Facultad de Ciencias Sociales y Administrativas

Ingeniería en Software

Tesis de grado

CONFIABILIDAD, RECUPERACIÓN DE FALLOS Y OPTIMIZACIÓN DE BASES DE DATOS “POSTGRES” EN SERVIDORES UTILIZANDO SISTEMAS DE REPLICACIÓN

Alumno: Marcelo A. Giuntoli

Legajo Nº: 16624

Tutor: Mgter. Alejandro Vazquez

CALIFICACIÓN

AGRADECIMIENTOS

Agradezco principalmente a Dios, a mi familia y los profesores que marcaron mi camino.

Tabla de Contenido

1- INTRODUCCIÓN

1.1 RESUMEN TÉCNICO

En este trabajo expongo la replicación de base de datos como un método para obtener mayor seguridad y disponibilidad de los datos almacenados en las distintas bases de datos Postgres. Como objetivo de este trabajo demuestro que utilizar replicación en bases de datos logra una mayor disponibilidad de los datos. Esto queda demostrado por medio de sistemas operativos (*Linux*) instalados en máquinas virtuales, los cuales cuentan con la configuración que se investiga según el tema que se desarrolla.

Este tipo de tecnología beneficia a los administradores de bases de datos, a los propietarios de la información y a los usuarios finales, dado que esto permite a los administradores de bases de datos manejar las bases de datos de una manera más segura, a los propietarios de la información de que garantizan la integridad de sus datos, dado que evita perdidas de información y que los usuarios de la misma cuentan con los datos cada vez que lo necesitan, aun cuando existen fallas en los servidores de bases de datos.

La replicación es el proceso de copiar y mantener objetos de la base de datos, como por ejemplo relaciones, en múltiples bases de datos que forman un sistema de bases de datos distribuido.

1.2 DENOMINACIÓN DEL PROYECTO

Confiabilidad, recuperación de fallos y optimización de bases de datos “Postgres” en servidores utilizando sistemas de replicación.

1.3 PALABRAS CLAVE

Servidor, bases de datos, replicación, disponibilidad, recuperación de fallos, confiabilidad, optimización.

1.4 DISCIPLINA

Las disciplinas del trabajo son las Bases de Datos. Este material de trabajo se centrará en la aplicación de modos de replicación de Bases de Datos Postgres instaladas en servidores Linux.

2- DESCRIPCION DEL PROYECTO

2.1 FORMULACIÓN Y FUNDAMENTACIÓN DEL PROBLEMA

En la actualidad, el Centro de Informática de la Universidad Nacional de Cuyo no cuenta con un sistema de base de datos que garantice la alta disponibilidad de los datos ante posibles fallas de hardware o de software, por lo que si se produjera algún desperfecto no se podrían realizar consultas sobre dichas bases de datos. Frente a esto, el administrador de bases de datos, tiene como desafío lograr en lo posible que el sistema de bases de datos sea consistente y se encuentre disponible siempre que sea requerido.

La base del problema radica en el hecho de que actualmente en el Centro de Informática de la Universidad Nacional de Cuyo sólo se dispone de backups que se realizan de manera periódica, por lo que si se interrumpe el servicio, este deberá estar detenido durante el lapso en que se logre restablecer el mismo o durante el tiempo que tome restaurar un backup corriendo el riesgo de perder los datos guardados en la base luego de realizada la última copia de seguridad.

2.2 HIPÓTESIS DE INVESTIGACIÓN

Mediante un sistema de replicación de datos en Postgres es posible aumentar la disponibilidad de las bases de datos, logrando un grado de consistencia en los datos replicados en estas bases de datos bajo condiciones de tolerancia a fallas.

2.3 OBJETIVOS

2.3.1 Objetivo General

Analizar, configurar e implementar un Sistema de replicación de bases de datos Postgres, que contribuya a mejorar la tolerancia a fallas, el almacenamiento y consulta de los datos.

2.3.2 Objetivos Específicos

- Instalar un sistema de replicación asincrónica maestro-esclavo (*master-slave*) en el que los nodos esclavos se pueden utilizar para realizar consultas de solo lectura.
- Proporcionar un método de balanceo de carga en los servidores Postgres.
- Realización de backups de las distintas bases de datos Postgres.
- Implementación en la Universidad Nacional de Cuyo, que constará de la instalación y configuración de un sistema de replicación conjuntamente con un método de balanceo de carga, además de un sistema de backups para las distintas bases de datos que la Universidad dispone.

2.4 METODOLOGÍA

El desarrollo de este trabajo de investigación se realizará en torno a las siguientes actividades:

- La primera etapa de investigación comienza con un relevamiento de la información tendiente a obtener un panorama de las distintas tecnologías y opciones que podemos encontrar para el mejor aprovechamiento de las bases de datos Postgres.

- La segunda etapa consiste en la selección de las mejores opciones y tecnologías en base a la información recolectada.

2.5 MARCO TEÓRICO

2.5.1 Características de Postgres

Qué es Postgres?

Los sistemas de Bases de Datos relacionales tradicionales soportan un modelo de datos que consisten en una colección de relaciones con nombre, que contienen atributos de un tipo específico. En los sistemas comerciales actuales, los tipos posibles incluyen numéricos de punto flotante, enteros, cadenas de caracteres, cantidades monetarias y fechas. Está generalmente reconocido que este modelo será inadecuado para las aplicaciones futuras de procesado de datos. Sin embargo, como se ha mencionado, esta simplicidad también hace muy difícil la implementación de ciertas aplicaciones.

