Trabajo **COS – Curso 2022/2023 – (4)**

Reparto de carga con alta disponibilidad (Corosync/Pacemaker/Pcs)

Antes de empezar, sacaremos una instantánea de todas las máquinas del clúster. Se sugiere que la instantánea se llame “preHA”.

Introducción

Una vez concluido el seminario 3, tenemos configurado un servidor web, pero no es altamente disponible, pues el diseño tiene varios *SPOF (Single Point Of Failure)* o puntos únicos de fallo. Estos puntos son elementos del sistema cuyo fallo supondría la paralización del servidor web en su conjunto. El primer *SPOF* lo encontramos en el balanceador de carga, si el nodo *master* deja de estar operativo, por un fallo hardware o por un fallo software que afecta al balanceo de carga, las peticiones *http* de los clientes no se podrán servir. El segundo *SPOF* se encuentra en el nodo de almacenamiento, pues todas las páginas web residen en él y si deja de funcionar paralizaría el servidor web completo. En este seminario se propone la eliminación del primer *SPOF*, dejando la eliminación del segundo para posibles ampliaciones.

Generalmente los puntos únicos de fallo se eliminan añadiendo redundancia. En este seminario añadiremos un nodo adicional (*standby*) que asumirá la función de balanceador de carga en caso de que el nodo *master* falle. Se utilizará una configuración *activo/pasivo*, siendo el nodo *master* el encargado de repartir las peticiones *http* entrantes entre los servidores de la granja y sólo en caso en el que el servicio deje de estar operativo en dicho nodo, asumiría esta función el nodo *standby*.

Para hacer el servicio altamente disponible es necesario disponer de elementos de reserva (pasivos) y el software necesario que permita la entrada en escena de estos, cuando se produzca la caída de algún elemento activo. El software que se va a utilizar consta básicamente de tres paquetes: ***Pacemaker, Corosync y Pcs***. Hay otras alternativas, pero estos paquetes tienen el apoyo de la comunidad *Red Hat*, ya que a partir de la versión *RHEL 7,* son los únicos que recomiendan para lograr la alta disponibilidad. Ver figura 1.

Corosync se encarga de controlar cuantos nodos en el clúster están activos. Lo hace pasando periódicamente unos mensajes (*totems*), que coloquialmente se conocen como latidos *(heartbeats).* Este software se instala en los nodos *master y standby.* Se configura para comunicarse con *Pacemaker* y avisarle en el caso der la caída de alguno de ellos.

Pacemaker es el gestor de recursos del clúster (Cluster Resource Manager). En nuestro caso vamos a definir tres recursos, ***dos direcciones IP flotantes***, una interna y otra externa, y el balanceador de carga ***Haproxy***. Los tres recursos se han de ejecutar **juntos** en el nodo *master* o en el nodo *standby*.

Pcs (Pacemaker/Corosync Configuration System) es la herramienta a través de la cual se configuran *Corosync y Pacemaker*.

