Replicación y alta disponibilidad de PostgreSQL con pgpool-II

Por Jaume Sabater, publicado el 1 de noviembre de 2008.

Índice

Introducción

En este artículo se muestra cómo instalar, configurar y mantener un clúster de servidores de bases de datos PostgreSQL gestionados mediante un middleware llamado pgpool-II sobre el sistema operativo Debian GNU/Linux. Dicho clúster ofrece capacidades de replicación, balanceo de carga y un pool de conexiones, y es capaz de realizar failover o degeneración de un nodo que deje de funcionar y de recuperar nodos caídos en línea (sin dejar de dar servicio). Se trata de un clúster activo-pasivo, si bien se hace uso del nodo pasivo para lectura con el propósito de mejorar la productividad del sistema.

Failover cluster

Un failover cluster (o clúster activo-pasivo) es un grupo de ordenadores independientes que trabajan conjuntamente para incrementar la disponibilidad de diversas aplicaciones y servicios. Los servidores en el clúster (llamados nodos) están interconectados mediante cables físicos y por software. Si uno de los nodos cae, otro empieza a dar servicio (proceso conocido como failover) sin necesidad de intervención humana. Esta guía describe los pasos para instalar y configurar un failover clúster con dos o más nodos.

Sobre PostgreSQL

PostgreSQL es la base de datos relacional de código abierto más avanzada del mundo. Distribuida bajo licencia BSD (del inglés, Berkeley Software Distribution), lleva más de 15 años desarrollándose y su arquitectura goza de una excelente reputación por su fiabilidad, integridad de datos y correctitud.

PostgreSQL dispone de versiones para prácticamente todos los sistemas operativos y cumple totalmente con ACID (del inglés, Atomicity, Consistency, Isolation, Durability). Tiene soporte para claves extranjeras, joins, vistas, disparadores y procedimientos almacenados (en múltiples lenguajes de programación). Incluye la mayoría de los tipos de datos de SQL92 y SQL99 y, asimismo, soporta el almacenamiento de grandes objetos binarios, como imágenes, sonidos y vídeos. Tiene interfaces de programación nativas para C/C++, Java, .Net, Perl, PHP, Python, Ruby, Tcl y ODBC, entre otros, y una excepcional documentación.

PostgreSQL ofrece sofisticadas características tales como control concurrente multiversión (MVCC), point in time recovery (PITR), tablespaces, replicación asíncrona, transacciones anidadas (savepoints), copias de seguridad en caliente/en línea, un sofisticado planificador/optimizador de consultas y write ahead logging para ser tolerante a fallos de hardware. Soporta juegos de caracteres internacionales, codificaciones de caracteres multibyte, Unicode y realiza ordenaciones dependiendo de la configuración de idioma local, de la diferenciación de mayúsculas y minúsculas y del formato. Es altamente escalable tanto en la cantidad bruta de datos que puede manejar como en el número de usuarios concurrentes que puede atender. Hay sistemas activos en producción con PostgreSQL que manejan más de 4 terabytes de datos.

Sobre pgpool-II

pgpool-II habla los protocolos de frontend y backend de PostgreSQL, y pasa las conexiones entre ellos. De ese modo, una aplicación de base de datos (frontend) cree que pgpool-II es el verdadero servidor de PostgreSQL, y el servidor (backend) ve a pgpool-II como uno de sus clientes. Debido a que pgpool-II es transparente tanto para el servidor como para el cliente, una aplicación de base de datos existente puede empezar a usarse con pgpool-II casi sin ningún cambio en su código fuente.

pgpool-II funciona sobre Linux, Solaris, FreeBSD y la mayoría de las arquitecturas UNIX. Windows no está soportado. Las versiones de PostgreSQL soportadas son de la 6.4 para arriba. Para usar la paralelización de consultas es necesaria la versión 7.4 o superior.

pgpool-II proporciona las siguientes características:

- **Limita el excedente de conexiones.** PostgreSQL soporta un cierto número de conexiones concurrentes y rechaza las que superen dicha cifra. Aumentar el límite máximo de conexiones incrementa el consumo de recursos y afecta al rendimiento del sistema. pgpool-II tiene también un límite máximo de conexiones, pero las conexiones extras se mantienen en una cola en lugar de devolver un error inmediatamente.

- **Pool de conexiones.** pgpool-II mantiene abiertas las conexiones a los servidores PostgreSQL y las reutiliza siempre que se solicita una nueva conexión con las mismas propiedades (nombre de usuario, base de datos y versión del protocolo). Ello reduce la sobrecarga en las conexiones y mejora la productividad global del sistema.
- **Replicación.** pgpool-II puede gestionar múltiples servidores PostgreSQL. El uso de la función de replicación permite crear una copia en dos o más discos físicos, de modo que el servicio puede continuar sin parar los servidores en caso de fallo en algún disco.
- **Balanceo de carga.** Si se replica una base de datos, la ejecución de una consulta SELECT en cualquiera de los servidores devolverá el mismo resultado. pgpool-II se aprovecha de la característica de replicación para reducir la carga en cada uno de los servidores PostgreSQL distribuyendo las consultas SELECT entre los múltiples servidores, mejorando así la productividad global del sistema. En el mejor caso, el rendimiento mejora proporcionalmente al número de servidores PostgreSQL. El balanceo de carga funciona mejor en la situación en la cuál hay muchos usuarios ejecutando muchas consultas al mismo tiempo.
- **Paralelización de consultas.** Al usar la función de paralelización de consultas, los datos pueden dividirse entre varios servidores, de modo que la consulta puede ejecutarse en todos los servidores de manera concurrente para reducir el tiempo total de ejecución. La paralelización de consultas es una solución adecuada para búsquedas de datos a gran escala.

Sobre Debian GNU/Linux

Debian GNU/Linux es un sistema operativo libre (el conjunto de programas básicos y utilidades que hacen que un ordenador funcione). Debian utiliza el núcleo Linux y las herramientas básicas de GNU. Para esta instalación se utilizará el sistema operativo Debian Lenny para la arquitectura x86_64 (AMD64/EM64T), partiendo de una instalación básica, sin ninguna tarea seleccionada en el selector de tareas del instalador. El sistema de ficheros elegido será XFS. La misma instalación puede obtenerse con Debian Etch y el repositorio de backports.

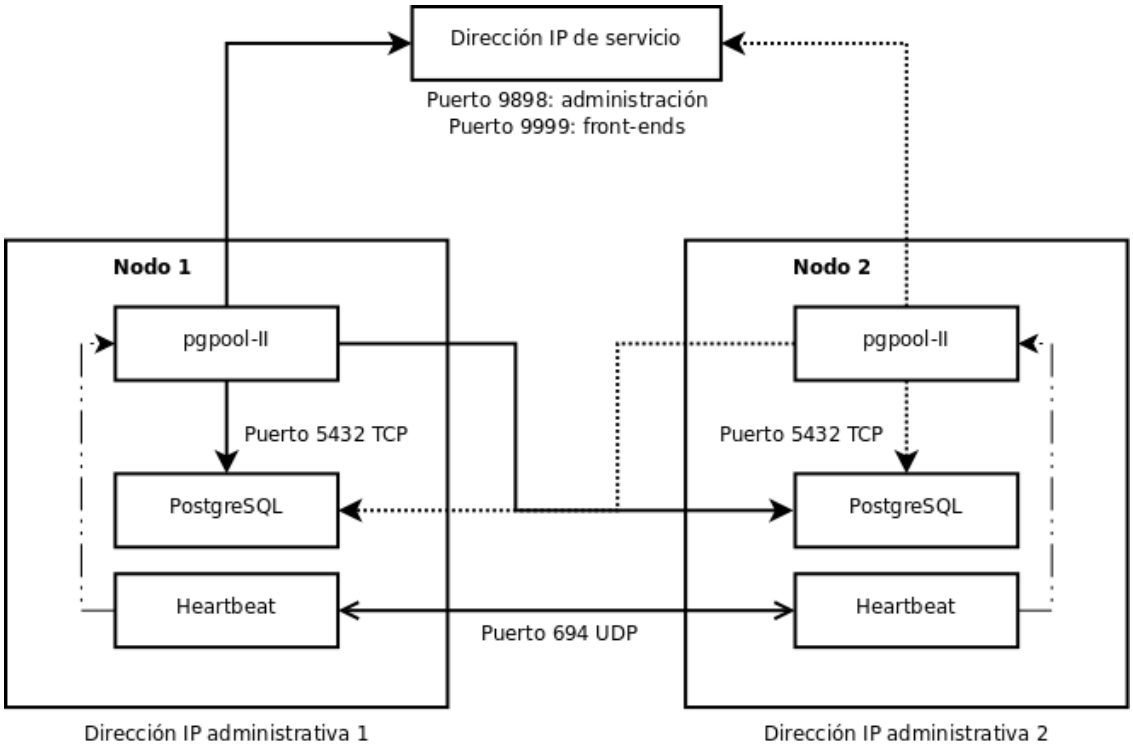
Arquitectura del sistema

El término clúster hace referencia a un conjunto de sistemas informáticos trabajando conjuntamente por un objetivo común. Como puede apreciarse, esta definición es muy genérica. No es, sino, un reflejo de la gran variedad y diversidad de acercamientos posibles a la hora de configurar un clúster y, por lo tanto, prueba de que el lector no debe tomar la arquitectura utilizada en el artículo más que como referencia y base para sus futuros trabajos.

En el clúster de este artículo se persiguen dos objetivos: uno, la alta disponibilidad, y dos, el rendimiento. La funcionalidad que persigue el clúster es únicamente la de servidor de base de datos, pero lo hace a través de tres aplicaciones:

- PostgreSQL, el sistema gestor de bases de datos (S.G.B.D.)
- pgpool-II, el middleware que gestiona la alta disponibilidad de los servidores de PostgreSQL.
- Heartbeat, un software que usaremos para dar alta disponibilidad a pgpool-II y a la dirección IP de servicio.

Esta configuración nos permite obtener alta disponibilidad de todos los servicios y recursos en las dos máquinas destinadas a este clúster. El diagrama de la arquitectura resultante sería el siguiente:



Instalación de paquetes y configuración de dependencias

Se toma como punto de inicio un sistema x86_64 con Debian GNU/Linux Etch o Lenny instalado. Se utilizarán las siguientes versiones de software:

- PostgreSQL 8.3.x
- pgpool-II 2.2.x
- Heartbeat 2.x

Antes de empezar, instalaremos un conjunto de paquetes básico para cualquier sistema, así como una serie de utilidades que nos harán falta para la gestión y el mantenimiento del clúster:

```
apt-get install ntp openssl file psmisc sysstat bzip2 unzip nmap dstat rsync wget
ccze tcpdump pciutils dnsutils host
```

rsync lo usaremos para la recuperación en línea de nodos, ntp para mantener el reloj del sistema sincronizado. El

prototipo inicial de este clúster se probó en una máquina con dos instancias de Xen y una base de datos de prueba creada con pgbench. Más tarde se configuró una versión de preproducción sobre sendos Intel Quad Core con 8 GB de RAM y la versión final del clúster se configuró sobre dos nodos con doble Intel Quad Core cada uno y 16 GB de RAM. La base de datos final ocupa algo más de 30 GB y se encuentra sobre dos discos SAS de 15.000 RPM en RAID 1.

A efectos de este artículo, a continuación se presenta un resumen de los datos de configuración que se usarán:

- Nodo 1
    - Hostname: pgsql1.dominio.com
    - Dirección IP de administración: 192.168.0.3
- Nodo 2
    - Hostname: pgsql2.dominio.com
    - Dirección IP de administración: 192.168.0.4
- Máscara de red de 24 bits.
- Dirección IP del router: 192.168.0.1
- pgpool-II
    - Dirección IP de servicio: 192.168.0.2
    - Puerto de gestión: 9898
    - Puerto de servicio: 9999

Los conceptos utilizados en esta descripción se irán explicando a lo largo del artículo. Es preciso que las entradas correspondientes a los datos anteriores existan en el fichero /etc/hosts:

```
127.0.0.1       localhost.localdomain    localhost
192.168.0.1     router.dominio.com       router
192.168.0.2     pgpool2.dominio.com      pgpool2
192.168.0.3     pgsql1.dominio.com       pgsql1
192.168.0.4     pgsql2.dominio.com       pgsql2
```

Empezaremos configurando PostgreSQL en ambos nodos pero pgpool-II sólo en el nodo pgsql1. Los comandos deberán ejecutarse como root o mediante sudo, a menos que se indique lo contrario (normalmente comandos que se ejecutarán con el usuario postgres).

Las dependencias de compilación de pgpool-II (las cabeceras de la librería de PostgreSQL, el paquete de desarrollo de PostgreSQL para pogramación de servidores y las utilidades de compilación de GNU) las resolveremos mediante la instalación de los siguientes paquetes:

```
apt-get install libpq-dev postgresql-server-dev-8.3 bison build-essential
```

Una vez resueltas las dependencias, empezaremos instalando PostgreSQL en ambos nodos:

```
apt-get install postgresql-8.3 postgresql-contrib-8.3 postgresql-doc-8.3 uuid
libdbd-pg-perl
```

La instalación por defecto de PostgreSQL en Debian ya nos deja un sistema gestor de base de datos funcionando. Finalmente, descargamos pgpool-II, lo descomprimimos e instalamos en */opt/pgpool2*:

```
cd /usr/local/src
wget http://pgfoundry.org/frs/download.php/2478/pgpool-II-2.2.6.tar.gz
tar --extract --gzip --file pgpool-II-2.2.6.tar.gz
cd pgpool-II-2.2.6
./configure --prefix=/opt/pgpool2
make
make install
cd /usr/local/src/pgpool-II-2.2.6/sql/pgpool-recovery
make
make install
su - postgres -c "psql -f /usr/local/src/pgpool-II-2.2.6/sql/pgpool-recovery/pgpool-
recovery.sql template1"
```

A partir de aquí, todas las bases de datos que creemos heredarán las funciones existentes en template1.

