



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



Escola Tècnica
Superior d'Enginyeria
Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Instalación, configuración y evaluación de un servidor de Internet de alta disponibilidad con equilibrado de carga basado en clúster

TRABAJO FINAL DE MASTER

Master Universitario de Ingeniería Informática

Autor: Mario David Cariñana Abasolo

Directores:

Pedro Juan López Rodríguez
María Elvira Baydal Cardona

9 de septiembre de 2016

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 con frecuencia han pasado a implementarse mediante clusters de computadores como una excelente alternativa coste-prestaciones. El acceso al clúster suele realizarse a través de un nodo maestro o 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 Master se plantea instalar, configurar y evaluar un servidor de Internet basado en un clúster de computadores, con sistema operativo Linux. El sistema incorporará dos nodos directores para garantizar un funcionamiento continuo, así como mecanismos para repartir la carga entre los servidores. Los nodos servidores estarán virtualizados para ofrecer flexibilidad en la configuración y ofrecerán al menos un servicio Web, con páginas estáticas y dinámicas. Por otra parte, también es habitual que los nodos servidores accedan a bases de datos para preparar la respuesta a las peticiones de los clientes. Por este motivo, en el TFM también se explorarán soluciones escalables basadas en clúster para bases de datos.

Palabras clave: clúster de computadores, servidor de Internet, equilibrado de carga, alta disponibilidad, bases de datos, virtualización.

Abstract

In recent years, the demand growth of Internet services has in many cases led to the inability of a single server to address the volume of client requests. This has sparked a change on the architecture of the servers, which have become to be implemented through computer clusters with frequency as an excellent cost-performances alternative. The access to the cluster usually takes place via a master (or manager) node, which is responsible for allocating the client requests from among a numer of servers that are the ones who actually implement the requested service.

This Final Master Project presents the installation, the setting and the testing of an Internet server based on a computer cluster, with a Lynux operating system. The system will incorporate two manager nodes to guarantee continuous operation, and mechanisms to divide the load between the servers. The server nodes will be virtualized in order to offer flexibility in the setting and will provide at least one Web service with static and dynamic pages. Moreover, it is also commonly that the server nodes have access to databases in order to prepare the response of the client requests. For this reason, in the FMP will also be explored scalable solutions based on cluster for databases.

Keywords: computer clusters, Internet server, load balancing, high availability, databases, virtualization.

Índice de abreviaturas

LB (*Load Balancer*), Repartidor de carga
CPD (*Centro de Proceso de Datos*)
HA (*High Availability*), Alta disponibilidad
HW (*Hardware*)
SW (*Software*)
BBDD (*Bases de Datos*)
LVS (*Linux Virtual Server*)
IPVS (*IP Virtual Server*)
KTCVPS (*Kernel TCP Virtual Server*)
VIP (*Virtual IP*)
MSS (*Microsoft Sql Server*), Servidor de Sql de Microsoft.
SSD (*Solid State Disk*), Disco duro de estado sólido.
MGM (*Management*), Administración.
NDB (*Node Data Base*), Nodo de base de datos.
PK (*Primary Key*), Clave primaria.

Índice general

1. Introducción, motivación y contexto	9
1.1. Introducción	9
1.2. Motivación y objetivo	10
1.3. Contexto del trabajo	10
2. Análisis de las alternativas	13
2.1. Repartidor de carga	13
2.2. Alta disponibilidad	15
2.3. Sistema de base de datos	16
3. Estructura del servidor	19
3.1. Configuración de HAProxy	20
3.2. Configuración de Keepalived	23
3.3. Configuración de los nodos servidores	25
4. Estructura de BBDD	27
4.1. Configuración de MySql Cluster	31
4.2. Cambios en la configuración de Keepalived	35
4.3. Cambios en la configuración de HAProxy	35
5. Verificación del servidor web	37
5.1. Todos los elementos activos	38
5.2. Fallo de servidores web	39
5.3. Fallo de un LB	40

6. Verificación del servidor de BBDD	43
6.1. Todos los nodos activos	45
6.2. Fallo de un SQL	47
6.3. Fallo de un NDB	49
6.4. Fallo en un MGM	51
7. Evaluación	55
7.1. Evaluación del servicio web	55
7.2. Evaluación del servicio de BBDD	59
8. Conclusiones	61

Índice de figuras

3.1. Estructura del servidor	20
4.1. Estructura de MySQL Cluster	27
4.2. Relación de los componentes de un MySQL Cluster	29
4.3. Interacciones en casos de caída en un sistema con cuatro nodos NDB y dos réplicas	30
4.4. Modelo de MySQL Cluster con la opción <i>cluster replication</i> . . .	31
4.5. Flujo en el sistema de una petición para MySQL Cluster	32
5.1. Resultado de consultar la página index.php	39
5.2. Resultado de consultar la página index.php con dos nodos fue- ra de servicio	40
5.3. Resultado de consultar la página index.php con un fallo en el LB1	41
6.1. Creación de la base de datos de prueba	44
6.2. Prueba de acceso entre SQL	45
6.3. Estado de los nodos del clúster de MySQL, todos los elementos activos	46
6.4. Prueba de funcionamiento del clúster de MySQL con todos los elementos activos	47
6.5. Estado de los nodos del clúster de MySQL con un nodo SQL fuera de servicio	48
6.6. Prueba de funcionamiento del clúster de MySQL con un nodo SQL fuera de servicio	49

6.7. Estado de los nodos del clúster de MySql con un nodo NDB fuera de servicio	50
6.8. Prueba de funcionamiento del clúster de MySql con un nodo NDB fuera de servicio	51
6.9. Estado de los nodos del clúster de MySql con un nodo MGM fuera de servicio	52
6.10. Prueba de funcionamiento del clúster de MySql con un nodo MGM fuera de servicio	53
7.1. Evaluación del servidor web para pi.php con número de intervalos variable	57
7.2. Evaluación del servidor web para pi.php con un número de usuarios concurrentes variable	58
7.3. Evaluación del servidor de BBDD con número de <i>threads</i> concurrentes variable	60

Índice de figuras

Capítulo 1

Introducción, motivación y contexto

1.1. Introducción

Este trabajo pretende implementar desde cero un servidor de alta disponibilidad con dos vertientes, una de ellas centrada en el reparto de tráfico web y la otra centrada en la explotación de BBDD (*Bases de Datos*). Desde un primer momento el objetivo no ha sido hacer una aproximación, ha sido utilizar las mismas herramientas y métodos que se usan en los grandes centros de computación (con la complicación que ello conlleva). A todos los niveles es un reto nada desdeñable: alta disponibilidad, escalabilidad, mantenimiento, etc.

En esta memoria se van a abordar diversos temas relacionados con el trabajo, el primero de ellos es la motivación y los objetivos del trabajo, después se tratan las diferentes opciones a la hora de diseñar nuestro servidor, posteriormente se describe la estructura del mismo, las diferentes configuraciones, el SW (*Software*) necesario para su correcto funcionamiento y la instalación y configuración del mismo.

Por último con ambos prototipos implementados también se detallan las pruebas efectuadas para la verificación de su correcto funcionamiento, así como una evaluación de sus prestaciones.

1.2. Motivación y objetivo

Hay diferentes motivaciones, una de las principales sin lugar a duda es explorar y comprender desde un punto de vista práctico cómo funciona y cómo está diseñado un sistema de esta magnitud. También es destacable la parte que ocupa la administración del mismo, puesto que en determinados entornos de gran magnitud y mucha carga de trabajo la tarea se complica notablemente.

Otra de las motivaciones fundamentales es trabajar en primera persona con las diferentes tecnologías y concepciones que utilizan CPD (*Centro de Proceso de Datos*), aunque la cantidad o la magnitud del sistema no sea la misma, a nivel software son exactamente las mismas herramientas.

Los objetivos que se propusieron en un primer momento eran: crear un prototipo funcional de un sistema de HA (*High Availability*) para simular un escenario real de tráfico web, desarrollar otro escenario en el cual se simulase un entorno de explotación de bases de datos (modificando el prototipo que mentaba anteriormente) y por último verificar su funcionamiento y sus prestaciones mediante una serie de pruebas.

1.3. Contexto del trabajo

En los últimos años ha habido un incremento notable en la demanda de servicios de Internet, incluyendo el auge de sistemas en la "nube", lo cual ha provocado que en muchos casos un único servidor sea incapaz de atender el volumen de peticiones de los clientes. Este fenómeno ha propiciado que haya un cambio en la arquitectura de los servidores, teniendo que escalar el volumen de los mismos, derivándose en última instancia a entornos *clusterizados*.

El acceso a los diferentes servidores que forman los *clusters* suele realizarse a través de un nodo director o distribuidor de carga, LB (*Load Balancer*), que se encarga de repartir las peticiones de los clientes entre dichos servidores que son los que realmente implementan el servicio demandado. Por otra parte, a mayor número de servidores, mayor posibilidad de que alguno de ellos falle.

Un factor clave en este tipo de entornos es la redundancia. Esto se debe a que si alguna de las piezas del sistema cae debido a algún fallo tiene que haber alguna medida para no provocar una caída de servicio. El primer punto crítico en este tipo de sistemas se encuentra en el repartidor de carga, puesto que si fallase y no hubiese otro para tomar el control, el sistema se quedaría parado por completo al fallar el punto de entrada al mismo. Por este motivo

se suele tener duplicado, de manera que haya siempre un nodo de *backup* por si el principal falla.

En el caso de los servidores, aunque las peticiones se distribuyan entre ellos, hay que considerar siempre la posibilidad de que alguno de ellos pueda estar fuera de servicio. En esa situación la prioridad es que el cliente no se vea afectado, o al menos minimizar el impacto de la caída al máximo posible de cara al mismo.

Todo ello nos lleva a un escenario en el cual, como se ha explicado con anterioridad, es necesaria la redundancia, pero también son necesarios otros factores como: tener nociones avanzadas acerca del funcionamiento y estado del sistema (a todos los niveles), conocer las herramientas que se usan (saber si son estables e intentar obtener el máximo provecho de las mismas), etc.

Capítulo 2

Análisis de las alternativas

2.1. Repartidor de carga

Actualmente existen diversas alternativas para el equilibrado de carga. Por un lado puede implementarse como un dispositivo HW (*Hardware*) de propósito específico, lo cual tiene como inconveniente un coste bastante elevado y un alto precio. Por otro lado también puede implementarse mediante SW (*Software*) en un ordenador de propósito general, muchas veces utilizando código abierto. La tendencia actual es hacia estas últimas soluciones, grandes empresas del sector como Twitter o Instagram están optando por ello por diversos motivos [1].

El primero de ellos es sin duda el coste de la solución, es mucho más barato configurar una máquina cualquiera (que tenga unas prestaciones suficientes) con un SW libre que comprar HW específico. Generalmente también son sistemas mucho más escalables y muy configurables (diferentes algoritmos de carga, repartir pesos entre los servidores, etc). Otro de los motivos es que las prestaciones de una y otra solución tampoco son muy diferentes. Algunas soluciones SW a día de hoy pueden soportar una cantidad de tráfico/peticiones muy elevada (miles de peticiones por segundo), [2, 3].

En este trabajo se ha optado por explorar la vía SW con herramientas de código abierto. Actualmente existen dos nombres propios que destacan sobre el resto, son LVS (*Linux Virtual Server*) [4] y HAProxy [5]. En ambos casos se pueden usar diferentes estrategias a la hora de repartir la carga *Round-Robin*, *Least-Connection*, etc. También se pueden asignar pesos a los servidores en función de la potencia de los mismos para hacer una distribución óptima de los recursos disponibles. Ambas soluciones son muy escalables y aceptan una cantidad de peticiones nada despreciable.

En el caso de LVS se trata de un conjunto de herramientas SW que permiten implantar un sistema de equilibrado de carga. Actualmente LVS se subdivide en tres desarrollos:

- IPVS (*IP Virtual Server*): módulo Linux de reparto de carga IP (capas 3 y 4).
- KTCPVS (*Kernel TCP Virtual Server*): módulo Linux de reparto de carga a nivel de aplicación (capa 7).
- Diferentes herramientas para la gestión del *cluster*.

IPVS se implementa directamente sobre el núcleo de Linux. Acepta tanto tráfico TCP como UDP. Para repartir los diferentes paquetes entre los servidores hay diferentes estrategias: NAT, *Direct Routing* y vía túneles IP. En función de en que modo queramos operar hay diferentes requisitos para la red. Cada uno de ellos ofrece prestaciones dispares.

Por otro lado HAProxy es una única utilidad SW. Únicamente puede utilizarse en el caso de los servicios TCP, pero proporciona alta disponibilidad para los servidores que dan el servicio en cuestión y servicios de proxy para aplicaciones basadas en TCP. Puede funcionar tanto a nivel de nivel 4 (TCP) como de 7 (aplicación), permitiendo incluir *cookies* de sesión. En su funcionamiento típico, el cliente establece una conexión con el LB (petición) y el LB establece una conexión con el servidor que va a proporcionar el servicio deseado. Al establecerse esta serie de conexiones el paquete que llega al servidor no incluye la IP del cliente, puesto que ha sido el propio LB quien ha enviado el paquete; no obstante existe una opción, llamada *X-Forwarder-For* mediante la cual la dirección IP del cliente se incluye en la cabecera (esto es de utilidad para algunas aplicaciones).

No obstante este tipo de funcionamiento también cuenta con una parte positiva en cuanto a la configuración de red se refiere. Al hacerlo de esta manera los servidores no necesitan tener el LB configurado como *router* de salida, lo cual permite simplificar notablemente la complejidad de la red al poder separar el LB del *router* de la red.

En este proyecto se utilizará como LB HAProxy, que además tiene la ventaja de monitorizar el estado de salud de los servidores.

2.2. Alta disponibilidad

Como se ha descrito anteriormente en este trabajo lo que se va a implementar es un clúster de HA. Para ello es necesario saber el estado de todas las máquinas en todo momento.

Si ponemos el foco sobre los servidores, cuando se de un fallo en alguno de ellos es necesario sacarlo de la lista de servidores a los que se reparten peticiones. Como se ha dicho antes para ello es necesario saber el estado de los servidores. Para ello existen herramientas específicas. En HAProxy no es necesario puesto que el propio SW se encarga de ello, sin embargo en IPVS sí que es necesario puesto que no cuenta con esta funcionalidad.

Si nos centramos en el LB es necesaria la duplicidad del mismo, puesto que al ser un servicio crítico ha de poder estar disponible siempre, aunque tenga cualquier clase de fallo. Para ello se configura en modo activo-pasivo, monitorizando el LB que está activo para que en caso de fallo tome el control el pasivo. Tanto para HAProxy como para IPVS se requiere una herramienta adicional. Para Linux existen diferentes alternativas de código abierto: Keepalived [6], Corosync [7] y Pacemaker [8].

Corosync y Pacemaker se usan juntas. Pacemaker es un gestor de recursos, monitoriza todo lo que le pasa al clúster y actúa en consecuencia, en nuestro caso cambiaría el servicio activo del LB (ya sea HAProxy o IPVS) de un nodo a otro. Dicha acción se sustenta en otros servicios, herramientas de mensajería entre los nodos y detección del estado de los nodos del cluster (proporcionada por Corosync).

Otra posibilidad es Keepalived, que incluye en una única utilidad SW todas las herramientas descritas anteriormente. Su funcionamiento se basa en la implementación del protocolo VRRP [9], lo cual le permite alternar la posesión de una VIP (*Virtual IP*) entre los repartidores de carga. También permite monitorizar el estado de los servidores mediante diferentes métodos: comprobar si se puede realizar una conexión a un servidor con un puerto determinado, solicitar un recurso a un servidor y comprobar que la respuesta del recurso recibida es el correcta, y por último, y lo más interesante permite ejecutar un script externo para realizar las comprobaciones.