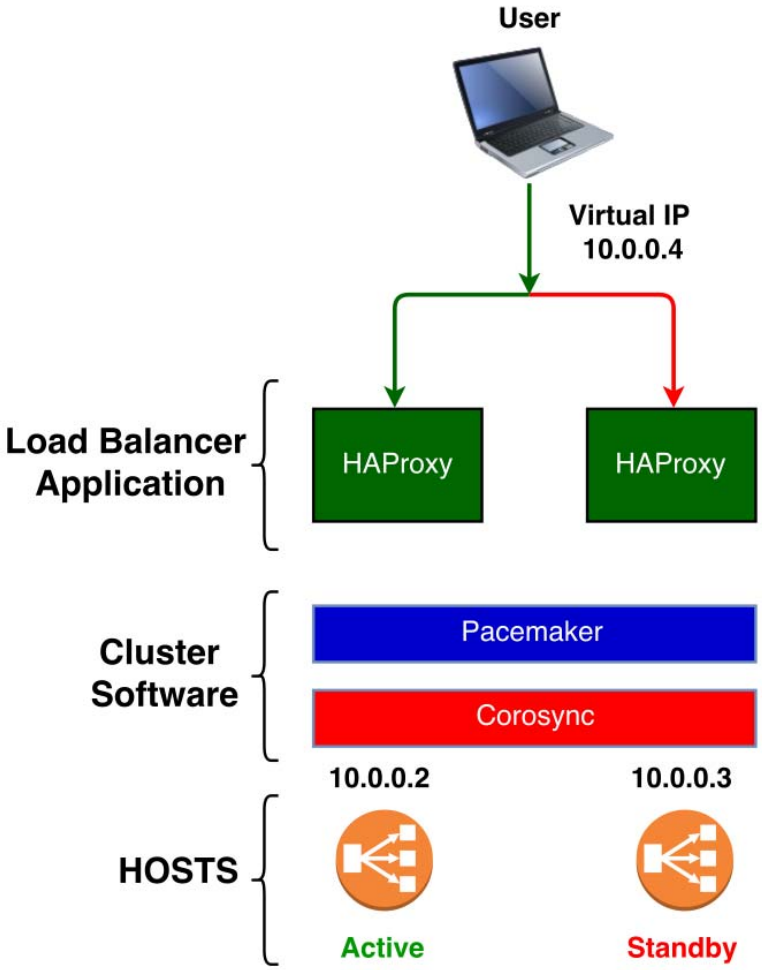

Figura 1. Software alta disponibilidad

Definiremos un par de direcciones IP virtuales (VIP), en algunos casos se les llama IP flotantes *(floating IP)*, una por cada interfaz de red, y el nodo que este activo es el que las tendrá configuradas. También habrá que elegir las direcciones IP de la máquina *standby*. En concreto las direcciones IP implicadas son las siguientes:

	Nombre	Dirección IP	
Director master	cluster1 (enp0s3)	RIP: 192.168.1.101	VIP: 192.168.1.200
	cluster1 (enp0s8)	RIP: 10.0.100.1	VIP: 10.0.100.200
	cluster1 (enp0s9)	RIP: 10.0.200.1	
Servidores	cluster2	RIP: 10.0.100.12	

Reales	cluster3 cluster4	10.0.100.13 10.0.100.14	
Director standby	standby (enp0s3) standby (enp0s8) standby (enp0s9)	RIP: 192.168.1.102 RIP: 10.0.100.2 RIP: 10.0.200.2	VIP: 192.168.1.200 VIP: 10.0.100.200

Los clientes acceden al servicio web a través de la VIP 192.168.1.200 (externa), esta dirección estará asociada al adaptador *enp0s3* del nodo *master* si está activo o se asociará al adaptador *enp0s3* del nodo *standby* si el nodo *master* deja de estar operativo, por eso a menudo se le llama *IP flotante*. Sin este recurso se podría acceder al servicio a través de la IP real *(RIP)* del nodo *master* 192.168.1.101, pero si el nodo dejase de funcionar el servicio quedaría inaccesible. El encargado de asociar esta IP virtual con el adaptador *enp0s3* del nodo *master* o del nodo *standby* de forma completamente automática es *Pacemaker*.

Las peticiones entrantes son repartidas por *HAproxy* entre los servidores de la granja, pero estos contestan a los clientes a través del nodo *master* o del nodo *standby*, depende de cuál de los dos este activo. Si no se dispone de una VIP interna, y en los servidores se configura como pasarela *(gateway)* la IP real asociada al adaptador *enp0s8* del nodo *master*, si este deja de funcionar, las respuestas del servidor web no llegarían a los clientes. La solución es definir un nuevo recurso, que será una VIP interna 10.0.100.200, que *Pacemaker* asociará al adaptador *enp0s8* del nodo *master* o *standby* en función de las necesidades.

El tercer recurso controlado por *Pacemaker* es el balanceador de carga *HAproxy*. Si el balanceador deja funcionar, las peticiones no llegarían a los servidores de la granja. *Pacemaker* se conecta de forma periódica con el recurso para ver su estado, si detecta que ha caído, procede a lanzarlo de nuevo a ejecución. El administrador puede definir un número máximo de caídas y si se supera, en lugar de levantar el recurso en el nodo *master*, se trasladaría el recurso al nodo *standby*, eso se conoce como una conmutación por fallo *(fail over).* El propio balanceador se encarga de controlar el estado de los servidores de la granja, si alguno de ellos cae, lo marca como no disponible y deja de enviarle peticiones. Para que el servicio dejase de estar disponible deberían caer simultáneamente todos los servidores de la granja.

1. Preparación: crea y configura la máquina standby

[`host`]

En primer lugar, vamos a añadir una nueva tarjeta de red (Intel PRO 1000 Server T) al nodo *master*. La conectamos a una red interna que llamaremos por ejemplo *latido.* A través de esta red, *Corosync* pasará periódicamente mensajes al nodo *standby* para asegurarse de que goza de buena salud. Se le puede asociar la MAC 0800270101C1.