Configuración de PostgreSQL

Tal y como se ha dicho anteriormente, los siguientes pasos aplican a ambas instancias de PostgreSQL en los nodos pgsql1 y pgsql2.

Empezaremos editando la configuración de PostgreSQL para permitir el acceso incondicional del usuario pgpool2, que será nuestro usuario de base de datos del clúster. Por incondicional nos referimos al uso del modo trust, que permite la validación del usuario sin necesidad de contraseña. Esto es un requerimiento de pgpool-II para el tipo de configuración del clúster que tendremos al final del artículo, por lo cuál deberemos prestar especial atención a los filtros de acceso por IP que configuremos tanto en los servidores PostgreSQL como en el cortafuegos de los nodos.

Comenzaremos, como usuario postgres, añadiendo el usuario de base de datos (role) pgpool2, sin contraseña:

```
su - postgres
createuser --superuser pgpool2
```

Para facilitar el trabajo en este artículo lo hemos dado de alta como superusuario. En realidad, si creamos la base de datos en todos los nodos como superadministrador postgres y le otorgamos propiedad al usuario pgpool2, éste puede ser un usuario con el role más básico (no superusuario, sin capacidad de crear bases de datos).

Editamos ahora el fichero */etc/postgresql/8.3/main/pg_hba.conf* y añadimos el acceso para el usuario pgpool2 desde la dirección IP donde se ejecutará pgpool-II (en estos momentos 192.168.0.3):

```
# TYPE  DATABASE    USER        CIDR-ADDRESS          METHOD
local   all         all                               ident sameuser
host    all         all         127.0.0.1/32          md5
host    all         pgpool2     192.168.0.3/32        trust
host    all         pgpool2     192.168.0.4/32        trust
host    all         all         ::1/128               md5
```

Al igual que durante la creación del usuario pgpool2, para facilitar este artículo se permite el acceso a todas las bases de datos a dicho usuario. En un entorno de producción sería preciso restringir dicho acceso a la base de datos que se vaya a usar.

Asimismo, si se quiere acceder directamente a la base de datos sin pasar por pgpool-II, por ejemplo para realizar pruebas de rendimiento o para su monitorización, deberán añadirse las direcciones IP de los clientes desde los que se conecten dichas aplicaciones. En este caso, si se desea, y siempre y cuando las direcciones IP de origen sean diferentes, puede cambiarse el método de autenticación a MD5 y crearse el usuario pgpool2 con contraseña.

A continuación activamos el archivado del Write-Ahead Log (WAL) de PostgreSQL, pues nos hará falta para poder usar PITR (Point-In-Time Recovery) desde pgpool-II. Editamos el fichero de configuración */etc/postgresql/8.3/main/postgresql.conf* y cambiamos los dos siguientes parámetros:

```
archive_mode = on
archive_command = 'exit 0'
```

Ya que sólo haremos uso de la característica de PITR cuando vayamos a recuperar un nodo caído o añadir uno nuevo, por defecto hemos configurado el parámetro archive_command para que no haga nada (exit 0). Esto lo hacemos debido a que la activación o desactivación del archivado de ficheros WAL requiere de un reinicio del servidor, pero la alteración del comando o script que realiza la tarea de archivar el fichero WAL rotado por PostgreSQL tan sólo requiere de una recarga de la configuración.

Así, PostgreSQL se comportará como si no estuviera archivando ficheros de log, generando dichos ficheros (de 16 MB cada uno) con normalidad en /var/lib/postgresql/8.3/main/pg_xlog y rotándolos a partir del octavo que almacene. Acto seguido crearemos el directorio /var/lib/postgresql/pg_xlog_archive, directorio donde archivaremos (copiaremos) los ficheros WAL cuando lo necesitemos, y le daremos permisos para el usuario postgres:

```
mkdir --mode=700 /var/lib/postgresql/pg_xlog_archive
chown postgres:postgres /var/lib/postgresql/pg_xlog_archive
```

Por último, indicaremos a PostgreSQL que escuche en todas las interfaces pues, por defecto, sólo lo hace en el localhost. Editamos el fichero /etc/postgresql/8.3/main/postgresql.conf y cambiamos la siguiente directiva:

```
listen_addresses = '*'
```

También podemos restringirlo a 127.0.0.1 y la dirección IP administrativa del nodo (192.168.0.3 en el primer nodo, 192.168.0.4 en el segundo) para asegurarnos de que no esté escuchando en la dirección IP de servicio del clúster (la 192.168.0.2, que queremos utilizar únicamente para pgpool-II).

Y reiniciamos PostgreSQL para activar los cambios:

```
/etc/init.d/postgresql-8.3 restart
```

Configuración de pgpool-II

La configuración de pgpool-II la realizaremos únicamente en el nodo pgsql1, pues sólo en ese host lo tenemos instalado.

pgpool-II proporciona una interfaz de gestión desde la cuál el administrador puede recoger el estado de pgpool-II y finalizar los procesos de pgpool-II a través de la red. El fichero pcp.conf es un fichero de nombres de usuario y contraseñas usado para autenticarse con la interfaz. Todos los comandos requieren que pcp.conf se haya configurado. Tras la instalación de pgpool-II se crea un fichero /opt/pgpool2/etc/pcp.conf.sample de ejemplo. Configurar ese fichero es tan sencillo como cambiarle el nombre al fichero y añadir el usuario y contraseña deseados.

En este artículo se usará root como nombre de usuario. Para generar la suma MD5 de nuestra contraseña podemos usar la utilidad pg_md5:

```
/opt/pgpool2/bin/pg_md5 -p
password: <password>
34b339799d540a72bf1c408c0e68afdd
```

Copiaremos el fichero de ejemplo a fichero deseado:

```
cp --archive /opt/pgpool2/etc/pcp.conf.sample /opt/pgpool2/etc/pcp.conf
```

Y lo editaremos para añadir nuestro usuario y contraseña en el formato:

```
root:34b339799d540a72bf1c408c0e68afdd
```

Luego creamos un fichero de configuración para pgpool-II a partir del que viene como ejemplo:

```
cp --archive /opt/pgpool2/etc/pgpool.conf.sample /opt/pgpool2/etc/pgpool.conf
```

Deberemos editar el fichero para configurarlo a nuestra medida. Para este artículo se configurarán las siguientes funcionalidades:

- Pool de conexiones.
- Replicación.
- Balanceo de carga.

Empezaremos con una configuración básica para arrancar pgpool-II e iremos añadiendo funcionalidades. Editamos el fichero */opt/pgpool2/etc/pgpool.conf* para dejarlo tal y como sigue:

```
listen_addresses = '*'
port = 9999
pcp_port = 9898
socket_dir = '/var/run/postgresql'
pcp_socket_dir = '/var/run/postgresql'
backend_socket_dir = '/var/run/postgresql'
pcp_timeout = 10
num_init_children = 32
max_pool = 4
child_life_time = 300
connection_life_time = 0
child_max_connections = 0
client_idle_limit = 0
authentication_timeout = 60
logdir = '/var/run/postgresql'
replication_mode = false
load_balance_mode = false
replication_stop_on_mismatch = false
replicate_select = false
reset_query_list = 'ABORT; RESET ALL; SET SESSION AUTHORIZATION DEFAULT'
print_timestamp = true
master_slave_mode = false
connection_cache = true
health_check_timeout = 20
health_check_period = 60
health_check_user = 'pgpool2'
failover_command = ''
failback_command = ''
insert_lock = false
ignore_leading_white_space = true
log_statement = false
log_connections = false
log_hostname = false
parallel_mode = false
enable_query_cache = false
pgpool2_hostname = 'pgsql1'
system_db_hostname = 'localhost'
system_db_port = 5432
system_db_dbname = 'pgpool'
system_db_schema = 'pgpool_catalog'
system_db_user = 'pgpool'
system_db_password = ''
backend_hostname0 = '192.168.0.3'
backend_port0 = 5432
backend_weight0 = 1
backend_hostname1 = '192.168.0.4'
backend_port1 = 5432
backend_weight1 = 1
enable_pool_hba = false
recovery_user = 'pgpool2'
recovery_password = ''
recovery_1st_stage_command = ''
recovery_2nd_stage_command = ''
recovery_timeout = 90
```

Todas las directivas de configuración vienen explicadas en la página web de pgpool-II. Como aspectos a destacar de la anterior configuración tenemos los siguientes:

- Mediante la directiva *listen_addresses* hacemos que, inicialmente, pgpool-II escuche en todas las interfaces.
- Mediante las directivas *logdir*, *socket_dir*, *pcp_socket_dir* y *backend_socket_dir*, configuramos, respectivamente, que el pid y todos los sockets de los diferentes procesos que forman pgpool-II se

guarden en el directorio por defecto de Debian para PostgreSQL, */var/run/postgresql*.
- Activamos el pool de conexiones (directiva connection_cache) pero dejamos todas las demás funcionalidades desactivadas (*replication_mode*, *load_balance_mode*, *replicate_select* y *master_slave_mode*).
- Mediante las directivas *health_check_timeout*, *health_check_period* y *health_check_user*, configuramos la comprobación de estado de los servidores PostgreSQL para que se haga con el usuario de base de datos pgpool2, cada 60 segundos y con un tiempo máximo de espera de 20 segundos.
- Dejamos todos los límites de conexiones, número de procesos, tiempos de espera y similares a sus valores por defecto.

El siguiente paso será crear el script de arranque de pgpool-II, que situaremos en */opt/pgpool2/etc/init.d/pgpool2*. A continuación se muestra un típico script, basado en el original del paquete pgpool de Debian:

```
#! /bin/sh

PATH=/opt/pgpool2/bin:/sbin:/bin:/usr/sbin:/usr/bin
DAEMON=/opt/pgpool2/bin/pgpool
PIDFILE=/var/run/postgresql/pgpool.pid

test -x $DAEMON || exit 0

# Include pgpool defaults if available
if [ -f /opt/pgpool2/etc/default/pgpool2 ] ; then
        . /opt/pgpool2/etc/default/pgpool2
fi

OPTS=""
if [ x"$PGPOOL_LOG_DEBUG" = x"yes" ]; then
        OPTS="$OPTS -d"
fi

. /lib/lsb/init-functions

is_running() {
        pidofproc -p $PIDFILE $DAEMON >/dev/null
}

d_start() {
        if is_running; then
                :
        else
                su -c "$DAEMON -n $OPTS 2 > &1 /dev/null | logger -t pgpool -p $
{PGPOOL_SYSLOG_FACILITY:-local0}.info > /dev/null 2 > &1 &" - postgres
        fi
}

d_stop() {
        killproc -p $PIDFILE $DAEMON -INT
        status=$?
        [ $status -eq 0 ] || [ $status -eq 3 ]
        return $?
}

case "$1" in
    start)
        log_daemon_msg "Starting pgpool-II" pgpool
        d_start
        log_end_msg $?
        ;;
    stop)
        log_daemon_msg "Stopping pgpool-II" pgpool
        d_stop
        log_end_msg $?
        ;;
    status)
```

```
        is_running
        status=$?
        if [ $status -eq 0 ]; then
                log_success_msg "pgpool-II is running."
        else
                log_failure_msg "pgpool-II is not running."
        fi
        exit $status
        ;;
    restart|force-reload)
        log_daemon_msg "Restarting pgpool-II" pgpool
        d_stop && sleep 1 && d_start
        log_end_msg $?
        ;;
    try-restart)
        if $0 status > /dev/null; then
                $0 restart
        else
                exit 0
        fi
        ;;
    reload)
        exit 3
        ;;
    *)
        log_failure_msg "Usage: $0 {start|stop|status|restart|try-restart|reload|
force-reload}"
        exit 2
        ;;
esac
```

Siguiendo el estándar Debian, crearemos el fichero */opt/pgpool2/etc/default/pgpool2* con los valores de configuración de arranque del daemon. Opcionalmente, aprovechamos para ponerlo en modo debug al arrancar:

```
# Defaults for pgpool initscript
# sourced by /opt/pgpool2/etc/init.d/pgpool2

# syslog facility for pgpool; see logger(1)
PGPOOL_SYSLOG_FACILITY=local0

# set to "yes" if you want to enable debugging messages to the log
PGPOOL_LOG_DEBUG=no
```

Ahora ya deberíamos de ser capaces de arrancar pgpool-II:

```
/opt/pgpool2/etc/init.d/pgpool2 start
```

Podremos observar el correcto arranque del daemon (o los errores en caso contrario) monitorizando el syslog, por ejemplo mediante el uso del comando tail:

```
/usr/bin/tail -f /var/log/syslog | ccze
```

A partir de este momento deberíamos de ser capaces de conectarnos al puerto 9999 de la dirección IP de administración del nodo pgsql1 (la dirección IP de servicio no estará disponible hasta que configuremos la alta disponibilidad con Heartbeat):

```
/usr/bin/psql -h pgsql1 -p 9999 -U pgpool2 -d postgres
```

Si deseamos monitorizar las conexiones a los nodos de PostgreSQL, podemos activar las directivas log_connections y log_disconnections en los ficheros de configuración */etc/postgresql/8.3/main/postgresql.conf* de cada nodo, reiniciando PostgreSQL para que los cambios surjan efecto.

Debido a que no tenemos ni replicación ni balanceo de carga activados , pgpool-II sólo se habrá conectado al

servidor PostgreSQL del nodo maestro, hecho que habremos podido comprobar monitorizando el fichero de log de PostgreSQL en */var/log/postgresql/postgresql-8.3-main.log*.

