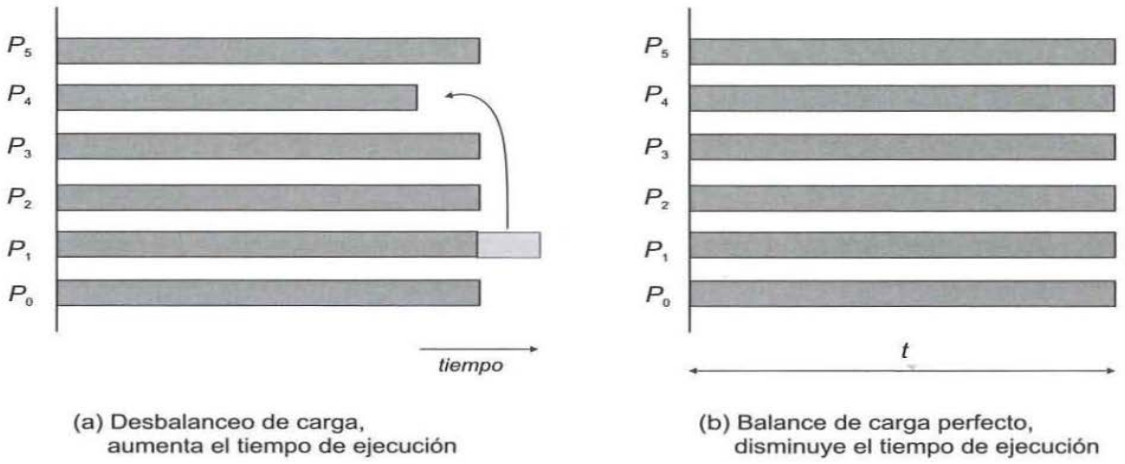


Ilustración 1 equilibrado de carga

El reparto de carga se puede realizar de manera estática, lo que quiere decir, que se realiza antes de la ejecución de cualquier proceso, y de forma dinámica, lo que quiere decir, que se lleva a cabo durante la ejecución de los procesos. Al reparto de carga estático también se le conoce como mapeado del problema o planificación del problema.

El reparto de carga estático presenta varias deficiencias comparado con el reparto de carga dinámico, siendo las más importantes los siguientes:

- ✓ Resulta complejo realizar una estimación a priori del tiempo de ejecución de las tareas.

- ✓ Hemos de tener en cuenta los retardos inherentes a la red de interconexión utilizada, siendo estos retardos difíciles de estimar.

- ✓ En el caso de algoritmos indeterministas es muy complejo estimar su tiempo de ejecución, lo que dificulta repartirlos equitativamente.

  Con el reparto de carga dinámico todos estos inconvenientes quedan resueltos, pues, el reparto de la carga depende de la naturaleza de las tareas en ejecución y no de estimaciones realizadas a priori. A pesar de que el reparto de carga dinámico acarrea una cierta sobrecarga, resulta una alternativa más eficiente que el reparto de carga estático.

  En el reparto de carga dinámico, las tareas se reparten entre los nodos en tiempo de ejecución. Dependiendo de dónde, cómo se almacenen, y se repartan las tareas, el equilibrado de carga dinámico puede ser:

  - ✓ **Reparto de carga dinámico centralizado:** Se implementa como una arquitectura del tipo maestro / esclavo, en la cual, el maestro es conocedor de la lista de tareas a realizar y se encarga de enviar dichas tareas a los esclavos. De manera que, cuando un esclavo finaliza una tarea pide una nueva tarea al maestro. A esta técnica se la conoce como programación bajo demanda o bolsa de trabajo, y es útil para problemas con tareas de cualquier tamaño.

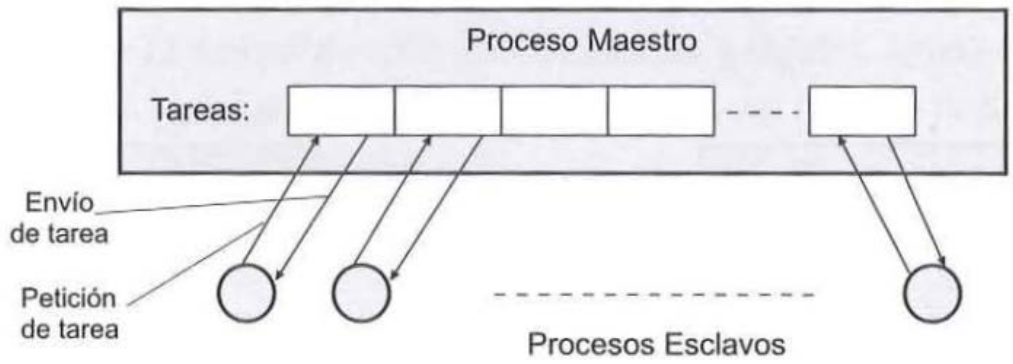


Ilustración 2 Reparto de carga dinámico centralizado

- ✓ **Reparto de carga dinámico distribuido o descentralizado:** Se emplean distintos maestros de manera que, cada uno controla a un grupo de esclavos. Tengamos en cuenta que, el reparto de carga dinámico distribuido tiene el inconveniente de que el maestro debe repartir las tareas de una en una, y lo mismo ocurre cuando recibe peticiones de trabajo por parte de los esclavos, lo que implica que pueden producirse colisiones en caso de que varios esclavos demanden nuevas tareas simultáneamente.

  En problemas con tareas de diferentes tamaños es conveniente asignar primero las tareas de mayor carga computacional, ya que, si la tarea más pesada se dejase para el final, las tareas pequeñas se completarían rápidamente con lo que los esclavos quedarían ociosos esperando a que se completase la tarea más compleja.

  Es posible utilizar una cola de tareas pendientes como se observa en la Figura 2, este enfoque es adecuado en caso de que las tareas sean similares en tamaño y prioridad, en otro caso, se debe analizar cuál es la estructura de datos más idónea.

  El enfoque centralizado es adecuado únicamente en caso de tener pocos esclavos y que las tareas no supongan una gran carga de trabajo. De no ser así, es más conveniente distribuir las tareas.

La Figura 3 muestra el esquema de un repartidor de carga dinámico distribuido. El maestro divide su trabajo inicial en varias partes y envía una a cada máquina que actúa como mini-maestro (de M_0 a M_{n-1}). Cada mini-maestro controla un conjunto de máquinas esclavas. Así, por ejemplo, para un problema de optimización, las máquinas que actúan como mini-maestros podrían encontrar un óptimo local y enviárselo al maestro para que seleccione la mejor solución.

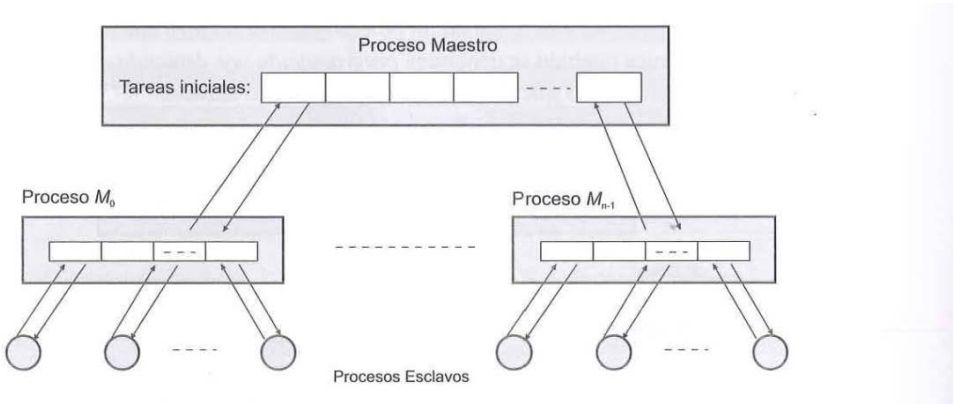


Ilustración 3 Reparto de carga dinámico distribuido

En este proyecto implementaremos un repartidor de cargar dinámico y centralizado.

3. Configuración del servidor de Internet

En este apartado vamos describir detalladamente todos los pasos involucrados en la configuración de nuestro cluster. El primer paso consiste en crear una red NAT que incluirá tres equipos: Master, Standby y un cliente desde el que realizar pruebas.

El esquema de trabajo que vamos a seguir consistirá en configurar en primer lugar, un nodo NFS que proporcionará almacenamiento compartido a todas las máquinas del cluster, para después pasar a configurar el nodo master y los servers. En este sentido, en lo referente a la configuración de los servidores hemos optado por que cada servidor ofrezca un único servicio, con lo que una vez tengamos configurado un servidor de un determinado tipo (HTTP / HTTPS, DNS), lo replicaremos mediante la opción de clonación de VirtualBox y realizaremos los ajustes de configuración en las réplicas tal y como describiremos más adelante. Respecto al direccionamiento de la red interna del cluster se han seguido los siguientes criterios:

- ✓ Se ha definido la red interna[1] **10.0.100.0/24**
- ✓ Se ha definido la red NAT[2] **192.168.1.0/24**
- ✓ Para los servidores se utilizan las direcciones MAC de la **080027010102** en adelante (la razón de esto se explicará mas adelante, al configurar el nodo master).
- ✓ Se han creado 3 servidores HTTP / HTTPS y 3 servidores DNS.
- ✓ Se ha dotado de interfaz gráfica de usuario a los nodos directores Master y Standby para facilitar las posteriores tareas de verificación y pruebas del cluster.

[1] Véase sección 3.1 donde se define el concepto y utilidad en la terminología de VirtualBox
[2] Ídem para la red nat

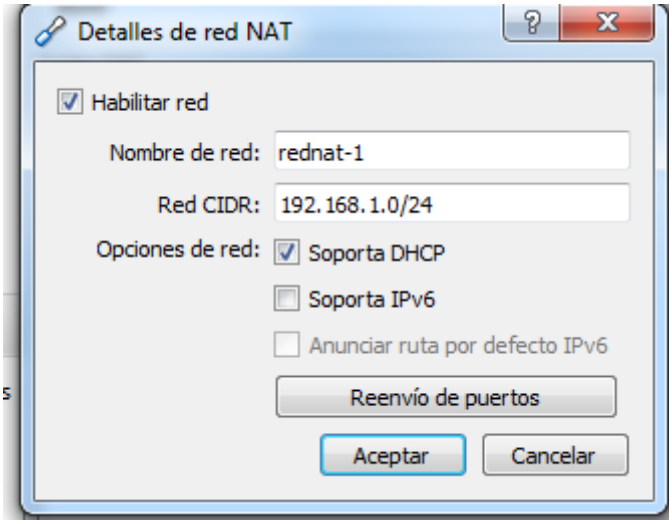


Ilustración 4 Configuración de la red NAT

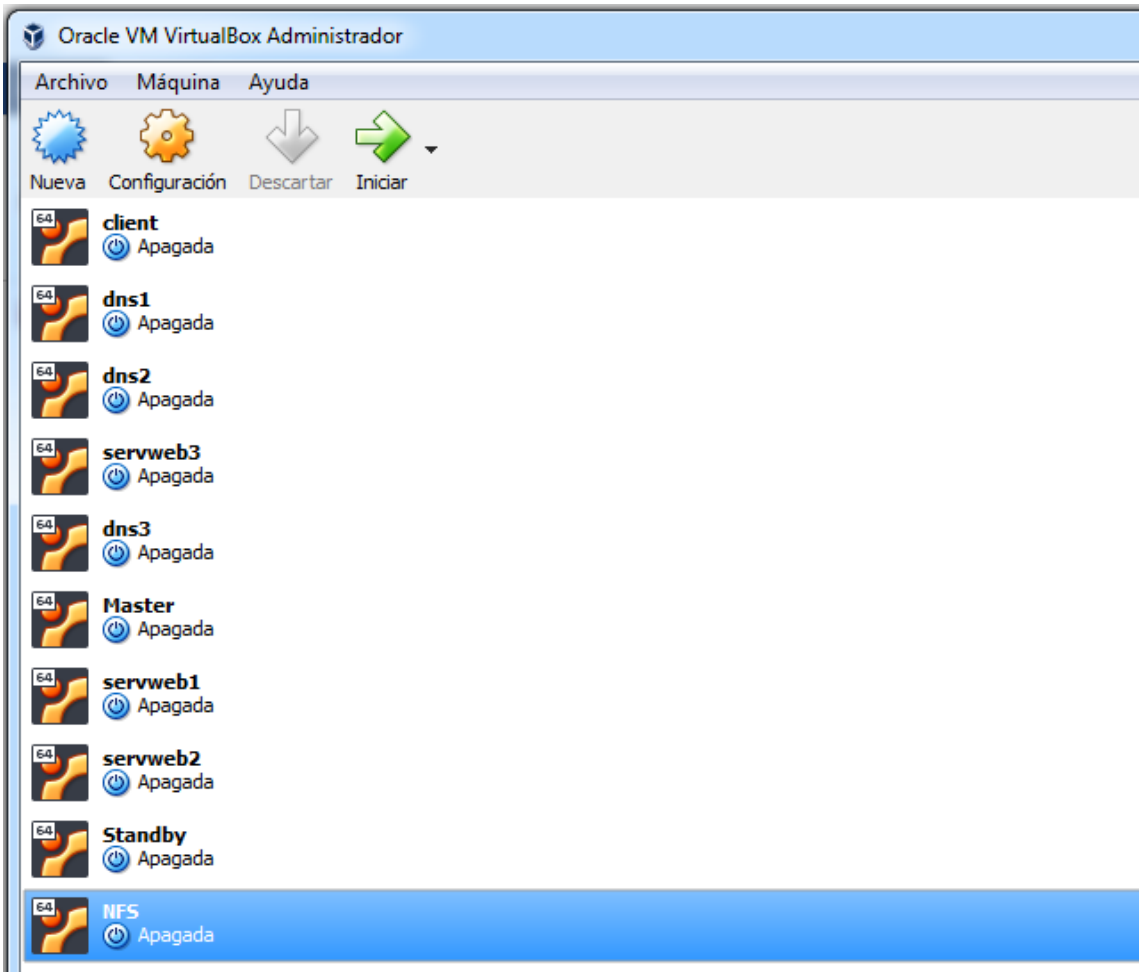


Ilustración 5 Interfaz de VirtualBox con todas las máquinas del cluster

Como se puede observar, el elevado número de máquinas virtuales creadas es lo que justifica que los nodos servidores tengan menos memoria RAM que los nodos master y standby ya que de no ser así, correríamos el riesgo de experimentar un bajo rendimiento en nuestro cluster virtual o incluso que la máquina anfitriona pese a disponer de 32GB de memoria RAM se quede sin recursos.

Comentar también que se ha hecho uso de la herramienta de toma de instantáneas de VirtualBox, característica que permite “congelar” la configuración de una máquina virtual asignándole un nombre para poder restaurar dicha configuración posteriormente en caso de tener problemas con la máquina virtual.

Como última anotación de este punto, comentar que sólo haremos mención a la instalación de Ubuntu server en la configuración del nodo de almacenamiento, que es el primero que vamos a instalar.

3.1. Estructura del servidor

Vamos a describir a continuación cuál va a ser el entorno de trabajo que se va a implementar. El entorno de trabajo estará totalmente virtualizado mediante la aplicación VirtualBox de Oracle versión 5.2.12, utilizando Microsoft Windows 7 x64 como sistema operativo anfitrión. En la siguiente figura podemos observar un esquema del entorno de trabajo.