Postgres ofrece una potencia adicional sustancial al incorporar los siguientes cuatro conceptos adicionales básicos en una vía en la que los usuarios pueden extender fácilmente el sistema: clases, herencia, tipos y funciones.

- ✓ ***Clases***

La noción fundamental en Postgres es la de clase, que es una colección de instancias de un objeto. Cada instancia tiene la misma colección de atributos y cada atributo es de un tipo específico. Más aún, cada instancia tiene un identificador de objeto (OID[1]) permanente, que es único a lo largo

[1] OID (*Object Identifiers*): Los identificadores de objetos se utilizan internamente por PostgreSQL como claves principales para diferentes tablas del sistema.

de toda la instalación. Ya que la sintaxis SQL[2] hace referencia a tablas, se usarán los términos tabla y clase indistintamente. Asimismo una fila SQL es una instancia y las columnas SQL son atributos. Las clases se agrupan en bases de datos y una colección de bases de datos gestionada por un único proceso postmaster constituye una instalación o sitio.

- ✓ ***Herencia***

En Postgres una clase puede heredar de ninguna o varias otras clases, y una consulta puede hacer referencia tanto a todas las instancias de una clase como a todas las instancias de sus descendientes.

- ✓ ***Tipos***

Postgres posee un conjunto de tipos de datos nativos disponibles para los usuarios. Los usuarios pueden agregar nuevos tipos a Postgres usando el comando CREATE TYPE.

- ✓ ***Funciones***

Las funciones escritas en cualquier lenguaje excepto SQL se ejecutan por el servidor de la base de datos con el mismo permiso que el usuario Postgres (el servidor de la base de datos funciona con el *usuario* de Postgres. Es posible que los usuarios cambien las estructuras de datos internas del servidor por medio de las funciones.

Es por ello que este tipo de funciones pueden, entre otras cosas, evitar cualquier sistema de control de acceso. Este es un problema inherente a las funciones definidas por los usuarios en C.

---

[2] SQL (*Structured Query Language*): El lenguaje de consulta estructurada es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones en ellas.

Otras características que aportan potencia y flexibilidad adicional son:

- ✓ ***Restricciones (Constraints)***

Las restricciones son una manera de limitar el tipo de datos que pueden ser almacenados en una tabla. Por ejemplo, una columna que contiene un precio de producto probablemente debería aceptar sólo valores positivos. Pero no hay ningún tipo de datos estándar que sólo acepta números positivos.

Otra posible cuestión es que se requieran restringir los datos de una columna con respecto a otras filas o columnas. Por ejemplo, en una tabla que contiene información de un producto, sólo debe haber una fila para cada tipo de producto.

Para ello, SQL permite definir restricciones sobre columnas y tablas. Si un usuario intenta guardar datos en una columna que viola una restricción, se producirá un error.

- ✓ ***Disparadores (triggers)***

Un disparador no es otra cosa que una acción definida en una tabla de la base de datos y ejecutada automáticamente por una función programada por el administrador de base de datos. Esta acción se activará cuando se realice un INSERT, un UPDATE o un DELETE en la tabla en cuestión.

Un disparador se puede definir de las siguientes maneras:

- Para que ocurra ANTES de cualquier INSERT, UPDATE o DELETE.
- Para que ocurra DESPUÉS de cualquier INSERT, UPDATE o DELETE.
- Para que se ejecute una sola vez por comando SQL (*statement-level trigger*).
- Para que se ejecute por cada línea afectada por un comando SQL (*row-level trigger*).

Antes de definir el disparador habrá que determinar el procedimiento almacenado que se ejecutará cuando el disparador se active.

El procedimiento almacenado usado por el disparador se puede programar en cualquiera de los lenguajes de procedimientos disponibles, entre ellos PL/Pgsql, que es el proporcionado por defecto cuando se instala Postgres.

Estas características colocan a Postgres en la categoría de las bases de datos identificadas como objeto-relacionales. Tales características son diferentes de las referidas como orientadas a objetos, que generalmente no son bien aprovechables para soportar lenguajes de bases de datos relacionales tradicionales.

Postgres tiene algunas características que son propias del mundo de las bases de datos orientadas a objetos. De hecho, algunas bases de datos comerciales han incorporado recientemente características en las que Postgres fue pionero.[3]

[3] Manual del usuario de PostgreSQL, página 1, Thomas Lockhart.

Conceptos de arquitectura

En la jerga de bases de datos, Postgres usa un modelo cliente-servidor conocido como "proceso por usuario". Una sesión de Postgres consiste en los siguientes procesos cooperativos (programas) de Unix:

- Un proceso demonio supervisor (*postmaster*).
- La aplicación sobre la que trabaja el usuario (*frontend*). Por ejemplo el programa *psql*.
- Uno o más servidores de bases de datos en segundo plano (el mismo proceso Postgres).

Un único postmaster controla una colección de bases de datos localizadas en un único host, debido a esto una colección de bases de datos se suele llamar una instalación o un sitio.

Las aplicaciones de *frontend* que quieren acceder a una determinada base de datos dentro de una instalación realizan llamadas a determinadas librerías, dichas librerías envían peticiones de usuario a través de la red al *postmaster*, el cual en respuesta a las peticiones inicia un nuevo proceso en el servidor (*backend*) y conecta el proceso de *frontend* al nuevo servidor. A partir de este punto, el proceso de *frontend* y el servidor en *backend* se comunican sin la intervención del *postmaster*.