La opción elegida para asegurar HA en los LB es Keepalived, con el VRRP se monitorizará y se pivotará entre los dos LB para asegurar en todo momento que haya uno disponible.

2.3. Sistema de base de datos

Actualmente en el mundo de las bases de datos estructuradas existen dos productos que destacan sobre el resto; MSS [10] (*Microsoft Sql Server*), que desde hace poco cuenta con versión para Linux y MySQL [11]. A día de hoy el más usado de ambos es MySQL. Esto se debe a diversos factores, el primero de ellos el precio. MySQL es de código abierto y la mayoría de versiones son gratuitas, por contrario MSS es de código cerrado y la mayoría de sus versiones son de pago. En entornos de clúster donde podemos tener el sistema de BBDD instalado en una gran cantidad de máquinas este es un factor crítico, puesto que el impacto es mucho mayor.

Otro de los factores clave es el *tuning* que se le puede hacer a la BBDD para que se ajuste a nuestras necesidades. En este aspecto MySQL también resulta mejor opción, es mucho más configurable y se puede ahondar mucho más que MSS.

Por ello, para este trabajo se ha decidido utilizar MySQL. Una vez dicho esto hay que recalcar que uno de los objetivos del mismo era explorar los *clusters* de BBDD, por ello no es suficiente con decidir que vamos a utilizar MySQL, es necesario indagar en las diferentes formas que existen para que la herramienta se pueda comportar como un *cluster*.

A día de hoy hay dos opciones bien diferenciadas:

- Hacer una instalación local en cada máquina del clúster de un servidor de MySQL y conectarlas entre ellas con una herramienta externa que se encargue de que funcionen externamente como un único sistema en común.
- Instalar MySQL Cluster. Para lo que se hace una instalación del producto en cada máquina, y la misma aplicación es la que se encarga de la administración, coherencia y cohesión entre los diferentes nodos.

Para trabajar con la primera opción destacan dos herramientas: Percona XtraDB [12] y Galera [13]. Ahondando en este tipo de solución, lo que aportan las herramientas anteriormente mencionadas es una capa superior a la de BBDD para que se comuniquen entre ellas, se verifique el estado y se hagan todas las operaciones a nivel de administración del clúster que sean necesarias.

En el caso tanto de Galera como de Percona, se necesitan herramientas externas para realizar diferentes tareas de administración tales como recuperar

la información actual de una base de datos para un nodo que ha fallado. Para esto se pueden usar herramientas como mysqldump o Percona Xtrabackup.

En MySQL Cluster todo esto está incluido en la misma herramienta. Los diferentes nodos se comunican mediante TCP para intercambiar información de los mismos.

A parte de lo ya mencionado, Percona XtraDB y Galera son sistemas que están más enfocados hacia el *mirroring* de las BBDD de manera que todos los servidores de BBDD contengan la misma información replicada. MySQL Cluster puede configurarse también de ese modo o se puede configurar para que cada servidor almacene una parte de la información, de manera que el contenido de todos los servidores no sea completamente el mismo (aunque si guarde información redundante). En este sentido su comportamiento es similar al de un RAID 5 en *arrays* de discos duros.

Por lo tanto, la opción escogida para el sistema de BBDD ha sido MySQL Cluster.

Capítulo 3

Estructura del servidor

Para realizar el presente trabajo se ha implementado un prototipo basado en máquinas virtuales, concretamente se ha usado VirtualBox 5.0.2. La máquina anfitrión cuenta con las siguientes características:

- Procesador: Intel Core i5-430M (2,26 Ghz / 3 Mb. Caché)
- Memoria: 4 Gb. DDR3 a 1066 Mhz.
- Disco Duro: Samsung 850 EVO SSD (*Solid State Disk*) (250 GB, Serial ATA III, 540 MB/s)
- Sistema operativo Windows 7 Ultimate edition (64 bits).

La estructura del servidor se puede apreciar en la figura 3.1.

El sistema consta de siete máquinas en total. Cuatro de las cuales son los servidores que proveerán las páginas web solicitadas y también devolverán el resultado de las consultas de BBDD. En la misma red están ubicados los dos LB, en configuración activo-pasivo. También se ha configurado una máquina virtual que hará las veces de cliente del servidor de Internet configurado.

Aparte de las direcciones IP nativas de cada máquina existen dos direcciones VIP proporcionadas por Keepalived que responderán a cada uno de los tipos de peticiones antes mencionados, una para peticiones de páginas web y el otro para peticiones de BBDD. El reparto de direcciones IP se indica en el cuadro 3.1.

En todas las máquinas virtuales se ha instalado un sistema operativo Ubuntu Server 16.04.1 (64 bits). Se ha creado también una red NAT en VirtualBox a la que pertenecen las IPs de las diferentes máquinas. Dicha red NAT se ha

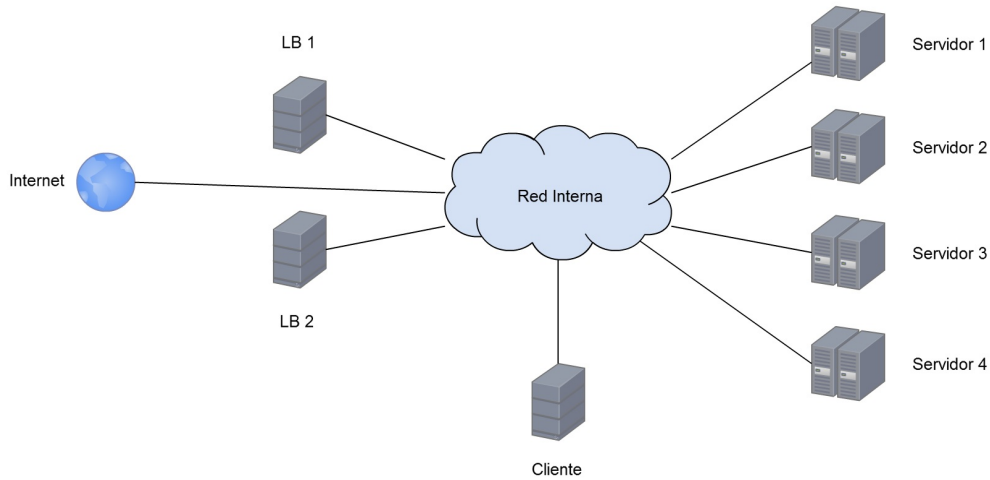


Figura 3.1: Estructura del servidor

Máquina	Alias	Dirección IP	
LB1	LB1	IP: 192.168.0.111	VIP1: 192.168.0.50 VIP2: 192.168.0.60
LB2	LB2	IP: 192.168.0.112	VIP1: 192.168.0.50 VIP2: 192.168.0.60
Cliente	Client	IP: 192.168.0.115	
Servidor 1	Mysql_1	IP: 192.168.0.101	
Servidor 2	Mysql_2	IP: 192.168.0.102	
Servidor 3	Mysql_3	IP: 192.168.0.103	
Servidor 4	Mysql_4	IP: 192.168.0.104	

Cuadro 3.1: Reparto de direcciones IP

llamado Cluster y tiene la dirección de red 192.168.0.0/24, y no cuenta con un servidor de DHCP.

3.1. Configuración de HAProxy

El paquete de HAProxy ha de estar instalado y corriendo en todo momento en ambos LB. Esto es debido a que si cambia el estado de uno de ellos y se convierte en *master*, el nodo tiene que estar ya preparado para poder comenzar a repartir las peticiones desde el primer momento. Cabe destacar que los pasos que se van a seguir en esta sección para configurar el HAProxy,

es necesario hacerlos tanto en LB1 como en LB2. En concreto en esta primera sección se va a configurar para el reparto web.

Se ha instalado la versión más actual del SW, concretamente se ha instalado la versión 1.6.3. Para ello es necesario ejecutar el siguiente comando:

```
root@LB1:~#apt-get install haproxy
```

Una vez instalado el paquete, hay que modificar el fichero de configuración para ajustarlo a nuestras necesidades, está ubicado en la siguiente ruta: `/etc/haproxy/haproxy.cfg`. Este es el fichero en cuestión (ya modificado):

```
global
    log /dev/log local0
    log /dev/log local1 notice
    chroot /var/lib/haproxy
    stats socket /run/haproxy/admin.sock mode 660 level admin
    stats timeout 30s
    user haproxy
    group haproxy
    daemon

defaults
    log global
    retries 3
    option redispatch
    timeout connect 50000
    timeout client 50000
    timeout server 50000
    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 apache
    mode http
    option httplog
    option httpclose
    option forwardfor
    balance roundrobin
```

```
bind 192.168.0.60:80
default_backend backend_apache

backend backend_apache
  mode http
  option tcp-check
  server mysql_client_01 192.168.0.101:80 check port 80
  server mysql_client_02 192.168.0.102:80 check port 80
  server mysql_client_03 192.168.0.103:80 check port 80
  server mysql_client_04 192.168.0.104:80 check port 80

listen stats 0.0.0.0:9000
  mode http
  stats show-desc Nodo LB1
  stats refresh 5s
  stats uri /haproxy?stats
  stats realm HAProxy\ Statistics
  stats auth admin:admin
  stats admin if TRUE
```

Como se puede observar en el fichero de configuración está dividido en diferentes secciones, son las siguientes:

- **Global:** en esta sección lo que se configura son parámetros en torno al proceso de HAProxy: cómo se ejecuta, con qué usuario y grupo lo hace, dónde se vierten los logs, también se configura un socket para hacer consultas al mismo de las estadísticas de la aplicación, una orden `chroot` para separar al HAProxy de otras áreas del sistema, y finalmente se indica que se ejecutará como un demonio.
- **Defaults:** dónde se configuran diferentes parámetros por defecto. Por ejemplo, el número de intentos de conexión antes de considerar un servidor caído (tres en nuestro caso), qué hacer con las peticiones dirigidas a un servidor caído, en nuestro caso se pasan a otro servidor que esté operativo, los *timeouts* por inactividad para conexión con los servidores y de actividad de cliente y servidores, y las páginas a devolver en caso de diferentes tipos de error.
- **Frontend:** aquí se define la dirección y puerto que atenderá HAProxy, así como algunas opciones para la recepción de conexiones. En este caso está escuchando en la dirección asignada para el tráfico web: 192.168.0.60, en el puerto 80. Las peticiones se reparten en modo http

y las máquinas a las que reparte son las asociadas al `backend_apache`. Se configuran las opciones:

- *httplog*: es el tipo de log (formato) en el que escribe la información, este en concreto es el recomendado por el fabricante para proxies http.
 - *httpclose*: con esta opción el HAProxy cierra todas las conexiones http una vez resueltas, aunque sean persistentes.
 - *forwardfor*: por esta opción se incluye la dirección del cliente en la cabecera http que se envía a los clientes.
- Backend: aquí se definen los grupos de servidores a los que los *frontends* envían las peticiones. Normalmente se crea uno por servicio. Para las consultas a BBDD se creará otro que estará compuesto por los mismos servidores pero diferente puerto, dado que MySQL escucha en el puerto 3306. Como se observa está habilitada la opción `tcp-check` configurando la comprobación para cada servidor en el puerto 80. Lo que hace es enviar un mensaje tcp para comprobar si el servidor responde correctamente. De este modo detecta si está activo, si no lo está no le enviará peticiones.

3.2. Configuración de Keepalived

Al igual que ocurre con el HAProxy, ambos LBs tienen que tener Keepalived instalado, pero el fichero de configuración difiere ligeramente el uno del otro. Se les ha puesto la última versión disponible, la 1.2.23. Se ha ejecutado el siguiente comando:

```
root@LB1:~#apt-get install keepalived
```

El fichero de configuración está ubicado en `/etc/keepalived/keepalived.cfg`. Tiene el siguiente aspecto una vez modificado para las necesidades del trabajo del LB1:

```
vrrp_script chk_haproxy {
    script "pidof haproxy"
    interval 2
}
```

```
vrrp_instance VI_E1 {
    state MASTER
    interface eth0
    virtual_router_id 51
    priority 150
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass superpass
    }
    virtual_ipaddress {
        192.168.0.60
    }
    track_script {
        chk_haproxy
    }
}
```

El fichero de configuración del LB2 es el siguiente:

```
vrrp_script chk_haproxy {
    script "pidof haproxy"
    interval 2
}
```

```
vrrp_instance VI_E1 {
    state BACKUP
    interface eth0
    virtual_router_id 51
    priority 100
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass superpass
    }
    virtual_ipaddress {
        192.168.0.60
    }
    track_script {
        chk_haproxy
    }
}
```

Se configura una VIP: 192.168.0.60. Esta irá cambiando entre los LBs en función de cual esté activo. Por defecto, la VIP la tendrá el LB1 por ser MASTER. En caso de fallo la tendrá el LB2, y si los dos están activos la tendrá el LB1 al contar con mayor prioridad. El cambio de VIP puede ser producido por diferentes factores, un fallo en la red, en el servidor, etc. Aparte de los fallos nombrados anteriormente también se configura un script para que si el HAProxy falla por cualquier circunstancia el nodo pierda la VIP. El mecanismo de dicho script consiste en ejecutar una orden que le permite averiguar si el HAProxy tiene un PID (`pidof`), si no lo tiene (es que no existe un proceso HAProxy) perderá la VIP.

Es necesario indicar que para que los nodos puedan compartir la VIP, es necesario dar permisos para que los procesos se puedan asociar a direcciones IPs no locales, para ello hay que modificar una línea del fichero `/etc/sysctl.conf` de la siguiente forma en ambos LBs:

```
net.ipv4.ip_nonlocal_bind=1.
```

Esto nos sirve para el próximo arranque de la máquina, si queremos que lo aplique desde este momento, es necesario ejecutar el siguiente comando:

```
root@LB1:~#sysctl -p
```

3.3. Configuración de los nodos servidores

Para completar el escenario de reparto web es necesario que en todos los nodos servidores se instalen y configuren un par de paquetes. El primero de ellos es Apache, un servidor http de código abierto, a día de hoy es uno de los más usados en todo el mundo, será el encargado de que nuestros servidores gestionen y devuelvan las páginas web. Para instalar el paquete hay que ejecutar el siguiente comando:

```
root@Mysql\_1:~#apt-get install apache2
```

Una vez instalado, todas las páginas que queramos que sirva deben de estar ubicadas en la siguiente ruta: `/var/www/html/`. Esta ruta estará habitualmente en un almacenamiento compartido, sin embargo, al tratarse de un clúster experimental, cada servidor la tiene localmente.

Para que en los logs del servidor se vea reflejada la IP del cliente y no la del LB, aparte de configurar el LB para que en las cabeceras http incluya la IP del cliente, como se ha visto anteriormente, hay que modificar el fichero de

Capítulo 3. Estructura del servidor

configuración y activar el modo *X-Forwarded-For* de apache. El fichero está en la ruta `/etc/apache2/apache2.conf/`, y hay que añadir las siguientes líneas:

```
RemoteIPHeader X-Forwarded-For
RemoteIPInternalProxy 192.168.0.60/32
```

Además, para activar el modo *X-Forwarded-For* hay que incluir el módulo que se encarga de la funcionalidad de dicho modo en la carpeta de módulos habilitados. Para ello hay que ejecutar los siguientes comandos:

```
root@Mysql\_1:~# cd /etc/apache2/mods-enabled
root@Mysql\_1:/etc/apache2/mods-enabled# ln -s
/etc/apache2/mods-available/remoteip.load remoteip.load
root@Mysql\_1:/etc/apache2/mods-enabled# service apache2 restart
```

Capítulo 4

Estructura de BBDD

Como se ha dicho antes, el sistema de BBDD elegido ha sido MySQL Cluster. Si lo considerásemos como una caja negra, su comportamiento sería idéntico al de una versión estándar de MySQL (con determinadas configuraciones incluso más rápido), con la misma manera de realizar consultas, misma manera de acceder a la BBDD desde la aplicación, etc. En la figura 4.1 se aprecia muy bien el flujo de datos que tendría una transacción tipo en un sistema que incluyese esta tecnología.