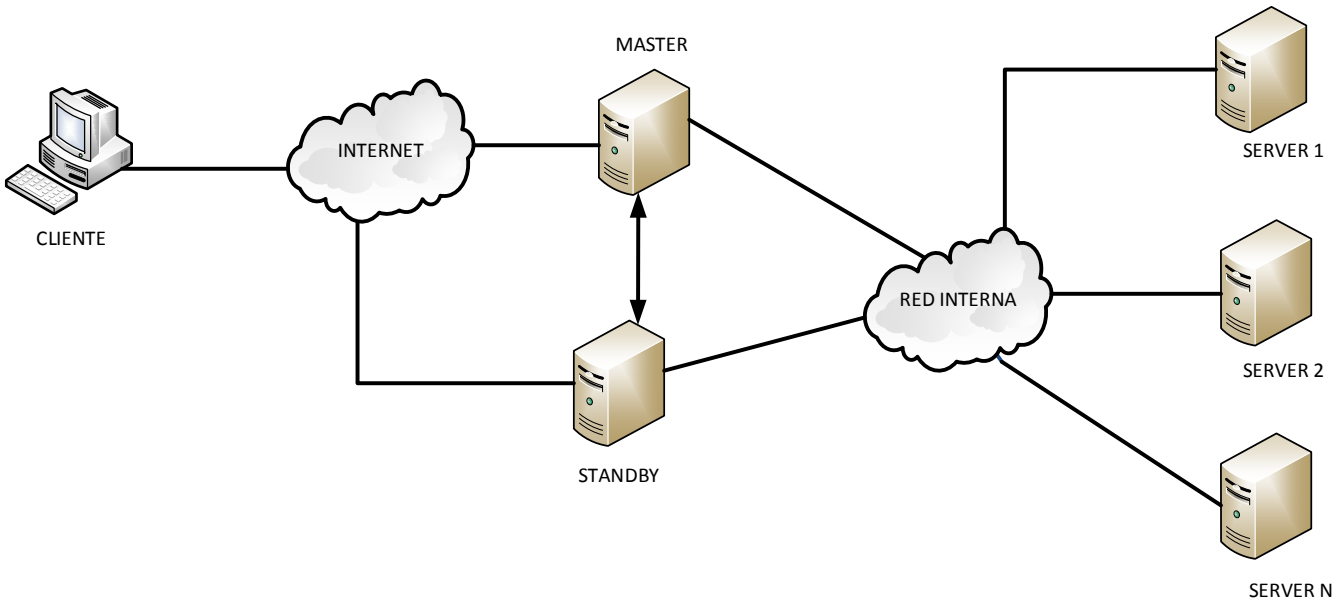


Ilustración 6. Entorno de trabajo virtualizado

Como se puede observar, tenemos un cliente que solicita desde Internet los servicios de nuestra red interna. En principio, el nodo master hace de pasarela entre la red interna donde residen los servidores que implementan los servicios que suministra el cluster e Internet que es desde donde nos llegan las peticiones.

Para que esto sea posible, el nodo master dispone de dos tarjetas de red, una conectada a la red interna y otra conectada a Internet. El nodo Stanby es una réplica del nodo master y actúa como nodo de respaldo en caso de que el nodo master deje de funcionar.

Todas las máquinas de nuestro cluster utilizan GNU / LINUX en su variante Ubuntu Server 16.0.4 LTS en versión de 64 bits, excepto el cliente que utiliza la misma distribución, pero en una versión para escritorio. A continuación, pasaremos a dar detalles sobre la creación de las distintas máquinas virtuales en la aplicación VirtualBox.

En primer lugar, veamos cual es la configuración básica de las máquinas virtuales que componen el cluster:

Nodo Master: Sistema operativo: GNU / Linux Ubuntu Server 16.0.4 LTS de 64bits

- 1GB de RAM
- 10GB de disco duro
- 2 tarjetas de red, la Intel PRO 1000 como adaptador1 conectado a la red NAT y una segunda tarjeta de red conectada a la red interna.

Nodo Standby: Misma configuración que nodo master.

Nodos servidores. La diferencia respecto al nodo master es que los nodos servidores sólo tienen una tarjeta de red que está conectada a la red interna y que disponen de 256MB de memoria RAM.

En este punto, es conveniente aclarar el significado de los términos **red interna** y **red NAT** en la terminología de VirtualBox. Una red interna es el equivalente a una red de área local, es decir, los nodos que la componen pueden verse entre sí, pero no tienen salida a otras redes para lo que precisan conectarse a una pasarela o router que les permita conectarse a otras redes como, por ejemplo, Internet. En el caso de la red NAT, los nodos que la componen pueden verse entre sí y además pueden conectarse a otras redes, dicho de otro modo, los nodos que componen una red NAT pueden actuar como pasarelas, razón por la cual los nodos Master y Standby tienen una de sus respectivas tarjetas de red conectadas a la red NAT, lo que les permite actuar como pasarelas para los servidores de la red interna del cluster.

Concretamente, los servicios que va a ofrecer nuestro cluster son un servidor HTTP / HTTPS y un servidor DNS. Como servidor HTTP / HTTPS utilizaremos apache, mientras que como servidor DNS utilizaremos bind.

Para ofrecer alta disponibilidad y equilibrado de carga, hacemos uso de HAProxy e IPVS configurando este último mediante el paquete keepalived. Una de las razones de utilizar estas dos herramientas es que combinándolas adecuadamente es posible ofrecer alta disponibilidad a la vez que nos permiten manejar tanto tráfico TCP como UDP, además de que, si bien keepalived sólo puede trabajar en capa de transporte, HAProxy puede hacerlo tanto en capa de transporte como en capa de aplicación, pero con la limitación de no poder gestionar tráfico UDP.

3.2. Configuración del nodo de almacenamiento

Partimos de una maquina virtual de nombre NFS, sin sistema operativo, por lo que el primer paso es instalar Ubuntu Server. Ha continuación indicamos las principales configuraciones realizadas durante este proceso.

- ✓ Nombre de máquina: **nas**
- ✓ Usuario por defecto: **marcos**
- ✓ Contraseña usuario: **123**
- ✓ Tipo de particionado: Manual, y definimos las particiones siguientes, todas ellas primarias:
  - ❖ /dev/sda1. Sistema (punto de montaje: /). 4GB. Formato ext4.
  - ❖ /dev/sda2. Almacenamiento local (punto de montaje: /home). 6GB. Formato ext4.
  - ❖ /dev/sda3. Swap. Como tamaño, el espacio restante. Formato “área de intercambio”.
- ✓ Configuración automática de red: Sin proxy
- ✓ Sin actualizaciones automáticas
- ✓ En la ventana de “Selección de programas”, elegimos openSSH Server
- ✓ Instalamos el cargador de arranque GRUB

Tras esto se descargarán e instalarán algunos paquetes y con ello ya tendremos Ubuntu Server instalado. La primera vez que arranquemos el sistema tendremos que iniciar sesión con el usuario que hemos creado durante la instalación, “marcos” en nuestro caso. Ahora, lo primero que tenemos que hacer es habilitar la cuenta de administrador ya que de esta manera podremos configurar la máquina sin tener que utilizar el comando sudo:

```
sudo passwd root
```

Una vez que le hemos asignado contraseña a la cuenta root, iniciamos sesión con ella, actualizaremos el sistema e instalaremos el servidor NFS:

```
apt-get update
apt-get install rpcbind nfs-kernel-server
```

Ahora, vamos configurar el servidor SSH para que nos permita conectarnos remotamente como root. Para ello, la línea PermitRootLogin debe quedar tal y como se muestra:

/etc/ssh/sshd_config:

```
…
PermitRootLogin yes
```

Reiniciamos el servicio para que los cambios de configuración surtan efecto:

```
service ssh restart
```

Vamos a configurar la red:

/etc/network/interfaces:

```
…
auto eth0
iface eth0 inet static
address 10.0.100.100
netmask 255.255.255.0
gateway 10.0.100.1
mtu 9000
dns-nameservers 10.0.100.1
```

Con la configuración anterior estamos indicando que la tarjeta de red “eth0” se configura estáticamente con la dirección IP y la máscara de red que se indica. Los parámetros dns-nameservers y gateway indican que la máquina nas utiliza al nodo master como pasarela y como servidor DHCP.

El parámetro mtu indica la máxima unidad de transferencia expresada en bytes y es un valor de interés para el protocolo de la capa de enlace de datos, ethernet en nuestro caso.

Ahora, vamos a configurar el nodo de almacenamiento de manera que exporte su directorio /home con el seudónimo /data a todas las máquinas de la red interna. Para ello, es necesario modificar el siguiente par de ficheros:

/etc/fstab:

```
…
/home /data none bind 0 0
```

/etc/exports:

```
…
/data 10.0.100.0/24(fsid=0,rw,sync,no_subtree_check,no_root_squash)
```

Por último, crearemos el directorio /data, lo montamos y comprobamos que el servidor NFS se inicia correctamente:

```
mkdir /data; mount /data; service nfs-kernel-server restart
```

Apagamos la máquina virtual, conectamos su tarjeta de red a la red interna e iniciamos nuevamente la máquina virtual NFS.

3.3. Configuración del nodo master

Partimos de la máquina virtual master la cual contiene una instalación limpia de Ubuntu Server en la cual hemos instalado soporte para openSSH y hemos habilitado la cuenta root tal y como se describe en el apartado anterior.

Vamos a instalar GRUB v2 y a configurarlo para que no busque automáticamente los sistemas instalados, sino que, en vez de ello, le indicaremos que sistemas deseamos arrancar.

Instalación de GRUB v2:

```
apt-get update
apt-get install grub2
grub-install /dev/sda
```

Deshabilitar la búsqueda automática de sistemas instalados:

```
chmod –x /etc/grub.d/10_linux
chmod –x /etc/grub.d/30_os-prober
```

Creamos el fichero 40_custom, en el que definiremos una entrada personalizada para GRUB indicándole que arranque desde la partición que contiene el sistema.

/etc/grub.d/40_custom:

```
menuentry “Ubuntu Servidor de Internet” {
set root=(hd0,1)
```

```
linux /vmlinuz root=/dev/sda1
initrd /initrd.img
}
```

Modificaremos la configuración de GRUB para indicarle que espere 10 segundos antes de arrancar el sistema. Hacemos esto para que los servidores DHCP y NFS puedan arrancar antes que el resto de nodos del cluster:

/etc/default/grub:

```
…

GRUB_TIMEOUT=10

…
```

Reiniciamos GRUB para que los cambios surtan efecto:

```
update-grub
```

Ahora vamos a configurar las dos tarjetas de red de que dispone el nodo maestro de manera que la primera de ellas (eth0), se configurará dinámicamente mediante DHCP para salir a Internet y la segunda tarjeta de red (eth1) se configurará de manera estática para conectarse a la red interna del cluster.

/etc/network/interfaces:

```
auto lo
iface lo inet loopback
auto eth0
iface eth0 inet dhcp
auto eth1
iface eth1 inet static
address 10.0.100.1
netmask 255.255.255.0
mtu 9000
```

Reiniciamos la segunda tarjeta de red (eth1) para que los cambios efectuados en su configuración surtan efecto:

```
ifdown eth1; ifup eth1
```

Como dijimos anteriormente, el nodo master debe actuar como pasarela a Internet para las máquinas de la red interna. Para que esto sea posible, es necesario ajustar adecuadamente la configuración del cortafuegos iptables de manera que permita ip-forwarding y añadir una regla para modificar los paquetes antes de que salgan por la tarjeta de red cambiando la dirección IP de origen por la que tenga la interfaz de salida.

/etc/rc.local:

```
sysctl –w net.ipv4.ip_forward=1
iptables –P FORWARD ACCEPT
iptables –-table nat –A POSTROUTING –o eth0 –j MASQUERADE
```

El fichero rc.local se carga durante el arranque de Linux, pero, para asegurarnos de que no hemos cometido ningún error sintáctico al modificarlo, podemos darle permiso de ejecución y ejecutarlo manualmente:

```
chmod +x /etc/rc.local
/etc/rc.local
```

Vamos a preparar el acceso mediante SSH y clave pública:

```
ssh-keygen
cat /root/.ssh/id_rsa.pub >> /root/.ssh/authorized_keys
```

Posteriormente difundiremos las claves autorizadas al resto de nodos de la red interna, por el momento vamos a editar el fichero de hosts.

/etc/hosts:

```
127.0.0.1   localhost
10.0.100.100    nas.cluster        nas
10.0.100.1      cluster1.cluster   cluster1    master    lb1
10.0.100.2      cluster2.cluster   cluster2    web1
10.0.100.3      cluster3.cluster   cluster3    web2
10.0.100.4      cluster4.cluster   cluster4    web3
10.0.100.5      cluster5.cluster   cluster5    dns1
10.0.100.6      cluster6.cluster   cluster6    dns2
10.0.100.7      cluster7.cluster   cluster7    dns3
10.0.100.50     standby.cluster    standby
```

Ya podemos copiar las claves al servidor NFS para que nos permita el acceso SSH mediante clave pública:

```
ssh nas “mkdir /root/.ssh”
scp /root/.ssh/id_rsa.pub nas:/root/.ssh/authorized_keys
```

Copiamos el fichero de hosts al servidor NFS:

```
scp /etc/hosts nas:/etc
```

Instalar y configurar el cliente NFS:

```
apt-get install  nfs-common
echo “nas:/ /nfs nfs auto,rsize=8192,wsize=8192 0 0” >> /etc/fstab
```

La primera entrada muestra el nombre del servidor NFS (“nas”) y la ruta (“/”). La segunda entrada indica el punto de montaje (“/nfs”). La tercera entrada indica el tipo de sistema de ficheros a emplear, al no especificar la versión de nfs (como nfs4 o nfs3) se comprueba primero si existe el directorio remoto según la sintaxis de la v4 y, si no se encuentra, se intenta empleando la versión 3. Ahora, sólo nos queda crear el directorio que actuará como punto de montaje:

```
mkdir /nfs
mount /nfs
```

En este punto, obtenemos una instantánea del nodo master que llamamos “MODELO” a partir de la cual clonaremos más adelante al resto de servidores de la red interna.

A continuación, vamos a instalar un entorno de escritorio en el nodo master, ya que así nos resultará más fácil realizar las tareas posteriores de verificación y pruebas en el cluster. Vamos a instalar lxde un escritorio muy completo que consume pocos recursos. También instalaremos el gestor de ventanas lxdm:

```
apt-get install lxde
apt-get install lxdm
apt-get install lxsession-logout
```

Ilustración 7 Interfaz del escritorio LXDE

Ahora, desde LXTerminal, continuaremos configurando el nodo master, concretamente instalaremos el paquete dnsmasq, que utilizaremos como servidor DNS y DHCP para los servidores de nuestra red interna:

```
apt-get install dnsmasq
service dnsmasq stop
```

Obsérvese que tras instalar el paquete detenemos el servicio para configurarlo:

/etc/dnsmasq.conf:

```
dhcp-range=10.0.100.2,10.0.100.50,infinite
log-dhcp
dhcp-option=26,9000

dhcp-option=6,10.0.100.1
interface=eth1
read-ethers
```

La opción interface=eth1 indica que sólo debe atender peticiones procedentes de la red interna. Las opciones dhcp-option=26, 3 y 6 fijan la mtu, la puerta de enlace y el servidor DNS, respectivamente. La opción read-ethers nos permite asignar parejas dirección MAC – dirección IP de manera que a cada dirección MAC siempre le corresponde la misma dirección IP. Para ello debemos editar el fichero /etc/ethers, que en nuestro caso alberga el siguiente contenido:

/etc/ethers:

```
08:00:27:01:01:02    10.0.100.2
08:00:27:01:01:03    10.0.100.3
```

```
08:00:27:01:01:04    10.0.100.4
08:00:27:01:01:05    10.0.100.5
08:00:27:01:01:06    10.0.100.6
08:00:27:01:01:07    10.0.100.7
```

Como se puede observar en este fichero, ya queda claro el sentido de la norma que impusimos anteriormente de asignar a los servidores de la red interna direcciones MAC de la 08:00:27:01:01:02 en adelante, y es que esto se hace para gestionar adecuadamente la asignación de direcciones IP en la red interna, tal y como hemos explicado unas líneas más arriba.