Tras haber comprobado que ya podemos conectarnos, vamos a proceder a activar la replicación y el balanceo de carga editando el fichero */opt/pgpool2/etc/pgpool.conf* y cambiando las directivas siguientes:

```
replication_mode = true
load_balance_mode = true
replication_stop_on_mismatch = true
```

Para activar los cambios reiniciaremos pgpool-II:

```
/opt/pgpool2/etc/init.d/pgpool2 restart
```

Pruebas de replicación

En el paquete *postgresql-contrib-8.3* podemos encontrar una utilidad llamada *pgbench*. Esta utilidad permite, en primer lugar, inicializar una base de datos con una serie de tablas sencillas y, en segundo lugar, realizar pruebas de rendimiento sobre servidores PostgreSQL mediante la ejecución de una cierta cantidad de consultas de varios tipos y con una concurrencia parametrizable.

A partir de este momento trabajaremos desde un tercer equipo, actuando ya como cliente del clúster. Por comodidad, daremos de alta las entradas del fichero */etc/hosts* mencionadas anteriormente en el artículo, igual que hicimos en ambos nodos del clúster. El primer paso consistirá en crear la base de datos *bench_replication*:

```
createdb -h pgsql1 -p 9999 -U pgpool2 bench_replication
createlang -h pgsql1 -p 9999 -U pgpool2 -d bench_replication plpgsql
```

Con *log_statement* y *log_connections* activados en */opt/pgpool2/etc/pgpool2.conf*, ésto nos mostrará entradas en */var/log/syslog* similares a las siguientes:

```
LOG: pid 4365: connection received: host=192.168.0.5 port=38024
LOG: pid 4365: statement: CREATE DATABASE bench_replication;
LOG: pid 4365: statement:  RESET ALL
LOG: pid 4365: statement:  SET SESSION AUTHORIZATION DEFAULT
```

Con *log_statement = 'all'* en */etc/postgresql/8.3/main/postgresql.conf*, en el log de cualquiera de los PostgreSQL nos aparecerán las siguientes líneas:

```
LOG:  connection received: host=192.168.0.3 port=33690
LOG:  connection authorized: user=pgpool2 database=postgres
LOG:  statement: CREATE DATABASE bench_replication;
LOG:  statement:  RESET ALL
LOG:  statement:  SET SESSION AUTHORIZATION DEFAULT
```

Como usuario postgres, en ambos nodos podremos usar psql para ver las bases de datos y verificar que se han creado:

```
$ su - postgres
$ psql -l
            List of databases
       Name        |  Owner   | Encoding
-------------------+----------+-----------
 bench_replication | pgpool2  | SQL_ASCII
 postgres          | postgres | SQL_ASCII
 template0         | postgres | SQL_ASCII
 template1         | postgres | SQL_ASCII
(4 rows)
```

Vamos a proceder ahora a rellenar las bases de datos con tablas e información de pruebas mediante el uso de

pgbench:

```
/usr/lib/postgresql/8.3/bin/pgbench -i -h pgsql1 -p 9999 -U pgpool2 -d
bench_replication
```

En el syslog podremos ver la siguiente información:

```
LOG: pid 4363: connection received: host=192.168.0.5 port=38124
LOG: pid 4363: statement: SET search_path = public
LOG: pid 4363: statement: drop table if exists branches
LOG: pid 4363: statement: create table branches(bid int not null,bbalance int,filler
char(88)) with (fillfactor=100)
LOG: pid 4363: statement: drop table if exists tellers
LOG: pid 4363: statement: create table tellers(tid int not null,bid int,tbalance
int,filler char(84)) with (fillfactor=100)
LOG: pid 4363: statement: drop table if exists accounts
LOG: pid 4363: statement: create table accounts(aid int not null,bid int,abalance
int,filler char(84)) with (fillfactor=100)
LOG: pid 4363: statement: drop table if exists history
LOG: pid 4363: statement: create table history(tid int,bid int,aid int,delta
int,mtime timestamp,filler char(22))
LOG: pid 4363: statement: begin
LOG: pid 4363: statement: insert into branches(bid,bbalance) values(1,0)
LOG: pid 4363: statement: insert into tellers(tid,bid,tbalance) values (1,1,0)
LOG: pid 4363: statement: insert into tellers(tid,bid,tbalance) values (2,1,0)
LOG: pid 4363: statement: insert into tellers(tid,bid,tbalance) values (3,1,0)
LOG: pid 4363: statement: insert into tellers(tid,bid,tbalance) values (4,1,0)
LOG: pid 4363: statement: insert into tellers(tid,bid,tbalance) values (5,1,0)
LOG: pid 4363: statement: insert into tellers(tid,bid,tbalance) values (6,1,0)
LOG: pid 4363: statement: insert into tellers(tid,bid,tbalance) values (7,1,0)
LOG: pid 4363: statement: insert into tellers(tid,bid,tbalance) values (8,1,0)
LOG: pid 4363: statement: insert into tellers(tid,bid,tbalance) values (9,1,0)
LOG: pid 4363: statement: insert into tellers(tid,bid,tbalance) values (10,1,0)
LOG: pid 4363: statement: commit
LOG: pid 4363: statement: begin
LOG: pid 4363: statement: truncate accounts
LOG: pid 4363: statement: copy accounts from stdin
LOG: pid 4363: statement: commit
LOG: pid 4363: statement: alter table branches add primary key (bid)
LOG: pid 4363: statement: alter table tellers add primary key (tid)
LOG: pid 4363: statement: alter table accounts add primary key (aid)
LOG: pid 4363: statement: vacuum analyze
LOG: pid 4363: statement:  RESET ALL
LOG: pid 4363: statement:  SET SESSION AUTHORIZATION DEFAULT
```

Con un sencillo script vamos a contar el número de registros insertados en cada instancia de PostgreSQL sin pasar por pgpool-II, de modo que podamos verificar que la replicación se ha realizado correctamente:

```
#!/bin/sh

PGSQL=/usr/bin/psql
HEAD=/usr/bin/head
TAIL=/usr/bin/tail
CUT=/usr/bin/cut
IP_LIST="192.168.0.3 192.168.0.4"
PORT=5432

for ip in $IP_LIST
do
    echo "ip address: $ip"
    for t in pgbench_branches pgbench_tellers pgbench_accounts pgbench_history
    do
        echo -n "table $t: "
        COUNT=`$PGSQL -h $ip -p $PORT -U pgpool2 -d bench_replication -c "SELECT
count(*) FROM $t" | $HEAD -n 3 | $TAIL -n 1`
```

```
        echo $COUNT
    done
done

exit 0
```

Para poder ver cómo se balancean las consultas, teniendo activada la directiva *log_statement = 'all'* en */etc/postgresql/8.3/main/postgresql.conf* de ambos PostgreSQL, podemos utilizar el siguiente script para ver qué consultas aparecen en el log de cada nodo:

```
#!/bin/sh

PGSQL=/usr/bin/psql
HEAD=/usr/bin/head
TAIL=/usr/bin/tail
CUT=/usr/bin/cut
IP_LIST="192.168.0.3"
PORT=9999

for ip in $IP_LIST
do
    echo "ip address: $ip"
    for t in pgbench_branches pgbench_tellers pgbench_accounts pgbench_history
    do
        echo -n "table $t: "
        COUNT=`$PGSQL -h $ip -p $PORT -U pgpool2 -d bench_replication -c "SELECT
count(*) FROM $t" | $HEAD -n 3 | $TAIL -n 1`
        echo $COUNT
    done
done

exit 0
```

En media, deberían haberse ejecutado dos (de las cuatro) consultas *SELECT* en cada una de las bases de datos, si bien esto no tiene porqué ser siempre así.

A continuación pasaremos a ejecutar el benchmark básico de *pgbench*, de modo que podamos apreciar el comportamiento del clúster bajo continuas inserciones, actualizaciones y consultas. Desde la consola ejecutaremos:

```
/usr/lib/postgresql/8.3/bin/pgbench -h pgsql1 -p 9999 -U pgpool2 -d
bench_replication -c 10 -t 1000
```

El resultado obtenido será similar al siguiente:

```
[..]
transaction type: TPC-B (sort of)
scaling factor: 1
number of clients: 1
number of transactions per client: 10
number of transactions actually processed: 10/10
tps = 119.105754 (including connections establishing)
tps = 126.179781 (excluding connections establishing)
```

Si monitorizamos el log de pgpool-II en */var/log/syslog* y los logs de ambas instancias de PostgreSQL, veremos como en el primer nodo se ejecutan todas las consultas (update, select, update, update, insert) mientras que en el segundo sólo las insert y las update. Esto se debe a que cada transacción está explícitamente declarada (BEGIN...END) y, en ese caso, pgpool-II no hace uso más que del nodo principal.

Recuperación en línea

El procedimiento de recuperación en línea de un nodo (del inglés, online recovery) permite volver a conectar nodos caídos al clúster. Es requisito, por lo tanto, que el nodo esté desconectado o caído (del inglés, dettached) antes de ejecutar este procedimiento (ver comandos pcp más adelante). Asimismo, el servidor de PostgreSQL del nodo caído debe estar parado.

En términos generales, el proceso de recuperación sigue los siguientes pasos, ejecutados en el nodo maestro (y activo):

- Activar el comando de archivado de ficheros WAL.
- Modificar el fichero */etc/postgresql/8.3/main/postgresql.conf*.
- Recargar la configuración de PostgreSQL.
- Ejecución de */opt/pgpool2/bin/pcp_recovery_node*.
    - Primera fase: copia del directorio base.
    - Segunda fase: sincronización de cambios.
    - Tercera fase: arranque del nuevo servidor.

El Write-Ahead Log

A partir de su versión 8, PostgreSQL almacena todas las operaciones que alteran los datos y los metadatos de la base de datos en forma de logs binarios en el directorio */var/lib/postgresql/8.3/pg_xlog*. Estos logs se escriben y sincronizan en disco antes de que se alteren los ficheros de las tablas (de ahí su nombre, Write-Ahead Log o WAL) y su función principal es la recuperación a un estado consistente de forma automática (de ahí su nombre, Point-In-Time Recovery o PITR) en caso de fallo del sistema por rotura de un disco, muerte de un proceso, etc. Su funcionamiento es análogo al del journaling de un sistema de ficheros. Cada uno de estos ficheros tiene un tamaño de 16 MB y PostgreSQL los rota y reutiliza automáticamente (almacena un máximo de 8 ficheros por defecto). El uso de estos logs binarios viene activado por defecto en PostgreSQL, pero no su archivado (copia a otra ubicación).

Debido a la forma en que se realiza la recuperación en línea de un nodo con pgpool-II, en realidad no es necesario tener una copia de todos los ficheros del WAL que se vayan generando, de ahí que se use un comando nulo durante la operación rutinaria del servidor. Además, el archivado de ficheros del WAL habitualmente supone un importante impacto en el rendimiento del servidor.

Para este artículo suponemos suficiente el uso de pg_dump una vez al día como mecanismo de copias de seguridad de los datos, luego no se estará haciendo uso de WAL y PITR como mecanismo de backup, sino que se utilizará únicamente para la recuperación online de un nodo. Por lo tanto, durante la operación normal del clúster, éste debe estar activo pero con un comando de archivado nulo que no haga copia alguna de los ficheros de log rotados, tal que:

```
archive_mode = on
archive_command = 'exit 0'
# archive_command = '/bin/cp %p /var/lib/postgresql/pg_xlog_archive/%f'
```

Ésto es así debido a que no puede activarse o desactivarse el uso de ficheros WAL por parte de PostgreSQL sin reiniciar el servicio (acción que queremos evitar en la medida de lo posible), mientras que cambiar el comando que se usa para el archivado de dichos ficheros requiere tan sólo una solicitud de recarga de la configuración al daemon de PostgreSQL. Según se especifica en la documentación en línea de PostgreSQL, es necesario que todo comando de archivado devuelva el valor 0 en caso de ejecución correcta y cualquier otro valor en caso contrario, de ahí que se haya optado por un simple "exit 0".

Procedimiento completo
Una vez introducidos los conceptos a utilizar, acto seguido se detalla el procedimiento completo de restauración de un nodo caído. Como paso previo activaremos el comando de archivado de ficheros WAL en el nodo maestro. Para ello, editaremos el fichero de configuración */etc/postgresql/8.3/main/postgresql.conf* y cambiaremos el valor de la directiva *archive_command*, según sigue:

```
archive_mode = on
# archive_command = 'exit 0'
archive_command = '/bin/cp %p /var/lib/postgresql/pg_xlog_archive/%f'
```

Luego solicitaremos al daemon de PostgreSQL que recargue la configuración mediante el siguiente comando:

```
/etc/init.d/postgresql-8.3 reload
```

En el nodo a recuperar, donde PostgreSQL debe estar parado, dejaremos la configuración original (comando de archivado nulo), pues no es necesario para una PITR. El siguiente paso es hacer una llamada al comando *pcp_recovery_node*, tal que:

```
/opt/pgpool2/bin/pcp_recovery_node 5 pgsql1 9898 root  1
```

En este ejemplo, la llamada se efectúa con los siguientes parámetros:

- Un timeout de 5 segundos.
- Sobre el puerto 9898.
- Al primer nodo del clúster, "pgsql1", actuando en estos momentos como maestro.
- Solicitando la recuperación del segundo nodo (el conteo empieza por el 0).

Debido a que parten de una llamada a un procedimiento almacenado de base de datos, todas las acciones que se realizan a raíz de esta llamada se ejecutan como usuario de sistema postgres. Divididas en tres fases, se detallan a continuación.

Primera fase

pgpool-II ejecuta `SELECT pgpool_recovery('recovery_1st_stage_command', 'target', 'PGDATA')` en un nodo maestro. Esto produce una llamada al script `/var/lib/postgresql/8.3/main/base-backup`. Durante la ejecución de este script se siguen atendiendo peticiones de los clientes (que PostgreSQL irá almacenando en su bitácora).

Los valores de los tres parámetros de la llamada se obtienen del fichero */opt/pgpool2/etc/pgpool.conf*. El primero de la directiva `recovery_1st_stage_command`; el segundo de la directiva *backend_hostname* correspondiente al nodo a recuperar; y el tercero de la directiva *backend_data_directory* correspondiente al nodo a recuperar.