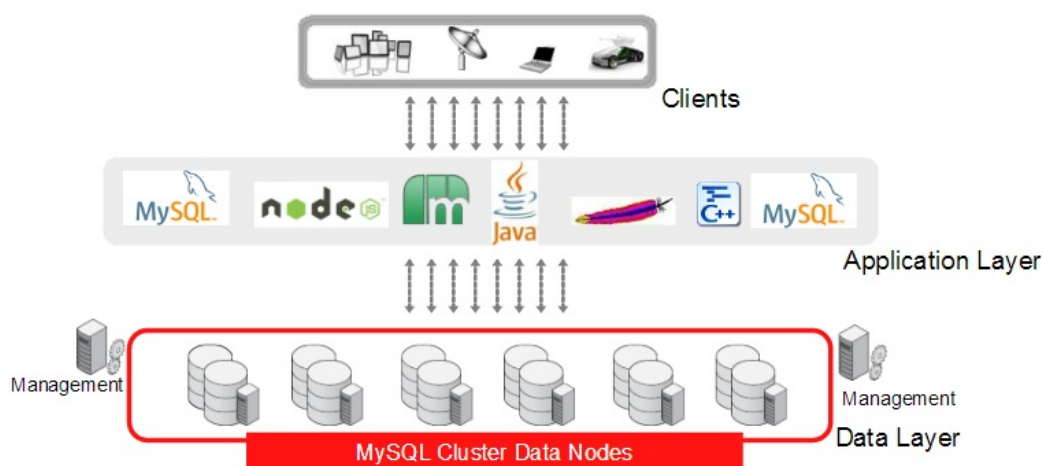


Figura 4.1: Estructura de MySQL Cluster

MySQL Cluster es una versión de alta disponibilidad de MySQL adaptada para entornos de *clustering*. Consiste en un conjunto de máquinas ejecutando diferentes tipos de procesos de MySQL Cluster. Permite redundancia y usa el motor NDB Cluster. Está pensada para grandes volúmenes de datos, para

entornos donde la escalabilidad sea un factor crítico y para situaciones en las que el sistema tenga que estar operativo siempre. Según sus diseñadores en un sistema bien administrado se pueden alcanzar unas tasas del 99.999 % de disponibilidad.

La arquitectura del sistema se define por nodos; realmente un nodo no es una máquina física como tal, en el contexto de Mysql Cluster hace referencia a un proceso. Existen tres tipos diferentes de nodos (en una arquitectura mínima tendría que haber al menos un nodo de cada tipo):

- **Nodo de administración, MGM (*Management*):** su función es administrar el resto de nodos, proporcionar y administrar la configuración del *cluster*, controlar el inicio o suspensión de los nodos, etc. Desde este nodo se pueden lanzar *backups*, es el único punto desde donde se pueden realizar. Al tener almacenados los los ficheros de configuración referentes a la topología del *cluster*, ha de ser el primer nodo en arrancarse.
- **Nodo de datos, NDB (*Node Data Base*):** son los nodos encargados de almacenar la información. Los datos no tienen porque estar en un único NDB, pueden estar repartidos e incluso un dato puede estar repetido en varios de ellos. Es configurable, depende de la topología y las características del clúster que se quiera desarrollar.
- **Nodo SQL:** la finalidad de este tipo de nodo es acceder a los datos, realmente es un servidor MySQL tradicional que usa el motor NDB Cluster.

Aparte de los nodos también existen clientes de administración, que pueden lanzar comandos sobre el MGM con el fin de administrar el *cluster*. El funcionamiento es igual que cuando en un MySQL estándar una aplicación lanza una consulta a la BBDD. En este caso ese tipo de consultas irán dirigidas al NDB, mientras que las que tengan que ver con la administración del clúster irán al MGM.

En la figura 4.2 se ilustra como interaccionan cada uno de los componentes del clúster entre si.

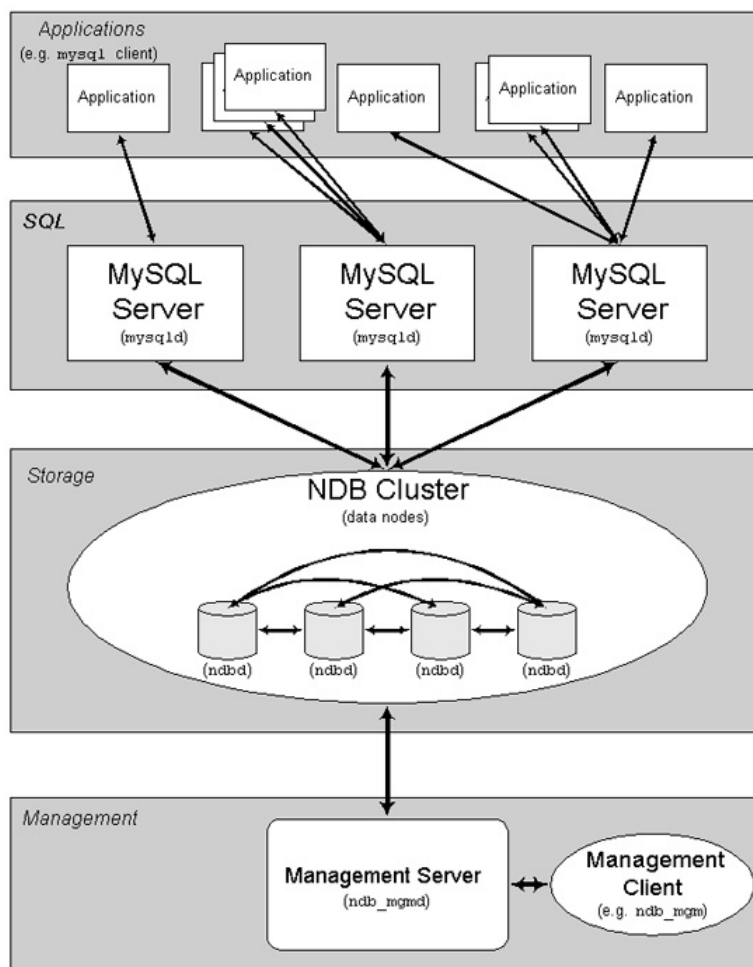


Figura 4.2: Relación de los componentes de un MySQL Cluster

Otro aspecto importante es cómo se puede almacenar la información. Anteriormente se ha mencionado que pueden haber determinados datos repetidos o incluso que la información puede estar distribuida entre diferentes nodos NDB. Esto es debido a que MySQL Cluster cuenta con réplicas y particiones. Esto está relacionado con el número de nodos que tengamos en nuestro clúster y los grupos que formen estos.

Una partición hace referencia a una sección de la información que el clúster almacena. Como mínimo existen tantas particiones como nodos que participen en el clúster, cada nodo debe de almacenar al menos una replica de su partición, aunque normalmente suelen almacenar más de una partición. Dichas particiones que no son las propias del nodo son las que se usarán para seguir dando servicio cuando un nodo falle. Aparte de las propias por número

de nodos, un usuario puede crear particiones de tablas. Estas particiones se crean subdividiendo por rangos la PK (*Primary Key*) de una tabla.

El número de réplicas es el número de veces que una determinada tabla se encuentra almacenada en el sistema. El tamaño de los grupos de nodos no es configurable, viene dado por la siguiente expresión:

$$\text{número de grupos de nodos} = \frac{\text{número de nodos NDB}}{\text{número de réplicas}}$$

Todos los nodos que están en un mismo grupo almacenan la misma información. Para asegurar la mayor disponibilidad de los nodos hay que procurar no tener dos procesos de un mismo grupo de nodos en una misma máquina física. En un clúster con cuatro nodos NDB, y dos réplicas habrán dos grupos de nodos, de manera que podrían llegar a fallar hasta un nodo de cada grupo y el sistema seguiría funcionando con normalidad. En la imagen 4.3 se muestran las diferentes interacciones que tendrían lugar en función de qué nodos fallasen en un escenario como el planteado.

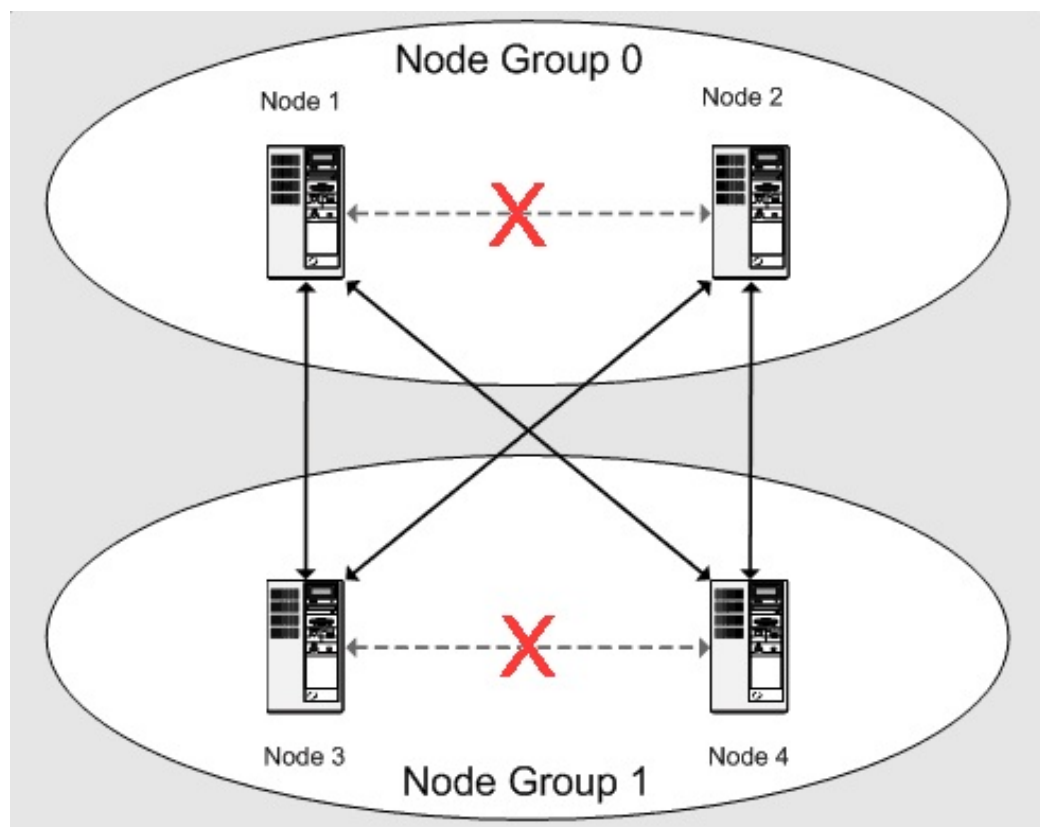


Figura 4.3: Interacciones en casos de caída en un sistema con cuatro nodos NDB y dos réplicas

MySQL Cluster también permite la replicación de un clúster completo, cosa que resulta muy útil en entornos reales de HA donde se suelen varios CPD (*Centro de Proceso de Datos*), separados físicamente entre sí para asegurar el servicio en caso de catástrofe natural, incendios que inhabiliten el CPD principal, etc.

Esta opción se llama *Cluster replication*, se trata de una sincronización asíncrona, de manera que en el peor de los casos únicamente se podrían perder los cambios que han tenido lugar desde la última sincronización entre los dos clusters. Para ello, ambos MySQL Cluster han de ser completamente idénticos, con la excepción de que el nodo maestro (MGM) en el clúster de *backup* ha de estar configurado como *slave* del MGM del clúster principal. En la imagen 4.4 se aprecia la configuración necesaria para un clúster con estas condiciones.

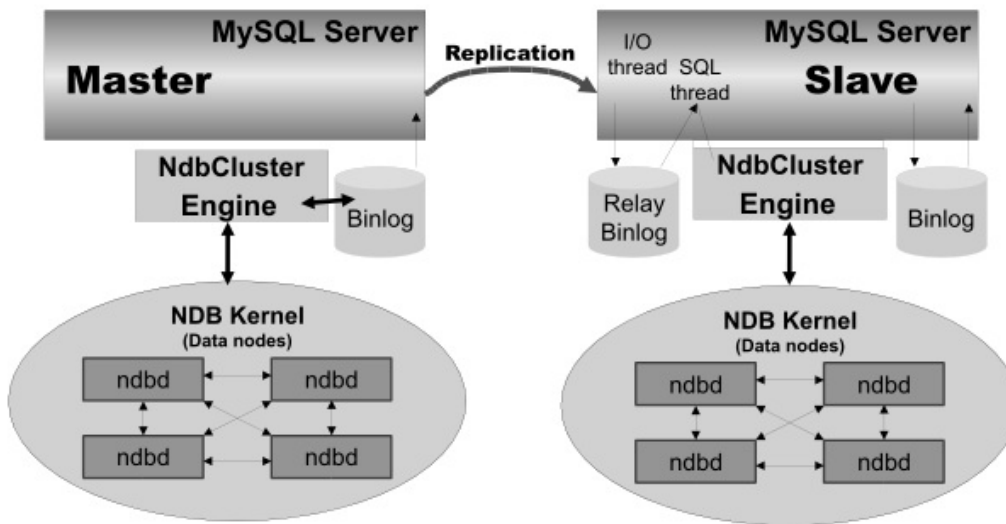


Figura 4.4: Modelo de MySQL Cluster con la opción *cluster replication*

4.1. Configuración de MySQL Cluster

En este trabajo se ha desarrollado un clúster de seis nodos ubicados en cuatro nodos (los servidores del clúster). Concretamente se han utilizado las máquinas Mysql_1, Mysql_2, Mysql_3 y Mysql_4. En la topología hay dos nodos MGM (uno activo y otro de *backup*), dos nodo NDB y dos nodos SQL. La distribución en las máquinas es la indicada en el cuadro 4.1.

Máquina	Alias	Dirección IP	Nodo MySql
LB1	LB1	IP: 192.168.0.111	No
LB2	LB2	IP: 192.168.0.112	No
Cliente	Client	IP: 192.168.0.115	No
Servidor 1	Mysql_1	IP: 192.168.0.101	MGM
Servidor 2	Mysql_2	IP: 192.168.0.102	NDB, SQL
Servidor 3	Mysql_3	IP: 192.168.0.103	NDB, SQL
Servidor 4	Mysql_4	IP: 192.168.0.104	MGM <i>backup</i>

Cuadro 4.1: Distribución de las máquinas

En la figura 4.5 se puede apreciar el ciclo de vida de una petición en el sistema. El Client lanzará la petición, la repartirá el LB1 o LB2 al SQL del Mysql_2 o Mysql_3. El SQL en cuestión puede enviar la consulta al NDB que está en su misma máquina física o al NDB ubicado en la otra máquina. Más tarde se devolverá el resultado de la consulta al LB que haya pasado la petición y este la mandará de vuelta al Client.