Desde el interfaz gráfico *VirtualBox*, clona la máquina `master` sobre la máquina `standby`. Cambia las direcciones MAC asociadas a todas las tarjetas, utilizando las que se detallaron en el seminario 1

para las dos primeras tarjetas de red y la MAC 0800270101C2 para la nueva tarjeta de red. También puede utilizarse la línea de órdenes:

```
# vboxmanage clonevm master --name standby
```

[master]

Una vez arrancado el nodo *master*, vamos a proceder a configurar la nueva tarjeta de red. Podéis averiguar el nombre que el sistema operativo le ha asociado con el comando:

```
# ip addr show
```

Vamos a suponer que el nombre asociado es *enp0s9*. Editamos el fichero de configuración */etc/sysconfig/network-scripts/ifcgf-enp0s9* y añadimos el siguiente contenido:

```
TYPE=Ethernet
BOOTPROTO=static
NAME=enp0s9
DEVICE=enp0s9
ONBOOT=yes
UUID=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
IPADDR=10.0.200.1
NETMASK=255.255.255.0
NM_CONTROLLED=no
```

Si el fichero no existe, se puede hacer una copia del fichero *ifcgf-enp0s8* y modificarlo según se ha indicado.

Para que los cambios se hagan efectivos ejecutamos:

```
# systemctl restart network
```

Y comprobamos que se ha asociado correctamente la IP elegida a la interfaz de red.

```
# ip addr show
```

Una vez iniciada la máquina virtual, modifica el archivo */etc/hosts* para añadir la nueva máquina.

- /etc/hosts

```
…….
10.0.100.2          standby.cluster             standby
10.0.200.1          master-cr
10.0.200.2          standby-cr
```

Los nombres *master-cr* y *standby-cr* se usarán a continuación para formar el *cluster*, que en este caso estará formado únicamente por dos nodos.

[standby]

Inicia la máquina *standby*. Vamos a asociar a los adaptadores de red, las direcciones IP elegidas.

- vi /etc/sysconfig/network-scripts/ifcfg-enp0s3:

```
TYPE=Ethernet
BOOTPROTO=static
NAME=enp0s3
DEVICE=enp0s3
ONBOOT=yes
UUID=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
IPADDR=192.168.1.102
NETMASK=255.255.255.0
GATEWAY=192.168.1.1
DNS1=192.168.1.1
NM_CONTROLLED=no
```

- vi /etc/sysconfig/network-scripts/ifcfg-enp0s8:

```
TYPE=Ethernet
BOOTPROTO=static
NAME=enp0s8
DEVICE=enp0s8
ONBOOT=yes
UUID=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
IPADDR=10.0.100.2
NETMASK=255.255.255.0
NM_CONTROLLED=no
```

- vi /etc/sysconfig/network-scripts/ifcfg-enp0s9:

```
TYPE=Ethernet
BOOTPROTO=static
NAME=enp0s9
DEVICE=enp0s9
ONBOOT=yes
UUID=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
IPADDR=10.0.200.2
NETMASK=255.255.255.0
NM_CONTROLLED=no
```

Una vez realizados los cambios reiniciamos el servicio:

```
# systemctl restart network
```

Comprobamos los cambios:

```
# ip addr show
```

Para finalizar comprobamos que tenemos conexión con el mundo exterior

```
# ping www.google.es
```

Anadir los nombres *master*, *master-cr* y *standby-cr* en el fichero */etc/hosts*

- `/etc/hosts`

```
…….
10.0.100.1          master.cluster       master        cluster1
10.0.200.1          master-cr
10.0.200.2          standby-cr
```

Modifica el nombre de la máquina.

```
# hostnamectl set-hostname standby
```

Comprueba que la red interna *latido* está bien configurada:

```
# ping master-cr
```

2. [master+standby] Instala corosync, pacemaker y pcs

En los dos nodos que van a formar el cluster instalamos los paquetes necesarios.

```
# yum install –y  pcs
```

Pacemaker y Corosync se instalarán cómo dependencias del paquete *pcs*.