Llegados a este punto, reiniciamos el nodo master y pasamos a configurar los servidores de la red interna, posteriormente volveremos a este nodo para realizar algunos ajustes.

Una puntualización a tener en cuenta es que para que los servidores de la red interna funcionen correctamente, es necesario que tanto el nodo NFS como el nodo master hayan arrancado previamente y se mantengan en funcionamiento.

3.4. Configuración de los servidores web

Realizamos una clonación completa de la instantánea “MODELO” que hemos creado previamente, bautizando a la nueva máquina virtual resultante como “servweb1”. Nos vamos a la configuración de su tarjeta de red para asignarle como dirección MAC la `08:00:27:01:01:02:`

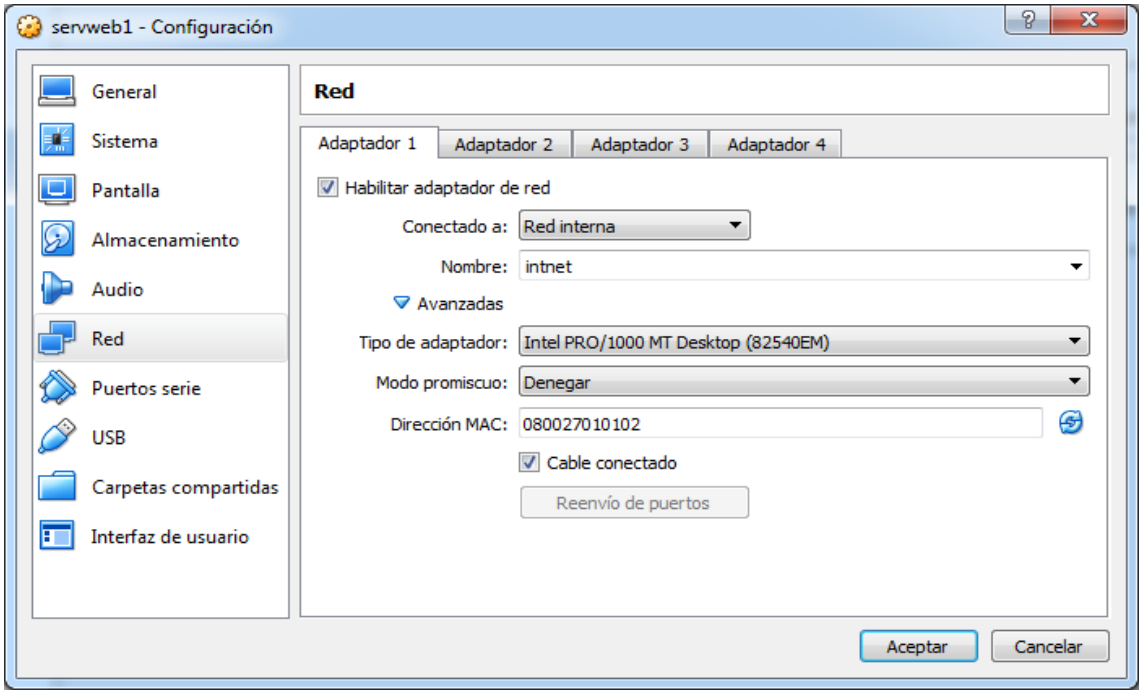


Ilustración 8 Configuración de la dirección MAC en VirtualBox

Arrancamos servweb1 y procedemos a instalar apache con soporte PHP para así poder trabajar con páginas web dinámicas:

```
apt-get update

apt-get install apache2

apt-get install php libapache2-mod-php
```

Además, habilitamos el módulo remoteip para poder acceder desde los servidores a la cabecera “XForwarded-For” que nos permitirá conocer la identidad de los clientes reales cuando utilicemos HAProxy en modo http:

```
a2enmod remoteip

service apache2 restart
```

Comprobamos mediante un navegador web gráfico o textual que el servidor web funciona correctamente. Para ello:

```
curl localhost
```

Ahora, vamos a cambiar la página de inicio de apache de manera que se trate de una página dinámica ubicada en el servidor NFS.

`/etc/apache2/sites-enabled/000-default.conf:`

```
…
DocumentRoot /nfs/www
…
```

/etc/apache2/apache2.conf:

```
…
<Directory /nfs/www/>
Options Includes
AllowOverride All
Require all granted
</Directory>
```

Utilizamos el siguiente script PHP como página de inicio:

/nfs/www/index.php:

```
<?php
header("Refresh: 10");
echo "Servidor Web cluster<br>";
echo "Hoy es " . date ("d/m/Y"). ".";
echo "Son las " . date ("H:i:s") . "<br>";
echo "Soy el servidor IP: ". $_SERVER['SERVER_ADDR'] . "<br>";
echo "Cliente IP: ". $_SERVER['REMOTE_ADDR'] . "<br>";
if ( isset( $_SERVER['HTTP_X_FORWARDED_FOR'] ) )
{
echo "Forwarded-For: ".
$_SERVER['HTTP_X_FORWARDED_FOR'] . "<br><br>";
} else {
echo "Forwarded-For: ". "Desconocido" . "<br><br>";
}
echo "Cookies:<br>";
print_r($_COOKIE);
?>
```

Para que los cambios realizados en la configuración de apache surtan efecto, reiniciamos el servicio:

```
service apache2 restart
```

curl localhost:

```
Servidor Web pruebas cluster<br>Hoy es 12/06/2018.Son las
11:21:30<br>Soy el servidor IP: 127.0.0.1<br>Cliente IP:
127.0.0.1<br>Forwarded-For: Desconocido<br><br>Cookies:<br>Array

(

)
```

En este punto tenemos ya nuestro servidor web configurado. Seguidamente, vamos configurar el soporte necesario para https mediante openssl, lo haremos mediante un certificado auto-firmado, que, si bien es cierto que no tiene validez en Internet al no estar respaldado por una entidad certificadora, es suficiente para hacer pruebas. Partiendo de que ya tenemos apache instalado hacemos lo siguiente:

- ✓ Habilitamos el módulo SSL
- ✓ Reiniciamos apache para que se active el módulo SSL

- ✓ Generamos un certificado con cifrado RSA de 2048 bytes y con un periodo de validez de 365 días

```
a2enmod ssl
service apache2 restart
mkdir /etc/apache2/ssl
openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/apache2/ssl/apache.key -out /etc/apache2/ssl/apache.crt
```

A continuación, nos pedirán una serie de datos para rellenar el certificado que estamos creando.

Editamos el fichero /etc/apache2/sites-enabled/default-ssl.conf:

```
…
SSLCertificateFile /etc/apache2/ssl/apache.crt
SSLCertificateKeyFile /etc/apache2/ssl/apache.key
…
```

Habilitamos un host virtual:

```
a2ensite default-ssl.conf
```

Otra cosa que tenemos que hacer es asegurarnos de que la ruta de la página de inicio de https apunte a /nfs/www igual que en la configuración de http. Para ello, en <VirtualHost default_:443>, editamos la siguiente línea:

/etc/apache2/sites-enabled/default-ssl.conf:

```
DocumentRoot /nfs/www
```

Reiniciamos el servicio:

```
service apache2 restart
```

3.4.1. Comprobación de funcionamiento

```
curl -k localhost
```

Para acceder desde la máquina master haríamos:

```
curl -k https://10.0.100.2
```

o bien:

```
curl http://10.0.100.2
```

En caso de usar un navegador gráfico como, por ejemplo, Firefox, al usar un certificado auto-firmado, la primera vez que accedemos a la página mediante https tenemos que añadir una excepción de seguridad:

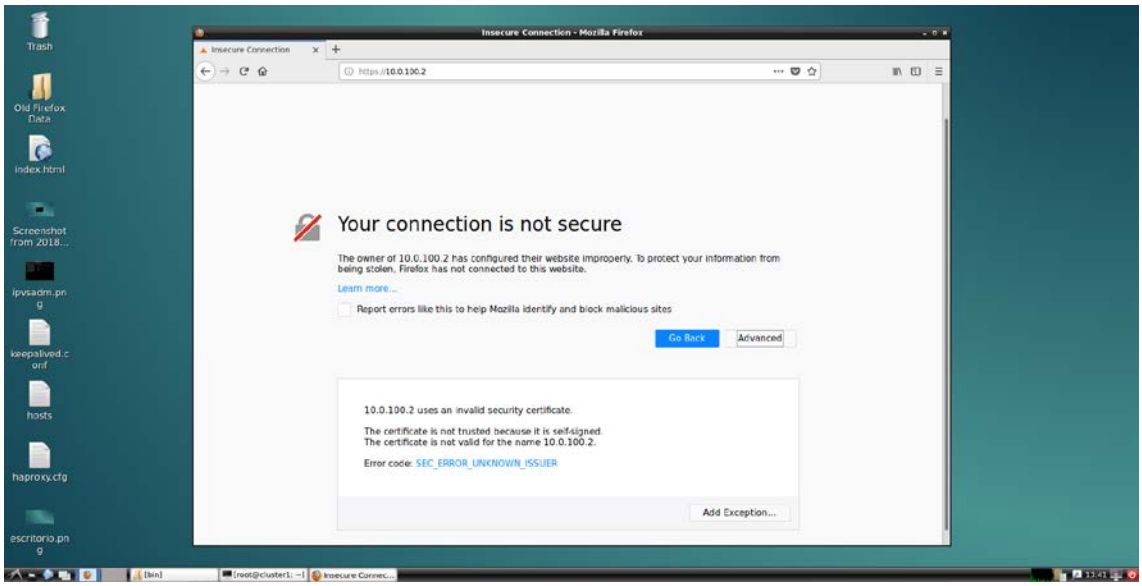


Ilustración 9 Advertencia de seguridad por certificado auto – firmado

Pulsamos en Add Exception…

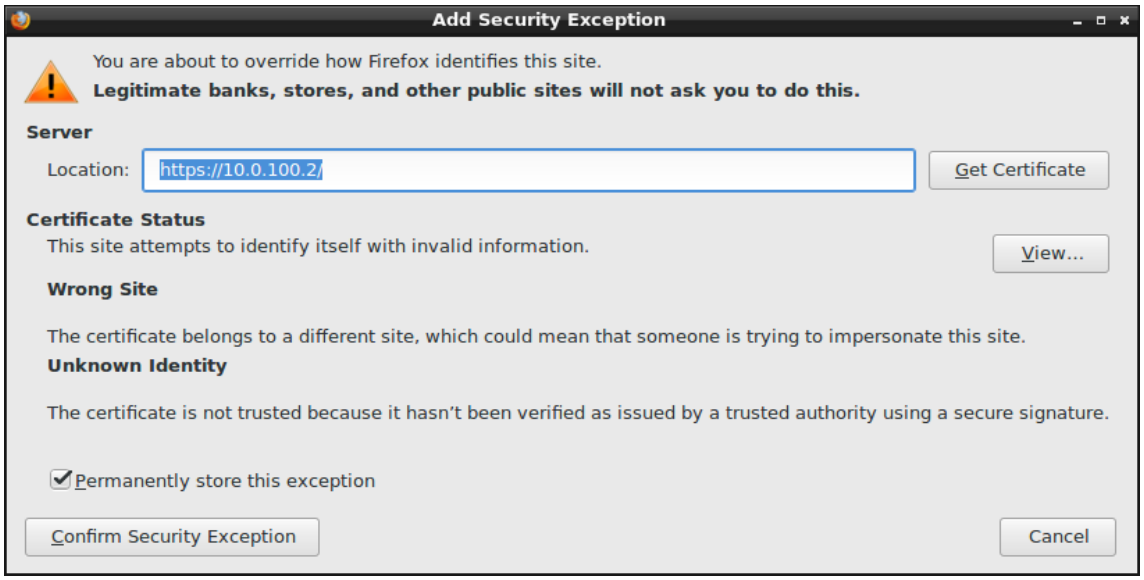


Ilustración 10 Aquí podemos ver los datos del certificado y confirmar la excepción

Pulsamos el botón “View…” para ver los datos del certificado que hemos creado, lo cual, nos permite comprobar que se ha generado correctamente según los datos que introdujimos al generarlo:

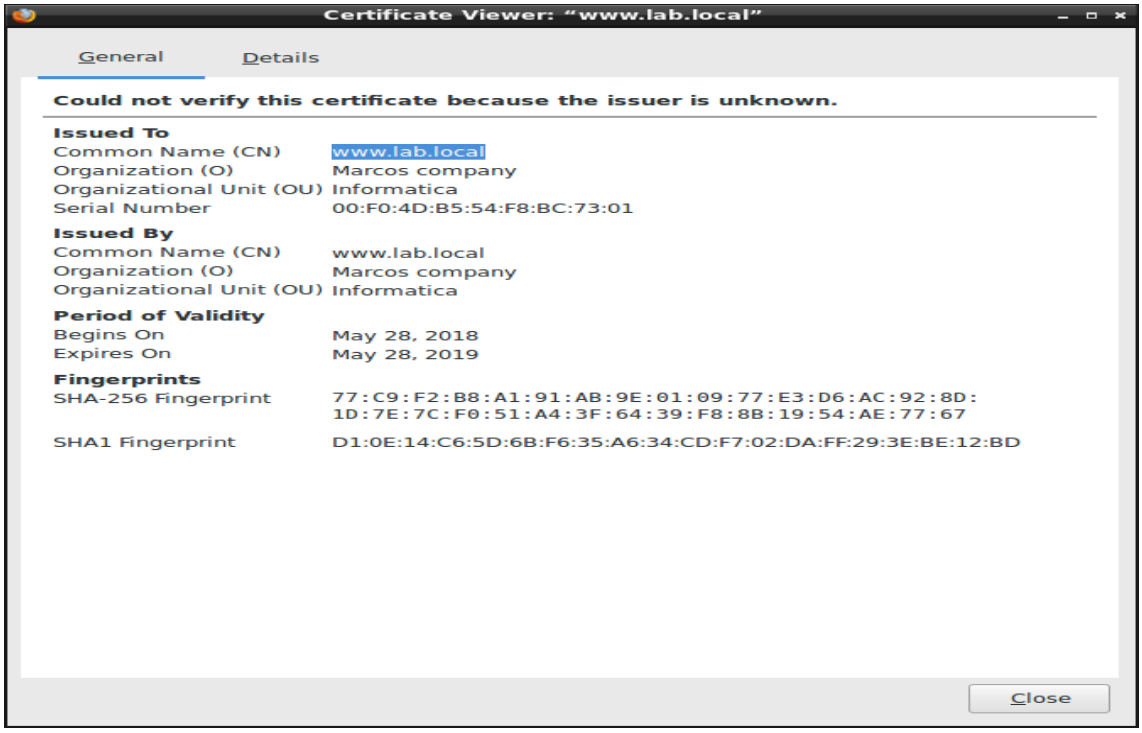


Ilustración 11 Datos del certificado

Cerramos el visor de certificados pulsando "close", y en la ventana "Add Security Exception", pulsamos el botón "Confirm Security Exception". Una vez hecho esto, ya podemos visualizar la página de inicio:

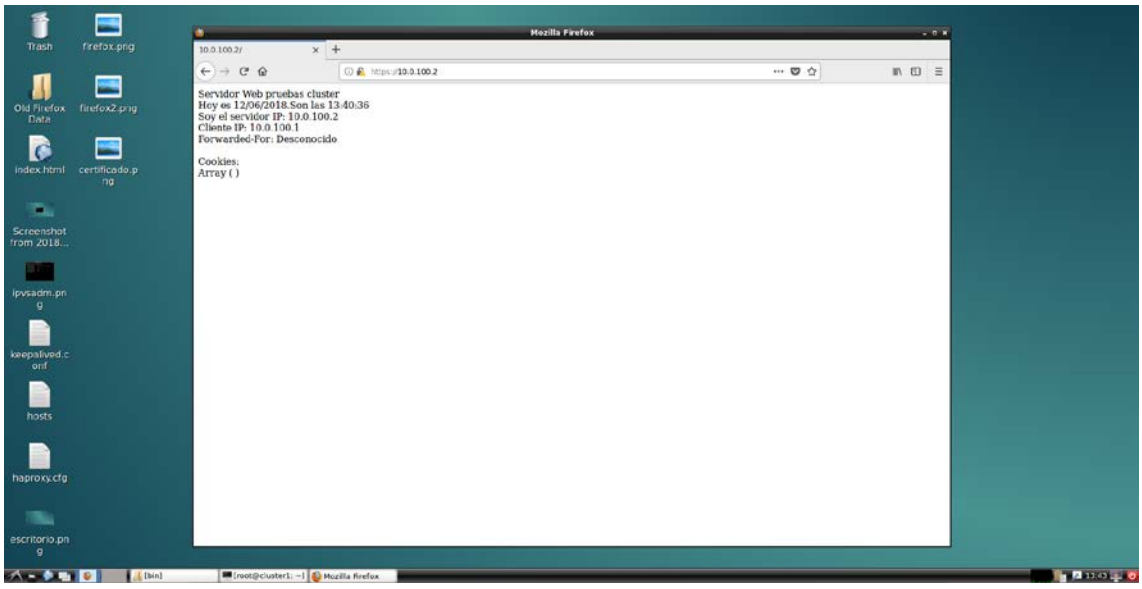


Ilustración 12 Página de inicio de nuestro servidor web

Para acabar, clonamos dos veces serweb1 obteniendo servweb2 y servweb3 y en las dos últimas configuramos la dirección MAC apropiada.

3.5. Configuración de los servidores DNS

Antes de empezar con la configuración de bind9 propiamente dicha, a fin de que este apartado sea lo más autocontenido posible, vamos explicar algunos conceptos básicos sobre el DNS, que ayudarán a entender mejor la configuración de estos servidores.

En particular, nos centraremos en explicar en que consiste la resolución iterativa y la resolución recursiva de nombres.

Por defecto bind no admite peticiones recursivas, si las necesitamos tenemos que indicarlo en uno de sus ficheros de configuración como veremos más adelante.

3.5.1. Resolución iterativa

El servidor DNS local devuelve la mejor respuesta que puede ofrecer al cliente en función del contenido de su caché. Si el servidor no dispone de la información solicitada, indicará la IP del siguiente servidor de nombres autorizado, comenzando siempre por un servidor raíz. Éste lo enviará al servidor del nivel siguiente que lo contenga y el servidor local volverá a lanzar la petición (iteración) al servidor referido. En el caso de que éste no disponga de la información solicitada, pasará la consulta al servidor del nivel siguiente que lo contenga y el servidor DNS local lanzará de nuevo la petición. El proceso se repetirá una y otra vez hasta que se llegue al servidor de nombres que contenga la información acerca del dominio solicitado.

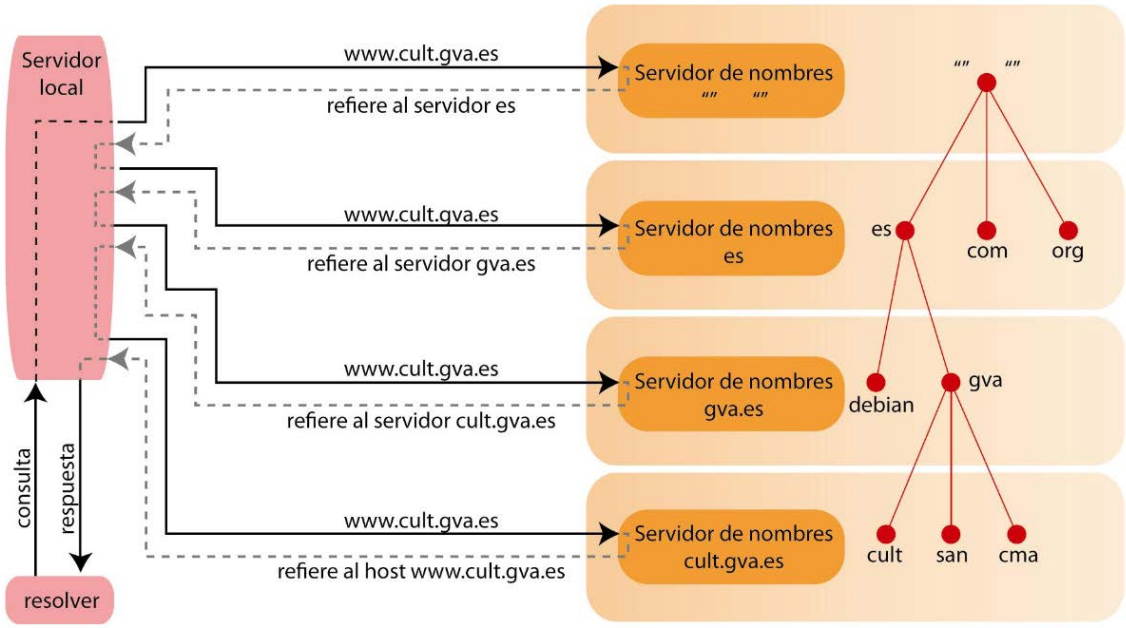


Ilustración 13 Petición iterativa

3.5.2 Resolución recursiva

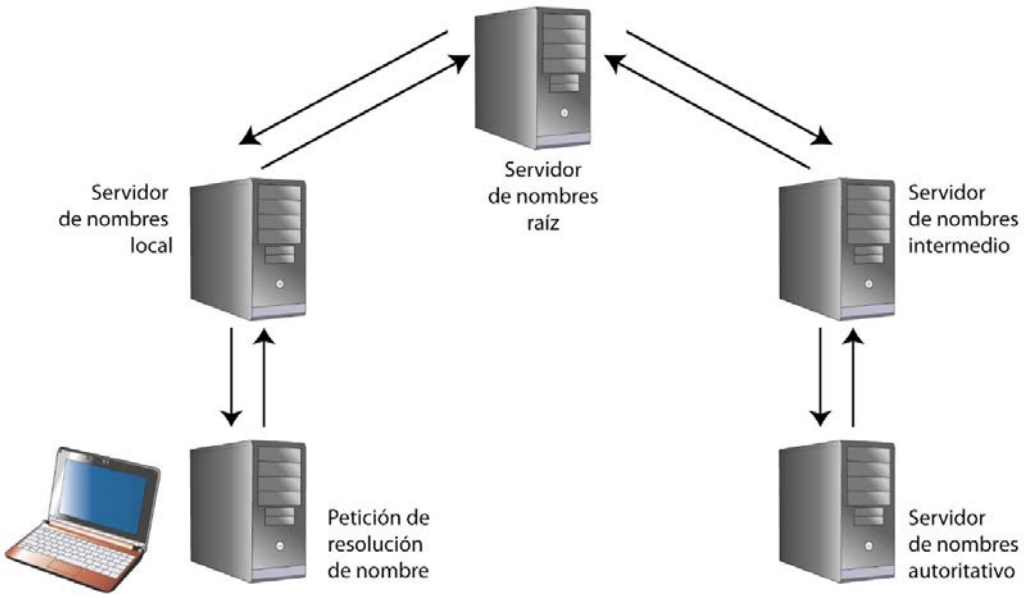


Ilustración 14 Petición recursiva

Cuando se solicita una resolución de nombre al servidor DNS local, es posible que éste no disponga de la información precisa. En ese caso, la buscará en el servidor autoritativo que la contenga.

Para ello, el servidor de nombres local enviará la consulta al servidor raíz, que le proporcionará información sobre los servidores de nombres autoritarios intermedios hasta que llegue al servidor que contiene el nombre del dominio en cuestión. En este caso, el servidor local se encargará de dar una respuesta al cliente y consultará a los demás servidores en su nombre.

3.5.3. Configuración de bind

Partimos de una máquina virtual de nombre “dns1” que hemos clonado a partir de la instantánea “MODELO”, asegurándonos de modificar su configuración de red para respetar las directrices de direccionamiento MAC que hemos establecido y arrancamos la máquina.

Instalamos el paquete bind9:

```
apt-get install bind9
```

Ahora, debemos modificar los siguientes ficheros:

/etc/bind/named.conf.options:

```
…
        forwarders {
                8.8.8.8;
                8.8.4.4;
        };


…
         listen-on {any;};
         allow-recursion {any;};
         allow-recursion-on{any;};
…
```

Como podemos observar, en la cláusula forwarders definimos a que servidores DNS externos redirigir aquellas peticiones que nuestro servidor no sepa resolver (usando en este caso los DNS de Google) y habilitamos las peticiones recursivas al mismo.

/etc/bind/named.conf.local:

```
...
zone "lab.local" {
  type master;
        allow-query {any;};
        file "/etc/bind/rd.lab.local";
};


zone "100.0.10.in-addr.arpa" {
  type master;
        allow-query {any;};
  file "/etc/bind/ri.lab.local";
};
```

Con esto, ya hemos definidos las zonas, vamos pues a editar los ficheros rd.lab.local y ri.lab.local

/etc/bind/rd.lab.local:

```
$TTL 38400

@ IN SOA dns.lab.local. admin.lab.local. (
       2014092900; //num serie
       28800; // refresco
       3600; // reintentos
       604800; // caducidad
       38400); // tiempo de vida en cache

                  IN NS dns1.lab.local.
                  IN NS dns2.lab.local.
                  IN NS dns3.lab.local.
dns1              IN A 10.0.100.5
dns2              IN A 10.0.100.6
dns3              IN A 10.0.100.7
master            IN A 10.0.100.1
standby           IN A 10.0.100.50
nas               IN A 10.0.100.100
web1              IN A 10.0.100.2
web2              IN A 10.0.100.3
web3              IN A 10.0.100.4
lb1               IN CNAME master
lb2               IN CNAME standby
cluster1          IN CNAME master
cluster2          IN CNAME web1
cluster3          IN CNAME web2
cluster4          IN CNAME web3
cluster5          IN CNAME dns1
cluster6          IN CNAME dns2
cluster7          IN CNAME dns3
```

Como podemos observar en este fichero indicamos el nombre de nuestro dominio, en este caso “lab.local” junto con varios parámetros del servidor DNS como el tiempo que deben mantenerse en caché las peticiones anteriormente resueltas, etc… En todos los casos todos los valores asociados con tiempos están expresados en segundos.

Luego, viene la tabla de consultas, en ella indicamos el nombre de máquina, el tipo de consulta que queremos realizar, y en su caso, la dirección IP de la máquina correspondiente. Utilizamos los tipos de registros siguientes:

- ✓ Registro A: En este caso dado un nombre de máquina, por ejemplo, “web1”, nos devuelve la dirección IP de esa máquina, que en este caso es la 10.0.100.2

- ✓ Registro NS: Devuelve el servidor DNS de nuestro dominio

- ✓ Registro CNAME: Permite definir alias o seudónimos, de manera que una máquina pueda tener varios nombres. En nuestro caso, por ejemplo, es posible dirigirse a la máquina cuya dirección IP es 10.0.100.2, bien por el nombre web1 que es su verdadero nombre o bien por el nombre cluster2 que es un alias

/etc/bind/ri.lab.local:

```
@ IN SOA dns.lab.local. admin.lab.local. (
       2014092900;
       28800;
       3600;
       604800;
       38400);
@   IN NS dns1.
@       IN NS dns2.
@       IN NS dns3.
100       IN PTR nas.
1 IN PTR master
2     IN PTR web1.
3     IN PTR web2.
4     IN PTR web3.
5     IN PTR dns1.
6     IN PTR dns2.
7     IN PTR dns3.
50    IN PTR standby
```

En este fichero la sección SOA tiene los mismos parámetros y con el mismo significado que en el fichero rd.lab.local, La @ significa “en este dominio”, que en nuestro caso es lab.local, indicando además que el dominio tiene 3 servidores DNS. Obsérvese que para realizar la resolución inversa de direcciones IP, basta con especificar el primer octeto empezando por la derecha de la dirección correspondiente, y esto es debido a que en el fichero de zonas hemos definido el prefijo "100.0.10.in-addr.arpa", que como vemos, contiene los 3 bytes más significativos y que son comunes a todas las direcciones IP.

3.5.3.1. Comprobación de funcionamiento

Vamos a realizar algunas pruebas para asegurarnos de que hemos realizado correctamente la configuración del servidor.

Ya hemos modificado y editado todos los ficheros necesarios, pero antes de reiniciar bind vamos a comprobar que no hemos cometido ningún error sintáctico al editar los ficheros. Para ello, utilizamos las siguientes órdenes:

Comprobar sintaxis named.conf.local y named.conf.options:

```
/etc/bind# named-checkconf
```

En caso de haber errores nos indicaría el nombre del fichero y el número de la línea que contiene el error, si no hay errores, no nos devuelve ninguna salida.

Comprobar sintaxis rd.lab.local y ri.lab.local:

```
/etc/bind# named-checkzone lab.local /etc/bind/rd.lab.local

/etc/bind# named-checkzone lab.local /etc/bind/ri.lab.local
```

Si todo va bien nos devolverá OK y el número de serie de nuestro servidor DNS, si hay errores se comporta igual que la orden named-checkconf.

Reiniciamos bind:

```
service bind9 restart
```

A continuación, utilizaremos los comandos dig y nslookup para asegurarnos de que el servidor resuelve nombres.

Comprobamos que resuelve nuestro propio dominio:

```
dig @127.0.0.1 lab.local
; <<>> DiG 9.10.3-P4-Ubuntu <<>> @127.0.0.1 web1.lab.local
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 24359
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL:
4
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;web1.lab.local.                    IN      A
;; ANSWER SECTION:
web1.lab.local.            38400   IN      A       10.0.100.2
;; AUTHORITY SECTION:
lab.local.         38400   IN      NS      dns3.lab.local.
lab.local.         38400   IN      NS      dns1.lab.local.
lab.local.         38400   IN      NS      dns2.lab.local.
;; ADDITIONAL SECTION:
dns1.lab.local.            38400   IN      A       10.0.100.5
dns2.lab.local.            38400   IN      A       10.0.100.6
dns3.lab.local.            38400   IN      A       10.0.100.7
;; Query time: 0 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Thu Jun 14 09:31:26 CEST 2018
;; MSG SIZE  rcvd: 164
```

Nótese que indicamos la dirección IP de bucle interno para forzar a que la consulta sea resuelta por nuestro servidor DNS local, ya que, de no hacerlo así, dado que los servidores web y DNS de la red interna reciben su configuración TCP / IP a través del servidor DHCP que hemos instalado en el nodo master, le estaríamos enviando la consulta a este último.