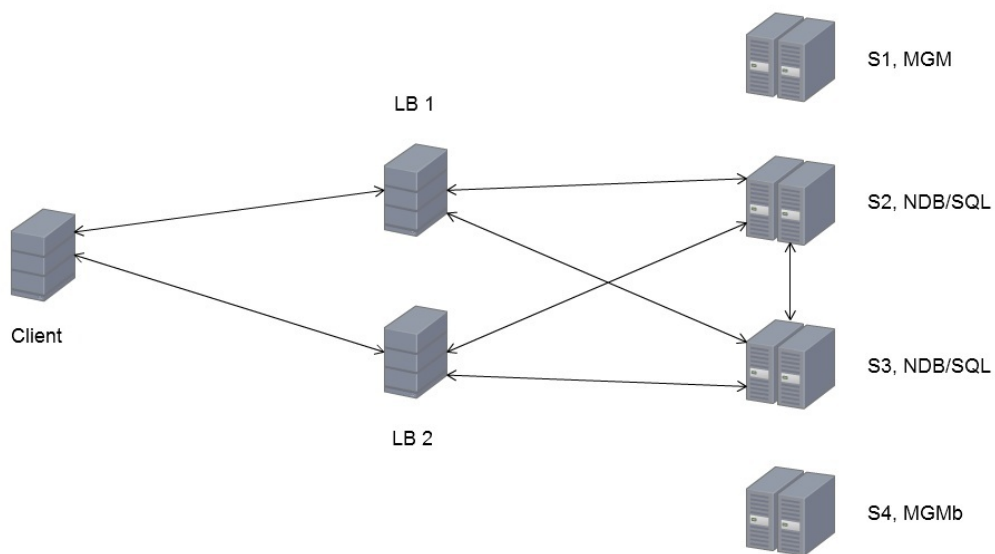


Figura 4.5: Flujo en el sistema de una petición para MySQL Cluster

Para instalar MySQL Cluster en los nodos hay que descargarlo de la página web de Mysql [14], puesto que no hay ningún repositorio desde donde se pueda instalar directamente. También es necesario instalar el paquete *libaio1* puesto que es una dependencia del SW. Para realizar ambas cosas hay que

4.1. Configuración de MySQL Cluster

ejecutar los siguientes comandos (en todos los nodos donde se quiera tener MySQL Cluster, Mysql_1, Mysql_2, Mysql_3 y Mysql_4):

```
root@Mysql\1:~# wget dev.mysql.com/get/Downloads/MySQL-Cluster-7.4/
mysql-cluster-gpl-7.4.12-debian7-x86_64.deb
root@Mysql\1:~# apt-get install libaio1
root@Mysql\1:~# dpkg -i mysql-cluster-gpl-7.4.12-debian7-x86_64.deb
```

Una vez instalados ambos paquetes, tendremos en `/opt/mysql/server-5.6/` el directorio de la instalación. A continuación se ha de crear el fichero donde se indica la estructura que va a tener nuestro clúster, las rutas donde van a depositar los logs los nodos MGM y la ruta donde van a almacenar los ficheros de datos los nodos NDB. Este fichero únicamente lo han de tener los nodos MGM, puesto que el resto de nodos utilizarán la configuración que les indique el MGM activo en ese momento. El fichero de configuración está ubicado en la siguiente ruta `/var/lib/mysql-cluster/config.ini`, y este es su contenido:

```
[ndb_mgmd]
hostname=Mysql_1                # Hostname del MGM
datadir=/var/lib/mysql-cluster # Directorio de los logs

[ndb_mgmd]
hostname=Mysql_4                # Hostname del MGMB
datadir=/var/lib/mysql-cluster # Directorio de los logs

[ndbd]
hostname=Mysql_2                # Hostname del NDB
datadir=/usr/local/mysql/data   # Ruta de los ficheros de datos

[ndbd]
hostname=Mysql_3                # Hostname del NDB
datadir=/usr/local/mysql/data   # Ruta de los ficheros de datos

[mysqld]
hostname=Mysql_2                # Hostname del SQL

[mysqld]
hostname=Mysql_3                # Hostname del SQL
```

En los servidores Mysql_2 y Mysql_3 (son nodos SQL y NDB) también es necesario añadir unas líneas al fichero `/etc/my.cnf` para indicar al nodo

NDB que tiene que usar el NDB Cluster como motor de las BBDD y para los SQL hay que configurar el *hostname* del MGM. Este es el contenido del fichero:

```
[mysql_cluster]
ndb-connectstring=Mysql_1
```

```
[mysqld]
ndbcluster
```

También es necesario crear el directorio donde se van a almacenar los ficheros de datos, porque por defecto no lo crea. Para ello hay que ejecutar el siguiente comando:

```
root@Mysql\_2:~# mkdir -p /usr/local/mysql/data
```

De la misma manera es necesario configurar el arranque automático de los procesos de MySQL para que cuando el servidor se inicie, los procesos se ejecuten automáticamente. Para ello hay que poner la orden correspondiente en el fichero */etc/rc.local* y activar para *systemctl* el servicio *rc-local* con el siguiente comando:

```
root@Mysql\_2:~# systemctl enable rc-local.service
```

Para los nodos MGM la orden con la que se ejecuta el proceso (y que habrá que añadir al servicio *rc-local*) es la siguiente:

```
root@Mysql\_1:~# /opt/mysql/server-5.6/bin/ndb_mgmd -f
/var/lib/mysql-cluster/config.ini
```

Como se puede apreciar se le pasa el fichero de configuración del clúster que hemos creado anteriormente. Para los nodos NDB esta es la orden que ejecuta el proceso:

```
root@Mysql\_1:~# /opt/mysql/server-5.6/bin/ndbd
```

En cuanto a los nodos SQL hay que crear un usuario y grupo que sea *mysql*, una vez hecho esto hay que crear la base de datos por defecto con el usuario previamente configurado:

```
root@Mysql\1:~# groupadd mysql
root@Mysql\1:~# useradd -r -g mysql -s /bin/false mysql
root@Mysql\1:~# /opt/mysql/server-5.6/scripts/
mysql_install_db --user=mysql
```

En el caso de los nodos SQL, vamos a configurar los procesos como servicios. Para ello hay que copiar el script de arranque a `/etc/init.d/`, habilitar el servicio para que se arranque cuando se inicie la máquina y arrancarlo:

```
root@Mysql\1:~# cp /opt/mysql/server-5.6/support-files/mysql.server
/etc/init.d/mysqld
root@Mysql\1:~# systemctl enable mysqld.service
root@Mysql\1:~# systemctl start mysqld
```

Si quisiéramos interactuar con el MySQL mediante consola para lanzar consultas o administrar las BBDD (tal y como se haría en una instalación estándar) habría que ejecutar el siguiente proceso:

```
root@Mysql\1:~#/opt/mysql/server-5.6/bin/mysql
```

4.2. Cambios en la configuración de Keepalived

Para este tipo nuevo de tráfico se va a crear una VIP adicional, de manera que el tráfico de un tipo y el de otro esté separado en dos VIPs diferentes. Para ello hay que añadir la dirección que queremos a la sección `virtual_ipaddress`. La VIP escogida es la 192.168.0.50. Así es como queda la sección modificada en el fichero de configuración:

```
virtual_ipaddress {
    192.168.0.50
    192.168.0.60
}
```

4.3. Cambios en la configuración de HAProxy

Para que los LBs repartan también el tráfico que queremos que llegue al MySQL Cluster hay que modificar el fichero de configuración de HAProxy.

Concretamente lo que hay que añadir es un *frontend* que escuche en la VIP nueva que se ha configurado en Keepalived para este tipo nuevo de tráfico. También es necesario configurar un *backend* para que reparta las peticiones. Para el tráfico correspondiente a BBDD los únicos servidores que podrán recibir peticiones son aquellos que estén configurados como nodos SQL (Mysql_2 y Mysql_3).

También hay que cambiar el puerto dónde se escucha (en este caso se pondrá el 3306 porque es donde escucha MySQL por defecto) y el script de comprobación puesto que en el puerto 80 contestaría el servidor Apache que hay instalado en las máquinas.

Por último, hay que cambiar el tipo de script que realiza las comprobaciones para ver si los servidores donde tiene que mandar las peticiones están activos. Se ha de configurar la comprobación del tipo *mysql-check*. Para realizar dicha comprobación hay que crear un usuario en la base de datos, dicho usuario es con el que se lanzan las comprobaciones. Para crear el usuario hay que ejecutar el siguiente comando en Mysql_2:

```
root@Mysql\2:~# /opt/mysql/server-5.6/bin/mysql -u root -p -e
'CREATE USER haproxy_check; FLUSH PRIVILEGES'
root@Mysql\2:~# /opt/mysql/server-5.6/bin/mysql -u root -p -h
Mysql\3 -e 'CREATE USER haproxy_check; FLUSH PRIVILEGES'
```

La segunda orden que se ejecuta es para añadir los permisos en Mysql_3, debido a que los permisos es lo único que MySQL Cluster no replica entre los NDB de manera automática.

Seguidamente, se muestran las modificaciones en el fichero de configuración de HAProxy:

```
frontend mysql
  mode tcp
  balance roundrobin
  bind 192.168.0.50:3306
  default_backend backend_mysql

backend backend_mysql
  option mysql-check user haproxy_check
  server mysql_client_01 192.168.0.102:3306 check
  server mysql_client_02 192.168.0.103:3306 check
```

Capítulo 5

Verificación del servidor web

En este apartado se va a poner a prueba el sistema para comprobar que responde de la manera esperada. Para ello se ha creado una página en php que devuelve la IP del servidor que ha sido el responsable de devolver dicha página, la IP del LB que le ha enviado la petición al servidor y la IP del cliente. Este es el código de la página, cuyo nombre es index.php:

```
<?php
echo "Servidor Web<br>";
echo "Server IP: ". $_SERVER['SERVER_ADDR'] . "<br>";
echo "LB 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>";
    }
?>
```

Como se puede apreciar en el código, para mostrar la IP del LB que ha gestionado la petición, es necesario extraer dicha dirección de quién está configurado como emisor del paquete. Como se ha explicado anteriormente esto es debido al comportamiento del HAProxy, no obstante gracias al *X-Forwarded-For* se incluye la dirección del cliente que realmente realiza la petición en la cabecera http.

Para poder visualizar dicha página en nuestro cliente va a ser necesario instalar varios paquetes en el cliente y en los servidores. El primero de ellos se

va a instalar en el cliente, se trata de un navegador de línea de comandos que puede interpretar código en php, se ha elegido *w3m*. También es necesario instalar varios paquetes en los servidores para que el código php se ejecute correctamente. Para instalar el navegador en el cliente:

```
root@Mysql\_1:~# apt-get install w3m
```

Y el comando para instalar los paquetes de php en los servidores (hay que instalarlos en todos los servidores):

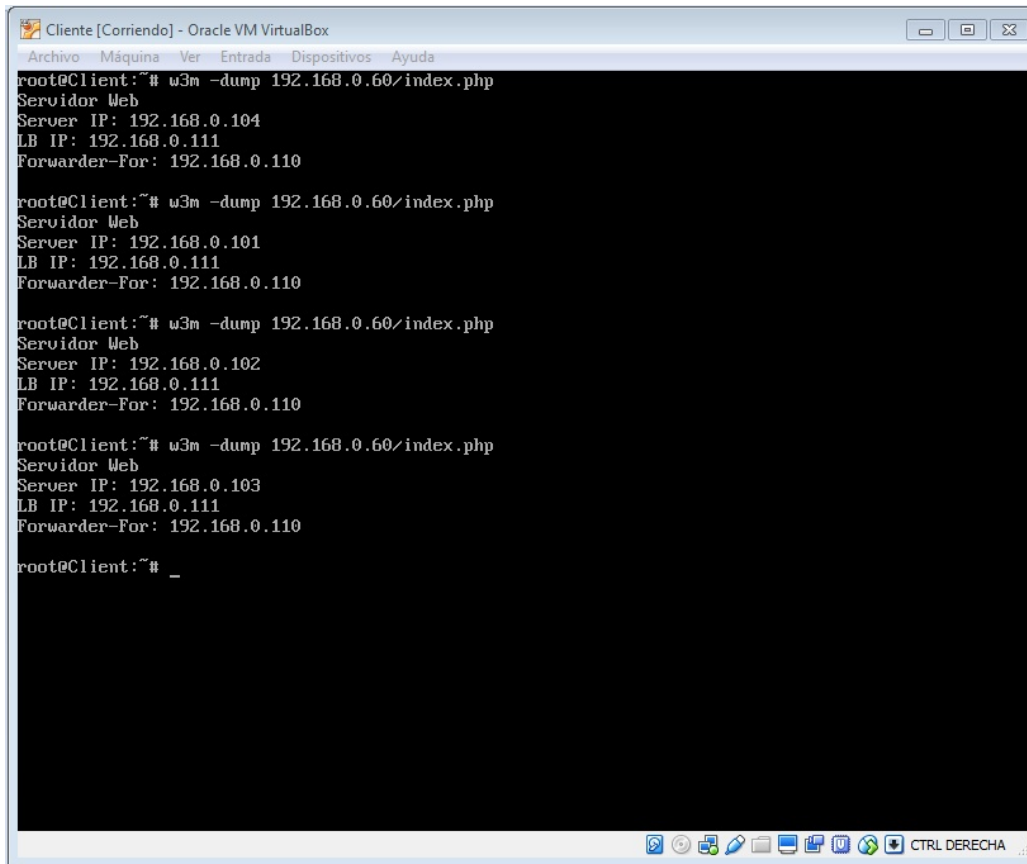
```
root@Mysql\_1:~# apt-get install php php-common libapache2-mod-php  
php-cli
```

5.1. Todos los elementos activos

En un escenario con todos los nodos activos, si realizamos cuatro peticiones deberían de ser atendidas una por cada nodo servidor. Esto es debido a que el reparto en el HAProxy está configurado con el algoritmo *roundrobin*. Todas estas peticiones deberían de ser distribuidas entre los diferentes servidores por el LB1 puesto que es el *master* en la configuración de Keepalived y éste será quien disponga inicialmente de la VIP. Para ello se ha de ejecutar la siguiente orden en el cliente cuatro veces:

```
root@Client:~#w3m -dump 192.168.0.60/index.php
```

En la figura 5.1 se ve el resultado de la ejecución. Como se puede apreciar el comportamiento es el deseado puesto que cada petición es devuelta por un servidor diferente y en todos los casos la IP asociada al LB es la misma.



```
Cliente [Corriendo] - Oracle VM VirtualBox
Archivo  Máquina  Ver  Entrada  Dispositivos  Ayuda
root@Client:~# w3m -dump 192.168.0.60/index.php
Servidor Web
Server IP: 192.168.0.104
LB IP: 192.168.0.111
Forwarder-For: 192.168.0.110

root@Client:~# w3m -dump 192.168.0.60/index.php
Servidor Web
Server IP: 192.168.0.101
LB IP: 192.168.0.111
Forwarder-For: 192.168.0.110

root@Client:~# w3m -dump 192.168.0.60/index.php
Servidor Web
Server IP: 192.168.0.102
LB IP: 192.168.0.111
Forwarder-For: 192.168.0.110

root@Client:~# w3m -dump 192.168.0.60/index.php
Servidor Web
Server IP: 192.168.0.103
LB IP: 192.168.0.111
Forwarder-For: 192.168.0.110

root@Client:~# _
```

Figura 5.1: Resultado de consultar la página index.php

5.2. Fallo de servidores web

Para este escenario se parará el servicio de apache en las máquinas Mysql_1 y Mysql_2, con la finalidad de recrear un problema por el cual dichas máquinas no están disponibles para recibir peticiones. Para ello hay que ejecutar la siguiente orden en ambos servidores:

```
root@Mysql\1:~# service apache2 stop
```

En este caso el comportamiento deseado al lanzar cuatro peticiones (mismo número de peticiones que en la anterior sección) es que dos de esas peticiones sean atendidas por Mysql_3 y las otras dos sean atendidas por Mysql_4. En la figura 5.2 se puede observar el resultado.