En primer lugar, este script genera un fichero llamado `recovery.conf`, que sitúa dentro del directorio PG_DATA (`/var/lib/postgresql/8.3/main` en Debian). Este fichero se crea en esa ubicación para que sea transferido al nodo a recuperar más adelante en este mismo script. Cuando el servidor PostgreSQL del nodo a recuperar arranque y realice una recuperación PITR, leerá ese fichero y ejecutará el valor de la variable `restore_command` tantas veces como sea necesario para obtener los ficheros WAL que le permitirán llegar desde el punto en la bitácora donde finalizó la copia de seguridad online hasta el momento en el cual se dejaron de atender peticiones (segunda fase de la recuperación online de un nodo). El origen de esos ficheros WAL será el directorio `/var/lib/postgresql/pg_xlog_archive` del nodo maestro, que es donde el comando de la directiva archive_command copia los ficheros WAL que se van rotando.

Durante la ejecución de este script, pgpool-II sigue antendiendo peticiones de los clientes, de ahí que se inicie un backup online de PostgreSQL antes de empezar la copia. De este modo, las diferencias entre ambos nodos cuando el script haya terminado su ejecución serán tan sólo los ficheros WAL generados durante su ejecución, que habrán sido archivados en un directorio aparte (`/var/lib/postgresql/pg_xlog_archive`) para evitar que más de 8 rotaciones de logs durante la ejecución del script sobreescriban ficheros WAL (en el caso de mucha actividad en el servidor o de tener un volumen de datos a copiar muy grande, o de ambas).

En segundo lugar, este script indica a PostgreSQL que se va a iniciar una copia de seguridad online mediante la ejecución de la consulta `SELECT pg_start_backup('etiqueta')` en un nodo maestro. Esto, a su vez, fuerza un CHECKPOINT. Un checkpoint es un punto en la secuencia del log de transacciones a partir del cual todos los ficheros de datos han sido actualizados para reflejar la información en el log o bitácora. Todos los ficheros de datos serán escritos en disco. Un checkpoint actúa como punto de inicio para una recuperación PITR. Es decir, un checkpoint más un conjunto de ficheros WAL es todo lo que se necesita para recuperar una base de datos PostgreSQL a un estado consistente, mediante el mecanismo de PITR.

En tercer lugar, el script realiza una copia del directorio `/var/lib/postgresql/8.3/main` desde el nodo maestro al nodo a recuperar mediante el uso de varias sentencias `rsync`.

Por último, el script indica a PostgreSQL que ya ha terminado el backup online mediante la ejecución de la sentencia `SELECT pg_stop_backup()`. Tras la anotación del punto donde finaliza el backup, el puntero de inserción del fichero de log actual se desplaza al siguiente fichero de log, de modo que el fichero actual puede rotarse y archivarse, completándose así la copia de seguridad.

Segunda fase

Una vez finalizada la copia del directorio base, pgpool-II deja de atender las peticiones de los clientes, quedando éstas acumuladas en una cola de peticiones pendientes de atender, y finaliza las que estuvieran en ejecución (durante un tiempo máximo definido en la directiva `client_idle_limit_in_recovery`).

Luego pgpool-II ejecuta `SELECT pgpool_recovery('recovery_2nd_stage_command', 'target', 'PGDATA')` en un nodo maestro. Esto produce una llamada al script `/var/lib/postgresql/8.3/main/pgpool-recovery-pitr`. Los valores de los tres parámetros de la llamada se obtienen del fichero `/opt/pgpool2/etc/pgpool.conf`. El primero de la directiva `recovery_2nd_stage_command`; el segundo de la directiva `backend_hostname` correspondiente al nodo a recuperar; y el tercero de la directiva `backend_data_directory` correspondiente al nodo a recuperar.

Este script fuerza una rotación de ficheros WAL mediante la ejecución de la sentencia SQL `SELECT pg_switch_xlog()`. Esta rotación de ficheros WAL se realiza en este instante (tras haber dejado de atender peticiones de clientes) a fin de generar un estado inalterado de la base de datos que pueda ser recuperado por el nuevo nodo a partir del backup del directorio base más los ficheros WAL generados por sucesivas rotaciones desde el inicio del backup hasta esta última rotación explícita).

Tercera fase

pgpool-II ejecuta `SELECT pgpool_remote_start('target', 'PGDATA')` en el nodo maestro. Esto produce una llamada al script `/var/lib/postgresql/8.3/main/pgpool_remote_start`. Los valores de los dos parámetros de la llamada se obtienen del fichero `/opt/pgpool2/etc/pgpool.conf`. El primero de la directiva `backend_hostname` correspondiente al nodo a recuperar; y el segundo de la directiva `backend_data_directory` correspondiente al nodo a recuperar. El nombre del script a recuperar está incluido en el código fuente del fichero `/usr/local/src/pgpool-II-2.2.6/sql/pgpool-recovery` y, si se desea cambiar, debe alterarse antes de compilar e instalar.

Este script arranca PostgreSQL en el nodo que se está recuperando mediante una llamada al binario `/usr/bin/pg_ctlcluster`, y PostgreSQL hace una recuperación PITR al arrancar. El tiempo de espera para el arranque no será superior a lo que indique la directiva `recovery_timeout` (en segundos) del fichero `/opt/pgpool2/etc/pgpool.conf`.

Una vez finalizado el proceso de PITR, el nodo de PostgreSQL recuperado acepta conexiones de nuevo. pgpool-II se conecta al nuevo nodo y, a partir de este momento, el nodo forma parte del clúster de nuevo. El clúster opera de nuevo con normalidad, todas las bases de datos son accesibles de nuevo y se procesan las peticiones acumuladas durante la segunda y tercera fases de la recuperación en línea.

Pasos finales

Una vez finalizada la recuperación, y habiendo comprobado el correcto funcionamiento del clúster con su nuevo nodo, procederemos a desactivar el comando de archivado de ficheros WAL del nodo maestro, alterando el fichero de configuración `/etc/postgresql/8.3/main/postgresql.conf` y dejando el valor nulo `exit 0` del parámetro, tal y como estaba antes de iniciar el procedimiento y solicitaremos una recarga de la configuración.

En este momento, nuestro clúster acaba de finalizar la operación llamada failback, es decir, se ha recuperado de una operación de failover que se inició al detectar que uno de los nodos no estaba disponible. pgpool-II permite configurar sendos scripts para que sean ejecutados al iniciarse y finalizar, respectivamente, los eventos de failover y failback. Estas directivas se encuentran en el fichero de configuración `/opt/pgpool2/etc/pgpool.conf` y son las siguientes:

```
failover_command = ''
failback_command = ''
```

En el apartado siguiente se ofrecen ejemplos de scripts que realizan todas las tareas descritas anteriormente.

Scripts necesarios

En este apartado se muestran varios scripts que son necesarios para realizar la recuperación en línea de un nodo. La documentación de pgpool-II describe los pasos específicos a seguir, que varían dependiendo de la versión de PostgreSQL que estemos usando y de las herramientas de sistema de que dispongamos, pero deja a discreción del administrador de sistemas su implementación. Como en muchos casos, no hay una única solución que se ajuste a todos los casos.

Listados en orden de ejecución, los scripts necesarios son los siguientes:

- pgpool-failover
- wall_archiving
- base-backup
- pgpool-recovery-pitr
- pgpool_remote_start
- pgpool-failback

Todos ellos deben residir en el directorio de datos de PostgreSQL (`/var/lib/postgresql/8.3/main`) de todos los nodos.

El script wall_archiving lo ejecutaremos manualmente antes de empezar la recuperación del nodo y tras finalizar la misma, bien como usuario root, bien como usuario postgres.

Los scripts base-backup y pgpool-recovery-pitr reciben tres parámetros, en este orden:

1. Directorio de datos de origen ($PG_DATA).
2. Nombre del host a recuperar ($PG_HOST).
3. Directorio de datos de destino ($PG_DATA).

El script `pgpool_remote_start` recibe sólo los dos últimos. Los tres scripts se ejecutan en algún nodo maestro del cual pgpool-II tenga constancia y esté activo (conste como `attached`), y corren como usuario postgres.

Por último, los scripts `pgpool-failover` y `pgpool-failback`, también ejecutados como usuario postgres, reciben un conjunto de parámetros, en el orden elegido por el usuario, de entre los siguientes:

- %d: identificador de nodo.
- %h: nombre del host.
- %p: puerto.
- %D: ruta al directorio de datos del clúster
- %m: identificador del nuevo nodo maestro.
- %M: identificador del nodo maestro antiguo.

Para escapar el carácter de porcentaje (%) se usa otro porcentaje (%%).

Estos parámetros se configuran en la propia llamada al script deseado en la configuración de pgpool-II (fichero `/opt/pgpool2/etc/pgpool.conf`). Por ejemplo, para este artículo se configuraron de la siguiente manera:

```
failover_command = '/var/lib/postgresql/8.3/main/pgpool-failover %d %h %p %D %m %M'
failback_command = '/var/lib/postgresql/8.3/main/pgpool-failback %d %h %p %D %m %M'
```

Los valores %m (identificador del nuevo nodo maestro) y %M (identificador del nodo maestro antiguo) tienen mucho valor cuando el nodo que cae (durante un failover) es el nodo maestro y pgpool-II decide, a partir de aquel momento, considerar otro nodo (antes esclavo) como nuevo nodo maestro.

En todos los casos, los parámetros pasados por pgpool-II al script que se esté llamando se reciben como parámetros de la manera habitual, tratándose como en cualquier otro caso y de la forma pertinente dependiendo del lenguage de scripting utilizado.

wal_archiving

El proceso de activación y desactivación del comando de archivado puede automatizarse mediante scripts de Bash, Perl o similar, e incluso integrarse dentro de la primera y última fases de la recuperación. En realidad, es una cuestión de gustos más que otra cosa. Personalmente, prefiero tenerlos por separado y ejecutarlos manualmente. A continuación se muestra un script de Bash que realiza dicha función:

```
#!/bin/sh

SED=/bin/sed
PSQL=/usr/bin/psql
PGDIR="/etc/postgresql/8.3/main"
PGCONF="postgresql.conf"
UNAME=/bin/uname
HOST=`$UNAME -n`

test -x $PSQL || exit 0
test -f $PGDIR/$PGCONF || exit 0

case "$1" in
    start)
        echo -n "Activating WAL archiving: "
        $SED -r -e "s/\s*archive_command\s*=.*/archive_command = '\/bin\/cp %p
\/var\/lib\/postgresql\/pg_xlog_archive\/%f'/" $PGDIR/$PGCONF > /tmp/$PGCONF
        chmod 644 /tmp/$PGCONF
        mv --force /tmp/$PGCONF $PGDIR/
        /etc/init.d/postgresql-8.3 reload 2>&1 > /dev/null
        echo "done."
        ;;

    stop)
        echo -n "Deactivating WAL archiving: "
        $SED -r -e "s/\s*archive_command\s*=.*/archive_command = 'exit 0'/" $PGDIR/
$PGCONF > /tmp/$PGCONF
        chmod 644 /tmp/$PGCONF
        mv --force /tmp/$PGCONF $PGDIR/
        /etc/init.d/postgresql-8.3 reload 2>&1 > /dev/null
        echo "done."
        ;;

    status)
        # Alternate way: SELECT setting FROM pg_settings WHERE name =
'archive_command'
        am=`$PSQL -t -U pgpool2 -h $HOST -d postgres -c 'SHOW archive_mode' | $SED
-r -e 's/^\s*//;s/\s*$//'`
        ac=`$PSQL -t -U pgpool2 -h $HOST -d postgres -c 'SHOW archive_command' |
$SED -r -e 's/^\s*//;s/\s*$//'`
```

```
        echo "archive_mode = $am"
        echo "archive_command = '$ac'"
        ;;

    *)
        echo "Usage: $0 {start|stop|status}"
        exit 1
        ;;
esac

exit 0
```

Este script podría crearse en `/var/lib/postgresql/8.3/main/wall_archiving` (propiedad del usuario y grupo `postgres` y permisos 755) sería llamado como usuario `root` o `postgres` pasándole por parámetro start, stop o status. Por ejemplo:

```
$ /var/lib/postgresql/8.3/main/wall_archiving
Usage: /var/lib/postgresql/8.3/main/wall_archiving {start|stop|status}
$ /var/lib/postgresql/8.3/main/wal_archiving status
archive_mode = on
archive_command = 'exit 0'
$ /var/lib/postgresql/8.3/main/wal_archiving start
Activating WAL archiving: done.
$ /var/lib/postgresql/8.3/main/wal_archiving stop
Deactivating WAL archiving: done.
$ tail -n 2 /var/log/postgresql/postgresql-8.3-main.log | ccze -A
LOG:  incomplete startup packet
LOG:  received SIGHUP, reloading configuration files
```

pgpool-failover

En términos generales, el mecanismo de failover permite asegurar la alta disponibilidad de diversos recursos críticos (como un sistema informático o un servicio de un sistema) incluyendo un sistema de respaldo paralelo que se mantiene en ejecución en todo momento, de modo que, en caso de detectarse un fallo en el sistema primario, las tareas a procesar puedan ser automáticamente desviadas hacia el sistema de respaldo, que seguirá dando servicio a los clientes.

En este artículo se configura un clúster activo-pasivo (primario-secundario), con la salvedad de que se hace uso también del nodo secundario de forma parcial para acelerar las consultas de tipo SELECT (balanceo de carga). El procedimiento de failover en pgpool-II puede suponer tanto un failover hacia el nodo secundario (si ha fallado el primario) como una degeneración del nodo secundario (desactivación del nodo de respaldo).