Probamos con una resolución local inversa:

```
nslookup 10.0.100.2 127.0.0.1
Server:          127.0.0.1
Address:         127.0.0.1#53
2.100.0.10.in-addr.arpa name = web1.
```

Ahora una directa:

```
dig @127.0.0.1 web1.lab.local

; <<>> DiG 9.10.3-P4-Ubuntu <<>> @127.0.0.1 web1.lab.local
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 24359
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL:
4

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;web1.lab.local.                        IN      A

;; ANSWER SECTION:
web1.lab.local.         38400   IN      A       10.0.100.2

;; AUTHORITY SECTION:
lab.local.              38400   IN      NS      dns3.lab.local.
lab.local.              38400   IN      NS      dns1.lab.local.
lab.local.              38400   IN      NS      dns2.lab.local.

;; ADDITIONAL SECTION:
dns1.lab.local.         38400   IN      A       10.0.100.5
dns2.lab.local.         38400   IN      A       10.0.100.6
dns3.lab.local.         38400   IN      A       10.0.100.7

;; Query time: 0 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Thu Jun 14 09:31:26 CEST 2018
;; MSG SIZE  rcvd: 164
```

Por último, comprobamos que nos resuelve dominios de Internet:

```
nslookup www.upv.es 127.0.0.1

Server:        127.0.0.1

Address:       127.0.0.1#53

Non-authoritative answer:
www.upv.es     canonical name = ias.cc.upv.es.
Name:   ias.cc.upv.es
Address: 158.42.4.23
```

Ya hemos realizado todas las comprobaciones necesarias, apagamos la máquina virtual dns1 y obtenemos dos clones de la misma a las que bautizamos como dns2 y dns3, sin olvidarnos de ajustar la dirección MAC de las dos nuevas máquinas virtuales según el convenio que hemos establecido.

3.6. Ajustes adicionales en el nodo master

Vamos a habilitar el acceso vía SSH desde el nodo master a los servidores web y DNS de la red interna. Recordemos que en este momento sólo nos podemos conectar al servidor NFS, aquí utilizaremos otro método para la difusión de claves, concretamente ssh-keyscan:

```
ssh-keyscan cluster2 >> /root/.ssh/known_hosts
ssh-keyscan cluster3 >> /root/.ssh/known_hosts
ssh-keyscan cluster4 >> /root/.ssh/known_hosts
ssh-keyscan web1 >>     /root/.ssh/known_hosts
ssh-keyscan web2 >>     /root/.ssh/known_hosts
ssh-keyscan web3 >>     /root/.ssh/known_hosts
ssh-keyscan cluster5 >> /root/.ssh/known_hosts
ssh-keyscan cluster6 >> /root/.ssh/known_hosts
ssh-keyscan cluster7 >> /root/.ssh/known_hosts
ssh-keyscan dns1 >>     /root/.ssh/known_hosts
ssh-keyscan dns2 >>     /root/.ssh/known_hosts
ssh-keyscan dns3 >>     /root/.ssh/known_hosts
```

Como se puede observar, hemos usado la orden ssh-keyscan una vez por cada alias que le hemos dado a cada máquina en el fichero /etc/hosts del nodo master, lo que nos permitirá hacer ssh con el nombre “canónico” de las máquinas o con sus alias indistintamente.

Ahora, vamos a configurar los nombres de host para los servidores web y DNS de la red interna:

```
ssh web1 “echo web1 > /etc/hostname”
ssh web2 “echo web2 > /etc/hostname“
ssh web3 “echo web3 > /etc/hostname”
ssh dns1 “echo dns1 > /etc/hostname”
ssh dns2 “echo dns2 > /etc/hostname“
ssh dns3 “echo dns3 > /etc/hostname”
```

Por último, sólo nos queda difundir el fichero hosts del nodo master al resto de máquinas del cluster:

```
scp   /etc/hosts  nas:/etc
scp   /etc/hosts  web1:/etc
scp   /etc/hosts  web2:/etc
scp   /etc/hosts  web3:/etc
scp   /etc/hosts  dns1:/etc
scp   /etc/hosts  dns2:/etc
scp   /etc/hosts  dns3:/etc
```

Con esto el nodo master queda configurado, ahora ya podemos instalar los paquetes necesarios para ofrecer equilibrado de carga y alta disponibilidad.

3.7. Fundamentos de HAProxy

HAProxy, que significa proxy de alta disponibilidad, es una solución de software libre que se puede ejecutar en Linux, Solaris y FreeBSD. Su uso más común es mejorar el rendimiento y la fiabilidad en un entorno cliente / servidor mediante la distribución de la carga de trabajo entre varios servidores (por ejemplo, sitios web, aplicaciones de base de datos, etc.). Se utiliza en muchas plataformas con un elevado volumen de clientes como, por ejemplo: GitHub, Imgur, Instagram y Twitter entre otras.

Se trata de un repartidor de carga que puede trabajar tanto en capa de transporte como en capa de aplicación, pero, con la limitación de que sólo puede gestionar tráfico TCP.

Antes de entrar en los detalles de la configuración que hemos realizado, es importante entender cuáles son las implicaciones que tiene el realizar un reparto de carga en capa de aplicación o en capa de transporte.

La manera más sencilla de repartir la carga consiste en realizar un reparto a nivel de capa de transporte. En este caso, el tráfico procedente de los clientes del cluster se reparte en función de la dirección IP y el número de puerto de destino. En este caso, como sólo se dispone de información de la capa de transporte, el equilibrado de carga se realiza en base a algún algoritmo, como, por ejemplo, round-robin y calculando el mejor servidor de destino en función de las conexiones mínimas o los tiempos de respuesta de los servidores disponibles.

Cuando trabaja en la capa de aplicación, el repartidor de carga tiene conocimiento de la aplicación que se está repartiendo y puede utilizar esta información adicional de la aplicación para tomar decisiones más complejas e informadas sobre el equilibrado de carga. Con un protocolo como HTTP, un equilibrador de carga puede identificar de manera única sesiones de cliente basadas en cookies y usar esta información para entregar todas las solicitudes de los clientes al mismo servidor. Incluso es posible que, si una cookie deja de ser válida, las peticiones del cliente sean reasignadas de forma totalmente transparente a otro servidor. Otra ventaja del reparto de carga en la capa de aplicación es que podemos conocer la identidad del cliente que realiza las peticiones.

3.7.1. Configuración de HAProxy

Instalamos en el nodo master el paquete de HAProxy:

```
apt-get install haproxy
```

Configuramos HAProxy para que sea iniciado por el script init. Para ello, es necesario editar el fichero HAProxy para establecer la opción ENABLED a 1:

/etc/default/haproxy:

```
ENABLED=1
```

A continuación, vamos a editar el fichero de configuración de HAProxy, que debe quedar tal y como se muestra:

/etc/haproxy/haproxy.cfg:

```
global
      log /dev/log local0
```

```
      log /dev/log local1 notice
      chroot /var/lib/haproxy
      user haproxy
      group haproxy
      daemon

defaults
      log  global
      option     dontlognull
      timeout connect 5000
      timeout client 3000
      timeout server 3000
      errorfile 400 /etc/haproxy/errors/400.http
      errorfile 403 /etc/haproxy/errors/403.http
      errorfile 408 /etc/haproxy/errors/408.http
      errorfile 500 /etc/haproxy/errors/500.http
      errorfile 502 /etc/haproxy/errors/502.http
      errorfile 503 /etc/haproxy/errors/503.http
      errorfile 504 /etc/haproxy/errors/504.http

frontend haproxy-http
       mode http
       option httplog
       option forwardfor
       bind 192.168.1.200:8000
       default_backend http-servers

frontend haproxy-tcp
       mode tcp
       option tcplog
       bind 192.168.1.200:80
       default_backend tcp-servers

frontend ft_https
      bind 192.168.1.200:443
      mode tcp
      default_backend bk_https

backend http-servers
      mode http
      balance roundrobin
      option redispatch
      cookie SERVERID insert
      option httpchk
      server cluster2 cluster2:80 cookie cluster2 check
```

```
      server cluster3 cluster3:80 cookie cluster3 check
      server cluster4 cluster4:80 cookie cluster4 check

backend tcp-servers
      mode tcp
      balance roundrobin
      server cluster2 cluster2:80 check
      server cluster3 cluster3:80 check
      server cluster4 cluster4:80 check

backend bk_https
      mode tcp
      balance roundrobin
      server cluster2 cluster2:443 check
      server cluster3 cluster3:443 check
      server cluster4 cluster4:443 check

listen stats
      bind 0.0.0.0:7000
      mode http
      stats show-desc Nodo Master
      stats refresh 5s
      stats uri /haproxy?stats
      stats realm HAProxy Statistics
      stats auth admin:123
      stats admin if TRUE
```

La configuración que hemos realizado, nos permite repartir un servidor web, tanto http como https (secciones **frontend** del fichero de configuración). En el caso de las peticiones http, si nos conectamos al puerto 8000 se realiza un reparto de carga de capa de aplicación, mientras que si nos conectamos al puerto 80 y 443, se realiza un reparto de capa de transporte.

A continuación, pasamos a explicar los principales componentes del fichero de configuración y las opciones utilizadas.

Un **backend** puede contener uno o más servidores: en términos generales, añadir más servidores a la sección backend aumentará la capacidad de carga potencial que puede gestionar el cluster mediante la distribución de la carga entre los servidores disponibles. También conseguiremos aumentar la disponibilidad del servicio en caso de que algún servidor deje de funcionar. Resumiendo, en la sección backend indicamos entre qué servidores reales se reparte la carga.

Los principales parámetros que hemos utilizado son:

- ✓ La cláusula `mode` especifica el tipo de reparto de carga que se va a realizar, `http` para capa de aplicación o `tcp` para capa de transporte.

- ✓ La cláusula `option redispatch` sólo se puede utilizar en modo http y hace que, si un servidor designado por una cookie no responde, el cliente pueda ser atendido por otro servidor de entre los disponibles.

- ✓ La cláusula `balance` especifica el algoritmo que se utilizará para llevar a cabo el reparto de carga. Su valor por defecto es `roundrobin` aunque admite otros algoritmos. En este caso, se realiza una asignación cíclica por turnos.

- ✓ La cláusula `check` permite comprobar si un servidor está escuchando tráfico por el puerto indicado. En caso de utilizarse en modo http, comprueba la validez de la cookie correspondiente.

- ✓ La cláusula `fordwardfor` permite conocer la identidad del cliente cuando el repartidor trabaja en la capa de aplicación

Un **frontend** define la manera en que las solicitudes se deben reenviar a los servidores de la sección backend correspondiente. Al igual que ocurre con los backend, también es posible tener varias secciones frontend. Las IP especificadas aquí pueden ser reales o virtuales (véase la sección sobre keepalived más adelante).

Otro aspecto importante de la configuración realizada reside en la gestión de las conexiones https. En concreto hay dos posibilidades:

- ✓ Que el certificado resida en el nodo repartidor y que por tanto éste se ocupe del cifrado, lo que redunda en una mayor carga computacional para el repartidor de carga.

- ✓ Que el certificado resida en los servidores web, con lo que el trabajo del repartidor consiste simplemente en redirigir las peticiones, dejando el cifrado para los servidores web.

Nosotros hemos optado por la segunda opción, en aras a descargar de trabajo al nodo repartidor.

Por último, comentar que si hacemos una petición http al puerto 7000, se carga la página de estadísticas de HAProxy que nos solicita un usuario y una contraseña para acceder a la misma y que en nuestro caso es: admin / 123 respectivamente.

La configuración de la página de estadísticas se realiza en la sección `listen stats`, indicando que el servidor de estadísticas escucha en el puerto 7000 de la máquina local.

3.8. Fundamentos de IPVS

IP virtual server es un repartidor de carga de capa de transporte incluido en el núcleo de Linux que trabaja sobre netfilter. Instalaremos el paquete keepalived, que, además de permitir la configuración de IPVS, ofrece soporte para alta disponibilidad mediante el protocolo VRRP (Virtual Router Redudancy protocol). Este protocolo permite definir una dirección IP que actuará como alias del servidor real, dicha “IP alias” es flotante, lo que quiere decir que, siempre está asignada al nodo que actualmente está actuando como repartidor de carga y dado que es posible definir grupos de nodos repartidores asignándoles prioridades, podemos tener varios repartidores configurados simultáneamente y hacer que sólo haya uno activo (el más prioritario) y los otros actúen como su respaldo para admitir tolerancia a fallos. Dado que como ya se comentó, HAProxy no gestiona tráfico UDP, utilizaremos IPVS para repartir tráfico DNS y para que actúe como router virtual de nuestro cluster.

3.8.1. Configuración de keepalived

Instalamos el paquete necesario:

```
apt-get install keepalived
```

Creamos el archivo de configuración /etc/keepalived/keepalived.conf:

```
vrrp_sync_group VG1 {
        group {
                VI_E
                VI_I
               }
         }

vrrp_instance VI_E {
        state MASTER
        interface eth0
        virtual_router_id 51
        priority 150
        authentication {
                auth_type PASS
                auth_pass mypasswd
         }
        virtual_ipaddress {
                192.168.1.200
                192.168.1.202

         }
 }
```

```
vrrp_instance VI_I {
        state MASTER
        interface eth1
        virtual_router_id 52
        priority 150
        authentication {
                auth_type PASS
                auth_pass mypasswd
         }
        virtual_ipaddress {
                10.0.100.200
         }
 }

virtual_server 192.168.1.202 80 {
        delay_loop 3
        lb_algo rr
        lb_kind NAT
        protocol TCP

        real_server 10.0.100.2 80 {
        weight 1
        TCP_CHECK {
                connect_timeout 3
                 }
         }
        real_server 10.0.100.3 80 {
        weight 1
        TCP_CHECK {
                connect_timeout 3
                 }
         }
        real_server 10.0.100.4 80 {
        weight 1
        TCP_CHECK {
                connect_timeout 3
                 }
         }
 }


virtual_server 192.168.1.200 53 {
        delay_loop 3
        lb_algo rr
        lb_kind NAT
        protocol UDP

        real_server 10.0.100.5 53 {
        weight 1
        TCP_CHECK {
                connect_timeout 3
                 }
```

```
        }
       real_server 10.0.100.6 53 {
       weight 1
       TCP_CHECK {
               connect_timeout 3
                }
        }
       real_server 10.0.100.7 53 {
       weight 1
       TCP_CHECK {
               connect_timeout 3
                }
        }
}
```

A continuación, vamos a explicar el significado de los distintos elementos que aparecen en el fichero de configuración. Para ofrecer alta disponibilidad, nuestro cluster debe tener dos direcciones VIP configuradas:

- ✓ La IP virtual en la red externa (Red-NAT). Como el servicio de distribución de carga puede migrar entre los nodos master (192.168.1.101) y standby (192.168.1.102), hay que definir una IP virtual (192.168.1.200) que será un alias de la máquina que, en cada momento, esté actuando como repartidor. Esta será la dirección "pública" de nuestro servidor y a la que accederán los clientes.

- ✓ La IP virtual en la red interna. Como el servicio de distribución de carga puede migrar entre lo mismo nodos master (10.0.100.1) y standby (10.0.100.50), hay que definir una IP virtual (por ejemplo: 10.0.100.200) que será un alias en la red interna de la máquina que, en cada momento, esté dando servicio.