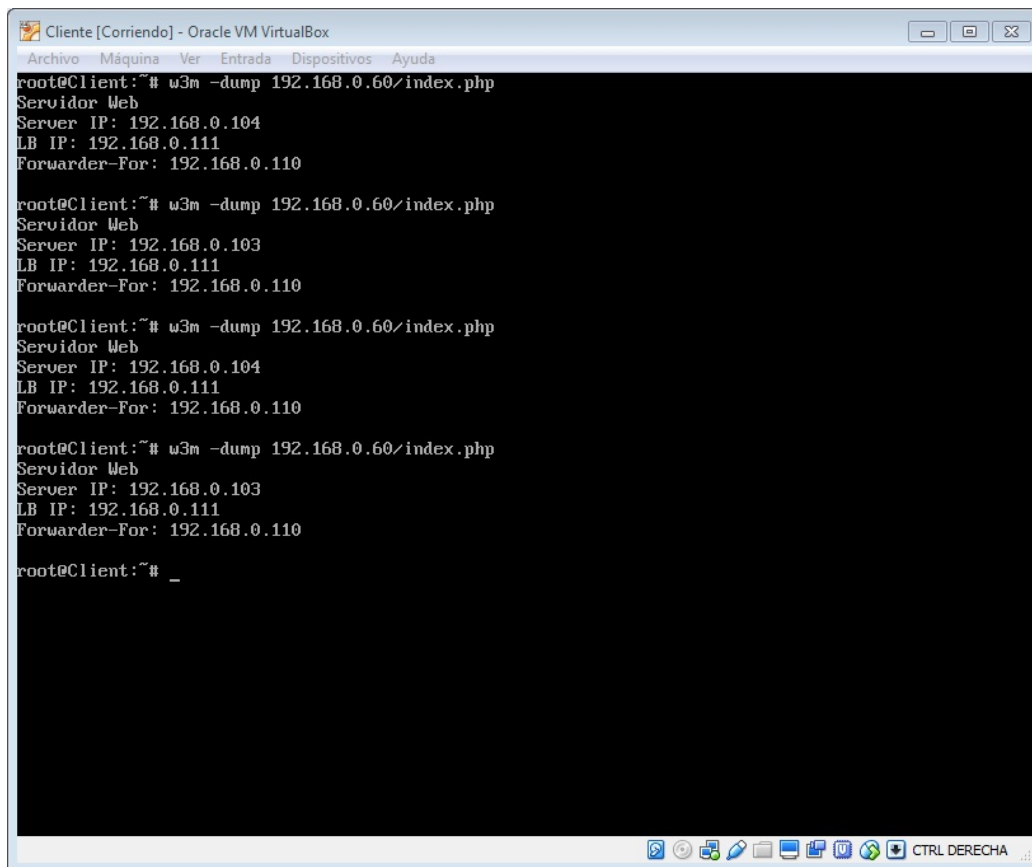


Figura 5.2: Resultado de consultar la página index.php con dos nodos fuera de servicio

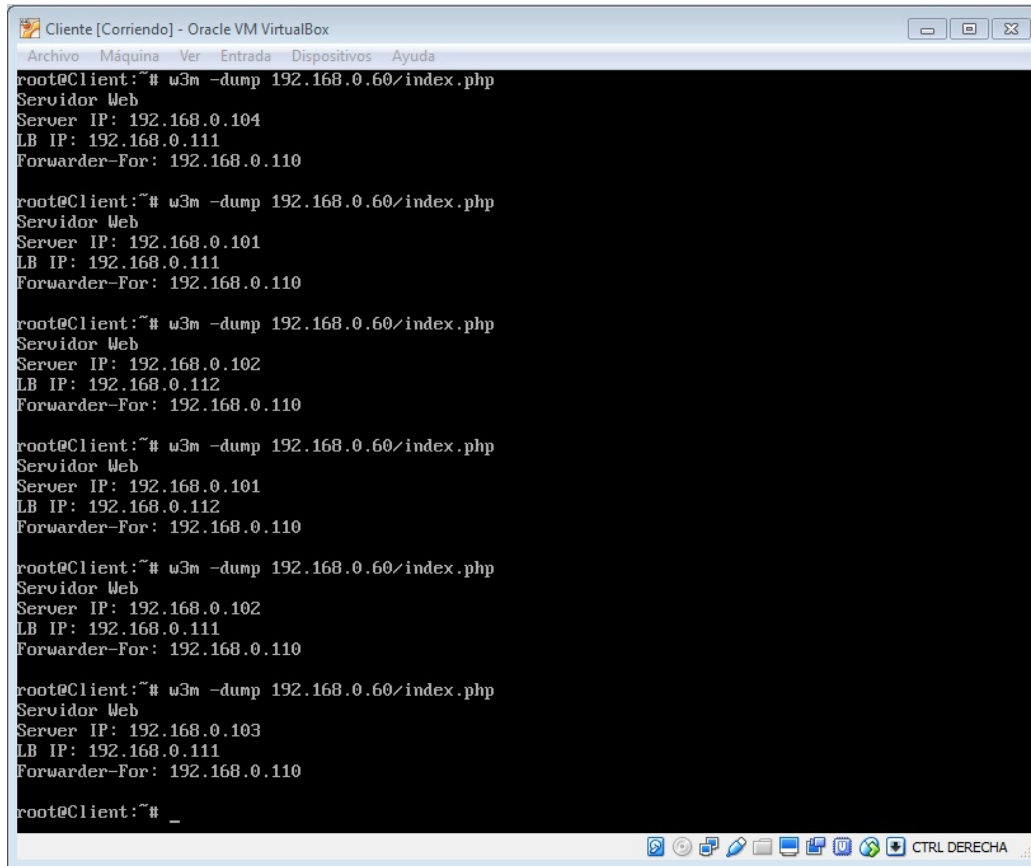
El comportamiento es el deseado, se da la situación antes descrita, dos peticiones tratadas por el Mysql_3 y dos peticiones tratadas por el Mysql_4.

5.3. Fallo de un LB

En este caso se va a parar el servicio de Keepalived de LB1, simulando un fallo en la máquina. Lo que debería suceder es que las VIPs que proporciona el Keepalived (192.168.0.50 y 192.168.0.60) han de migrar del LB1 al LB2. Y a continuación se volverá a habilitar el servicio de manera que las VIPs volverán al LB1.

Para esta prueba se van a lanzar seis peticiones. Se realizarán dos peticiones que serán gestionadas por el LB1, se parará el servicio y se realizarán otras dos peticiones, que deberían ser atendidas por el LB2. Posteriormente se

volverá a habilitar el servicio y se lanzarán las dos últimas peticiones, que deberían de ser tratadas por el LB1.



```
Ciente [Corriendo] - Oracle VM VirtualBox
Archivo  Máquina  Ver  Entrada  Dispositivos  Ayuda
root@Client:~# w3m -dump 192.168.0.60/index.php
Servidor Web
Server IP: 192.168.0.104
LB IP: 192.168.0.111
Forwarder-For: 192.168.0.110

root@Client:~# w3m -dump 192.168.0.60/index.php
Servidor Web
Server IP: 192.168.0.101
LB IP: 192.168.0.111
Forwarder-For: 192.168.0.110

root@Client:~# w3m -dump 192.168.0.60/index.php
Servidor Web
Server IP: 192.168.0.102
LB IP: 192.168.0.112
Forwarder-For: 192.168.0.110

root@Client:~# w3m -dump 192.168.0.60/index.php
Servidor Web
Server IP: 192.168.0.101
LB IP: 192.168.0.112
Forwarder-For: 192.168.0.110

root@Client:~# w3m -dump 192.168.0.60/index.php
Servidor Web
Server IP: 192.168.0.102
LB IP: 192.168.0.111
Forwarder-For: 192.168.0.110

root@Client:~# w3m -dump 192.168.0.60/index.php
Servidor Web
Server IP: 192.168.0.103
LB IP: 192.168.0.111
Forwarder-For: 192.168.0.110

root@Client:~# _
```

Figura 5.3: Resultado de consultar la página index.php con un fallo en el LB1

En la figura 5.3 se ve claramente como cada dos peticiones cambia el LB que atiende las peticiones.

Capítulo 6

Verificación del servidor de BBDD

En este capítulo se va a verificar el comportamiento del clúster de MySQL de manera similar a como se ha realizado anteriormente la comprobación para el servicio web. Se verificará mediante peticiones realizadas en el cliente que repartirá el LB y el clúster de MySQL responderá a la petición. Concretamente se ejecutará una consulta muy sencilla: «SHOW DATABASES», la cual nos mostrará las BBDD que hay creadas en ese momento.

Para ello es necesario instalar tres clientes de MySQL, el primero en el nodo cliente y los otros dos restantes en los LBs, puesto que sin un cliente MySQL no reparten peticiones de este tipo. Para instalar los clientes es necesario ejecutar la siguiente orden en los tres nodos ya nombrados:

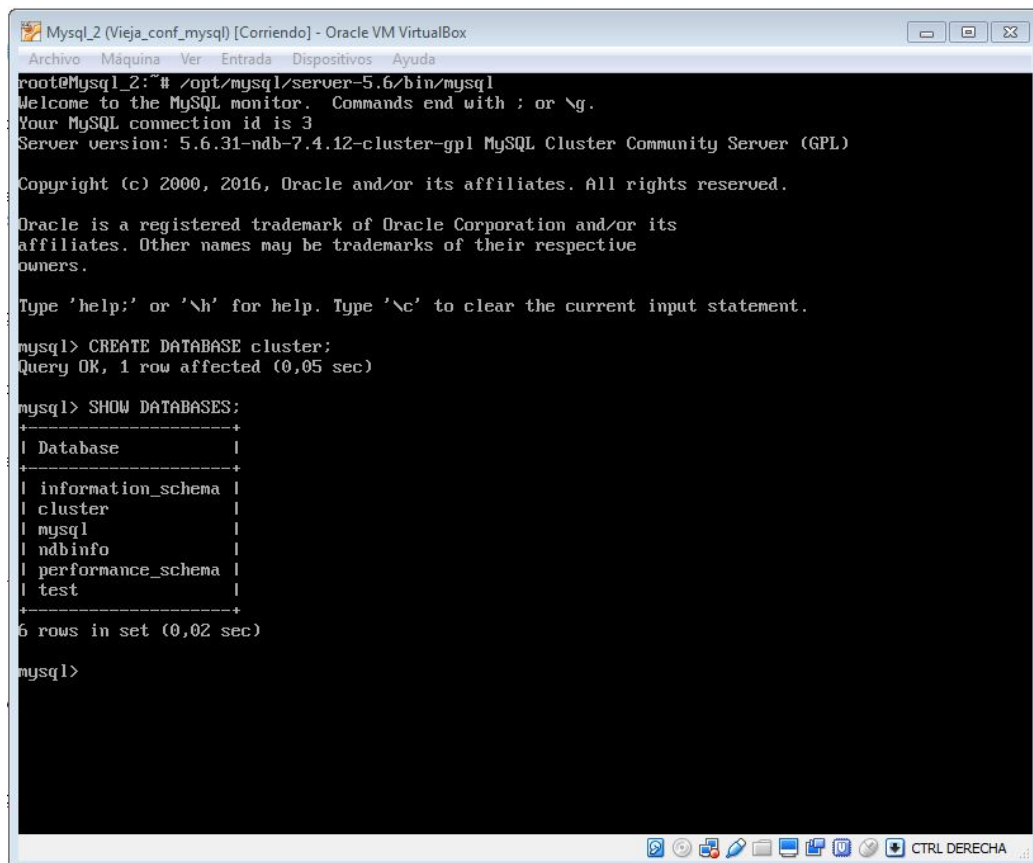
```
root@Client:~# apt-get install mysql-client
```

Para ver el estado de los nodos del clúster en todo momento es necesario acceder a la interfaz de gestión que tiene cualquiera de los nodos MGM. Para ello hay que ejecutar esta orden en cualquiera de ellos, y una vez en la consola de administración se ha de poner «SHOW»:

```
root@Mysql\_1:~/opt/mysql/server-5.6/bin/ndb_mgm
```

Para comprobar que la replicación entre los diferentes nodos SQL funciona correctamente se va a crear una BBDD llamada «cluster» en uno de los nodos y posteriormente se accederá al otro para comprobar si realmente se ha creado en ese también. Para la creación hay que conectarse en Mysql_2 a

un cliente MySQL y posteriormente (una vez conectados) ejecutar la siguiente query: «CREATE DATABASE cluster». En la figura 6.2 se aprecia todo el proceso.



```
Oracle VM VirtualBox
Mysql_2 (Vieja_conf_mysql) [Corriendo]
Archivo  Máquina  Ver  Entrada  Dispositivos  Ayuda
root@Mysql_2:~# /opt/mysql/server-5.6/bin/mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3
Server version: 5.6.31-ndb-7.4.12-cluster-gpl MySQL Cluster Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> CREATE DATABASE cluster;
Query OK, 1 row affected (0,05 sec)

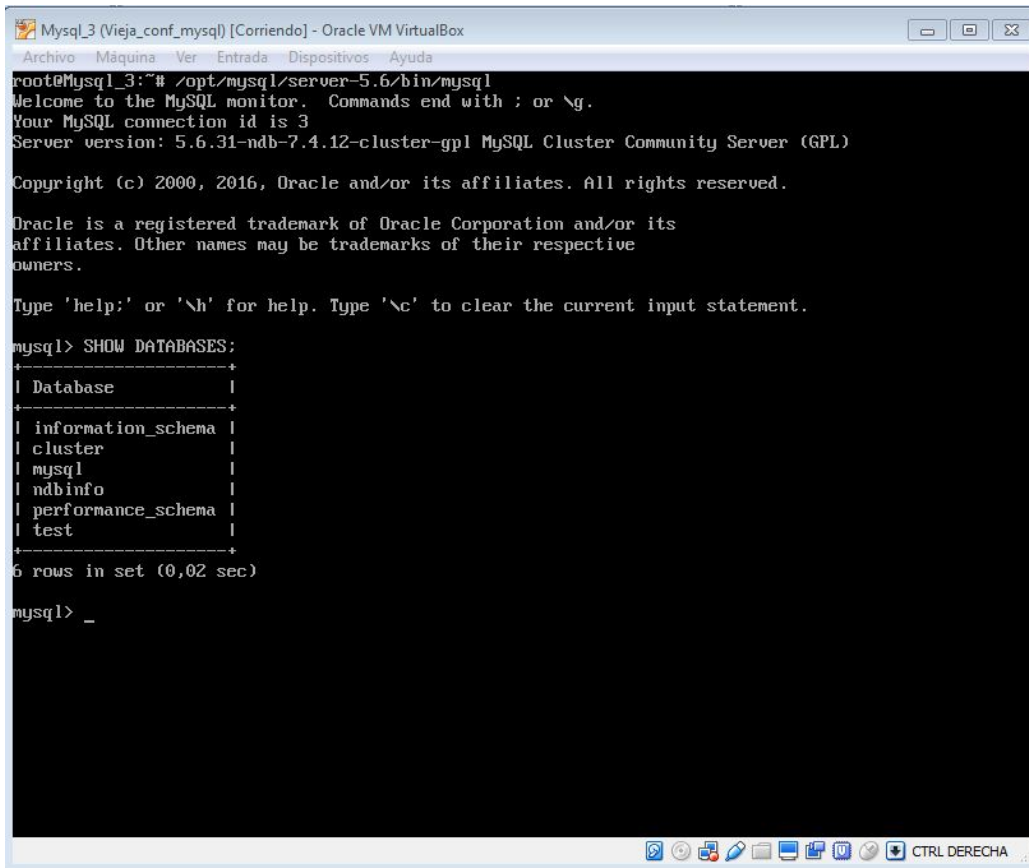
mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| cluster      |
| mysql        |
| ndbinfo      |
| performance_schema |
| test         |
+-----+
6 rows in set (0,02 sec)

mysql>
```

Figura 6.1: Creación de la base de datos de prueba

Una vez creada la base de datos satisfactoriamente se ha de conectar a un cliente MySQL en Mysql_3 y comprobar mediante un «SHOW DATABASES» si es accesible desde este nodo.