Aunque, el *postmaster* siempre se está ejecutando, esperando peticiones, tanto los procesos de *frontend* como los de *backend* vienen y se van.

La librería *libpq* permite a un único proceso en *frontend* realizar múltiples conexiones a procesos en *Backend*, aunque la aplicación *frontend* todavía es un proceso en un único thread. Conexiones multithread entre el *frontend* y el *backend* no están soportadas de momento en *libpq*. Una implicación de esta arquitectura es que el *postmaster* y el proceso *backend* siempre se ejecutan en la misma máquina (el servidor de base de datos), mientras que la aplicación en *frontend* puede ejecutarse desde cualquier sitio. Es importante tener esto en mente, porque los archivos que pueden ser accedidos en la máquina del cliente pueden no ser accesibles (o sólo pueden ser accedidos usando un nombre de archivo diferente) en la máquina del servidor de base de datos.

Los servicios *postmaster* y Postgres se ejecutan con el identificador de "*superusuario*" Postgres, donde el superusuario Postgres no necesita ser un usuario especial (Ejemplo: Un usuario llamado "Postgres").

En cualquier caso, todos los archivos relacionados con la base de datos deben pertenecer a este superusuario Postgres.[4]

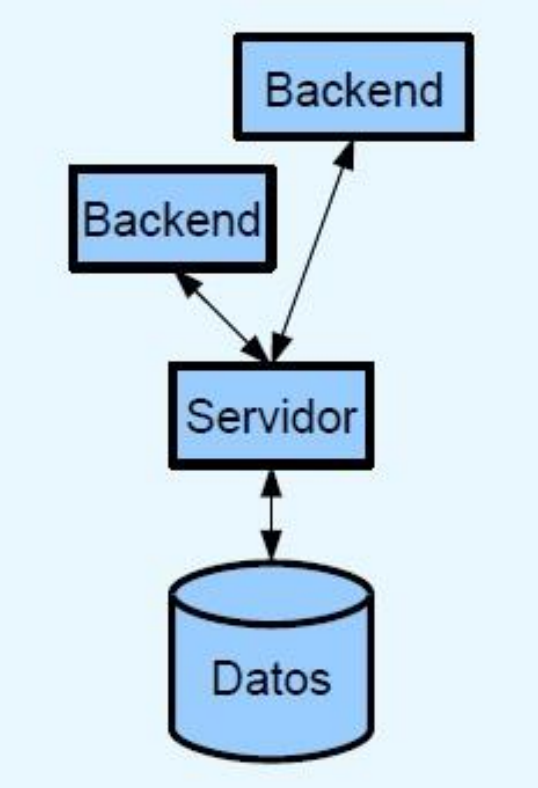

Imagen 1 - Material Capacitación SIU, página 9 – Agosto de 2009.

[4] Tutorial de PostgreSQL, página 31, Thomas Lockhart.

Procesamiento interno

La ruta para ejecutar una consulta es la siguiente:

1. Se debe establecer una conexión desde una aplicación al servidor Pgsql, luego la aplicación transmite la consulta y espera a los resultados.

2. El parseador chequea que la consulta transmitida por la aplicación tenga una sintaxis correcta y entonces crea el árbol de consulta (ADC[5]).

3. El sistema de reescritura (SDR[6]) toma el ADC y busca cualquier regla (almacenada en los catálogos de sistema) para aplicar el ADC. Posteriormente ejecuta ciertas transformaciones en los cuerpos de las reglas. Una aplicación del SDR es la realización de vistas, cuando es una consulta sobre una vista, el SDR reescribe la consulta para acceder a la tabla.

4. El planeador/optimizador recibe la reescritura de la consulta y crea un plan, el cual será la entrada para el ejecutor. Como primera instancia creará todas las posibles rutas que tengan el mismo resultado. Luego, se estima el costo por cada ejecución para elegir el menos costoso.

5. El ejecutor pasa recursivamente a través del árbol del plan y devuelve las filas. Hace uso del sistema de almacenamiento mientras realiza las tareas del plan y finalmente retorna los resultados.[7]

---

[5] ADC (*Árbol de Consulta*): Es una estructura de árbol que corresponde a una expresión del álgebra relacional, en el que las tablas se representan como nodos hojas y las operaciones del álgebra relacional como nodos intermedios.

[6] SDR por sus siglas *Sistema de Reescritura*.

[7] Material Capacitación SIU– Agosto de 2009.