En el caso de haber más de un nodo secundario, lo arriba expuesto no varía pues sólo hay un nodo maestro. El siguiente script envía dos entradas al `syslog`, que luego pueden ser capturadas por una herramienta de monitorización como Zabbix o Nagios, las cuales enviarían las alertas pertinentes. Este es un acercamiento que utilizo normalmente, alternativo al tradicional envío de una notificación por correo electrónico.

```
#! /bin/sh

LOGGER="/usr/bin/logger -i -p local0.info -t pgpool"
BASENAME=`/usr/bin/basename $0`
ID=`/usr/bin/id -un`

# $1 = node id
# $2 = host name
# $3 = port number
# $4 = database cluster path
# $5 = new master node id
# $6 = old master node id

$LOGGER "Executing $BASENAME as user $ID"
$LOGGER "Failover of node $1 at hostname $2. New master node is $5. Old master node
was $6."
```

```
exit 0
```

pgpool-II ejecuta este script como usuario `postgres` al final del proceso, una vez se ha completado la degeneración del nodo.

pgpool-failback

Análogamente, failback es el proceso mediante el cual se devuelve un sistema, componente o servicio en un estado de failover a su estado original (antes del fallo).

pgpool-II ejecuta el script configurado como usuario postgres una vez concluido el procedimiento de failback, es decir, la recuperación en línea de un nodo en tres pasos. Del mismo modo que el caso del failover, a continuación se muestra un sencillo script de Bash que añade una entrada al syslog, de modo que pueda ser capturada por una herramienta de monitorización de logs que envíe las alertas pertinentes.

```
#! /bin/sh

LOGGER="/usr/bin/logger -i -p local0.info -t pgpool"
BASENAME=`/usr/bin/basename $0`
ID=`/usr/bin/id -un`

# $1 = node id
# $2 = host name
# $3 = port number
# $4 = database cluster path
# $5 = new master node id
# $6 = old master node id

$LOGGER "Executing $BASENAME as user $ID"
$LOGGER "Failback of node $1 at hostname $2. New master node is $5. Old master node
was $6."

exit 0
```

base-backup

De entre todas las tareas a realizar por parte del script base-backup, la única que está sujeta a variación en su implementación es la copia inicial de todo el directorio de datos de PostgreSQL (`$PG_DATA`, que en Debian es `/var/lib/postgresql/8.3/main`) desde el nodo maestro al nodo a recuperar.

En el caso del script que se presenta a continuación, la herramienta elegida es rsync sobre un túnel SSH con un par de claves pública/privada DSA de 1024 bits sin contraseña. Esta elección se basa en dos puntos:

- La eficiencia de rsync (impacto sobre los discos frente al tiempo necesario), si bien el uso de un canal cifrado ralentiza el proceso.
- La mayor parte de los datos a copiar ya existirán en el nodo a recuperar. Podemos considerar que esta premisa será falsa únicamente en el caso de que el motivo de la caída del nodo hubiera sido la rotura de los discos.

A continuación se presenta un script de ejemplo:

```
#!/bin/sh

PSQL=/usr/bin/psql
SCP=/usr/bin/scp
SSH=/usr/bin/ssh
LOGGER="/usr/bin/logger -i -p local0.info -t pgpool"
RSYNC="/usr/bin/rsync --archive --quiet --compress --rsh=$SSH --delete"
BASENAME=`/usr/bin/basename $0`
```

```
HOSTNAME=`/bin/hostname`
ID=`/usr/bin/id -un`

# $1 = Database cluster path of a master node.
# $2 = Hostname of a recovery target node.
# $3 = Database cluster path of a recovery target node.

PG_HOME=/var/lib/postgresql
SRC_DATA=$1
DST_HOST=$2
DST_DATA=$3

$LOGGER "Executing $BASENAME as user $ID"

$LOGGER "Executing pg_start_backup"
$PSQL -d postgres -c "select pg_start_backup('pgpool-recovery')"

$LOGGER "Creating file recovery.conf"
echo "restore_command = '$SCP $HOSTNAME:$PG_HOME/pg_xlog_archive/%f %p'" >
$SRC_DATA/recovery.conf

$LOGGER "Rsyncing directory base"
$RSYNC $SRC_DATA/base/ $DST_HOST:$DST_DATA/base/
$LOGGER "Rsyncing directory global"
$RSYNC $SRC_DATA/global/ $DST_HOST:$DST_DATA/global/
$LOGGER "Rsyncing directory pg_clog"
$RSYNC $SRC_DATA/pg_clog/ $DST_HOST:$DST_DATA/pg_clog/
$LOGGER "Rsyncing directory pg_multixact"
$RSYNC $SRC_DATA/pg_multixact/ $DST_HOST:$DST_DATA/pg_multixact/
$LOGGER "Rsyncing directory pg_subtrans"
$RSYNC $SRC_DATA/pg_subtrans/ $DST_HOST:$DST_DATA/pg_subtrans/
$LOGGER "Rsyncing directory pg_tblspc"
$RSYNC $SRC_DATA/pg_tblspc/ $DST_HOST:$DST_DATA/pg_tblspc/
$LOGGER "Rsyncing directory pg_twophase"
$RSYNC $SRC_DATA/pg_twophase/ $DST_HOST:$DST_DATA/pg_twophase/
$LOGGER "Rsyncing directory pg_xlog"
$RSYNC $SRC_DATA/pg_xlog/ $DST_HOST:$DST_DATA/pg_xlog/
$LOGGER "Rsyncing file recovery.conf (with source deletion)"
$RSYNC --remove-source-files $SRC_DATA/recovery.conf $DST_HOST:$DST_DATA/

$LOGGER "Executing pg_stop_backup"
$PSQL -d postgres -c 'select pg_stop_backup()'

exit 0
```

pgpool-recovery-pitr

El único propósito de este script es forzar la rotación de un fichero WAL (y su consecuente archivado al directorio `/var/lib/postgresql/pg_xlog_archive`). El script `pgpool-recovery-pitr` se ejecuta una vez se han dejado de atender nuevas peticiones de clientes (que quedan en una cola de peticiones pendientes de atender) y se han atendido las que estaban en curso (o se ha llegado al timeout y se han desechado).

A continuación se presenta un script de ejemplo:

```
#! /bin/sh

PSQL=/usr/bin/psql
LOGGER="/usr/bin/logger -i -p local0.info -t pgpool"
BASENAME=`/usr/bin/basename $0`
ID=`/usr/bin/id -un`

$LOGGER "Executing $BASENAME as user $ID"
$LOGGER "Executing pg_switch_xlog"
$PSQL -d postgres -c 'select pg_switch_xlog()'
```

```
exit 0
```

pgpool_remote_start

La función del script `pgpool_remote_start` es la de arrancar remotamente la instancia de PostgreSQL en el nodo a recuperar. Para poder utilizar el mismo usuario `postgres` y su par de claves pública/privada, he optado por hacer una llamada a través de SSH directamente al binario `/usr/bin/pg_ctlcluster`. Nótese que el script de arranque sito en `/etc/init.d/postgresql-8.3` realiza la misma llamada, pero precedida de una serie de verificaciones que, en este caso, asumimos como correctas (básicamente que se dispone de todos los binarios y librerías necesarios y que no hay una instancia ya ejecutándose):

```
#! /bin/sh

SSH=/usr/bin/ssh
LOGGER="/usr/bin/logger -i -p local0.info -t pgpool"
BASENAME=`/usr/bin/basename $0`
ID=`/usr/bin/id -un`

DST_HOST=$1
DST_DIR=$2

$LOGGER "Executing $BASENAME as user $ID"
$LOGGER "Starting remote PostgreSQL server"
$SSH -T $DST_HOST '/usr/bin/pg_ctlcluster 8.3 main start' 2>/dev/null 1>/dev/null

exit 0
```

Comandos de PCP

pgpool-II proporciona una interfaz de control desde la consola mediante la cual el administrador puede recoger el estado de pgpool-II y gestionar sus procesos a través de la red. Todos los comandos PCP siguen el mismo patrón:

```
<comando_pcp> <timeout> <hostname> <puerto> <username> <password> [<núm_nodo>]
```

El número de nodo es necesario sólo en algunos comandos. Todos los comandos se encuentran en `/opt/pgpool2/bin`, por lo que se eliminará esta ruta en todos los ejemplos (queda a discreción del usuario añadir dicho directorio al PATH o preceder los comandos con la misma). El hostname será siempre el de la máquina donde se esté ejecutando pgpool-II y el puerto el 9898. El usuario y la contraseña serán los definidos en el fichero `/opt/pgpool2/etc/pcp.conf`.

A continuación se muestra el funcionamiento de algunos de estos comandos, utilizados para la gestión del clúster explicado en el artículo.

Para saber el número de nodos controlados por pgpool-II (activos o no), podemos ejecutar el siguiente comando:

```
pcp_node_count 5 pgsql1 9898 root <password>
```

Nos devolverá un número entero mayor o igual que cero. Es útil a la hora de crear scripts. Para saber el estado de un nodo podemos usar el siguiente comando:

```
pcp_node_info 5 pgsql1 9898 root <password> <núm_nodo>
```

El último parámetro será el número de nodo del cuál se desea información. Téngase en cuenta que se empieza a contar por el cero. La salida del comando ofrece cuatro campos, en este orden:

1. Nombre del host
2. Número de puerto
3. Estado
4. Peso en el balanceo de carga

El estado viene representado por un dígito de 0 a 3:

0. Este estado se usa únicamente durante la inicialización y PCP no lo mostrará nunca.
1. El nodo está activo pero no ha recibido conexiones todavía.
2. El nodo está activo y hay conexiones activas (en el pool o fondo común).
3. El nodo está caído.

Para iniciar la recuperación de un nodo usaremos el siguiente comando:

```
pcp_recovery_node 5 pgsql1 9898 root <password> <núm_nodo>
```

El último parámetro será el número de nodo a recuperar. Téngase en cuenta que se empieza a contar por el cero. El comando pcp_node_info nos permitirá conocer el estado de cada nodo (nos interesan los nodos en estado 3). Todos los comandos PCP finalizan con un código de salida 0 si todo va sobre ruedas. Si ha ocurrido algún error, deberá consultarse la siguiente tabla de códigos de error:

Nombre	Código	Descripción
unknownerr	1	error desconocido (no debería ocurrir)
eoferr	2	error de fin de fichero (end of file)
nomemerr	3	memoria insuficiente
readerr	4	error durante la lectura de datos del servidor
writeerr	5	error durante la escritura de datos al servidor
timeouterr	6	timeout
invalerr	7	parámetros del comando pcp incorrectos
connerr	8	error de conexión con el servidor
noconnerr	9	no existe ninguna conexión
sockerr	10	error de socket
hosterr	11	error de resolución del nombre de host
backenderr	12	error de proceso de pcp en el servidor (se especificó un id inválido, etc.)
autherr	13	error de autorización

Simulación de caída y recuperación de un nodo

Para simular la caída de un nodo, vamos a apagar el servidor PostgreSQL del nodo secundario:

```
/etc/init.d/postgresql-8.3 stop
```

En estos instantes, lo más probable es que pgpool-II aún no se haya dado cuenta de que ese nodo está caído. Bien podemos esperar a que se ejecute un health check (cada 60 segundos o según se haya configurado la directiva `health_check_period`) o podemos forzarlo manualmente lanzando una consulta SQL de tipo INSERT, UPDATE o DELETE. En cualquier caso, veremos aparecer las siguientes líneas en el `/var/log/syslog`:

```
ERROR: pid 27928: connect_inet_domain_socket: connect() failed: Connection refused
ERROR: pid 27928: health check failed. 1 th host 192.168.0.4 at port 5432 is down
LOG:   pid 27928: set 1 th backend down status
LOG:   pid 27928: starting degeneration. shutdown host 192.168.0.4(5432)
LOG:   pid 27928: failover_handler: do not restart pgpool. same master node 0 was
selected
LOG:   pid 27928: failover done. shutdown host 192.168.0.4(5432)
```

Con la consulta SQL siguiente obtendremos el mismo resultado:

```
psql -h 192.168.0.4 -p 9999 -U pgpool2 -d bench_replication -c "UPDATE ACCOUNTS SET
abalance = 1009 WHERE aid = 10"
```

Como puede observarse en el log, para llevar a cabo el proceso de failover, pgpool-II tan sólo tiene que degenerar el nodo caído y, tal y como informa, no es necesario reinicio alguno. A efectos prácticos, lo único que se ha perdido es la capacidad de balancear la carga. Éste es el caso más sencillo posible al que podemos enfrentarnos. Ahora, para iniciar la recuperación del nodo pgsql2, realizaremos los siguientes pasos (según lo explicado anteriormente en este artículo):

1. Activar el archivado de ficheros WAL.
2. Ejecutar el comando `pcp_recovery_node` en el nodo 0 (pgsql1) o en algún cliente con autorización en el fichero `pg_hba.conf`.
3. Desactivar el archivado de ficheros WAL y borrar aquellos que se hayan copiado al directorio `/var/lib/postgresql/pg_xlog_archive` del nodo maestro durante el proceso.