6.1. Todos los nodos activos



```
Mysql_3 (Vieja_conf_mysql) [Corriendo] - Oracle VM VirtualBox
Archivo  Máquina  Ver  Entrada  Dispositivos  Ayuda
root@Mysql_3:~# /opt/mysql/server-5.6/bin/mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3
Server version: 5.6.31-ndb-7.4.12-cluster-gpl MySQL Cluster Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| cluster |
| mysql |
| ndbinfo |
| performance_schema |
| test |
+-----+
6 rows in set (0,02 sec)

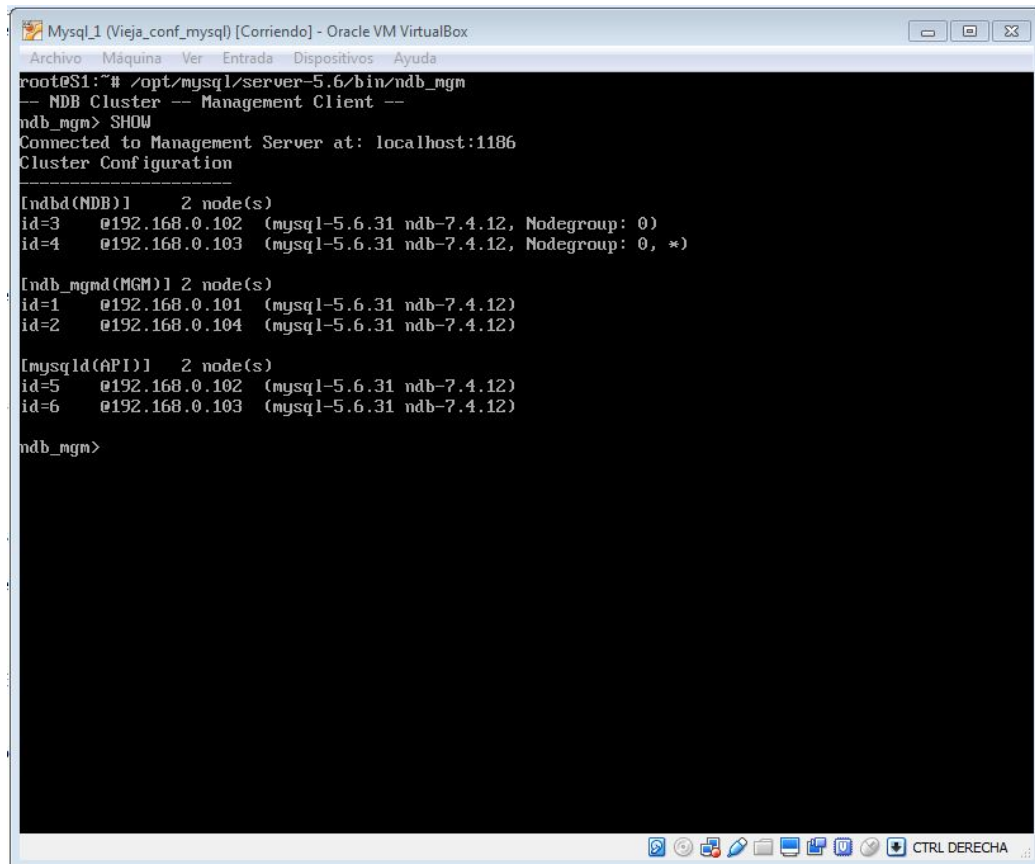
mysql> _
```

Figura 6.2: Prueba de acceso entre SQL

Como se aprecia en la figura 6.3, la prueba es satisfactoria, puesto que se puede ver en este nodo la BBDD anteriormente creada.

6.1. Todos los nodos activos

Con todos los elementos activos el estado que nos muestra el MGM del Mysql_1 en la consola de administración es el de la figura 6.1.



```
root@S1:~# /opt/mysql/server-5.6/bin/ndb_mgm
-- NDB Cluster -- Management Client --
ndb_mgm> SHOW
Connected to Management Server at: localhost:1186
Cluster Configuration
-----
[ndbd(NDB)] 2 node(s)
id=3 @192.168.0.102 (mysql-5.6.31 ndb-7.4.12, Nodegroup: 0)
id=4 @192.168.0.103 (mysql-5.6.31 ndb-7.4.12, Nodegroup: 0, *)

[ndb_mgmd(MGM)] 2 node(s)
id=1 @192.168.0.101 (mysql-5.6.31 ndb-7.4.12)
id=2 @192.168.0.104 (mysql-5.6.31 ndb-7.4.12)

[mysqld(API)] 2 node(s)
id=5 @192.168.0.102 (mysql-5.6.31 ndb-7.4.12)
id=6 @192.168.0.103 (mysql-5.6.31 ndb-7.4.12)

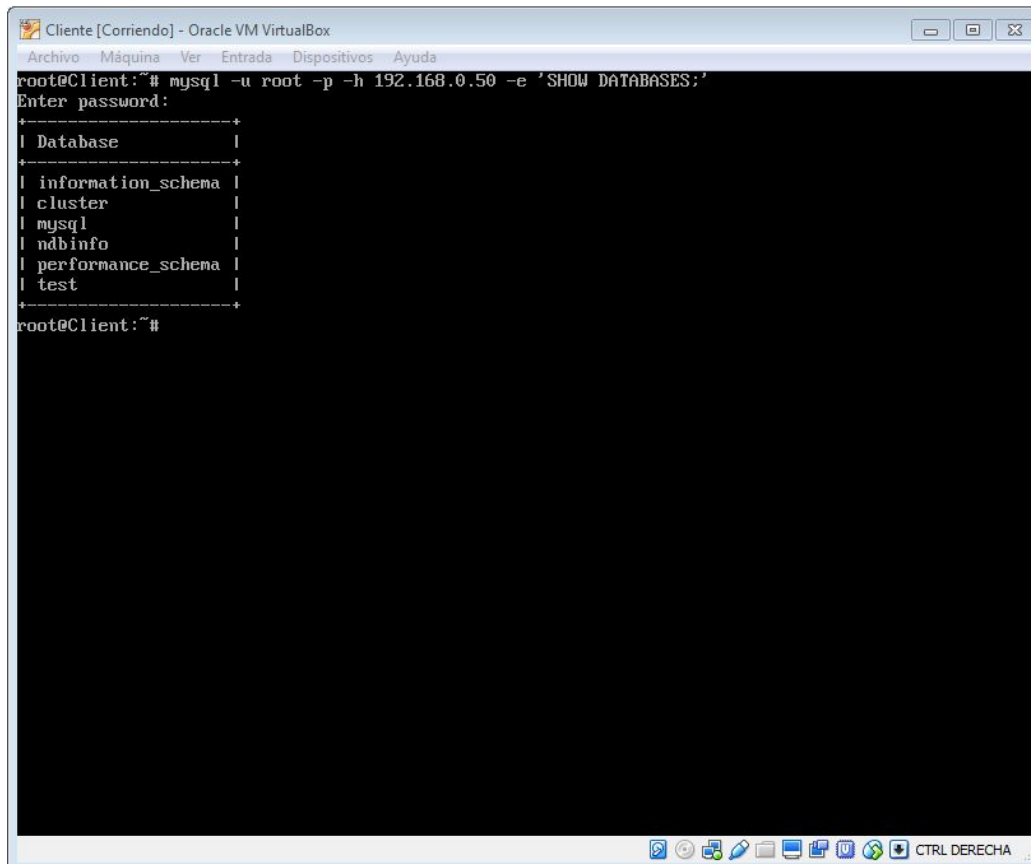
ndb_mgm>
```

Figura 6.3: Estado de los nodos del clúster de MySQL, todos los elementos activos

Para la prueba inicial se va a utilizar la misma consulta, pero esta vez se ha de lanzar la petición desde el cliente. Para ello hay que usar el cliente MySQL, lanzando la petición a la VIP correspondiente:

```
root@Client:~# mysql -u root -p -h 192.168.0.50 -e 'SHOW DATABASES;'
```

El resultado de la ejecución de la petición se ve en la figura 6.4. Como se observa ha funcionado correctamente, ya que se muestra el listado de las BBDD que hay en el momento de la consulta.

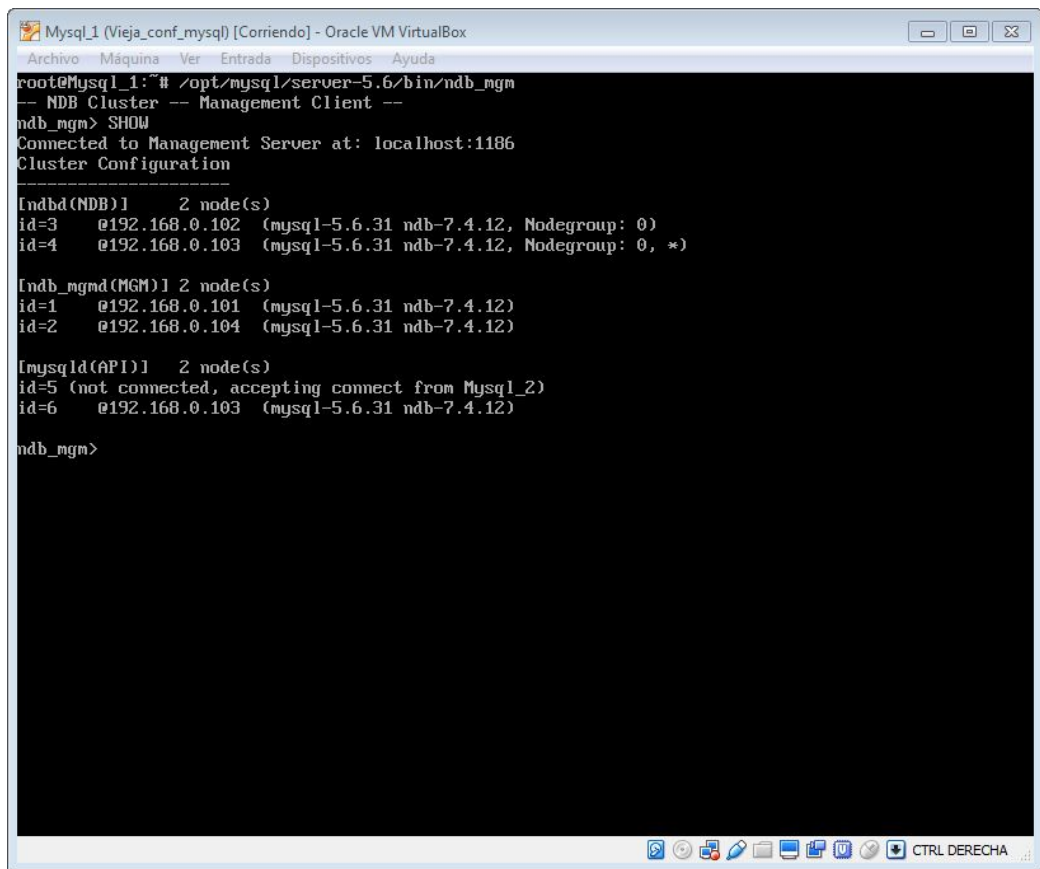


```
Cliente [Corriendo] - Oracle VM VirtualBox
Archivo  Máquina  Ver  Entrada  Dispositivos  Ayuda
root@Client:~# mysql -u root -p -h 192.168.0.50 -e 'SHOW DATABASES;'
Enter password:
+-----+
| Database |
+-----+
| information_schema |
| cluster |
| mysql |
| ndbinfo |
| performance_schema |
| test |
+-----+
root@Client:~#
```

Figura 6.4: Prueba de funcionamiento del clúster de MySQL con todos los elementos activos

6.2. Fallo de un SQL

Para simular un fallo en uno de los nodos se ha matado el proceso correspondiente al nodo SQL de la máquina Mysql_2. En la figura 6.5 se aprecia el estado sin el nodo SQL.



```
Mysql_1 (Vieja_conf_mysql) [Corriendo] - Oracle VM VirtualBox
Archivo  Máquina  Ver  Entrada  Dispositivos  Ayuda
root@Mysql_1:~# /opt/mysql/server-5.6/bin/ndb_mgm
-- NDB Cluster -- Management Client --
ndb_mgm> SHOW
Connected to Management Server at: localhost:1186
Cluster Configuration
-----
[ndbd(NDB)] 2 node(s)
id=3 @192.168.0.102 (mysql-5.6.31 ndb-7.4.12, Nodegroup: 0)
id=4 @192.168.0.103 (mysql-5.6.31 ndb-7.4.12, Nodegroup: 0, *)

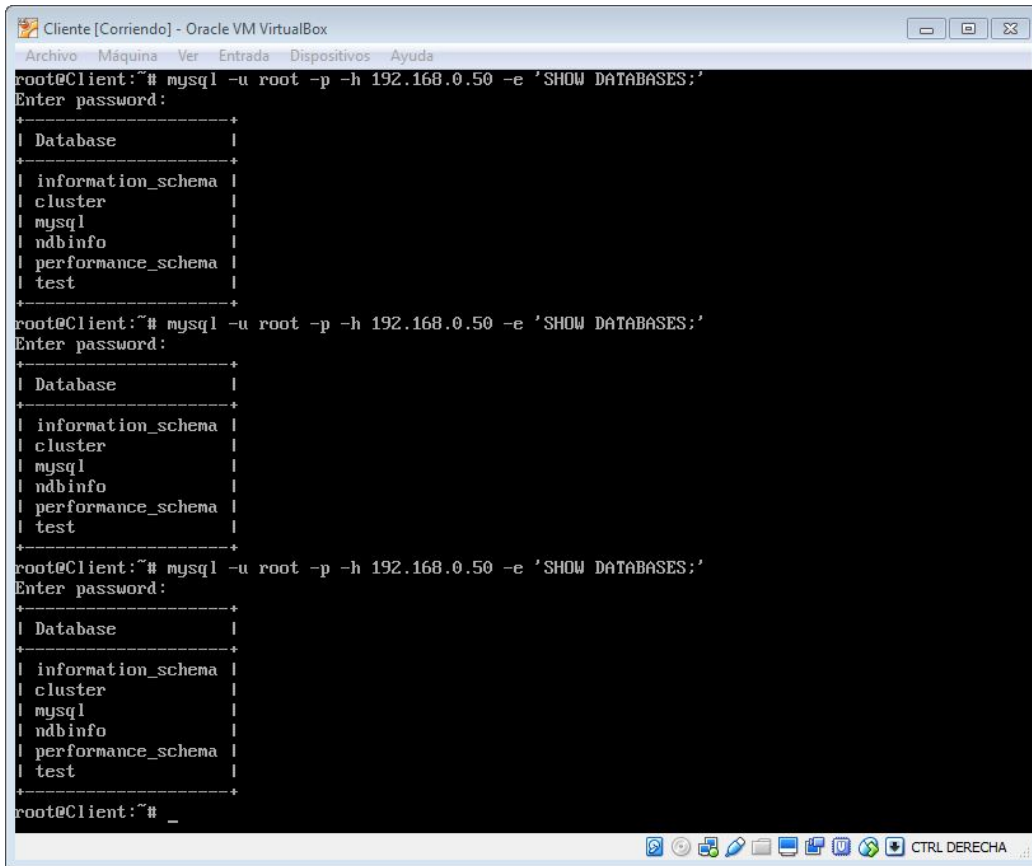
[ndb_mgmd(MGM)] 2 node(s)
id=1 @192.168.0.101 (mysql-5.6.31 ndb-7.4.12)
id=2 @192.168.0.104 (mysql-5.6.31 ndb-7.4.12)

[mysqlid(API)] 2 node(s)
id=5 (not connected, accepting connect from Mysql_2)
id=6 @192.168.0.103 (mysql-5.6.31 ndb-7.4.12)

ndb_mgm>
```

Figura 6.5: Estado de los nodos del clúster de MySQL con un nodo SQL fuera de servicio

Si todo funciona correctamente el HAProxy comprobará, por el *check* que tiene configurado, que el nodo está fuera de servicio y únicamente le enviará las peticiones al nodo SQL activo. Para demostrar esto se lanzarán tres peticiones como las del apartado anterior a través del cliente. En la figura 6.6 se puede apreciar que el comportamiento es correcto.