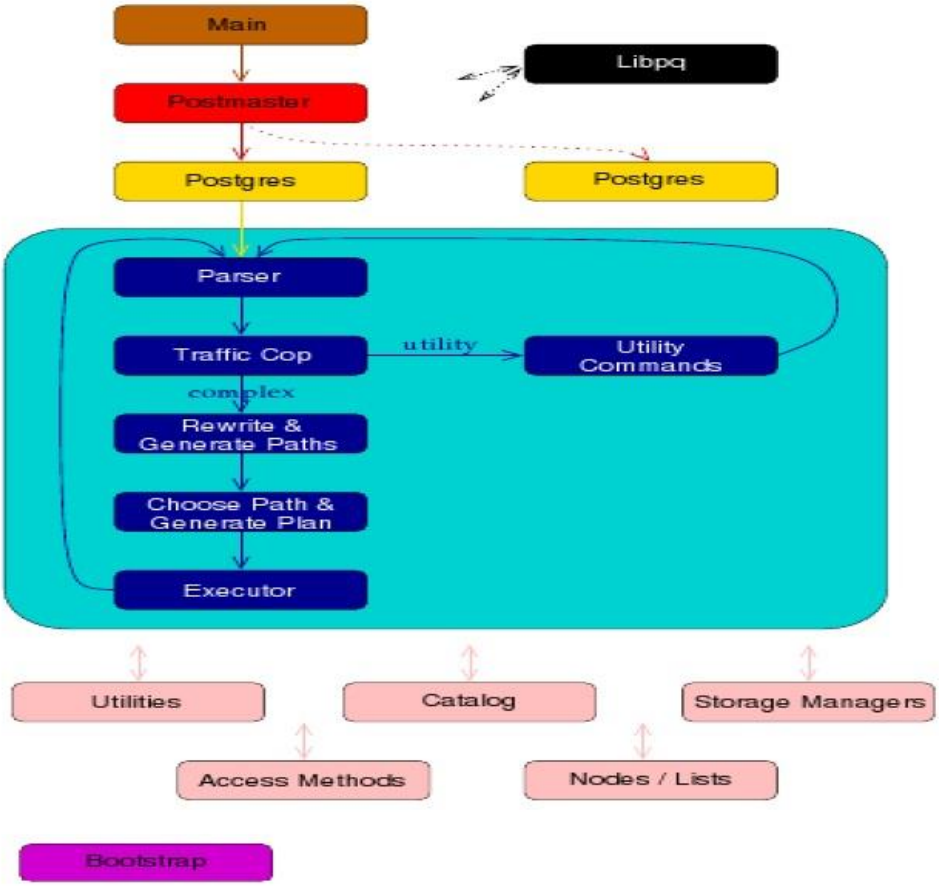

Imagen 2 - Material Capacitación SIU, página 39 – Agosto de 2009.

Atomicidad, Consistencia, Aislamiento y Durabilidad

La atomicidad, consistencia, aislamiento y durabilidad (ACID[8]) son un conjunto de características necesarias para que una serie de instrucciones puedan ser consideradas como una transacción. Así pues, si un sistema de gestión de bases de datos es ACID quiere decir que el mismo cuenta con las funcionalidades necesarias para que sus transacciones tengan las características ACID.

- *Atomicidad* (Indivisible) es la propiedad que asegura que la operación se realice o no, y por lo tanto ante un fallo del sistema no puede quedar a medias.
- *Consistencia* es la propiedad que asegura que sólo se empieza aquello que se puede acabar. Por lo tanto se ejecutan aquellas operaciones que no van a romper la reglas y directrices de integridad de la base de datos.
- *Aislamiento* es la propiedad que asegura que una operación no puede afectar a otras. Esto asegura que dos transacciones sobre la misma información no generarán ningún tipo de error.
- *Durabilidad* es la propiedad que asegura que una vez que se confirma una transacción, sus resultados no se perderán, independientemente de los fallos posteriores.[9]

[8] ACID por sus siglas en inglés *Atomicity, Consistency, Isolation and Durability*.

[9] http://www.postgresql.org/files/developer/transactions.pdf, página 6, Tom Lane.

Control de Concurrencia Multi Versión

El control de concurrencia multi versión (*MVCC*[10]) es una técnica avanzada para mejorar las prestaciones de una base de datos en un entorno multiusuario.

A diferencia de la mayoría de otros sistemas de bases de datos que usan bloqueos para el control de concurrencia, Postgres mantiene la consistencia de los datos con un modelo multiversión, esto significa que mientras se consulta una base de datos, cada transacción ve una imagen de los datos (una versión de la base de datos), sin tener en cuenta el estado actual de los datos que hay por debajo. Esto evita que la transacción se encuentre con datos inconsistentes que puedan ser causados por la actualización de otra transacción concurrente en la misma fila de datos, proporcionando aislamiento transaccional para cada sesión de la base de datos.

La principal diferencia entre multiversión y el modelo de bloqueo, es que en los bloqueos MVCC derivados de una consulta (lectura) de datos no entran en conflicto con los bloqueos derivados de la escritura de datos y de este modo la lectura nunca bloquea la escritura, y la escritura nunca bloquea la lectura.[11]

[10] MVCC por sus siglas en inglés Multi-Version Concurrency Control.

[11] Manual del usuario de PostgreSQL, página 91, Thomas Lockhart.

Write Ahead Logging

Postgres utiliza los denominados ficheros WAL[12] para guardar toda la información sobre las transacciones y cambios realizados en la base de datos.

Los ficheros WAL se utilizan para garantizar la integridad de los datos grabados en la base de datos. También se utilizan para reparar automáticamente posibles inconsistencias en la base de datos después de una caída súbita del servidor.

Estos ficheros tienen un nombre único y un tamaño por defecto de 16MB y se generan en el subdirectorio *pg_xlog* que se encuentra en el directorio de datos que utiliza Postgres.

El número de ficheros WAL contenidos en el subdirectorio *pg_xlog* dependerá del valor asignado al parámetro *checkpoint_segments* en el fichero de configuración *postgresql.conf*.