Esto es necesario para el caso de que falle uno de los distribuidores de carga. En tal caso, el distribuidor superviviente debe hacer de router de los servidores reales (cluster2, cluster3, etc.). Por tanto, será necesario también configurar estos servidores reales para que su router sea 10.0.100.200, como veremos posteriormente.

Hemos definido dos interfaces de red virtuales, correspondientes al interfaz externo (VI_E) e interno (VI_I) de los nodos repartidores de carga. Además, están agrupados, de manera que un fallo en cualquiera de ellos provocará la migración de las dos direcciones VIP al repartidor superviviente.

Ahora, analizaremos el objetivo de las diferentes cláusulas y secciones utilizadas en el fichero de configuración de keepalived:

- ✓ `state MASTER:` Indica que, en caso de tener varios nodos repartidores de carga, este nodo es el repartidor maestro o principal. Es decir, en nuestro cluster tenemos los repartidores Master y Standby, pero el nodo Standby sólo adquirirá el rol de repartidor en caso de que el nodo master deje de funcionar. En el caso del nodo standby, esta cláusula llevará el valor

`BACKUP` para indicar que se trata de un nodo de respaldo para el nodo master.

- ✓ Con la cláusula `interface` especificamos qué tarjeta de red atiende las peticiones. Como podemos ver, las peticiones procedentes de los clientes son atendidas por la tarjeta de red primaria, `eth0`, mientras que todo el tráfico generado por los servidores de la red interna es gestionado por la tarjeta de red secundaria, `eth1`.

- ✓ Con la cláusula `virtual_router_id`, definimos una interfaz virtual para el router virtual del cluster, puede tener cualquier valor siempre y cuando no definamos dos interfaces con el mismo número.

- ✓ La cláusula `priority` indica el nivel de prioridad del repartidor, de manera que si tenemos varios repartidores del mismo tipo (varios master) el tráfico será repartido por el nodo repartidor de mayor prioridad.

- ✓ La sección de autenticación es necesaria para que los servidores de keepalived confíen entre sí.

- ✓ La sección `virtual_ipaddress` contiene las vips a los que se pueden conectar los clientes. Nótese que en el caso de la IP virtual externa, hemos definido dos IP (192.168.1.200, 192.168.1.202), hemos hecho esto para demostrar que nada nos impide definir un pool de direcciones IP para acceder a un servidor, en nuestro ejemplo, los clientes pueden acceder al servidor web del cluster mediante cualquiera de las dos direcciones IP, mientras que al servidor DNS sólo pueden acceder dese la IP 192.168.1.200.

La sección virtual server define los parámetros de reparto de carga de IPVS y qué servidores de la red interna ofrecen realmente cada servicio. En concreto:

- ✓ La cláusula `delay_loop` indica cada cuántos segundos se comprobará la disponibilidad de los servidores de la red interna.

- ✓ La cláusula `lb_algo` indica que algoritmo se utilizará para llevar a cabo el reparto de carga. En nuestro caso, el parámetro `rr` indica que utilizaremos el algoritmo round robin.

- ✓ La cláusula `lb_kind` indica el modo de trabajo del repartidor de carga. En nuestro caso, usamos el modo `NAT` lo que quiere decir que cuando llega una petición desde el exterior del cluster, el repartidor reenvía la petición al servidor de la red interna adecuado, según el algoritmo de planificación que hayamos establecido previamente.

- ✓ La cláusula protocol indica cuál es el protocolo de capa de transporte de cada servidor virtual. En nuestro caso, el protocolo http trabaja sobre TCP, mientras que el DNS lo hace sobre UDP.

En la sección real server tenemos:

- ✓ La cláusula `weight` mediante la indicamos la importancia o “peso” que cada servidor tiene de cara a ser seleccionado para atender una petición por parte del algoritmo de planificación. En nuestro caso, le hemos dado el mismo peso a todos los servidores debido a que utilizamos el algoritmo de planificación round robin, y, por lo tanto, buscamos la equidad para que el reparto de carga se realice de manera uniforme.

- ✓ En la sección TCP_CHECK establecemos un tiempo de espera máximo en caso de que un servidor de la red interna no responda, lo que hará que se asigne la petición a otro servidor de entre los disponibles.

Arrancamos keepalived:

```
service keepalived restart
```

Comprobamos que el reparto de carga está funcionando:

```
ipvsadm -L

IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  192.168.1.202:http rr
  -> cluster2.cluster:http        Masq    1      0          0
  -> cluster3.cluster:http        Masq    1      0          0
  -> cluster4.cluster:http        Masq    1      0          0
UDP  192.168.1.200:domain rr
  -> cluster5.cluster:domain      Masq    1      0          0
  -> cluster6.cluster:domain      Masq    1      0          0
  -> cluster7.cluster:domain      Masq    1      0          0
```

Otra comprobación que podemos hacer consiste en lanzar la orden `ip addr` fijándonos en que todas las direcciones IP virtuales del cluster se encuentren activas:

```
…

2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
state UP group default qlen 1000
    link/ether 08:00:27:d7:13:ec brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.101/24 brd 192.168.1.255 scope global eth0
```

```
       valid_lft forever preferred_lft forever
    inet 192.168.1.200/32 scope global eth0
       valid_lft forever preferred_lft forever
    inet 192.168.1.201/32 scope global eth0
       valid_lft forever preferred_lft forever
    inet 192.168.1.202/32 scope global eth0
       valid_lft forever preferred_lft forever
…
```

Cuando las máquinas del cluster arranquen mediante DHCP, el servidor DNS debería estar asignado a la dirección VIP interna. Por otra parte, el router del sistema también debería estar asignado a la VIP interna. Recordemos que la dirección VIP interna estará activa sólo en uno de los nodos repartidores de carga. Esto lo conseguiremos cambiando la configuración del servidor DHCP, proporcionado por el paquete dnsmasq.

/etc/dnsmasq.conf:

```
…
dhcp-option=3,10.0.100.200
dhcp-option=6,10.0.100.200
…
```

En este punto ya tenemos el nodo master totalmente configurado, por lo que, apagamos esta máquina virtual.

3.9 Configuración del nodo Standby

Clonamos la máquina master sobre la máquina standby, asegurándonos de que en la clonación marcamos la opción de reiniciar la dirección MAC de todas las tarjetas de red.

Arrancamos la máquina standby que en este momento es un clon de la máquina master, lo que hace necesario ajustar su configuración. Para ello, en primer lugar, modificaremos la configuración de red, que debe quedar tal y como se muestra:

/etc/network/interfaces:

```
auto lo
iface lo inet loopback
auto eth0
iface eth0 inet static
address 192.168.1.102
netmask 255.255.255.0
gateway 192.168.1.1
```

```
dns-nameservers 192.168.1.1
mtu 9000
auto eth1
iface eth1 inet static
address 10.0.100.50
netmask 255.255.255.0
mtu 9000
```

Ahora, modificamos el nombre de la máquina:

```
echo standby > /etc/hostname
```

Hacemos un pequeño cambio en la configuración de HAProxy:

```
…
listen stats 0.0.0.0:7000
…
stats show-desc Nodo Standby
…
```

De este modo, podremos conocer en todo momento qué máquina está repartiendo el tráfico desde la página de estadísticas de HAProxy. Por último, modificamos la configuración de keepalived.

/etc/keepalived/keepalived.conf:

```
…
vrrp_instance VI_E {
state BACKUP
…}
vrrp_instance VI_I {
state BACKUP
…
priority 100
…}
```

Reiniciamos la máquina standby para que todos los cambios de configuración que hemos realizado surtan efecto, y con esto, ya tenemos el nodo standby totalmente configurado.

4. Evaluación del servidor

Realizada ya toda la configuración de nuestro cluster, en este punto vamos a realizar una comprobación de funcionamiento completa del cluster desde un cliente externo.

Para ello, realizaremos pruebas a dos niveles:

- ✓ Verificaremos que nuestro cluster ofrece servicio web y DNS.
- ✓ Evaluaremos las prestaciones del cluster implementado.

  Para ello, hemos preparado una nueva máquina virtual denominada client en la que hemos instalado una versión para escritorio de GNU / Linux Ubuntu 16.04 LTS con una tarjeta de red conectada a la red NAT. Las peticiones de los servicios las realizaremos con un usuario sin privilegios administrativos para emular el comportamiento de un cliente conectado desde Internet.

  En el cliente instalaremos apache benchmark para poder realizar posteriormente la evaluación de prestaciones del servidor web. Para ello, instalamos el paquete necesario:

```
apt-get install apache2-utils
```

4.1. Verificación de funcionamiento

Vamos a comprobar en primer lugar, el funcionamiento del servidor web para lo que utilizaremos el navegador Firefox:

Para comenzar, comprobamos que funciona el servicio web, para ello en la barra de direcciones de Firefox colocamos:

```
http://192.168.1.200
```

Obteniendo la siguiente salida:

```
Servidor Web pruebas cluster
Hoy es 06/07/2018.Son las 03:43:07
Soy el servidor IP: 10.0.100.3
Cliente IP: 10.0.100.1
Forwarded-For: Desconocido

Cookies:
Array ( )

Servidor Web pruebas cluster
Hoy es 06/07/2018.Son las 03:43:18
```

```
Soy el servidor IP: 10.0.100.4
Cliente IP: 10.0.100.1
Forwarded-For: Desconocido

Cookies:
Array ( )

Servidor Web pruebas cluster
Hoy es 06/07/2018.Son las 03:43:27
Soy el servidor IP: 10.0.100.2
Cliente IP: 10.0.100.1
Forwarded-For: Desconocido

Cookies:
Array ( )
```

Como observamos, cada vez que solicitamos la hora nos atiende un servidor distinto debido a que, como ya dijimos, en la configuración de HAProxy, estamos utilizando el algoritmo round robin para decir qué servidor de la red interna atiende una petición. Nótese también que, dado que en este caso el reparto de carga se hace en la capa de transporte, no podemos ver la IP del cliente que realmente ha lanzado la petición. Téngase en cuenta que, en el caso de lanzar las peticiones con https, los resultados son análogos a los que acabamos de comentar debido a que el reparto también se hace en la capa de transporte.

Veamos ahora si HAProxy funciona correctamente cuando hace el reparto en la capa de aplicación, para ello, basta con lanzar la petición al puerto 8000:

```
http://192.168.1.200:8000
```

Y obtenemos lo siguiente:

```
Servidor Web pruebas cluster
Hoy es 06/07/2018.Son las 03:52:59
Soy el servidor IP: 10.0.100.4
Cliente IP: 10.0.100.1
Forwarded-For: 192.168.1.10

Cookies:
Array ( )

Servidor Web pruebas cluster
Hoy es 06/07/2018.Son las 03:53:09
Soy el servidor IP: 10.0.100.4
Cliente IP: 10.0.100.1
Forwarded-For: 192.168.1.10

Cookies:
Array ( [SERVERID] => cluster4 )

Servidor Web pruebas cluster
```

```
Hoy es 06/07/2018.Son las 03:53:30
Soy el servidor IP: 10.0.100.4
Cliente IP: 10.0.100.1
Forwarded-For: 192.168.1.10

Cookies:
Array ( [SERVERID] => cluster4 )
```

Como observamos, siempre nos contesta el mismo servidor, “cluster4”, esto se debe a la utilización de las cookies. Nótese también que en la cláusula `Forwarded-For` podemos ver la IP del cliente que realmente ha lanzado la petición.

Ahora, vamos a probar el funcionamiento de IPVS, para ello, en la barra de direcciones del navegador colocamos:

```
http://192.168.1.202
```

Obteniendo la siguiente salida:

```
Servidor Web pruebas cluster
Hoy es 06/07/2018.Son las 05:57:04
Soy el servidor IP: 10.0.100.3
Cliente IP: 192.168.1.10
Forwarded-For: Desconocido

Cookies:
Array( )

Servidor Web pruebas cluster
Hoy es 06/07/2018.Son las 05:57:14
Soy el servidor IP: 10.0.100.2
Cliente IP: 192.168.1.10
Forwarded-For: Desconocido

Cookies:
Array( )

Servidor Web pruebas cluster
Hoy es 06/07/2018.Son las 05:58:24
Soy el servidor IP: 10.0.100.4
Cliente IP: 192.168.1.10
Forwarded-For: Desconocido

Cookies:
Array( )
```

Como podemos observar, aunque IPVS hace un reparto en capa de transporte, nos muestra la dirección IP del cliente mientras que por el hecho de utilizar round robin, cada vez nos contesta un servidor distinto.

Para comprobar el funcionamiento del servicio DNS hacemos lo siguiente:

```
dig @192.168.1.200 www.upv.es
```

Obteniendo la siguiente salida:

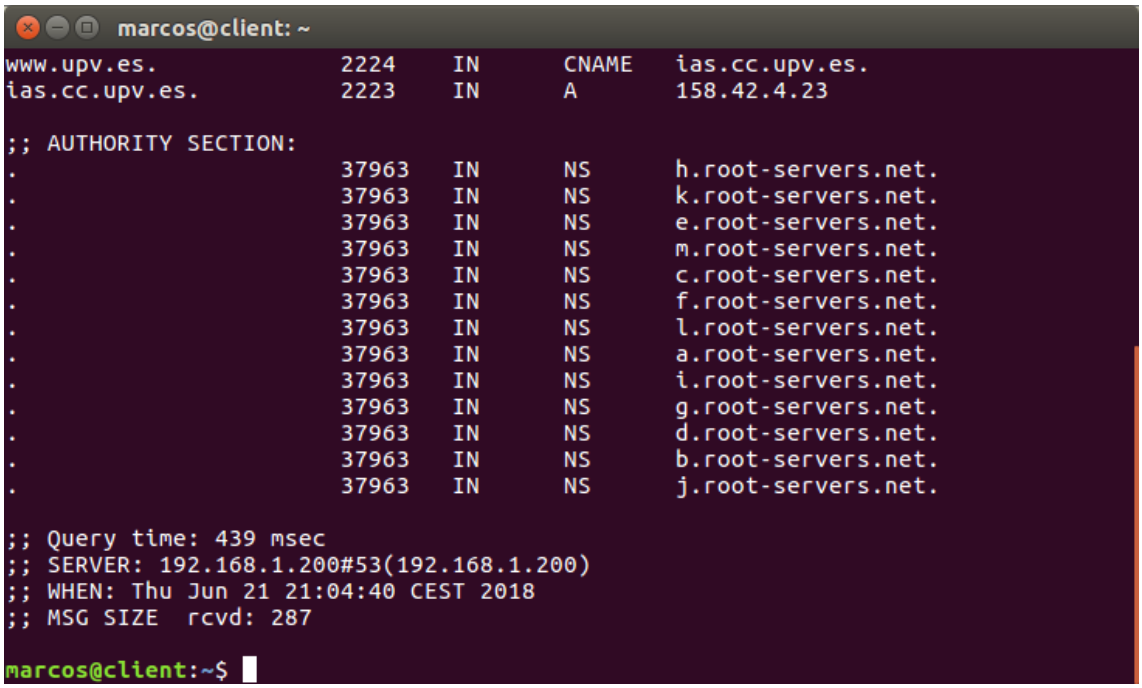


Ilustración 15 Consulta DNS

Podemos comprobar que los tres servidores DNS de la red interna están activos lanzando la siguiente orden desde el nodo maestro:

```
ipvsadm -L
```