Tras la instalación se pondrá en ejecución en ambos nodos el demonio *pcsd* y se indicará que se inicie siempre en el arranque:

```
# systemctl start pcsd && systemctl enable pcsd
```

Cuando se instala *pacemaker* se crea automáticamente el usuario *hacluster*. Se debe configurar una contraseña para este usuario en los nodos *master* y *standby*, **usando la misma contraseña para ambos**:

```
# passwd hacluster
```

Para que los distintos demonios que gestionan el cluster en ambos nodos, no tengan problemas para comunicarse entre ellos, configuramos adecuadamente el cortafuegos y el módulo de seguridad mejorada:

```
# firewall-cmd --permanent --add-service=high-availability
# firewall-cmd --reload
# setsebool -P daemons_enable_cluster_mode 1
```

A partir de este momento los comandos se ejecutan sólo en el nodo master.

Tras tener el usuario *hacluster* configurado, desde el nodo *master* se autoriza al nodo *standby.*

```
# pcs cluster auth master-cr standby-cr
Usename: hacluster
Password:
master-cr: Authorized
standby-cr: Authorized
```

Creamos el cluster, al que llamaremos *cluster_lb* (load balancer) desde el nodo *master*:

```
# pcs cluster setup --name cluster_lb master-cr standby-cr --force
```

Este comando genera automáticamente el fichero de configuración */etc/corosync/corosync.conf* y lo copia en el nodo *standby.*

El contenido del fichero será el siguiente:

```
totem {
version: 2
secauth: off
cluster_name: cluster_lb
transport: udpu
}

nodelist {
      node {
            ring0_addr: master-cr
            nodeid: 1
            }
```

```
      node {
            ring0_addr: standby-cr
            nodeid: 2
            }
}
quorum {
provider: corosync_votequorum
two_node: 1
}
logging {
to_syslog: yes
}
```

Ponemos en ejecución *corosync y pacemaker* en los dos nodos:

```
# pcs cluster start --all
```

Si queremos que arranquen en todos los nodos que forman cluster cada vez que se reinicien:

```
# pcs cluster enable –all
# pcs status
```

Verificamos la instalación de *corosync*:

```
# corosync-cfgtool –s
# corosync-cmapctl | grep members
# pcs status corosync
```

Para comprobar que la configuración de *pacemaker* es correcta:

```
# crm_verify –L -V
```

Como no hemos configurado ningún mecanismo de STONITH/FENCING (Shoot-The-Other-Node-In-The-Head) nos informará de que la configuración no es válida. En un entorno en producción habría que definir un recurso de STONITH que fuera gestionado por *pacemaker*. Ese recurso se encarga de apagar el nodo que falla dentro del clúster, antes de levantar los recursos en el nodo que permanecía inactivo. Esto se realiza para evitar una situación conocida como *Split-Brain*, done los recursos están activos en los dos nodos del clúster simultáneamente.

Desactivamos el mecanismo de STONITH

```
# pcs property set stonith-enabled=false
```

Y comprobamos que ya no causa problemas en la verificación de la configuración.

Quorum es el número mínimo de nodos activos en el clúster para que este pueda funcionar. Normalmente este parámetro se establece con un valor de la mitad de nodos + 1. Cómo sólo

tenemos dos nodos en el clúster no tiene sentido definir este parámetro. Si uno de los dos nodos dejara de estar operativo el cluster no tendría quorum y los recursos no estarían disponibles.

```
# pcs property set no-quorum-policy=ignore
```

Comprobamos la configuración del cluster:

```
# pcs status
```

La salida de este comendo nos informará del estado de *PCSD*, que debe estar *Online* en ambos nodos. La lista de recursos *Full list of resources* aparecerá vacía, pues aún no hemos definido ningún recurso. Y en *Daemon Status* debe indicar que *corosync/pacemaker/pcsd: active/enable.*.

4. [master] Configura los recursos de pacemaker

Para ver una lista de todos los recursos que puede gestionar *pacemaker*:

```
# pcs resource list
```

Para obtener información sobre un tipo de recurso concreto:

```
# pcs resource describe ocf:heartbeat:IPaddr2
```

Los agentes de recursos son scripts que se encuentran normalmente en el directorio */usr/lib/ocf/resource.d/heartbeat* y le indican a *pacemaker* la forma correcta de iniciar, parar y monitorizar cada uno de los recursos que es capaz de gestionar.