Los ficheros WAL generados en *pg_xlog* se reciclan continuamente y en un sistema muy ocupado solo tendremos disponibles en *pg_xlog* los últimos cambios ocurridos en la base de datos durante el periodo de tiempo registrado en los ficheros WAL existentes en *pg_xlog*.

Los ficheros WAL se pueden archivar automáticamente como copia de seguridad o para usarlos con PITR[13].

Para activar el archivo automático de ficheros WAL hay que definir los parámetros *wal_level*, *archive_mode* y *archive_command* en *postgresql.conf*. Un fichero WAL se archivará antes que sea reciclado en *pg_xlog*, pero no antes de que tenga registrado 16MB de información en el mismo.[14]

---

[12] WAL por sus siglas en inglés *Write Ahead Log*.

[13] PITR (*Point in time recovery*): Es un tipo de backup avanzado utilizado en sistemas PostgreSQL que trabajan con datos importantes, los cuales no pueden perderse en caso de fallo.

[14] http://www.postgresql.org.es/node/483

2.5.2 Descripción de la replicación

Replicación

Existen varios métodos de replicación basados principalmente en la arquitectura (Maestro/Esclavo ó Multimaestro):

Métodos Síncronos: Solo se aplican cambios en una base de datos maestra, cuando pueden ser aplicados a las bases de datos esclavas, garantizando que las bases de datos son siempre idénticas. Pero dado que las prioridades en la actualización son altas, se puede ver afectada la aplicación que usa la base de datos.

Métodos Asíncronos: Se aplican los cambios a la base de datos maestra y luego a las bases esclavas, teniendo un periodo para lograr convergencia de las bases de datos, entonces existe un peligro con la ganancia de impactar menos a las aplicaciones.

La replicación de bases de datos es la acción de tener bases de datos redundantes en distintos servidores, los cuales pueden estar ubicados en instalaciones diferentes. Esto es distinto a los respaldos, una base de datos replicada esta en producción y refleja todos sus cambios a sus copias que también están funcionando. Un backup es una instantánea de una base de datos en un momento determinado. Ninguna presupone a la otra.

"También se puede decir que la replicación de bases de datos es el término que se usa para describir la tecnología que mantiene una copia de un conjunto de datos en un sistema remoto. Por lo general hay dos razones principales para querer hacer esto, y generalmente esas razones se combinan:

- ***Alta disponibilidad***: Reducir las posibilidades de falta de disponibilidad de datos por tener múltiples sistemas cada uno con una copia completa de los datos.
- ***El movimiento de datos***: Permite que los datos puedan ser utilizados por otras aplicaciones, un ejemplo es la gestión de datos de referencia, donde un servidor central podría proporcionar información para otras aplicaciones. En el movimiento de datos, no hay transformación de los datos, sino simplemente copiarlos de un servidor de base de datos a otro.

En la arquitectura básica, los servidores individuales de base de datos se denominan nodos. Todo el grupo de servidores de bases de datos implicados en la replicación se conoce como un *Cluster*. Aunque el término *Cluster* también se utiliza para otros significados, como por ejemplo, se suele utilizar a veces para hacer referencia a la instancia de base de datos completa, es preferible el término servidor de base de datos.

El servidor de base de datos primario es también conocido como maestro, el servidor de base de datos secundario se conoce como esclavo. Las designaciones de maestro y esclavo son los roles que cada nodo puede tener en algún momento, para cambiar de nodo principal a otro nodo, se realiza un procedimiento denominado conmutación (*Switchover*). Si el maestro cae, y no se recupera, entonces el cambio de rol se conoce como una recuperación de fallos (*Failover*). El software que administra el clúster, y en algunos casos inicia automáticamente el *Failover*, se suele conocer como *Clusterware*, el *Clusterware* también puede realizar otras funciones, tales como balanceo de carga. El aspecto clave de la replicación de datos es que los cambios se capturan en el maestro, y luego son transferidos a los nodos en espera.

Luego de que una transacción en el maestro es confirmada (*commits*), el tiempo necesario para transferir los cambios sobre los datos que se envían del maestro al esclavo es importante, normalmente se conoce como latencia, o demora de replicación, que se mide en segundos. Los cambios que deben ser aplicados al esclavo, que se llevan a cabo durante un lapso de tiempo se conocen como demora de replicación, estos cambios de datos son enviados a menudo por lotes. El aumento de tamaño de lote puede aumentar la eficiencia de transferencia, aunque también aumentará el retraso de la replicación. El tiempo total que toma un registro del maestro al esclavo es el de la demora de replicación más el de aplicar el retardo.

Si los cambios de datos se reconocen como enviados del maestro al esclavo antes de que la transacción se confirme (*commit*), se produce una replicación síncrona. Por el contrario si los cambios de datos se envían después de una transacción, se trata de una replicación asíncrona. Con la replicación sincrónica, la demora de replicación afecta directamente el rendimiento en el maestro. Con la replicación asíncrona el maestro puede continuar a toda velocidad, aunque esto abre un posible riesgo de que el esclavo no sea capaz de mantener el ritmo con el maestro. Toda replicación asíncrona debe ser monitoreada para asegurar que no se desarrolle un retraso significativo, por lo que se hay que controlar la demora de la replicación.