El siguiente comando instruye a pgpool-II que inicie el proceso de recuperación del nodo 1 (pgsql2):

```
/opt/pgpool2/bin/pcp_recovery_node 5 pgsql1 9898 root <password> 1
```

Éste es el log de pgpool-II (`/var/log/syslog`) producido por la ejecución del anterior comando (los comandos de failover y failback no estaban configurados en el fichero de configuración durante la ejecución):

```
LOG:   pid 27964: starting recovering node 1
LOG:   pid 27964: CHECKPOINT in the 1st stage done
LOG:   pid 27964: starting recovery command: "SELECT pgpool_recovery('base-backup',
'192.168.0.4', '/var/lib /postgresql/8.3/main')"
pgpool[28094]: Executing base-backup as user postgres
pgpool[28095]: Executing pg_start_backup
pgpool[28098]: Creating file recovery.conf
pgpool[28099]: Rsyncing directory base
pgpool[28103]: Rsyncing directory global
pgpool[28106]: Rsyncing directory pg_clog
pgpool[28109]: Rsyncing directory pg_multixact
pgpool[28112]: Rsyncing directory pg_subtrans
pgpool[28115]: Rsyncing directory pg_tblspc
pgpool[28118]: Rsyncing directory pg_twophase
pgpool[28121]: Rsyncing directory pg_xlog
pgpool[28124]: Rsyncing file recovery.conf (with source deletion)
pgpool[28127]: Executing pg_stop_backup
```

```
LOG:   pid 27964: 1st stage is done
LOG:   pid 27964: starting 2nd stage
LOG:   pid 27964: all connections from clients have been closed
LOG:   pid 27964: CHECKPOINT in the 2nd stage done
LOG:   pid 27964: starting recovery command: "SELECT pgpool_recovery('pgpool-
recovery-pitr', '192.168.0.4', '/var/lib/postgresql/8.3/main')"
pgpool[28138]: Executing pgpool-recovery-pitr as user postgres
pgpool[28147]: Executing pgpool_remote_start as user postgres
pgpool[28148]: Starting remote PostgreSQL server
LOG:   pid 27964: 1 node restarted
LOG:   pid 27964: send_failback_request: fail back 1 th node request from pid 27964
LOG:   pid 27964: recovery done
LOG:   pid 27928: starting fail back. reconnect host 192.168.0.4(5432)
LOG:   pid 27928: failover_handler: do not restart pgpool. same master node 0 was
selected
LOG:   pid 27928: failback done. reconnect host 192.168.0.4(5432)
```

Tal y como podemos ver en el log, pgpool-II realiza las siguientes acciones:

- Hace un checkpoint de la base de datos. Este checkpoint en realidad no es necesario, pues ya lo lanza el propio pg_start_backup que se ejecuta al principio del script base-backup, pero está aquí por compatibilidad con PostgreSQL 7.4.
- Ejecuta el script base-backup (primera fase de la recuperación en línea) mediante la ejecución de la función pgpool_recovery añadida durante la configuración de pgpool-II. Durante la ejecución de este fichero, sigue aceptando y atendiendo peticiones de los clientes.
- Corta las nuevas conexiones de clientes, que quedan encoladas, a la espera de poder ser atendidas.
- Espera a que se hayan atendido las peticiones que ya estaban en marcha.
- Realiza un checkpoint en la base de datos (pone fin al backup online).
- Ejecuta el script `pgpool-recovery-pitr` (segunda fase de la recuperación en línea), también mediante la función pgpool_recovery. Esto fuerza la rotación del fichero WAL actual.
- Ejecuta el script `pgpool_remote_start`.
- Reconoce el nuevo nodo y lo añade al clúster.
- Vuelve a operar con normalidad, atendiendo a las peticiones que se habían ido encolando durante la segunda fase y el inicio remoto del nuevo nodo.
- En el caso de que se hayan producido variaciones de los datos durante la primera fase de la recuperación (se hayan atendido consultas INSERT, UPDATE o DELETE), será necesario ejecutar REINDEX sobre todos los índices de la base de datos afectados una vez recuperado el nodo, puesto que las operaciones sobre índices de tipo tablas de dispersión (hash) no se guardan en el WAL.
- Si realizamos la misma operación con el nodo principal, el proceso de failover consistirá en la degeneración del nodo principal y en cambiar el papel de maestro al nodo pasivo del clúster (a partir de ahora el nodo maestro será el 1). Tras realizar la recuperación online, pgpool-II mantendrá al nodo secundario como maestro.

Alta disponibilidad de pgpool-II

Gracias a pgpool-II tenemos la posibilidad de seguir dando servico tras el fallo de N-1 servidores de PostgreSQL en nuestro clúster. Ahora bien, si la alta disponibilidad completa es uno de los requerimientos (u objetivos) de nuestro sistema, debemos garantizar la continuidad del servicio en caso de caída del middleware.

Para ello, instalaremos otra copia de pgpool-II en el nodo pgsql2, con una configuración casi idéntica a la del ya instalado, y utilizaremos Heartbeat para detectar la caída completa de un nodo (no sólo del servidor de PostgreSQL en dicho nodo) y la gestión de dicho evento. A partir de este momento, entra en juego una nueva dirección IP, la 192.168.0.2, que es la dirección IP que ambos nodos compartirán y que Heartbeat se encargará de gestionar, de modo que sólo uno de los nodos (aquel actuando como maestro) tenga configurada dicha dirección IP en un momento dado. Esta dirección IP es conocida como dirección IP de servicio.

Los pasos a seguir para instalar pgpool-II en pgsql2 son los mismos que en pgsql1:

- Bajar los fuentes al directorio `/usr/local/src`.
- Instalar las dependencias de compilación.
- Descomprimir, compilar e instalar.
- Compilar e instalar la función y la librería
- pgpool-recovery.
- Crear los ficheros de configuración y el script de arranque.

Los ficheros de configuración y el script de arranque pueden copiarse desde pgsql1 a pgsql2. El único fichero que precisará algunas modificaciones será `/opt/pgpool2/etc/pgpool.conf`. A continuación se presentan los valores que habrá que cambiar en pgsql2:

```
listen_addresses = '192.168.0.2'
pgpool2_hostname = 'pgsql2'
backend_hostname0 = '192.168.0.4'
backend_hostname1 = '192.168.0.3'
```

Y en el nodo pgsql1 tan sólo habrá que cambiar la dirección IP de servicio:

```
listen_addresses = '192.168.0.2'
```

Hay que tener en cuenta que, si en algún momento el nodo pgsql2 pasase a ser el nodo maestro, entonces los índices de los nodos se invertirían, ya que en el pgpool-II de pgsql2 los nodos están configurados al revés (pgsql1 pasaría a ser el nodo 1 y pgsql2 sería el nodo 0).

El proyecto Linux High Availability

El objetivo fundamental del proyecto Linux-HA es desarrollar una solución de alta disponibilidad (clustering) para Linux que proporcione y promocione la fiabilidad, la disponibilidad y la calidad de servicio o usabilidad (RAS, en sus siglas en inglés) a través del esfuerzo de una comunidad de desarrolladores.

El programa Heartbeat es uno de los componentes principales del proyecto. Fácilmente portable, corre en todos los Linux conocidos, así como en FreeBSD y Solaris. Heartbeat es una de las implementaciones principales del estándar Open Cluster Framework (OCF).

Heartbeat fue la primera pieza de software que se escribió para el proyecto Linux-HA. Puede llevar a cabo la detección de la caída de nodos, las comunicaciones y la gestión del clúster en un solo proceso. Actualmente soporta un modelo de dependencias muy sofisticado para clústeres de N nodos, y es muy útil y estable. La unidad de gestión de Heartbeat es el recurso. Los recursos pueden ser, por ejemplo, direcciones IP o servicios (aplicaciones). Los siguientes tipos de aplicaciones son típicos ejemplos:

- Servidores de bases de datos.
- Servidores web.
- Aplicaciones ERP.
- Servidores de correo electrónico.
- Cortafuegos.
- Servidores de ficheros.
- Servidores de DNS.
- Servidores de DHCP.
- Servidores de proxy-caché.

Un recurso es la unidad básica de la alta disponibilidad. Un recurso es un servicio o facility el cuál pasa a tener alta disponibilidad mediante el gestor de recursos del clúster de alta disponibilidad.

Un recurso es una abstracción que puede ser de diferentes tipos. Puede ser algo muy concreto, como un volumen de disco o un lector de tarjetas, o puede ser más abstracto, como una dirección IP, un conjunto de reglas de firewall o un servicio de software (como un servidor web o un servidor de base de datos).

Las operaciones básicas que los recursos deben soportar son las siguientes:

- **start**: iniciar o adquirir el control del recurso.
- **stop**: finalizar o ceder el control del recurso.
- **status**: consultar si el recurso está iniciado o parado.
- **monitor**: consultar de manera más detallada si el recurso está operando correctamente.

Nótese que los recursos del tipo R1 (versión 1 de Linux-HA) deben soportar status, mientras que los del tipo R2 (versión 2) deben soportar la operación monitor.

El gestor de recursos del clúster de alta disponibilidad intenta conseguir que todos los recursos estén disponibles para los usuarios asegurándose de que estén ejecutándose en alguno de los nodos del clúster.

El gestor de recursos de Heartbeat R1 (y muchos otros gestores de recursos en clúster) aúna diversos recursos en grupos, llamados ResourceGroups. En ese caso, cada grupo es iniciado, detenido o movido en su conjunto por el gestor de recursos del clúster.

Instalación de Heartbeat

Pasamos ahora a la instalación y configuración de la alta disponibilidad con Heartbeat. El primer paso será instalarlo en ambos nodos:

```
apt-get install heartbeat
```

Los ficheros de configuración de Heartbeat están en /etc/ha.d. Necesitamos crear tres ficheros:

- **ha.cf**: fichero de configuración principal.
- **haresources**: fichero de configuración de recursos.
- **authkeys**: información de autenticación.

Llegados a este punto, es preciso introducir dos nuevos conceptos de uso frecuente con Heartbeat, dirección IP de servicio y dirección IP de administración.

Una dirección de servicio es una dirección que es gestionada por el sistema de HA, y que es movida por el clúster allí donde los servicios correspondientes se estén ejecutando. Estas direcciones de servicio son direcciones a través de las cuales los clientes y usuarios de los servicios en HA acceden a dichos servicios. Típicamente se almacenan en DNS con nombres conocidos.

Es importante que la dirección de servicio no sea gestionada por el sistema operativo, sino que sea el software de HA el único que la maneje. Si se le da una dirección administrativa al sistema de HA para que la gestione, ésto causará problemas pues se confundirá al sistema de HA y el sistema operativo y el sistema de HA se pelearán por el control de esta dirección.

El agente de recursos IPaddr2 es capaz de levantar una interfaz desde cero, incluso si no se ha establecido ninguna dirección base (la versión anterior necesitaba de una dirección base en cualquier interfaz, pues tan sólo era capaz de añadir o eliminar direcciones a interfaces ya levantados). Además, no existe límite en el número de direcciones IP por interfaz que IPaddr2 puede gestionar.

En cambio, una dirección administrativa es una dirección que está permanentemente asociada a un nodo específico del clúster.

Tales direcciones son muy útiles, y se recomienda encarecidamente que una dirección de este tipo sea reservada para cada nodo del clúster, de manera que el administrador de sistemas pueda acceder al nodo del clúster incluso si no hay servicios ejecutándose. Para la mayoría de sistemas, una dirección de este tipo es obligatoria.

Asimismo, se recomienda que se reserve una de estas direcciones para cada interfaz, de modo que se puedan testear las interfaces incluso cuando no estén activas.

Tal y como se ha especificado al inicio de este artículo, la configuración de direcciones administrativas y de servicio es la siguiente:

- Dirección de servicio (inicialmente en pgsql1): 192.168.0.2
- Dirección administrativa de pgsql1: 192.168.0.3
- Dirección administrativa de pgsql2: 192.168.0.4

Si lo deseamos, para facilitar las pruebas o en vistas al futuro, podemos configurar la dirección de servicio en el fichero `/etc/network/interfaces` siempre y cuando no incluyamos su autoconfiguración (en forma de directiva auto o allow-hotplug). El fichero `/etc/network/interfaces` del nodo pgsql1 podría quedar tal y como sigue:

```
auto lo
iface lo inet loopback

# Dirección administrativa
allow-hotplug eth0
iface eth0 inet static
    address 192.168.0.3
    netmask 255.255.255.0
    network 192.168.0.0
    broadcast 192.168.0.255
    gateway 192.168.0.1

# Dirección de servicio
iface eth0:0 inet static
    address 192.168.0.2
    netmask 255.255.255.0
    network 192.168.0.0
    broadcast 192.168.0.255
```

De nuevo, nótese que la declaración de la dirección de servicio en el fichero `/etc/network/interfaces` es completamente prescindible al usar el agente de recursos IPaddr2. El único propósito es dejar constancia de ella en el sistema fuera de la configuración de Heartbeat.

Configuración de Heartbeat

Vamos ahora a editar los tres ficheros de configuración de Heartbeat. Tomaremos los ficheros de ejemplo de `/usr/share/doc/heartbeat` y los modificaremos a nuestro gusto. Empezaremos con el fichero `ha.cf`:

```
cd /etc/ha.d/
cp /usr/share/doc/heartbeat/ha.cf.gz /etc/ha.d/
gunzip ha.cf.gz
```

Editamos el fichero `/etc/ha.d/ha.cf` y lo configuramos a nuestro gusto, por ejemplo tal y como sigue:

```
logfacility local0
keepalive 2
deadtime 30
warntime 10
initdead 120
udpport 694
bcast eth0
auto_failback off
node pgsql1 # uname -n
node pgsql2 # uname -n
ping 192.168.0.1 # router
respawn hacluster /usr/lib/heartbeat/ipfail
```

Nos aseguramos, una vez más, de que los nodos pgsql1 y pgsql2 existen en el fichero `/etc/hosts` de ambos hosts pgsql1 y pgsql2:

```
127.0.0.1      localhost.localdomain localhost
192.168.0.1    router.dominio.com       router
192.168.0.2    pgpool2.dominio.com      pgpool2
192.168.0.3    pgsql1.dominio.com       pgsql1
192.168.0.4    pgsql2.dominio.com       pgsql2
```

Editamos el fichero `/etc/ha.d/haresources` y añadimos la siguiente línea:

```
pgsql1 192.168.0.2 pgpool2
```

Esto indica a Heartbeat que el nodo maestro es pgsql1 y que debe gestionar dos recursos:

- La dirección IP de servicio 192.168.0.2.
- El servicio pgpool2.

El orden es muy importante. Primero se especifica el hostname del nodo que consideramos maestro. Segundo, los recursos. El orden de los recursos también es crítico, pues Heartbeat los iniciará en orden de izquierda a derecha, y los detendrá en orden de derecha a izquierda (y no queremos que intente arrancar el servicio `pgpool2` antes de disponer de la dirección IP en la cual pgpool-II debe escuchar).