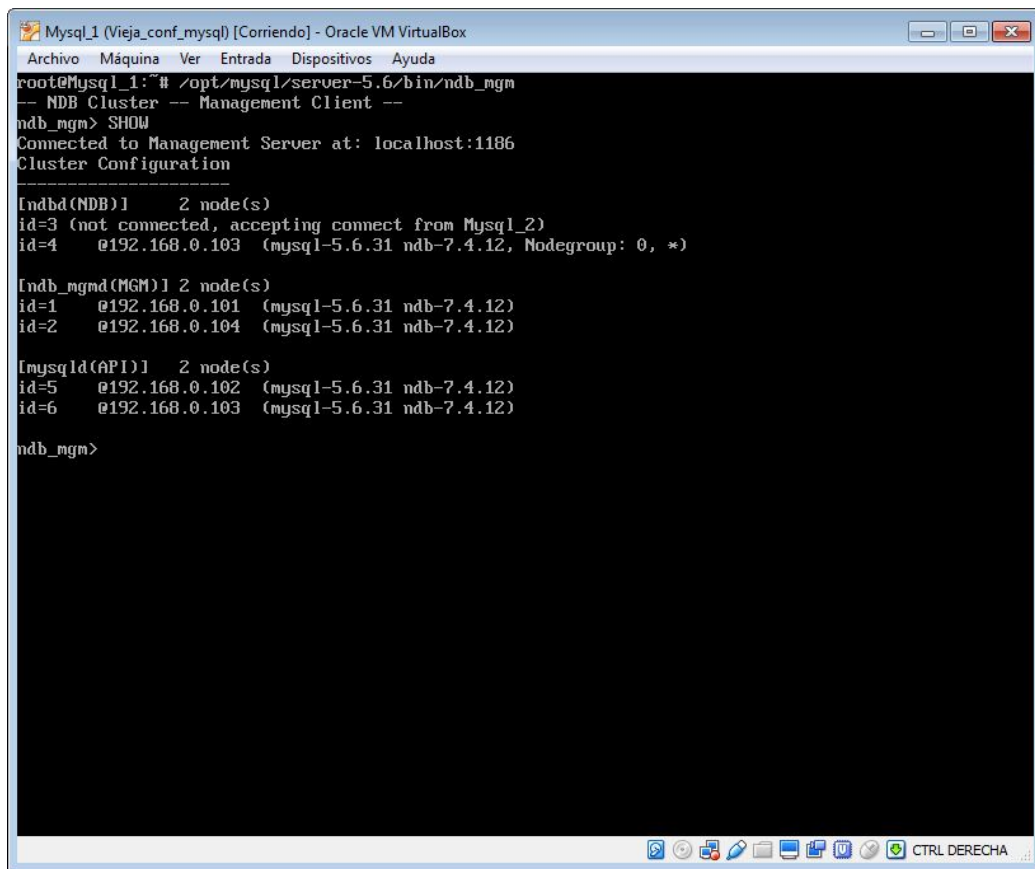


```
Cliente [Corriendo] - Oracle VM VirtualBox
Archivo  Máquina  Ver  Entrada  Dispositivos  Ayuda
root@Client:~# mysql -u root -p -h 192.168.0.50 -e 'SHOW DATABASES;'
Enter password:
+-----+
| Database |
+-----+
| information_schema |
| cluster |
| mysql |
| ndbinfo |
| performance_schema |
| test |
+-----+
root@Client:~# mysql -u root -p -h 192.168.0.50 -e 'SHOW DATABASES;'
Enter password:
+-----+
| Database |
+-----+
| information_schema |
| cluster |
| mysql |
| ndbinfo |
| performance_schema |
| test |
+-----+
root@Client:~# mysql -u root -p -h 192.168.0.50 -e 'SHOW DATABASES;'
Enter password:
+-----+
| Database |
+-----+
| information_schema |
| cluster |
| mysql |
| ndbinfo |
| performance_schema |
| test |
+-----+
root@Client:~# _
```

Figura 6.6: Prueba de funcionamiento del clúster de MySQL con un nodo SQL fuera de servicio

6.3. Fallo de un NDB

Al igual que en la sección anterior se va a matar el proceso para simular que el nodo está fuera de servicio, la única diferencia es que esta vez se matará el proceso correspondiente al NDB de Mysql_2. En la figura 6.7 se aprecia el estado tras el fallo de un nodo NDB.



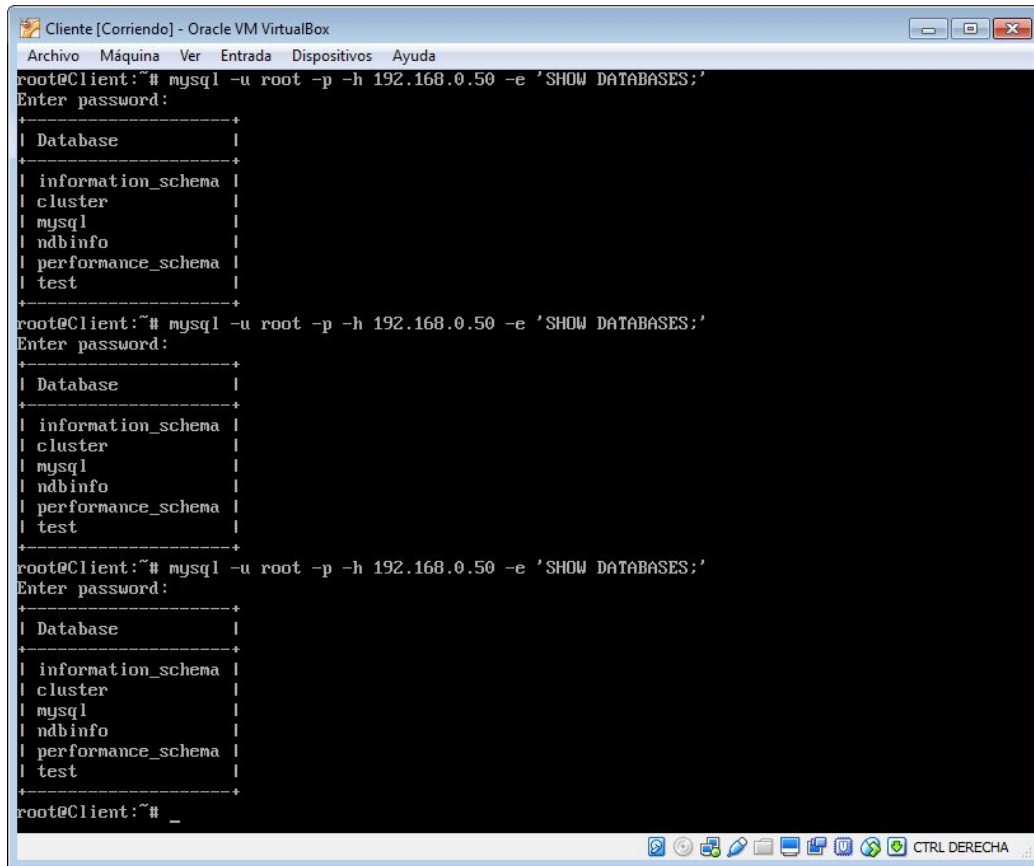
```
Mysql_1 (Vieja_conf_mysql) [Corriendo] - Oracle VM VirtualBox
Archivo  Máquina  Ver  Entrada  Dispositivos  Ayuda
root@Mysql_1:~# /opt/mysql/server-5.6/bin/ndb_mgm
-- NDB Cluster -- Management Client --
ndb_mgm> SHOW
Connected to Management Server at: localhost:1186
Cluster Configuration
-----
[ndbd(NDB)] 2 node(s)
id=3 (not connected, accepting connect from Mysql_2)
id=4 @192.168.0.103 (mysql-5.6.31 ndb-7.4.12, Nodegroup: 0, *)

[ndb_mgmd(MGM)] 2 node(s)
id=1 @192.168.0.101 (mysql-5.6.31 ndb-7.4.12)
id=2 @192.168.0.104 (mysql-5.6.31 ndb-7.4.12)

[mysqlid(API)] 2 node(s)
id=5 @192.168.0.102 (mysql-5.6.31 ndb-7.4.12)
id=6 @192.168.0.103 (mysql-5.6.31 ndb-7.4.12)
ndb_mgm>
```

Figura 6.7: Estado de los nodos del clúster de MySQL con un nodo NDB fuera de servicio

Del mismo modo que en la prueba con un nodo SQL fuera de servicio, se van a realizar tres peticiones desde el cliente. En este caso el HAProxy si que mandará peticiones a los dos nodos SQL (puesto que están activos), pero ambos nodos realizarán todas las consultas sobre el nodo NDB que está activo. Como se muestra en la figura 6.8 funciona tal y como se espera.

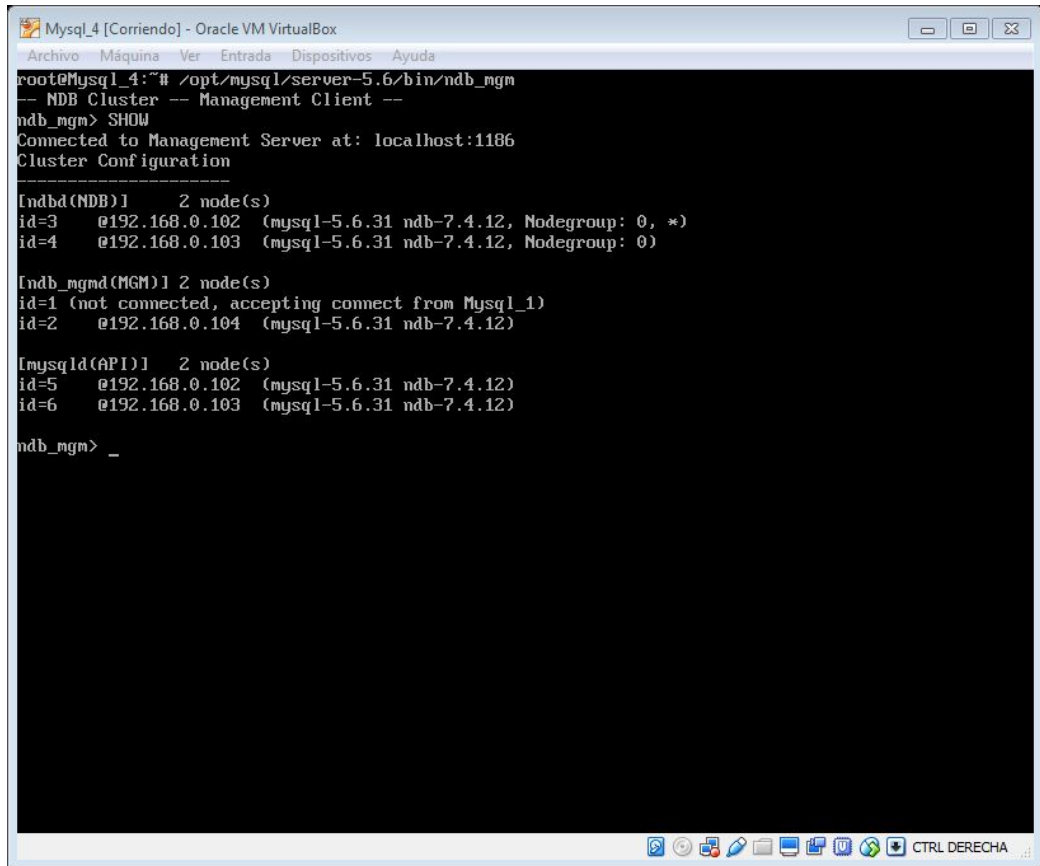


```
Cliente [Corriendo] - Oracle VM VirtualBox
Archivo Máquina Ver Entrada Dispositivos Ayuda
root@Client:~# mysql -u root -p -h 192.168.0.50 -e 'SHOW DATABASES;'
Enter password:
+-----+
| Database |
+-----+
| information_schema |
| cluster |
| mysql |
| ndbinfo |
| performance_schema |
| test |
+-----+
root@Client:~# mysql -u root -p -h 192.168.0.50 -e 'SHOW DATABASES;'
Enter password:
+-----+
| Database |
+-----+
| information_schema |
| cluster |
| mysql |
| ndbinfo |
| performance_schema |
| test |
+-----+
root@Client:~# mysql -u root -p -h 192.168.0.50 -e 'SHOW DATABASES;'
Enter password:
+-----+
| Database |
+-----+
| information_schema |
| cluster |
| mysql |
| ndbinfo |
| performance_schema |
| test |
+-----+
root@Client:~# _
```

Figura 6.8: Prueba de funcionamiento del clúster de MySQL con un nodo NDB fuera de servicio

6.4. Fallo en un MGM

Para recrear un fallo en un nodo MGM se ha matado el proceso correspondiente en la máquina Mysql_1. Una vez dicho proceso esté muerto solo quedará como MGM el correspondiente a la máquina Mysql_4. Para verificarlo, se consulta el estado del clúster tal y como se ha hecho en las secciones anteriores, pero esta vez en el Mysql_4. El estado se muestra en la figura 6.9.



```
root@Mysql_4:~# /opt/mysql/server-5.6/bin/ndb_mgm
-- NDB Cluster -- Management Client --
ndb_mgm> SHOW
Connected to Management Server at: localhost:1186
Cluster Configuration
-----
[ndbd(NDB)] 2 node(s)
id=3 @192.168.0.102 (mysql-5.6.31 ndb-7.4.12, Nodegroup: 0, *)
id=4 @192.168.0.103 (mysql-5.6.31 ndb-7.4.12, Nodegroup: 0)

[ndb_mgmd(MGM)] 2 node(s)
id=1 (not connected, accepting connect from Mysql_1)
id=2 @192.168.0.104 (mysql-5.6.31 ndb-7.4.12)

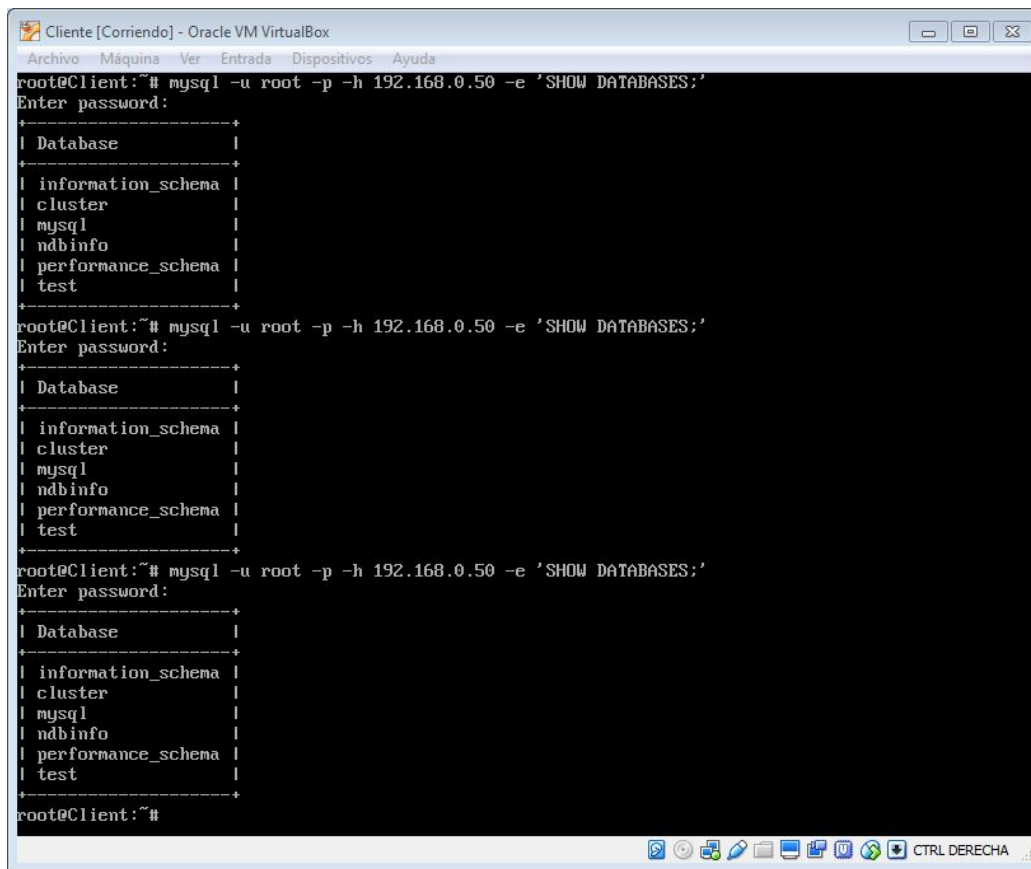
[mysqlid(API)] 2 node(s)
id=5 @192.168.0.102 (mysql-5.6.31 ndb-7.4.12)
id=6 @192.168.0.103 (mysql-5.6.31 ndb-7.4.12)

ndb_mgm> _
```

Figura 6.9: Estado de los nodos del clúster de MySQL con un nodo MGM fuera de servicio

El comportamiento con un único MGM debe de ser idéntico al que presenta el sistema cuando hay dos MGM. Para verificarlo se van a relizar tres peticiones desde el cliente. Como se puede apreciar en la figura 6.10 el comportamiento es el mismo.