Todas las formas de replicación Maestro-Esclavo se inicializan aproximadamente de la misma manera. En primer lugar, se habilita la captura de cambios, y luego se hace una réplica completa del conjunto de datos en el nodo esclavo, para luego empezar a aplicar los cambios. El esclavo comenzará a ponerse al día con el maestro, y el retraso de la replicación comenzará a reducirse. Si el maestro está ocupado, seguirá produciendo muchos cambios, y eso podría prolongar el tiempo que tarda el esclavo en actualizarse.

En la replicación o bien se copian todas las tablas, o en algunos casos, se puede copiar un subconjunto de las tablas, en cuyo caso se llama replicación selectiva. Cuando se utiliza la replicación selectiva se pueden agrupar los objetos seleccionados en una serie de replicación. Si se opta por una replicación selectiva, se debe tener en cuenta que la sobrecarga de administración aumenta a medida que aumenta el número de objetos manejados.

PostgreSQL ha incluido tipos de replicación nativa desde la versión 8.2, las cuales han ido mejorando con el tiempo, además PostgreSQL ha contado con el apoyo de una amplia gama de herramientas de replicación.

En muchos casos, se utiliza una técnica en un servidor y otra técnica diferente para proteger a otros servidores. Incluso los desarrolladores de determinadas herramientas utilizan otras utilidades cuando es conveniente.”[15]

[15] PostgreSQL 9 Administration Cookbook, página 300, Simon Riggs - Hannu Krosing, Packt Publishing.

Mejores Prácticas de Replicación

Como se puede aplicar:

- ***Utilizar hardware y sistema operativo similar en todos los sistemas***: La replicación permite que los nodos intercambien roles. Si se produce un *switchover* o un *failover* con un hardware distinto, se puede obtener un mal rendimiento y será más difícil mantener una aplicación funcionando sin problemas.

- ***Configurar todos los sistemas de forma idéntica, en la medida de lo posible***: Utilizar los mismos puntos de montaje, mismos nombres de directorio, los mismos usuarios, mantener igual todo lo posible. No tener la tentación de hacer un sistema de alguna manera más importante que los demás.

- ***Dar buenos nombres a los sistemas para reducir la confusión***: Nunca llamar a uno de los sistemas "Maestro" y al otro "Esclavo". Tratar de elegir los nombres de sistemas que no tienen nada que ver con su papel. Si un sistema falla, y se añade un nuevo sistema, no volver a utilizar el nombre del viejo sistema, elegir otro. No escoger nombres que se refieren a algo en el negocio. Los colores son una mala opción, porque si se tienen dos servidores llamados amarillo y rojo, puede ocurrir un caso como "hay una alerta roja en el servidor amarillo ", lo que puede resultar confuso.

- ***Mantener sincronizados los relojes del sistema***: Esto ayuda a mantener la concordancia al buscar en los log de archivos generados por varios servidores. También se puede sincronizar manualmente sobre una base regular.

- ***Utilizar solamente una zona horaria, sin ambigüedades***: Usar por ejemplo UTC (Tiempo Universal Coordinado) o alguno similar. No elegir una zona horaria que tiene el horario de verano, esto puede conducir a una confusión con la replicación, puede ocurrir que los servidores estén en diferentes países, y las diferencias de zona horaria varían a lo largo del año.

- ***Monitorear cada uno de los servidores de bases de datos***: Si se desea una alta disponibilidad, entonces se deberá comprobar regularmente que los servidores están funcionando correctamente.

- ***Supervisar el retraso de la replicación entre los servidores***: Todas las formas de replicación son útil solamente si los datos fluyen correctamente entre los servidores. Monitorear el tiempo que toma pasar los datos de un servidor a otro es esencial para entender si la replicación está funcionando o no. La replicación puede ser por ráfagas, por lo que debe supervisarse para estar seguro de que se mantiene dentro de límites razonables.

El punto esencial es que el retraso de la replicación está directamente relacionado con la cantidad de datos que es probable perder cuando se ejecuta la replicación asíncrona.[16]

[16] PostgreSQL 9 Administration Cookbook, página 304, Simon Riggs - Hannu Krosing, Packt Publishing.

Balanceo de Carga

La replicación a menudo se combina con el balanceo de carga para mejorar el rendimiento de las operaciones de lectura. Algunas soluciones de replicación tienen un balanceo de carga integrado, en cambio otras soluciones se apoyan en el sistema operativo o sobre balanceadores de carga de terceros.

Métodos de replicación

Las bases de datos pueden mantenerse coherente de varias maneras, pueden encontrarse los siguientes enfoques:

Replicación por transacciones: Consiste en copiar cada una de las transacciones ejecutadas en el sistema donde se encuentran conectados los usuarios hacia un conjunto de bases de datos secundarias. Estas bases de datos secundarias reciben los cambios propagados por la base de datos principal por medio de un canal de comunicación.

Replicación por bitácoras (logs): Es otro método comúnmente utilizado, que consiste en el envío de logs (bitácoras de la base de datos). Este método presenta el problema de que a veces dichas bitácoras no siempre son pensadas como elementos de intercambio para efectos de replicación, este método generalmente se delega para porciones de la base de datos.

Divergencia

Mantener la coherencia de los datos a través de múltiples nodos replicados es un proceso costoso debido a la latencia de red. Es así como algunos sistemas tratan de evitar el costo por latencia permitiendo a los nodos divergir levemente, lo que implica que es posible que ocurran conflictos en las transacciones.