Ahora definimos los recursos del clúster que debe gestionar `pacemaker`. Son tres:

- La IP virtual en la red externa (*red-nat-1*). Ya que el servicio de distribución de carga puede migrar entre `master` (`192.168.1.101`) y `standby` (`192.168.1.102`), hay que definir una IP virtual (por ejemplo: `192.168.1.200`) que será un alias de la máquina que, en cada momento, esté dando servicio. Esta será la dirección "pública" del servidor web y a la que accederán los clientes.

```
# pcs resource create VIP_externa ocf:heartbeat:IPaddr2 ip=192.168.1.200
nic=enp0s3 cidr_netmask=24 op monitor interval=10s
```

- La IP virtual en la red interna: ya que el servicio de distribución de carga puede migrar entre `master` (`10.0.100.1`) y `standby` (`10.0.100.2`), 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 en que falle uno de los distribuidores de carga, el distribuidor superviviente debe hacer de "gateway" de los servidores reales (`cluster2`, `cluster3 y cluster4`). Por tanto, será necesario también configurar estos servidores reales para que su "gateway" sea `10.0.100.200`.
-

```
# pcs resource create VIP_interna ocf:heartbeat:IPaddr2 ip=10.0.100.200
nic=enp0s8 cidr_netmask=24 op monitor interval=10s
```

- El propio servicio de distribución de carga: *HAproxy*

  Antes de que *pacemaker* lance a ejecución el demonio *haproxy* hay que hacer algunos ajustes en los nodos *master* y *standby*. En ambos nodos paramos y deshabilitamos el servicio *HAproxy*, pues ahora *Pacemaker* se encarga de su puesta en marcha, su parada y la monitorización de su estado.

```
# systemctl stop haproxy && systemctl disable haproxy
```

En ambos nodos cambiamos el contenido del fichero */etc/haproxy/haproxy.cnf* para que ahora *HAproxy* realice la escucha de peticiones entrantes en el puerto 80 de la VIP externa. Este fichero ha de ser exactamente igual en ambos nodos. Su contenido es igual que el fichero utilizado en el seminario 3, con sólo una modificación:

Cambiamos esta línea:

```
bind 192.168.1.101:80
```

Por esta:

```
bind 192.168.1.200:80
```

Ahora creamos el recurso:

```
# pcs resource create HAproxy systemd:haproxy op monitor interval=5s
```

Es muy importante que los tres recursos que hemos creado sean ubicados por *pacemaker* en el mismo nodo, para lo cual establecemos una serie de restricciones que lo garanticen:

```
# pcs constraint colocation --help
# pcs constraint colocation add VIP_externa with VIP_interna INFINITY
# pcs constraint colocation add VIP_interna with HAproxy INFINITY
```

Establecemos un orden de puesta en marcha de los recursos, para que *HAproxy* no se inicie hasta que las direcciones virtuales se hayan asignado a las interfaces de red.

```
# pcs constraint order --help
# pcs constraint order set VIP_externa VIP_interna HAproxy
```

Si quereos que un determinado recurso resida en un nodo por defecto:

```
# pcs  constraint location HAproxy prefers master-cr
```

Si el nodo *master* deja de estar operativo, los tres recursos se moverán al nodo *standby*, pero en el momento el nodo *master* vuelva a funcionar correctamente recuperará los recursos.

Llegados a este punto vamos a comprobar que la configuración es correcta, que se han creado los tres recursos que gestionará *pacemaker* y que se han tenido en cuenta las restricciones.

```
# pcs status
# pcs config
# pcs constraint show
```

Reiniciamos *pacemaker* y *corosync* en ambos nodos para que contemple los cambios introducidos en su configuración:

```
# pcs cluster stop --all
# pcs cluster start --all
```

Podéis comprobar con la orden *pcs status* qué nodo de los dos que forman el cluster tiene asignados los tres recursos. En el nodo que los tenga ejecutáis:

```
# ip addr show
```

Podéis comprobar que las dos direcciones virtuales están asignadas a las interfaces de red *enp0s3* y *enp0s8*.

8. [Servidores] Configura el nuevo gateway de los servidores reales

Accediendo a cada nodo por ssh o utilizando un fichero de comandos para lanzar la orden simultáneamente a todos los servidores, sustituimos el gateway real por el virtual.

En el fichero */etc/sysconfig/network-scripts/ifcfg-enpos3* de cada uno de los servidores hay que cambiar:

```
GATEWAY=10.0.100.1
```

Por:

```
GATEWAY=10.0.100.200
```

Y reiniciamos el servicio de red:

```
# systemctl restart network
```

9. Verifica el funcionamiento del servidor Web

Para comprobar que la instalación es correcta, lanzaremos en una de las máquinas, `cluster1` o `standby` un navegador solicitando la URL `http://192.168.1.200/index.php`. Podemos utilizar el navegador `w3m` o bien instalar otro (por ejemplo `firefox`).