Obteniendo la siguiente salida:

```
…
UDP  192.168.1.200:domain rr
  -> cluster5.cluster:domain      Masq    1      0          0
  -> cluster6.cluster:domain      Masq    1      0          0
  -> cluster7.cluster:domain      Masq    1      0          0
```

Apagamos el servidor dns1, volvemos a lanzar `ipvsadm -L` y obtenemos:

```
…
UDP 192.168.1.200:domain rr
  -> cluster6.cluster:domain      Masq    1      0          0
  -> cluster7.cluster:domain      Masq    1      0          0
```

Ahora lanzamos otra petición distinta:

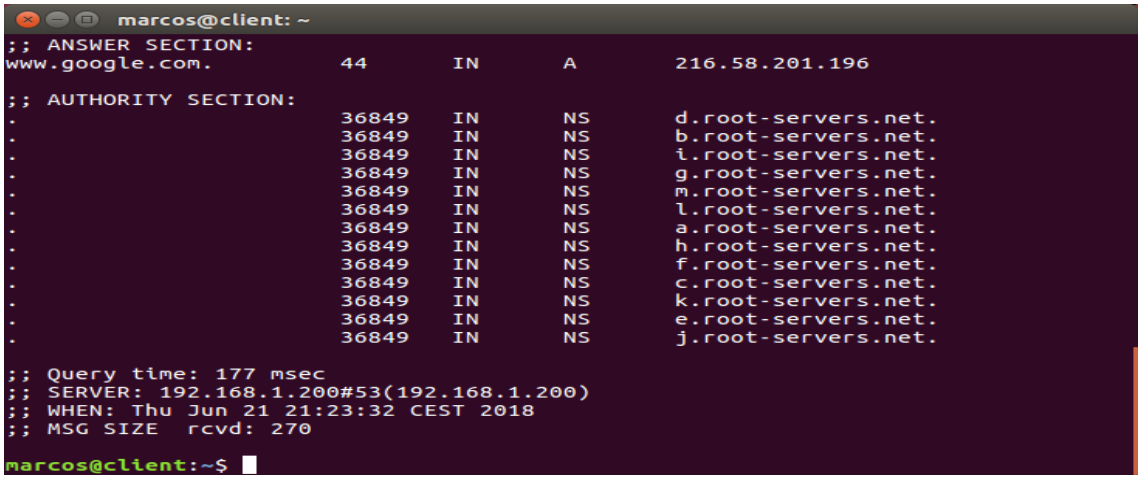


Ilustración 16 Otra consulta DNS con un servidor caído

Encendemos de nuevo el servidor dns1. Veamos ahora qué ocurre si provocamos un fallo en un servidor web, concretamente apagando el cluster4, tras lo que lanzamos nuevamente `ipvsadm -L`:

```
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  192.168.1.202:http rr
  -> cluster2.cluster:http        Masq    1      0          0
  -> cluster3.cluster:http        Masq    1      0          0
UDP  192.168.1.200:domain rr
  -> cluster5.cluster:domain      Masq    1      0          0
  -> cluster6.cluster:domain      Masq    1      0          0
  -> cluster7.cluster:domain      Masq    1      0          0
```

A continuación, repetiremos nuevamente la prueba del servidor http, prestando atención al hecho de que hemos apagado precisamente el cluster4 y por lo tanto, este no nos responde nunca:

```
Servidor Web pruebas cluster
Hoy es 06/07/2018.Son las 07:49:37
Soy el servidor IP: 10.0.100.2
Cliente IP: 10.0.100.1
Forwarded-For: Desconocido
```

```
Cookies:
Array ( )

Servidor Web pruebas cluster
Hoy es 06/07/2018.Son las 07:49:48
Soy el servidor IP: 10.0.100.3
Cliente IP: 10.0.100.1
Forwarded-For: Desconocido

Cookies:
Array ( )

Servidor Web pruebas cluster
Hoy es 06/07/2018.Son las 07:50:18
Soy el servidor IP: 10.0.100.2
Cliente IP: 10.0.100.1
Forwarded-For: Desconocido

Cookies:
Array ( )
```

Como se aprecia, el servicio web sigue funcionando y evidentemente, las peticiones se reparten entres los servidores web supervivientes. También podemos hacer esta comprobación visitando la página de estadísticas de HAProxy:

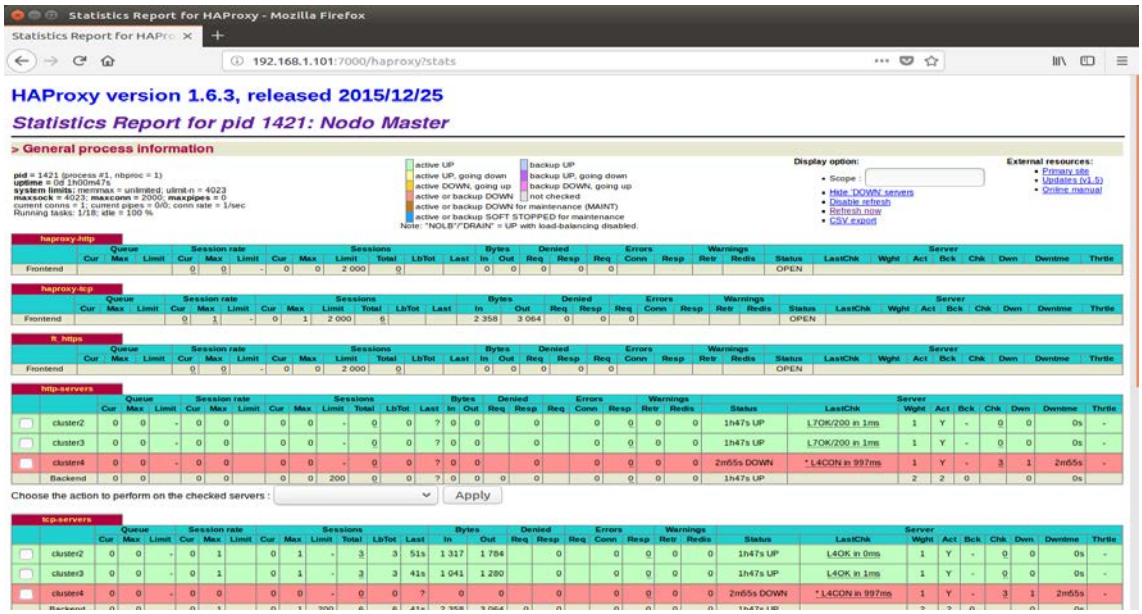


Ilustración 17 Página de estadísticas de HAProxy

Ahora vamos a comprobar qué ocurre en caso de que falle el nodo master, apagándolo, pero antes, podemos comprobar que en este momento el servidor standby no está repartiendo tráfico, para ello, basta con lanzar desde el nodo standby la orden `ip addr`:

```
...
```

```
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
state UP group default qlen 1000
    link/ether 08:00:27:3c:ec:b2 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.102/24 brd 192.168.1.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe3c:ecb2/64 scope link
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9000 qdisc pfifo_fast
state UP group default qlen 1000
    link/ether 08:00:27:72:79:67 brd ff:ff:ff:ff:ff:ff
    inet 10.0.100.50/24 brd 10.0.100.255 scope global eth1
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe72:7967/64 scope link
       valid_lft forever preferred_lft forever
```

Se observa que en este momento el nodo standby no tiene ninguna IP virtual activa. A continuación, apagamos el nodo master y desde el nodo standby volvemos a lanzar la orden `ip addr,` cuyo resultado es ahora el siguiente:

```
…

2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
state UP group default qlen 1000
    link/ether 08:00:27:3c:ec:b2 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.102/24 brd 192.168.1.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet 192.168.1.200/32 scope global eth0
       valid_lft forever preferred_lft forever
    inet 192.168.1.201/32 scope global eth0
       valid_lft forever preferred_lft forever
    inet 192.168.1.202/32 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe3c:ecb2/64 scope link
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9000 qdisc pfifo_fast
state UP group default qlen 1000
    link/ether 08:00:27:72:79:67 brd ff:ff:ff:ff:ff:ff
    inet 10.0.100.50/24 brd 10.0.100.255 scope global eth1
       valid_lft forever preferred_lft forever
    inet 10.0.100.200/32 scope global eth1
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe72:7967/64 scope link
       valid_lft forever preferred_lft forever
```

Como se observa, las IP virtuales han migrado al nodo standby. Por otro lado, si abrimos la página de estadísticas de HAProxy, veremos que el nodo standby ha adquirido el rol de repartidor:

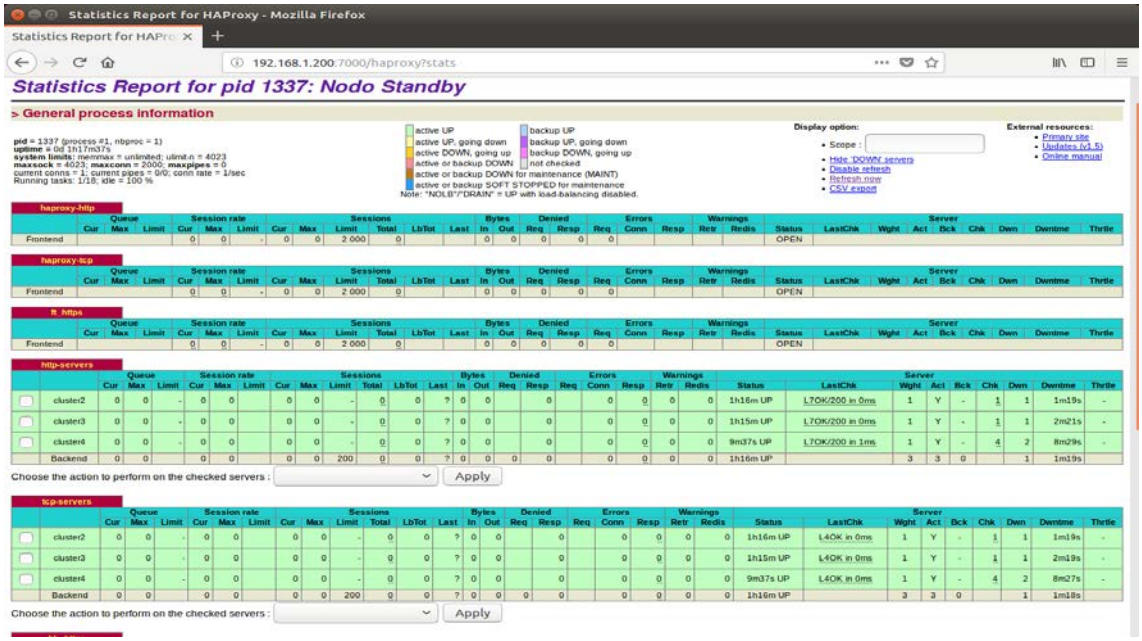


Ilustración 18 El nodo standby actuando como repartidor

Por último, vamos a repetir la prueba de funcionamiento del servidor web, esta vez en modo https obteniendo:

```
Servidor Web pruebas cluster
Hoy es 06/07/2018.Son las 09:46:07
Soy el servidor IP: 10.0.100.4
Cliente IP: 10.0.100.50
Forwarded-For: Desconocido

Cookies:
Array ( )

Servidor Web pruebas cluster
Hoy es 06/07/2018.Son las 09:40:11
Soy el servidor IP: 10.0.100.2
Cliente IP: 10.0.100.50
Forwarded-For: Desconocido

Cookies:
Array ( )

Servidor Web pruebas cluster
Hoy es 06/07/2018.Son las 09:40:22
Soy el servidor IP: 10.0.100.3
Cliente IP: 10.0.100.50
Forwarded-For: Desconocido

Cookies:
Array ( )
```

Como se observa, aparece como cliente la dirección IP del nodo standby.

4.2. Evaluación de prestaciones del servidor web

Para evaluar las prestaciones de nuestro servidor web, disponemos de varias herramientas de benchmarking, de entre ellas, hemos escogido apache benchmark por ser una de las más populares. Otras opciones son:

- Siege: Permite cargar varias páginas distintas con probabilidades configurables.

- Weighttp: Es multihilo, lo que evita que el cliente se convierta en el cuello de botella.

- G-wan: web server: Dispone de mucha documentación.

A continuación, se muestra una invocación de ejemplo de apache benchmark:

```
ab –n 1000 –c 3 http://192.168.1.200/index.php
```

Donde:

- ✓ -n indica el número de peticiones que se van a lanzar.
- ✓ -c indica el número de peticiones que se van a lanzar concurrentemente.

Para realizar la evaluación utilizaremos dos páginas webs dinámicas, una que realiza una tarea trivial como es devolver la fecha y la hora y que es la página que ya hemos utilizado con anterioridad para validar el funcionamiento del servidor web, y, otra página que calcula una aproximación de la constante PI mediante integración numérica por el método de los rectángulos. A continuación, mostramos el último de los scripts citados:

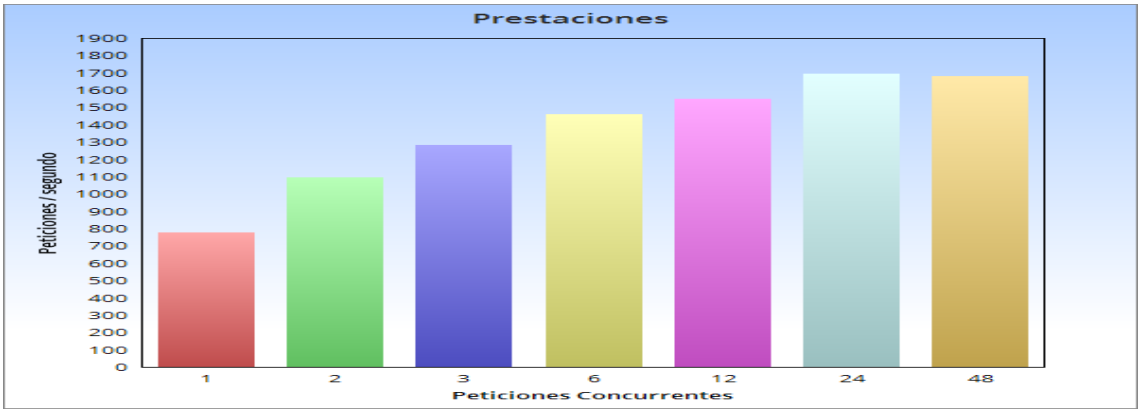
```
 <html>
<body>
<?php
   $start=microtime(true);
   $area=0.0;
   $n= $_GET["n"];
   for($i = 0; $i<$n; $i++) {
         $x=($i+0.5)/$n;
         $area=$area+ 4.0/ (1.0+$x*$x);
   }
   $result=$area/$n;
```

```
   $end=microtime(true);
   $exectime=$end-$start;
   echo "<br>Calculo de PI<br><br>";
   printf ("La cte. PI con n= %d es igual a %f<br>", $n, $result);
   printf ("Tiempo de ejecucion= %.5f segundos<br>",$exectime);
   printf ("<br>El servidor es %s<br>", $_SERVER['SERVER_ADDR']);
?>
</body>
</html>
```

Como se puede observar, lo que hacemos es medir cuánto tiempo tarda en ejecutarse el bucle, y lógicamente, a mayor valor de n, mejor aproximación obtendremos a costa eso sí, de alargar el tiempo de ejecución.