Para tener una base de datos coherente y consistente, estos conflictos deben ser resueltos cuanto antes de forma automática o manual. Tales conflictos violan la propiedad *ACID* del sistema de base de datos, por lo que la aplicación tiene que ser consciente de ello.

Replicación Síncrona vs Asíncrona

En el contexto de la replicación de bases de datos, la definición más común de la replicación síncrona es cuando los datos se transfieren de un sistema maestro, a otro esclavo, y el maestro espera el acuse de recibo del esclavo antes de hacer disponibles en el maestro los datos replicados. En un sistema de replicación sincrónico, todos los datos disponibles en el maestro están disponibles en los esclavos. Esto es muy costoso en términos de latencia y la cantidad de mensajes a ser enviados, pero evita la divergencia.

En los sistemas de replicación asincrónica, los datos se transfieren de un sistema maestro a otro esclavo, sin esperar por el acuse de recibo del esclavo antes de hacer disponibles en el maestro los datos replicados. En un sistema de replicación asincrónico puede existir un cierto retraso o demora en la disponibilidad de los datos en el sistema esclavo.[17]

Consulta Distribuida

Es prácticamente otro nombre que recibe el Balanceo de Carga, dentro de la documentación de PostgreSQL se le conoce como "*Multi-Server Parallel Query Execution*"

Clustering

Mientras que el clustering es un término muy popular, no tiene un significado bien definido en lo que respecta a los sistemas de bases de datos. Diferentes técnicas, como el balanceo de carga, la replicación, consultas distribuidas, etc. se llaman clustering.[18]

---

[17] http://www.postgresql.org.es/node/483

[18] http://www.postgres-r.org/documentation/terms

2.5.3 Distintos métodos de replicación

2.5.3.1 Slony

Qué es Slony-I?

Slony-I es un sistema de replicación asíncrono para bases de datos Postgres de una base de datos maestra hacia múltiples bases de datos hijas. Las características de Slony-I incluyen:

- Slony-I puede replicar datos entre distintas versiones de Postgres.
- Slony-I puede replicar datos entre diferente hardware o sistemas operativos.
- Slony-I permite replicar sólo algunas de las tablas al esclavo.
- Slony-I permite replicar algunas tablas a un esclavo y otras tablas a otro esclavo.
- Slony-I permite a diferentes servidores de bases de datos ser maestros para diferentes tablas.

Como funciona

Realiza las actualizaciones utilizando *triggers*, lo que significa que no puede propagar cambios de esquemas y operaciones con objetos. Actualmente Slony-I replica solamente cambios a tablas y secuencias.

No se recomienda Slony-I para:

- Sitios con redes altamente inestables.
- Bases de datos donde regularmente se hagan cambios a esquemas y definiciones de tablas (DDL[19]).
- Para quienes requieran un sistema de replicación de múltiples maestros a esclavos.
- Para quienes quieran un sistema de replicación que haga algo automáticamente al detectar el fallo de un nodo.

Conceptos Slony

Antes de poner en marcha la replicación con Slony-I, es importante manejar los siguientes conceptos:

- ***Clúster*:** Un clúster es un conjunto de bases de datos Postgres dentro de las cuales sucede la replicación. El nombre del clúster se especifica en todos y cada uno de los scripts Slonik mediante la directiva: ***cluster name = base_uno;*** Si el nombre del clúster es ***base_uno***, entonces Slony-I creará en cada instancia de base de datos del clúster, el namespace[20]/schema[21] ***_ base_uno***.

---

[19] DDL (*Data Definition Language*): El Lenguaje de Definición de Datos, sirve para modificar la estructura de los objetos en una base de datos.

[20] Namespace (*Espacio de nombres*): Es la estructura subyacente de esquemas SQL, cada espacio de nombres puede tener una colección separada de las relaciones, tipos, etc., sin haber conflictos de nombres.

[21] Schema (*Esquema*): Son usados en las bases de datos para separarlas de manera lógica, dando la opción en un momento determinado de tener corriendo un sistema real y uno de prueba dentro de la misma base, pero separados mediante esquemas, también es posible tener en dos esquemas distintos los mismos nombres de tablas sin que esto no represente un error.

- ***Nodo*:** Un nodo es una base de datos que participará en el proceso de replicación. Se define cerca del comienzo de cada script Slonik usando la directiva: ***NODE 1 ADMIN CONNINFO = 'dbname=prueba host=server1 user=slony'.*** Por lo tanto, un clúster Slony-I consta de un nombre de clúster y de un conjunto de nodos Slony-I, donde cada uno de los cuales tiene un espacio de nombres basado en el nombre del clúster.

- ***Set de replicación*:** Es un conjunto de tablas y secuencias que se replicarán entre nodos de un clúster Slony-I. Se pueden tener varios set donde el flujo de replicación no necesita ser idéntico entre los set de replicación. No se replican todas las tablas, sino que tienen que especificarse explícitamente.

- ***Origen, proveedores y suscriptores*:** Cada set de replicación tiene un nodo origen, que es la base de donde se obtiene la información. Los nodos que reciben la información se denominan nodos suscriptores, dichos nodos suscriptores se pueden convertir en proveedores si a su vez tienen otros nodos suscriptores.