Alternativamente, podemos establecer una redirección de puertos en la aplicación VirtualBox, de forma similar a lo que hicimos en la sesión anterior:

```
VBoxManage natnetwork modify -t red-nat-1 -p "http2:tcp:[]:5280:[192.168.1.200]:80"
```

En este caso, la URL a la que acceder desde un navegador de la maquina física (host) es `http://localhost:5280/index.php`. Para acceder desde otra máquina del laboratorio http://direccion_IP_host:5280/index.php.

Para comprobar el correcto funcionamiento del servidor Web de alta disponibilidad con equilibrado de carga, deberíamos realizar las siguientes pruebas:

- Que el sistema funciona con todos sus elementos activos. En este caso, uno de los distribuidores de carga, el que esté activo, reparte la carga entre los servidores reales.
- Que el sistema funciona cuando se apaga un distribuidor de carga. El distribuidor superviviente se hace cargo del servicio y reparte la carga. El "apagado" de un distribuidor de carga se consigue fácilmente desconectando (virtualmente) el cable conectado al interfaz de red *enp0s9* o apagándolo con la orden *poweroff*.
- Que el sistema funciona cuando se apaga uno o varios servidores reales. En ese caso, los servidores supervivientes se hacen cargo del servicio. En el caso extremo de que todos los servidores reales estén apagados, el distribuidor de carga debería ofrecer una página local de error. Se pueden apagar los servidores con la orden *poweroff* o también desconectando (virtualmente) el cable conectado al interfaz de red.
- Que el sistema funciona con un distribuidor de carga y sólo un servidor real.
- Simular un fallo software matando el proceso *haproxy*.

```
# systemctl stop haproxy
```

Probamos a recargar la página y vemos que no es posible. Pasados 5 segundos pacemaker se da cuenta de que le recurso HAproxy ha caído y lo lanza de nuevo a ejecución. Pasado ese corto periodo de tiempo ya es posible recargar la página.

Una vez verificado el correcto funcionamiento, podemos ejecutar un *benchmark* para medir sus prestaciones. Por ejemplo, podemos utilizar el *apachebench*, ejecutando desde el nodo *master*:
Instalamos el paquete en el nodo *master* y lanzamos la simulación.

```
# yum -y install httpd-tools
```

```
# ab –n 100 –c 10 http://192.168.1.200/index.php
```

o también desde otra máquina del laboratorio con (si tiene instalado el paquete)

```
ab –n 100 –c 10 http://direccion_IP_host:5280/index.php
```

donde el parámetro –n indica el número de peticiones total y –c el número de peticiones concurrentes.

Se pueden realizar pruebas teniendo activos uno, dos o los tres servidores de la granja. Comprobar que el tiempo de respuesta por petición disminuye a medida que aumentamos el número de servidores. Si tenemos una estimación del número de peticiones simultaneas que pueden llegar al servidor web y queremos mantener un tiempo de respuesta razonable, es posible estimar el número de servidores que debería tener la granja.

Para someter al servidor a pruebas de carga alternativas ante las distintas configuraciones y analizar los resultados obtenidos, se pueden usar también otras herramientas de pruebas de carga http cómo por ejemplo **Siege.**
Instalamos el paquete en el nodo *master*.

```
# yum –y install epel-release
# yum –y install siege
```

Lanzamos la simulación:

```
# siege –c 10 –r 100 http://192.168.1.200/index.php
```

donde el parámetro –c indica el número de peticiones concurrentes y –r el número de peticiones total. Observar que los resultados obtenidos son coherentes con los obtenidos en la anterior simulación.

También podríamos ubicar en el servidor una página que genere algo más de carga. Por ejemplo, el siguiente programa PHP calcula el valor de la constante pi mediante integración numérica. El programa acepta como dato el número de intervalos y lo guardaremos con el nombre *pi.php*.

```
#nano pi.php

<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>
```

Por ejemplo, para n=1000000 intervalos, podemos ejecutarlo en un navegador solicitando la URL `http://192.168.1.200/pi.php?n=1000000` desde un nodo del cluster (*master o standby*) , o la URL `http://localhost:5280/pi.php?n=1000000` desde la máquina física (host). Para simular su comportamiento con *apachebench* la orden sería, lanzada desde el nodo *master*:

```
ab –n 100 –c 10 -r http://192.168.1.200/pi.php?n=1000000
```

Ahora podríamos probar con diferente número de peticiones totales, concurrentes y número de intervalos.