Heartbeat buscará el script de arranque del servicio que debe gestionar en `/etc/init.d` y en `/etc/ha.d/resource.d`. Siempre y cuando no creemos los enlaces en los directorios `/etc/rcX.d`, tanto da que lo coloquemos en uno u otro, pues el sistema operativo no lo arrancará automáticamente. Entonces, creamos un enlace débil al script de arranque `pgpool2` para que Heartbeat pueda encontrarlo:

```
cd /etc/ha.d/resource.d
ln --symbolic /opt/pgpool2/etc/init.d/pgpool2
```

De este modo pgpool2 no se arrancará al iniciarse el nodo, sino que será Heartbeat quien lo arranque si detecta que así tiene que hacerlo, es decir, si decide que éste nodo debe asumir el papel de maestro.

A continuación editamos el fichero `/etc/ha.d/authkeys`:

```
auth 1
1 sha1 57ef7ac02cf6aef7e13131622598f94778fb07d6
```

Para obtener la clave SHA1 de la contraseña en texto plano que querramos usar, utilizaremos el comando sha1sum:

```
$ echo -n "<password>" | sha1sum
57ef7ac02cf6aef7e13131622598f94778fb07d6  -
```

Authkeys es obligatorio que sea accesible sólo por el usuario root y que tenga permisos 600. Los demás, 664. Les damos los permisos adecuados a los ficheros:

```
chown root:root /etc/ha.d/authkeys /etc/ha.d/haresources /etc/ha.d/ha.cf
chmod 600 /etc/ha.d/authkeys
chmod 664 /etc/ha.d/haresources /etc/ha.d/ha.cf
```

Finalmente, configuraremos el logger daemon, específico de Heartbeat. Tomamos el fichero de ejemplo en `/usr/share/doc/heartbeat`:

```
cp --archive /usr/share/doc/heartbeat/logd.cf /etc/
```

Editamos el fichero `/etc/logd.cf`:

```
debugfile /var/log/ha-debug
logfile /var/log/ha-log
logfacility daemon
```

```
entity logd
useapphbd no
sendqlen 256
recvqlen 256
```

Repetiremos los anteriores pasos para el nodo pgsql2. Los ficheros de configuración serán idénticos en ambos nodos, sin diferencia alguna, por lo que podemos copiar los ficheros de configuración desde pgsql1 a pgsql2 sin problemas.

Ahora ya estamos listos para iniciar la alta disponibilidad. Debido a que el logd puede estar iniciado con una configuración por defecto, vamos a asegurarnos primero de que Heartbeat está completamente parado en ambos nodos:

```
/etc/init.d/heartbeat stop
```

Es posible que debamos esperar a algún timeout. Ahora, con una diferencia máxima de 120 segundos (directiva initdead en `/etc/ha.d/ha.cf`), ejecutaremos el script de inicio, primero en pgsql1 y después en pgsql2:

```
/etc/init.d/heartbeat start
```

Podemos monitorizar el arranque de ambos Heartbeats a través del fichero de log `/var/log/ha-log`. En el nodo pgsql1 debería aparecernos la siguiente (o parecida) información en dicho log:

```
logd[4197]: info: logd started with /etc/logd.cf.
logd[4197]: WARN: Core dumps could be lost if multiple dumps occur.
logd[4197]: WARN: Consider setting non-default value in
/proc/sys/kernel/core_pattern (or equivalent) for maximum supportability
logd[4197]: WARN: Consider setting /proc/sys/kernel/core_uses_pid (or equivalent) to
1 for maximum supportability
logd[4197]: info: G_main_add_SignalHandler: Added signal handler for signal 15
logd[4197]: info: G_main_add_SignalHandler: Added signal handler for signal 15
heartbeat[4272]: info: Enabling logging daemon
heartbeat[4272]: info: logfile and debug file are those specified in logd config
file (default /etc/logd.cf)
heartbeat[4272]: WARN: Core dumps could be lost if multiple dumps occur.
heartbeat[4272]: WARN: Consider setting non-default value in
/proc/sys/kernel/core_pattern (or equivalent) for maximum supportability
heartbeat[4272]: WARN: Consider setting /proc/sys/kernel/core_uses_pid (or
equivalent) to 1 for maximum supportability
heartbeat[4272]: info: Version 2 support: false
heartbeat[4272]: WARN: logd is enabled but logfile/debugfile/logfacility is still
configured in ha.cf
heartbeat[4272]: info: **************************
heartbeat[4272]: info: Configuration validated. Starting heartbeat 2.1.3
heartbeat[4273]: info: heartbeat: version 2.1.3
heartbeat[4273]: info: Heartbeat generation: 1225899638
heartbeat[4273]: info: glib: UDP Broadcast heartbeat started on port 694 (694)
interface eth0
heartbeat[4273]: info: glib: UDP Broadcast heartbeat closed on port 694 interface
eth0 - Status: 1
heartbeat[4273]: info: glib: ping heartbeat started.
heartbeat[4273]: info: G_main_add_TriggerHandler: Added signal manual handler
heartbeat[4273]: info: G_main_add_TriggerHandler: Added signal manual handler
heartbeat[4273]: info: G_main_add_SignalHandler: Added signal handler for signal 17
heartbeat[4273]: info: Local status now set to: 'up'
heartbeat[4273]: info: Link 192.168.0.1:192.168.0.1 up.
heartbeat[4273]: info: Status update for node 192.168.0.1: status ping
heartbeat[4273]: info: Link pgsql1:eth0 up.
heartbeat[4273]: info: Link pgsql2:eth0 up.
heartbeat[4273]: info: Status update for node pgsql2: status up
harc[4283][4289]: info: Running /etc/ha.d/rc.d/status status
heartbeat[4273]: info: Comm_now_up(): updating status to active
heartbeat[4273]: info: Local status now set to: 'active'
heartbeat[4273]: info: Starting child client "/usr/lib/heartbeat/ipfail" (107,111)
```

```
heartbeat[4273]: info: Starting "/usr/lib/heartbeat/ipfail" as uid 107  gid 111 (pid
4294)
heartbeat[4273]: info: Status update for node pgsql2: status active
harc[4298][4303]: info: Running /etc/ha.d/rc.d/status status
ipfail[4294]: info: Status update: Node pgsql2 now has status active
ipfail[4294]: info: Asking other side for ping node count.
heartbeat[4273]: info: remote resource transition completed.
heartbeat[4273]: info: remote resource transition completed.
heartbeat[4273]: info: Initial resource acquisition complete (T_RESOURCES(us))
IPaddr[4346][4374]: INFO:  Resource is stopped
heartbeat[4311]: info: Local Resource acquisition completed.
harc[4378][4383]: info: Running /etc/ha.d/rc.d/ip-request-resp ip-request-resp
ip-request-resp[4378][4388]: received ip-request-resp 192.168.0.2 OK yes
ResourceManager[4389][4399]: info: Acquiring resource group: pgsql1 192.168.0.2
pgpool2
IPaddr[4411][4439]: INFO:  Resource is stopped
ResourceManager[4389][4455]: info: Running /etc/ha.d/resource.d/IPaddr 192.168.0.2
start
IPaddr[4472][4502]: INFO: Using calculated nic for 192.168.0.2: eth0
IPaddr[4472][4507]: INFO: Using calculated netmask for 192.168.0.2: 255.255.255.0
IPaddr[4472][4529]: INFO: eval ifconfig eth0:0 192.168.0.2 netmask 255.255.255.0
broadcast 192.168.0.255
IPaddr[4457][4548]: INFO:  Success
ResourceManager[4389][4574]: info: Running /etc/ha.d/resource.d/pgpool2  start
ipfail[4294]: info: No giveup timer to abort.
```

En el nodo pgsql2 veremos información similar a la que sigue en el fichero de log:

```
logd[3793]: info: logd started with /etc/logd.cf.
logd[3793]: WARN: Core dumps could be lost if multiple dumps occur.
logd[3793]: WARN: Consider setting non-default value in
/proc/sys/kernel/core_pattern (or equivalent) for maximum supportability
logd[3793]: WARN: Consider setting /proc/sys/kernel/core_uses_pid (or equivalent) to
1 for maximum supportability
logd[3794]: info: G_main_add_SignalHandler: Added signal handler for signal 15
logd[3793]: info: G_main_add_SignalHandler: Added signal handler for signal 15
heartbeat[3868]: info: Enabling logging daemon
heartbeat[3868]: info: logfile and debug file are those specified in logd config
file (default /etc/logd.cf)
heartbeat[3868]: WARN: Core dumps could be lost if multiple dumps occur.
heartbeat[3868]: WARN: Consider setting non-default value in
/proc/sys/kernel/core_pattern (or equivalent) for maximum supportability
heartbeat[3868]: WARN: Consider setting /proc/sys/kernel/core_uses_pid (or
equivalent) to 1 for maximum supportability
heartbeat[3868]: info: Version 2 support: false
heartbeat[3868]: WARN: logd is enabled but logfile/debugfile/logfacility is still
configured in ha.cf
heartbeat[3868]: info: **************************
heartbeat[3868]: info: Configuration validated. Starting heartbeat 2.1.3
heartbeat[3869]: info: heartbeat: version 2.1.3
heartbeat[3869]: info: Heartbeat generation: 1225899699
heartbeat[3869]: info: glib: UDP Broadcast heartbeat started on port 694 (694)
interface eth0
heartbeat[3869]: info: glib: UDP Broadcast heartbeat closed on port 694 interface
eth0 - Status: 1
heartbeat[3869]: info: glib: ping heartbeat started.
heartbeat[3869]: info: G_main_add_TriggerHandler: Added signal manual handler
heartbeat[3869]: info: G_main_add_TriggerHandler: Added signal manual handler
heartbeat[3869]: info: G_main_add_SignalHandler: Added signal handler for signal 17
heartbeat[3869]: info: Local status now set to: 'up'
heartbeat[3869]: info: Link 192.168.0.1:192.168.0.1 up.
heartbeat[3869]: info: Status update for node 192.168.0.1: status ping
heartbeat[3869]: info: Link pgsql2:eth0 up.
heartbeat[3869]: info: Link pgsql1:eth0 up.
heartbeat[3869]: info: Status update for node pgsql1: status up
heartbeat[3869]: info: Status update for node pgsql1: status active
```

```
heartbeat[3869]: info: Comm_now_up(): updating status to active
heartbeat[3869]: info: Local status now set to: 'active'
heartbeat[3869]: info: Starting child client "/usr/lib/heartbeat/ipfail" (107,111)
heartbeat[3881]: info: Starting "/usr/lib/heartbeat/ipfail" as uid 107  gid 111 (pid
3881)
harc[3880][3889]: info: Running /etc/ha.d/rc.d/status status
harc[3894][3899]: info: Running /etc/ha.d/rc.d/status status
heartbeat[3869]: info: local resource transition completed.
heartbeat[3869]: info: Initial resource acquisition complete (T_RESOURCES(us))
heartbeat[3904]: info: No local resources [/usr/share/heartbeat/ResourceManager
listkeys pgsql2] to acquire.
heartbeat[3869]: info: remote resource transition completed.
ipfail[3881]: info: Ping node count is balanced.
```

Podemos observar como Heartbeat se ha encargado de asociar la dirección IP 192.168.0.2 al primer alias disponible de la interfaz `eth0`, es decir, `eth0:0`. También podremos observar como pgpool-II está levantado y es capaz de servir conexiones.

En el syslog podremos observar que ha establecido la conexión con ambos servidores de PostgreSQL y está esperando peticiones:

```
$ tail -n 500 /var/log/syslog | grep pgpool | ccze -A
pgsql1 pgpool: LOG: pid 4594: pgpool successfully started
```

Simulación de la caída del servicio

A continuación vamos a simular la caída del nodo maestro, pgsql1. Si estamos trabajando con dos máquinas, esto es tan sencillo como desconectar dicho nodo de la red. En el caso de este artículo, se usó Xen Hypervisor para crear las dos máquinas virtuales dentro de la misma máquina física, por lo que se ejecutó el siguiente comando (las máquinas virtuales se habían creado con las xen-tools de Steve Kemp):

```
xm destroy pgsql1
```

Al cabo de un máximo de 30 segundos, deberíamos ver información parecida a la siguiente en el fichero de log de Heartbeat del nodo pgsql2:

```
heartbeat[3869] WARN: node pgsql1: is dead
ipfail[3881] 2008/11/05_17:08:10 info: Status update: Node pgsql1 now has status
dead
heartbeat[3869] WARN: No STONITH device configured.
heartbeat[3869] WARN: Shared disks are not protected.
heartbeat[3869] info: Resources being acquired from pgsql1.
heartbeat[3869] info: Link pgsql1:eth0 dead.
harc[3942][3954] info: Running /etc/ha.d/rc.d/status status
heartbeat[3943] info: No local resources [/usr/share/heartbeat/ResourceManager
listkeys pgsql2] to acquire.
mach_down[3964][3984] info: Taking over resource group 192.168.0.2
ResourceManager[3985][3995] info: Acquiring resource group: pgsql1 192.168.0.2
pgpool2
IPaddr[4007][4035] INFO:  Resource is stopped
ResourceManager[3985][4051] info: Running /etc/ha.d/resource.d/IPaddrpg_hba
192.168.0.2 start
IPaddr[4068][4098] INFO: Using calculated nic for 192.168.0.2: eth0
IPaddr[4068][4103] INFO: Using calculated netmask for 192.168.0.2: 255.255.255.0
IPaddr[4068][4125] INFO: eval ifconfig eth0:0 192.168.0.2 netmask 255.255.255.0
broadcast 192.168.0.255
IPaddr[4053][4144] INFO:  Success
ResourceManager[3985][4172] info: Running /etc/ha.d/resource.d/pgpool2  start
mach_down[3964][4199] info: /usr/share/heartbeat/mach_down: nice_failback: foreign
resources acquired
mach_down[3964][4203] info: mach_down takeover complete for node pgsql1.
heartbeat[3869] info: mach_down takeover complete.
```

```
ipfail[3881] info: NS: We are still alive!
ipfail[3881] info: Link Status update: Link pgsql1/eth0 now has status dead
ipfail[3881] info: Asking other side for ping node count.
ipfail[3881] info: Checking remote count of ping nodes.
```