6.4. Fallo en un MGM



```
Cliente [Corriendo] - Oracle VM VirtualBox
Archivo  Máquina  Ver  Entrada  Dispositivos  Ayuda
root@Client:~# mysql -u root -p -h 192.168.0.50 -e 'SHOW DATABASES;'
Enter password:
+-----+
| Database |
+-----+
| information_schema |
| cluster |
| mysql |
| ndbinfo |
| performance_schema |
| test |
+-----+
root@Client:~# mysql -u root -p -h 192.168.0.50 -e 'SHOW DATABASES;'
Enter password:
+-----+
| Database |
+-----+
| information_schema |
| cluster |
| mysql |
| ndbinfo |
| performance_schema |
| test |
+-----+
root@Client:~# mysql -u root -p -h 192.168.0.50 -e 'SHOW DATABASES;'
Enter password:
+-----+
| Database |
+-----+
| information_schema |
| cluster |
| mysql |
| ndbinfo |
| performance_schema |
| test |
+-----+
root@Client:~#
```

Figura 6.10: Prueba de funcionamiento del clúster de MySQL con un nodo MGM fuera de servicio

Capítulo 7

Evaluación

Pese a que el prototipo realizado a lo largo de este proyecto esté basado en máquinas virtuales, resulta interesante ver la capacidad del sistema que se ha construido. Los resultados obtenidos no serán comparables con respecto a los que se pueden obtener en nodos servidores reales con la misma configuración. La virtualización ofrece muchas ventajas a cambio de prestaciones.

7.1. Evaluación del servicio web

Para la evaluación a nivel de tráfico web se utilizará el Apache Benchmark, un SW que obtiene estadísticas de uso de un sistema para unas determinadas condiciones de carga. Para esta utilidad se pueden configurar el número de peticiones totales a realizar, la cantidad de peticiones simultáneas, etc. El test se lanzará desde el Client, luego solo es necesario instalar el Apache Benchmark en dicho nodo. Para ello hay que ejecutar la siguiente orden:

```
root@Client:~# apt-get install apache2-utils
```

Existen diferentes opciones para la herramienta que configuran el *benchmark* que se va a realizar. Algunas permiten añadir campos a la cabecera de los paquetes enviados, configurar conexiones persistentes, etc. Las más comunes, y las que vamos a usar son *c* (número de peticiones concurrentes), y *n* (número total de peticiones servidas para realizar la prueba).

Para realizar las pruebas se ha desarrollado una página en php que calcula el número *pi* mediante integración numérica, se ajusta en la llamada el número de intervalos (*n*), de manera que mediante este parámetro se puede regular el tiempo de ejecución. Este es el código de la página *pi.php*:

```
<?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']);
?>
```

Para ejecutar dicho comando en el Client para 2 peticiones concurrentes, un total de 1000 peticiones y 5000 intervalos en el calculo de la constante π se haría de la siguiente forma:

```
root@Client:~# ab -c2 -n1000 http://192.168.0.60/index.php?5000
```

Se van a realizar dos tipos de pruebas diferentes, en los cuales se medirán el número de peticiones por segundo atendidas:

- Número de peticiones constante, número de peticiones concurrentes constante y número de intervalos considerados en pi.php variable.
- Número de peticiones constante, número de peticiones concurrentes variable y número de intervalos considerados en pi.php constante.

Para el primer grupo de pruebas se van a ejecutar todas pruebas con 2 peticiones concurrentes y un total de 1000 peticiones. Los valores el número de intervalos considerados son: 10, 100, 500, 1000, 5000, 10000. Los resultados están en la siguiente figura:

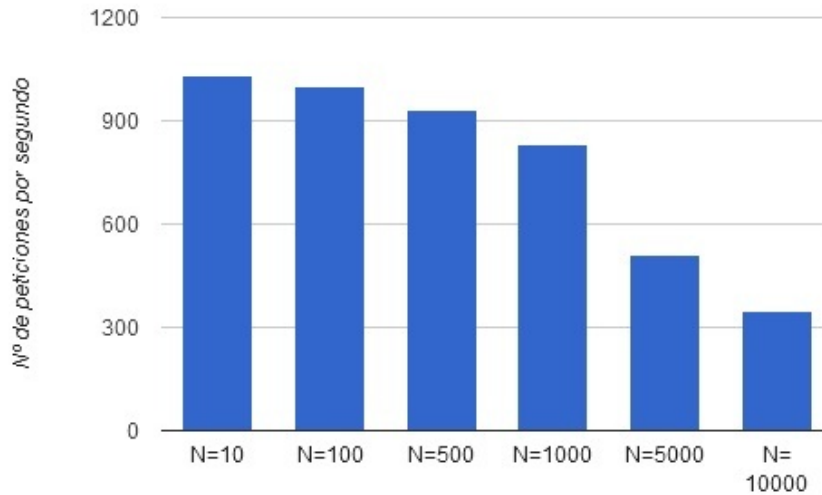


Figura 7.1: Evaluación del servidor web para pi.php con número de intervalos variable

Como era de esperar conforme el número de intervalos considerados va aumentando, la tasa de peticiones atendidas se va reduciendo. Esto es debido a que cada vez las peticiones tardan más en resolverse, puesto que los nodos servidores tardan más tiempo en realizar el cálculo.

Para el segundo grupo de pruebas se han considerado 5000 intervalos en el cálculo de π , y un total de 1000 peticiones, mientras que los valores que va a tomar el número de peticiones concurrentes van a ser: 2, 4, 8, 16, 32, 64. En la siguiente figura se pueden apreciar los resultados obtenidos:

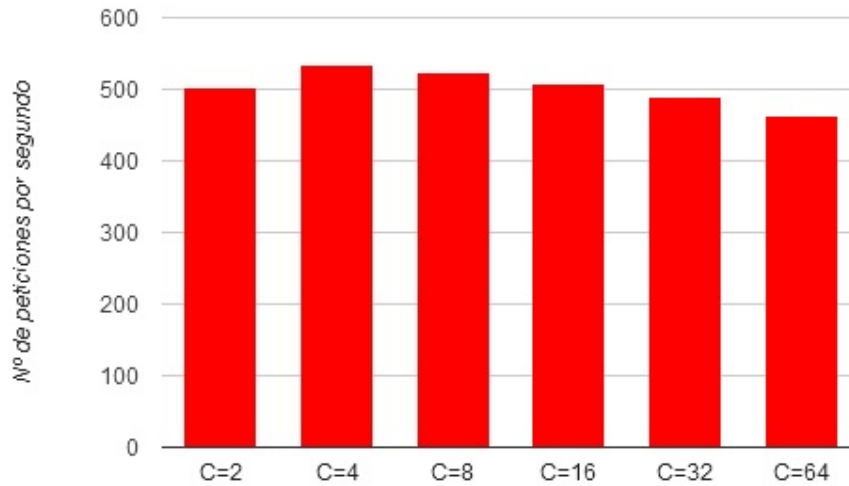


Figura 7.2: Evaluación del servidor web para pi.php con un número de usuarios concurrentes variable

En este caso se ve que el número de peticiones por segundo llega a su máximo cuando hay cuatro peticiones concurrentes. Conforme este valor va aumentando, las peticiones por segundo comienzan a decaer, esto es debido a que la gran cantidad de peticiones concurrentes no pueden ser tratadas con la misma efectividad al mismo tiempo, lo cual satura al LB y hace que se obtengan peores resultados.

En este caso se ve que el número de peticiones por segundo llega a su máximo para cuatro peticiones concurrentes. En general, la tasa de transacciones atendidas debería crecer hasta que el número de usuarios concurrentes alcanza el número de procesadores disponible en los nodos servidores. En nuestro prototipo tenemos 4 nodos servidores, cada uno con un único procesador. Conforme el número de peticiones concurrentes sobrepasa este valor las peticiones por segundo atendidas comienzan a decaer. Esto puede deberse a saturación de los propios nodos servidores, del propio nodo repartidor de carga o también relacionado con el hecho de que las pruebas se han hecho sobre un entorno virtualizado que se ejecuta sobre un computador anfitrión cuya procesador sólo tiene dos núcleos.

7.2. Evaluación del servicio de BBDD

La evaluación del sistema con tráfico de BBDD se va a realizar con una herramienta SW que se llama Sysbench. Se trata de una herramienta de *benchmarking* que sirve para medir diferentes parámetros del sistema. Se va a usar en este contexto porque puede generar tráfico (consultas de BBDD) para estimar el rendimiento de la BBDD. Para instalarla hay que ejecutar el siguiente comando en la máquina Client:

```
root@Client:~# apt-get install sysbench
```

Esta herramienta tiene varios modos de funcionamiento. El primero de ellos es el de preparación, en este modo crea la base de datos y la tabla donde se van a lanzar las pruebas. El segundo es cuando se lanza ya la prueba para obtener resultados. Para ejecutar ambos modos se haría de la siguiente manera:

```
root@Client:~# sysbench --test=oltp --oltp-table-size=10000
--mysql-host=192.168.0.50 --mysql-db=cluster
--mysql-user=root --mysql-table-engine=ndbcluster
--mysql-password= prepare
root@Client:~# sysbench --test=oltp --oltp-table-size=10000
--mysql-host=192.168.0.50--mysql-db=cluster --mysql-user=root
--mysql-password= --num_threads=8 run
```

Entre los diferentes parámetros de la llamada es necesario destacar varios: `oltp` es como se denomina el tipo de test para BBDD, y el más importante, el parámetro `mysql-table-engine=ndbcluster`, si no se ejecuta con ese motor seleccionado el clúster no replicará entre los nodos, lo hará en local el nodo que le reparte el LB.

En este caso se va a probar el comportamiento del sistema para ver los resultados con 1, 2, 4, 8, 16 y 32 *threads*. Se pueden ver en la siguiente imagen:

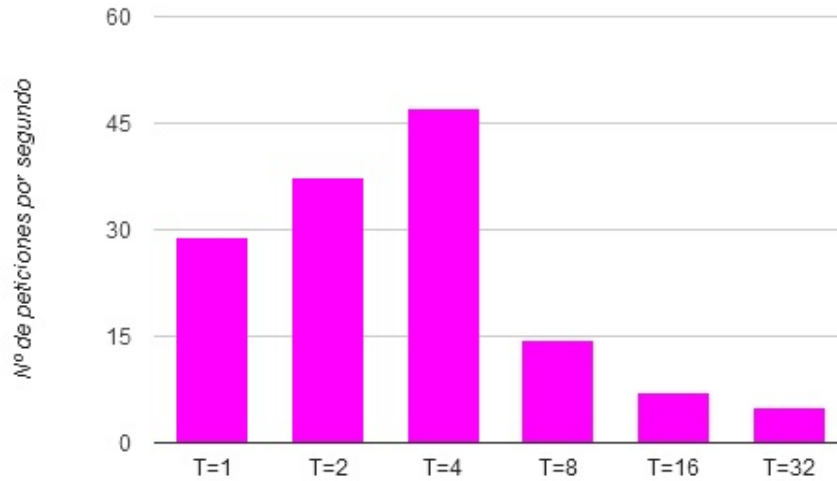


Figura 7.3: Evaluación del servidor de BBDD con número de *threads* concurrentes variable

El comportamiento es muy similar al observado con el servidor Web. El número de peticiones atendidas por segundo aumenta hasta un determinado número (4 en este caso). A partir de ese punto comienzan a descender. La justificación cabe buscarla nuevamente en una saturación de los nodos servidores, del nodo LB o en los efectos laterales generados por la ejecución del prototipo virtualizado sobre un computador anfitrión con un escaso número de núcleos.

Capítulo 8

Conclusiones

A lo largo del documento se han ido desgranando una a una muchas ideas y conceptos que han culminado en un clúster completamente operativo. El clúster ofrece servicios web y de acceso a bases de datos MySQL y ofrece equilibrado de carga con alta disponibilidad. Este resultado ha sido fruto de mucho trabajo e investigación, puesto que he tenido que manejar diversas tecnologías adaptándolas a las necesidades de la situación. En muchos casos, muchas de ellas se entrelazan y complementan.

Desde un punto de vista académico ha resultado muy interesante e instructivo el enfrentarse con una problemática real, que responde a las necesidades y retos actuales del sector informático.

En el trabajo se ha pasado por muchos niveles diferentes, desde la configuración e instalación de paquetes, hasta el desarrollo y definición de configuraciones de computadores. No obstante quedan muchas puertas abiertas hacia un futuro.

Una opción muy interesante sería el ampliar el sistema, tanto a nivel de nodos servidores como de nodos MySQL para intentar ahondar más todavía en la tecnología. También se podría ampliar el clúster con un sistema de almacenamiento escalable. Finalmente, todas las pruebas se han realizado sobre máquinas virtuales. Un paso más sería configurar un clúster de computadores basado en máquinas reales.

Bibliografía

- [1] <http://www.haproxy.org/they-use-it.html>
- [2] <http://www.haproxy.org/#perf>
- [3] <https://loadbalancer.org/blog/load-balancer-performance-benchmarking-haproxy-on-ec2-quick-and-dirty-style>
- [4] <http://www.linuxvirtualserver.org/>
- [5] <http://www.haproxy.org/>
- [6] <http://www.keepalived.org/>
- [7] <http://corosync.github.io/corosync/>
- [8] <http://clusterlabs.org/>
- [9] <https://tools.ietf.org/html/rfc5798>
- [10] www.microsoft.com/sql
- [11] <https://www.mysql.com/>
- [12] <https://www.percona.com/software/mysql-database/percona-xtradb-cluster>
- [13] <http://galeracluster.com/products/>
- [14] <http://dev.mysql.com/downloads/cluster/>