Por ejemplo, con un millón de iteraciones obtenemos el siguiente resultado:

Ahora, pasamos a analizar los resultados obtenidos con apache benchmark en el caso de la página sencilla, para lo cual, hemos ido duplicando el parámetro -c, empezando en c = 1, c = 2 y c= 3, a partir de ahí vamos doblando el valor de c finalizando en c = 48, tal y como muestra la siguiente gráfica:

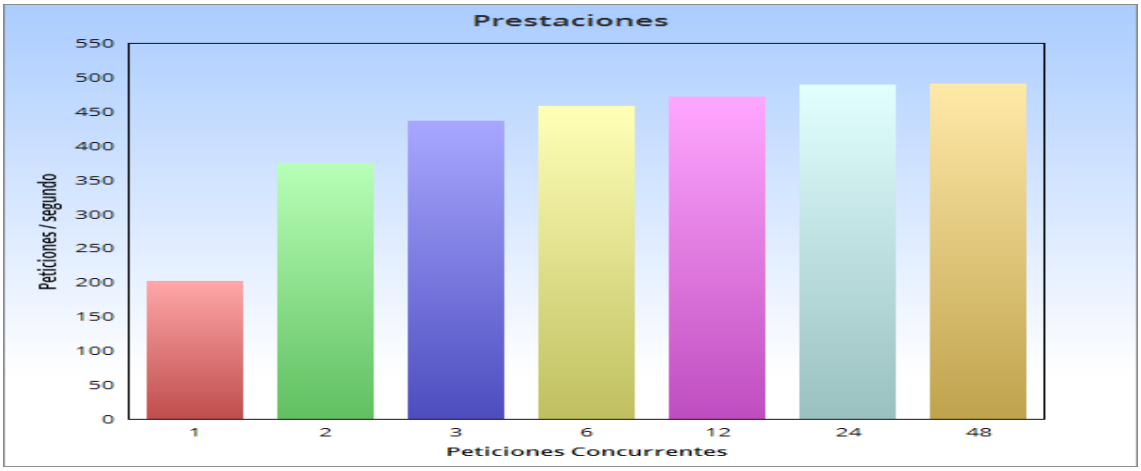


Como se observa, a partir de 24 peticiones concurrentes, el número de peticiones por segundo deja de aumentar. Durante las pruebas hemos monitorizado la actividad del sistema mediante la orden:

```
top
```

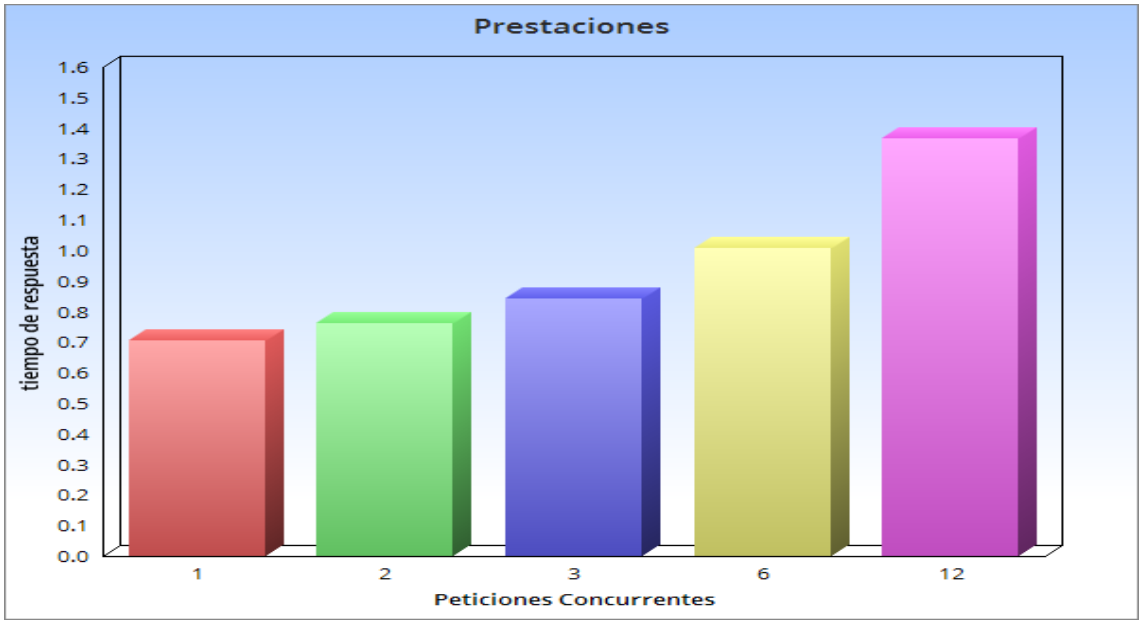
Observando claramente que en este caso el cuello de botella reside en el repartidor de carga, dada la incesante actividad del proceso de HAProxy, llegando a consumir el 80% del tiempo de CPU, de hecho, cuanto más sencilla sea la página, más trabajo tiene que realizar el repartidor de carga. Por otra parte, el tiempo de respuesta se mantendrá constante mientras el número de peticiones concurrentes no exceda el número de servidores reales disponibles. Cuando esto ocurra, el tiempo de respuesta aumentará.

En el caso de https obtenemos:



Como observamos, el volumen de peticiones / segundo atendidas es mucho menor que en el caso de http, esto se debe a que como ya se explicó anteriormente, la tarea de cifrar recae en los servidores web, que, al tener más trabajo que hacer no pueden atender tantas peticiones por unidad de tiempo como en el caso de http.

En cuanto al tiempo de respuesta, obtenemos:

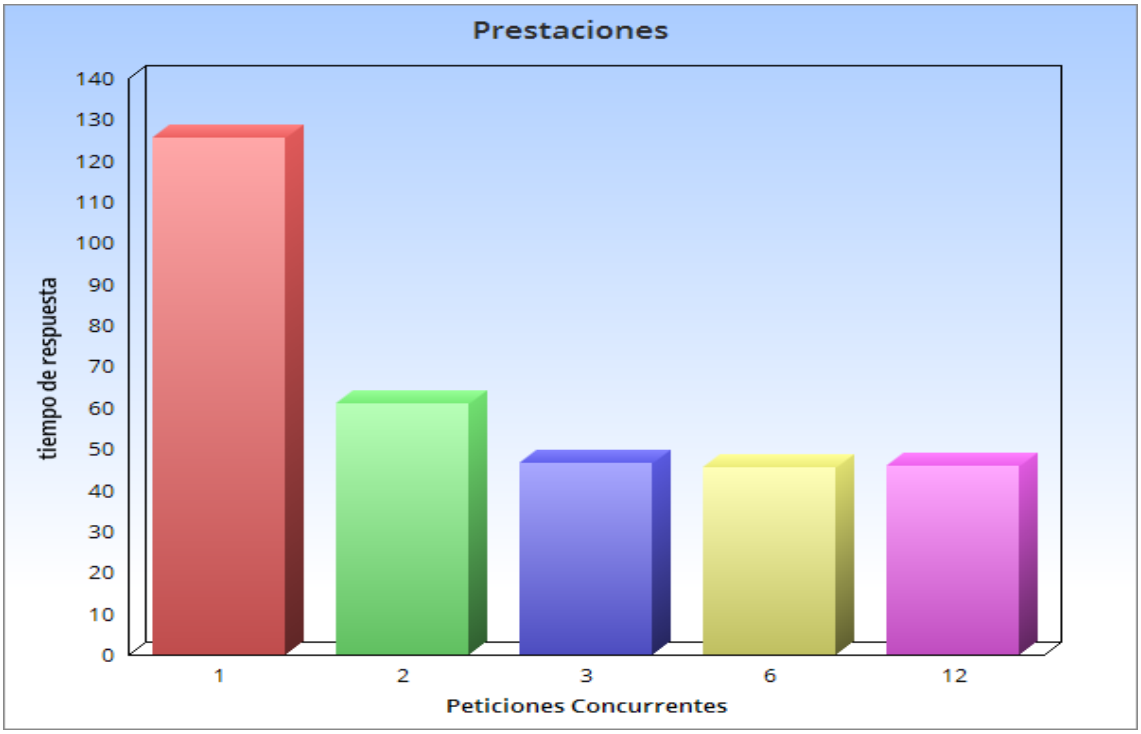


Como es de esperar, mientras el número de peticiones concurrentes no supere el número de servidores disponibles, el tiempo de respuesta varia muy poco, aumentando cuando se excede este valor.

Analicemos ahora el caso del cálculo del número PI:

Como apreciamos, el número de peticiones por segundo disminuye drásticamente respecto al caso de la página que devuelve la fecha y la hora y es que ahora el repartidor de carga permanece ocioso, consumiendo alrededor del 4% del tiempo de CPU, mientras que los servidores web consumen mucha CPU.

Analicemos ahora lo que ocurre con el tiempo de respuesta:



Como se observa, al tratarse de un cálculo costoso, el tiempo de respuesta disminuye mientras el número de peticiones concurrentes no supere el número de servidores disponibles, momento a partir del cual, el tiempo de respuesta permanece prácticamente constante.

Sin embargo, en este caso la utilización de https no tiene un impacto significativo en las prestaciones, ya que el cuello de botella sigue estando en los servidores web y el número de peticiones / segundo es idéntico a lo que se muestra en la gráfica para http.

4.3. Evaluación de prestaciones del servidor DNS

Para evaluar las prestaciones del servidor DNS, haremos usa de la herramienta de benchmarking dnsperf, que se distribuye en forma de código fuente que debemos compilar, una vez hecho, lanzamos una orden de prueba como la siguiente:

```
dnsperf -s 192.168.1.200 -d queryfile-example-current -l 60 -c 1 -Q 10
```

A continuación, pasamos a explicar que indica cada parámetro:

- El parámetro -s indica el servidor al que lanzamos las consultas.

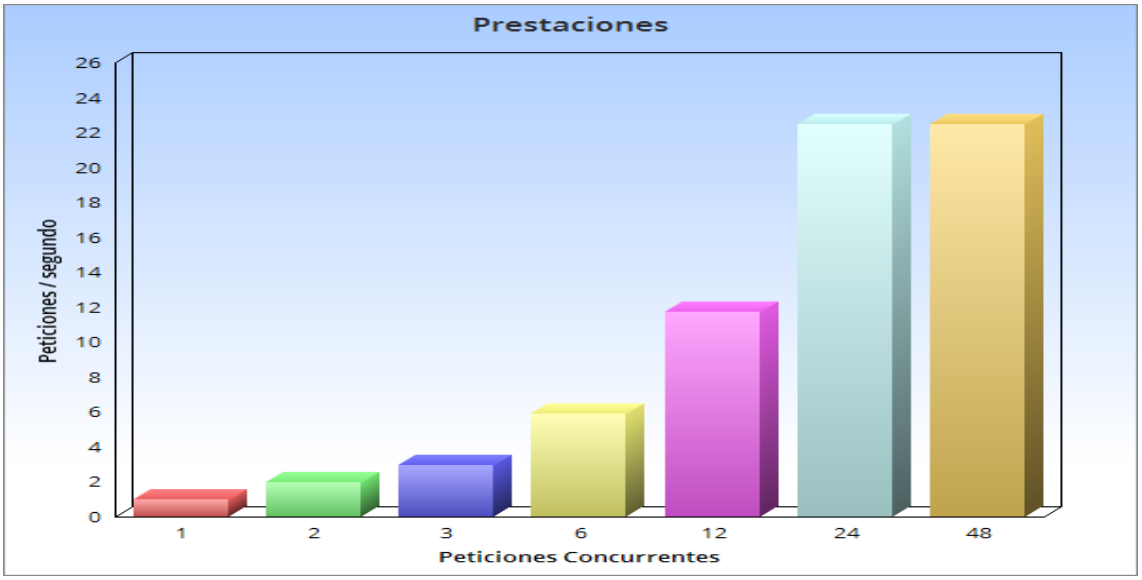
- El parámetro -d indica los nombres a resolver, que, en este caso se leen del fichero queryfile-example-current.

- El parámetro -l indica durante cuanto tiempo vamos a estar lanzando consultas, 60 segundos en nuestro caso.

- El parámetro -c indica la cantidad de clientes que lanzan peticiones.

- El parámetro -Q indica el número de peticiones concurrentes que se van a lanzar.

  Según las pruebas realizadas, en las cuales se han lanzado peticiones durante un minuto, se observa que el número de peticiones por segundo ha ido aumentando, hasta llegar a 24 peticiones concurrentes tal y como muestra la siguiente gráfica:



5. Conclusiones

En este proyecto fin de grado hemos abordado una problemática fundamental en los servicios de Internet como es la alta disponibilidad y el equilibrado de carga, que, si bien son mecanismos totalmente transparentes para los usuarios, son la columna vertebral de todos los servicios de Internet ofrecidos a gran escala, desde buscadores a servicios en la nube de todo tipo.

El proyecto nos ha permitido profundizar en el conocimiento de la herramienta de virtualización Oracle virtual box, haciéndonos ahondar en temas como los tipos de red que soporta, etc.

También nos ha permitido adquirir una valiosa experiencia práctica en lo referente a la administración de sistemas Linux, brindándonos la posibilidad de configurar servicios de red ampliamente utilizados, así como participar en todas las decisiones involucrados en el diseño del cluster.

Además de ser una valiosa introducción practica al mundo del clustering y los sistemas de alta disponibilidad, introduciendo herramientas ampliamente utilizadas como HAProxy e IPVS.

Este proyecto no está ni mucho menos cerrado, dado que la amplia variedad de servicios que se pueden beneficiar de estas tecnologías es enorme:

- Correo electrónico
- Bases de datos
- Redes de distribución de contenidos
- Etc.…

Hay varios temas en los que no hemos entrado como son, por ejemplo, el de la seguridad (HAProxy tiene mecanismos para evitar ataques de denegación de servicio).

Otra cuestión importante es la de que si bien este trabajo se ha limitado a realizar demostraciones conceptuales, en entornos como el que hemos implementado es fundamental también realizar optimizaciones a nivel hardware como por ejemplo, hacer un uso óptimo de la afinidad de las memorias caché, la realización de mejoras en la red de interconexión e incluso en el almacenamiento compartido dentro del cluster, donde nos hubiera gustado incluir por ejemplo, glusterFS en vez de NFS por tratarse de un sistema de ficheros replicado y distribuido.

Por último, comentar que, durante la realización de este proyecto, hemos podido constatar que las distintas áreas de conocimiento de la ingeniería informática están más interrelacionadas de lo que uno percibe cuando cursa las asignaturas: Comenzando por la necesaria planificación tanto temporal como material, este proyecto ha exigido desde conocimientos de redes, hardware y sistemas operativos hasta la utilización de técnicas y herramientas para el análisis de los resultados obtenidos.

6. Bibliografía

LVS DOCUMENTATION. (1998). “Manuals”. https://cbonte.github.io/haproxy-dconv/1.7/configuration.html, recuperado el 05- 06- 2018.

NOMINUM. (2007). “DNS Performance Tool Manual”. ftp://ftp.nominum.com/pub/nominum/dnsperf/1.0.1.0/dnsperf-1.0.1.0-Info-20071228.pdf, recuperado el 27- 06- 21018.

RED HAT INC. (2018). “Load Balancer Administration”. https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/load_balancer_administration/index, recuperado el 06- 06- 2018.

TARREAU, W. (2018). “HAProxy Configuration Manual”. https://cbonte.github.io/haproxy-dconv/1.7/configuration.html, recuperado el 02- 06- 2018.

THE APACHE SOFTWARE FOUNDATION. (2018). “Documentation”. https://httpd.apache.org/docs/2.4/programs/ab.html, recuperado el o8- 06- 2018.

W. SMITH, R. LPIC-2. Linux Professional Institute Certification, Madrid: Anaya Multimedia, 2011.