Vemos como ahora el nodo pgsql2 tiene la dirección IP 192.168.0.2 (podemos verificarlo con el comando *ifconfig*) y como pgpool-II está arrancado (podemos verificarlo con el comando *ps xua |grep ^postgres*). Asimismo, podemos observar como pgpool-II ha realizado un health check nada más iniciarse y, al ver que el servidor de PostgreSQL del nodo 192.168.0.3 no estaba disponible, ha realizado la pertinente degeneración de ese nodo (que, recordemos, a efectos del nuevo pgpool-II, es un nodo secundario):

```
$ tail -n 500 /var/log/syslog | grep pgpool | ccze -A
pgsql2 pgpool: LOG:   pid 4195: pgpool successfully started
pgsql2 pgpool: ERROR: pid 4195: connect_inet_domain_socket: connect() failed: No
route to host
pgsql2 pgpool: ERROR: pid 4195: health check failed. 1 th host 192.168.0.3 at port
5432 is down
pgsql2 pgpool: LOG:   pid 4195: set 1 th backend down status
pgsql2 pgpool: LOG:   pid 4195: starting degeneration. shutdown host
192.168.0.3(5432)
pgsql2 pgpool: LOG:   pid 4195: failover_handler: do not restart pgpool. same master
node 0 was selected
pgsql2 pgpool: LOG:   pid 4195: failover done. shutdown host 192.168.0.3(5432)
pgsql2 pgpool: LOG:   pid 4195: execute command:
/var/lib/postgresql/8.3/main/pgpool-failover 1 192.168.0.3 5432
/var/lib/postgresql/8.3/main 0 0
pgsql2 pgpool[4243]: Executing pgpool-failover as user postgres
pgsql2 pgpool: /var/lib/postgresql/8.3/main/pgpool-failover: 15: Failover of node 1
at hostname 192.168.0.3. New master node is 0. Old master node was 0.: not found
```

Nótese que, a raíz de la instalación de Heartbeat, pgpool-II está ahora escuchando en la IP 192.168.0.2, por lo cual hay que mandar al puerto 9898 de esa IP los comandos de PCP. La ventaja es que siempre lanzaremos nuestros comandos PCP contra esa dirección IP, pues Heartbeat se encargará de llevarla de un nodo a otro según sea necesario. Además, las peticiones de pgpool-II a PostgreSQL vienen ahora desde esa dirección IP, por lo que es conveniente revisar que nuestros ficheros */etc/postgresql/8.3/main/pg_hba.conf* permiten las conexiones desde ambas con el usuario *pgpool2*.

Por otra parte, los comandos que nuestros clientes quieran mandar contra el clúster usarán, a partir de ahora, la dirección IP 192.168.0.2 y el puerto 9999. Al igual que antes, dado que Heartbeat se encargará de mover esta dirección IP y el servicio pgpool2 de nodo a nodo, nuestros clientes siempre funcionarán de la misma manera. Este es uno de los propósitos principales de la alta disponibilidad que acabamos de configurar (además de la alta disponibilidad de la que gozamos, por supuesto).

Podemos probar nuestro clúster con un simple comando como el que sigue:

```
psql -h 192.168.0.2 -p 9999 -U pgpool2 -d bench_replication -c "UPDATE ACCOUNTS SET
abalance = 1009 WHERE aid = 10"
```

Herramientas de monitorización

En Internet pueden encontrarse un gran número de herramientas de gestión y monitorización de PostgreSQL, tanto de código abierto como propietarias. A continuación se presenta una lista de algunas de ellas que, personalmente, me resultan de gran utilidad.

pg_osmem y pg_buffercache

pg_osmem es un script hecho en Python por Kenny Gorman que permite obtener información sobre la memoria caché que un sistema Linux está dedicando al servidor PostgreSQL. Utiliza un script en Perl llamado fincore (pronunciado en inglés *eff in core*), creado por David Plonka, Archit Gupta y Dale Carder de la universidad de Wisconsin-Madison y que fue presentado en la LISA '07.

Su instalación y uso son muy sencillos:

- Crear el directorio */root/bin* y cambiarse a él.
- Descargar fincore desde http://net.doit.wisc.edu/~plonka/fincore/fincore
- Crear el fichero */root/pg_osmem/pg_osmem* a partir del script que hay en la página web de Kenny Gorman.
- Crear el directorio */root/pg_osmem/data*.
- Instalar la dependencia *psycopg2* (paquete *python-psycopg2* en Debian).
- Instalar la dependiencia *inline* (paquete *libinline-perl* en Debian).
- Modificar el script */root/pg_osmem/pg_osmem* para adecuarlo a nuestro sistema:
    - Cambiar */home/postgres/python/bin/python* por */usr/bin/python*.
    - Cambiar la variable *fincore* por */root/bin/fincore*.
    - Cambiar la variable *mydir* por */var/lib/postgresql/8.3/main/base*.
- Ejecutar con la siguiente sentencia (suponemos que */root/bin* está en el PATH del usuario *root*): *pg_osmem --username=pgpool2 --password='' --machine=pgsql1 --dbname=bench_replication*

El fichero de configuración */etc/postgresql/8.3/main/pg_hba.conf* deberá estar configurado adecuadamente.

pg_buffercache es un módulo de PostgreSQL, habitualmente hallado en el paquete *postgresql-contrib-8.3*, que proporciona maneras de examinar lo que está ocurriendo en la cache de memoria compartida de PostgreSQL en tiempo real (no a nivel de sistema operativo, pues para eso necesitamos *pg_osmem*). Su instalación es muy sencilla:

```
$ psql -d bench_replication < /usr/share/postgresql/8.3/contrib/pg_buffercache.sql
SET
CREATE FUNCTION
CREATE VIEW
REVOKE
REVOKE
```

Entonces, mediante el script pg_osmem podemos obtener una salida similar a la siguiente:

```
$ pg_osmem --username=pgpool2 --password='' --machine=pgsql1
--dbname=bench_replication
OS Cache Usage:
bench_replication:accounts_pkey:566272
bench_replication:pg_proc:102400
bench_replication:pg_proc_proname_args_nsp_index:79872
bench_replication:pg_depend:79872
bench_replication:pg_depend_reference_index:65536
[..]
```

Luego, gracias al módulo *pg_buffercache*, podemos lanzar la siguiente consulta SQL:

```
psql -d bench_replication
# SELECT current_database(),c.relname, count(*)*8192 as bytes
    FROM pg_buffercache b INNER JOIN pg_class c
      ON b.relfilenode = c.relfilenode
     AND b.reldatabase IN (0, (
         SELECT oid FROM pg_database
          WHERE datname = current_database()))
GROUP BY c.relname
ORDER BY 3 DESC LIMIT 25;
 current_database  |    relname    |  bytes
-------------------+---------------+----------
 bench_replication | accounts      | 13434880
 bench_replication | accounts_pkey |   262144
 bench_replication | pg_attribute  |   155648
 bench_replication | pg_statistic  |   122880
 bench_replication | pg_operator   |   106496
[..]
(25 rows)
```

Gracias a estos dos códigos, podemos ver el conjunto de buffers más utilizado en la caché del sistema operativo y compararlo con la caché de PostgreSQL. La salida de ambos comandos es en bytes.

pg_top

pg_top es un *top* para PostgreSQL, derivado del *top* de UNIX. Similar a *top*, pg_top permite monitorizar los procesos de PostgreSQL y, además, ofrece estas otras funcionalidades:

- Ver las consultas SQL que está ejecutando un proceso.
- Ver el query plan de una consulta SQL que se está ejecutando.
- Ver los bloqueos realizados por un proceso.
- Ver la estadística de las tablas.
- Ver la estadística de los índices.

Los fuentes de pg_top pueden descargarse de su página web en PgFoundry. También podemos bajarnos la última versión de desarrollo del repositorio Git de pgFoundry mediante el siguiente comando:

```
cd /usr/local/src
git clone git://git.postgresql.org/git/pg_top.git
cd pg_top
./autogen.sh
```

Podemos instalar Git mediante el siguiente comando:

```
apt-get install git-cvs
```

Instalaremos las dependencias de compilación con el siguiente comando:

```
apt-get install libncurses5-dev
```

Los paquetes *build-essential*, *linux-headers-server*, *libpq-dev*, *libpq5* y *postgresql-server-dev-8.3* ya los habíamos instalado durante la instalación de pgpool-II y de PostgreSQL.

pg_top se instala por defecto en */usr/local/bin*, pero podemos cambiar la ruta mediante el parámetro *--prefix* del script *configure*:

```
cd /usr/local/src
wget http://pgfoundry.org/frs/download.php/1780/pg_top-3.6.2.tar.bz2
tar -xjf pg_top-3.6.2.tar.bz2
cd pg_top-3.6.2
./configure --prefix=/opt/pg_top
make
make install
```

El uso de pg_top es análogo al de top. Tan sólo hay que invocar el ejecutable y usar las teclas para obtener diferentes vistas e información de los procesos deseados. El ejecutable requiere tres parámetros:

- La base de datos (parámetro -d).
- El usuario (parámetro -U).
- La contraseña, si es necesaria (parámetro -W).

Entonces, la ejecución de pg_top sería tal y como sigue:

```
/opt/pg_top/bin/pg_top -U pgpool2 -d bench_replication
```

pg_top asume *localhost* como hostname y 5432 como puerto por defecto. Usa el usuario de sistema como medio de autenticación por defecto, por lo cual deberemos pasarle datos de usuario y contraseña o ejecutarlo como usuario *postgres*, dependiendo de la configuración que tengamos en */etc/postgresql/8.3/main/pg_hba.conf*.

Podemos crearnos un script */opt/pg_top/pg_top* donde hacer la llamada completa con todos los datos y hacernos el trabajo más sencillo, por ejemplo:

```
#!/bin/bash
/usr/bin/sudo /bin/su - postgres -c '/opt/pg_top/bin/pg_top'
```

Y luego añadir la ruta al PATH del sistema en */etc/profile*.

ps

ps, o process status, la famosa utilidad presente en todos los UNIX, es una útil herramienta para tener una primera impresión de qué está haciendo PostgreSQL. Un sencillo comando tal que el que sigue será suficiente:

```
$ ps auxww | grep ^postgres
postgres   960  0.0  1.1  6104 1480 pts/1    SN   13:17   0:00 postgres -i
postgres   963  0.0  1.1  7084 1472 pts/1    SN   13:17   0:00 postgres: writer
process
postgres   965  0.0  1.1  6152 1512 pts/1    SN   13:17   0:00 postgres: stats
collector process
postgres   998  0.0  2.3  6532 2992 pts/1    SN   13:18   0:00 postgres: tgl runbug
127.0.0.1 idle
postgres  1003  0.0  2.4  6532 3128 pts/1    SN   13:19   0:00 postgres: tgl
regression [local] SELECT waiting
postgres  1016  0.1  2.4  6532 3080 pts/1    SN   13:19   0:00 postgres: tgl
regression [local] idle in transaction
```

pgd

pgstat, otra utilidad desarrollada por Kenny Gorman, es muy similar a iostat pero orientada a bases de datos. Es una utilidad muy útil para realizar diagnósticos rápidos, tests de rendimiento, etc. La salida es adecuada para ser importada en hojas de cálculo o en programas que puedan generar gráficas.

En la página web de Kenny Gorman se explica cómo obtener gráficas de la salida de pgd usando gnuplot. Ver la bibliografía para el enlace.

La instalación de pgd es muy sencilla:

- Descargar pgstat desde pgFoundry.
- Descomprimirlo en /root/bin o donde consideremos oportuno.
- Editar el fichero y cambiar los valores por defecto de los parámetros en la línea 16. De esta forma podremos llegar a no tener que especificar ningún parámetro o, como máximo, la base de datos.

La utilidad la llamaremos con un sencillo *pgstat -d bench_replication*.

iotop

Iotop es un script de Python con una interfaz de usuario similar a la de top que sirve para mostrar qué proceso está originando qué carga de lecturas y escrituras de disco (E/S). Requiere Python 2.5 o superior y un kernel de Linux versión 2.6.20 o superior. Este script muestra la misma información que el comando vmstat, pero asociando la carga al proceso que la genera y mostrando la información de una forma mucho más útil para el administrador de sistemas.

Su instalación es muy sencilla pues, a partir de Lenny, viene como paquete Debian:

```
apt-get install iotop python-pkg-resources
```

Bibliografía

- HomePage: Linux HA
- Managing Kernel Resources
- El blog de Juanky Moral: PostgreSQL: HW Tunning
- PostgreSQL Hardware Performance Tuning
- PostgreSQL: The world's most advanced open source database
- PostgreSQL: Documentation: Manuals: Performance Tips
- pgpool-II README
- PgFoundry: pgpool: Project Info
- PgFoundry: pgpool: Mailing Lists for pgpool
- Python script showing PostgreSQL objects in linux memory: pg_osmem.py | kennygorman.com
- pgd: a database utility like iostat | kennygorman.com
- Fincore
- PostgreSQL top (pg_top): Project Home Page
- Graphing pgd output | kennygorman.com
- Iotop's homepage
- Setting shared buffers
- PostgreSQL Performance Tuning
- Performance Tuning PostgreSQL
- psql tricks
- pgFincore

Historial de revisiones

Fecha	Versión	Cambios
2008-10-01	1.0	Documento inicial.
2009-07-09	1.1	Actualizado a pgpool-II versión 2.2.2.
2009-12-13	1.2	Actualizado a pgpool-II version 2.2.6. Añadida nota al principio del apartado de recuperación en línea especificando que PostgreSQL debe estar parado en el nodo a recuperar.