- ***Demonios Slon*:** Para cada nodo en el clúster, existe un proceso llamado *Slon* que administra la actividad de replicación para ese nodo. *Slon* es un programa implementado en el lenguaje de programación *C*, el cual procesa los eventos de replicación. Hay dos tipos principales de eventos: *Eventos de configuración* que normalmente ocurren cuando se ejecuta un script slonik y envían actualizaciones a la configuración del clúster, y *Eventos SYNC* en donde las actualizaciones de las tablas se replican juntas en grupos *SYNC*, estos grupos de transacciones se aplican conjuntamente en los nodos suscriptores.

- ***Comandos Slonik*:** Son un pequeño intérprete de comandos, que incluye comandos utilizados para la manipulación del clúster de replicación.

- ***Configuración del Procesador Slonik:*** Los comandos del procesador slonik son scripts en un pequeño lenguaje, que son usados para enviar eventos que actualizan la configuración de un clúster Slony. Esto incluye acciones como agregar y quitar nodos, modificación de vías de comunicación y añadir o eliminar suscripciones.

- ***Tunneling SSH[22]*:** Si una conexión directa a Postgres no puede establecerse debido a un firewall, entonces se puede establecer mediante un túnel SSH para que Slony-I siga operando. Esto permite que el tráfico de reenvío de Postgres a un puerto local, sea transmitido a través de una conexión encriptada y comprimida, utilizando SSH.

- ***Path de Comunicaciones Slony*:** Slony-I utiliza DSN[23] Postgres en tres contextos para establecer el acceso a la bases de datos:

  - ***SLONIK ADMIN CONNINFO:*** Controla la forma de cómo un script slonik accede a los diferentes nodos. Estas conexiones son las que van desde la estación de trabajo administrativa a todos los nodos de un clúster Slony-I. Es fundamental tener las conexiones desde la ubicación central en la que se ejecuta slonik para cada nodo en la red. Estas conexiones sólo se usan brevemente, para enviar las peticiones SQL necesarias que controlan la administración del clúster.

  - ***El parámetro slon DSN***: El parámetro DSN que se pasa a cada proceso slon indica que ruta de red se debe utilizar desde el proceso slon a la base de datos que administra.

  - ***SLONIK STORE PATH***: Controla cómo los demonios slon se comunican con los nodos remotos, estas rutas son almacenadas en el *sl_path*. Debe haber un camino entre cada nodo suscriptor y su proveedor y otras rutas opcionales, y que no se utilizarán a menos que se determine en *sl_listen* que es necesario utilizar una ruta en particular.

---

[22] SSH (*Secure Shell*): Intérprete de órdenes segura, es el nombre de un protocolo y del programa que lo implementa, y sirve para acceder a máquinas remotas a través de una red.

[23] DSN (*Data Source Name*): Nombre de Fuente de Datos, todo lo relativo a una fuente de datos configurada por el usuario para conectarse a una Base de datos. Es decir, por cada conexión que el usuario quiera establecer con algún(os) fabricante(s), tiene que especificar una serie de información que permitan al Controlador o Driver saber con qué fabricante(s) se tiene que conectar y la cadena de conexión que tiene que enviarle a dicho fabricante(s) para establecer la conexión con la fuente de datos ODBC accedida por el proveedor en cuestión.

Las diferencias y posibles complejidades de rutas no son normalmente un problema para las personas con redes simples donde todos los hosts pueden verse entre sí a través de un conjunto de direcciones de red. Es mucho más importante para los que tienen complejas configuraciones de firewall, nodos en múltiples ubicaciones, y el tema de que los nodos no pueden ser capaces de comunicarse con los otros nodos por medio de un conjunto uniforme de direcciones de red.

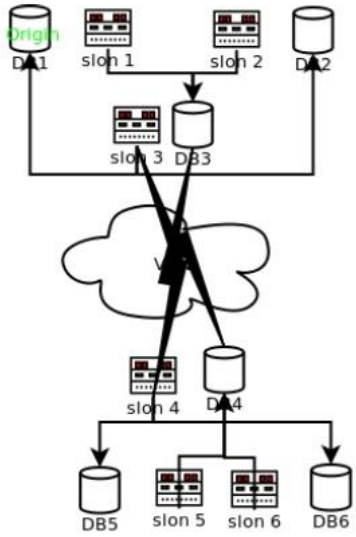

Imagen 3 - http://slony.info/adminguide/2.1/doc/adminguide/slony.pdf, página 5 – 2004-2010

En el diagrama anterior, el cual describe un conjunto de seis nodos:

- DB1 y DB2 son bases de datos que residen en un entorno seguro de capa base de datos, un firewall protegiendo el acceso externo con la excepción de lugares controlados, y suponiendo que DB1 es el nodo de origen para el sistema de replicación.
- DB3 se ubica en un DMZ[24] en el mismo sitio, y suele ser utilizado como un proveedor Slony-I para ubicaciones remotas.

[24] DMZ (*Demilitarized Zone*): En seguridad informática, una zona desmilitarizada o red perimetral es una red local que se ubica entre la red interna de una organización y una red externa, generalmente en Internet.

- DB4 se encuentra en una zona de distensión, en un sitio secundario. Su función, en esta ocasión es de proveer a los servidores en las capas de base de datos ubicados en el sitio secundario.

- DB5 y DB6 son equivalentes a DB1 y DB2, pero son configurados como suscriptores. Suponiendo que el sitio primario es atacado o tiene algún problema, el sitio secundario debería estar preparado para asumir el servicio de las aplicaciones que utilizan estos datos.

- La simetría de la configuración significa que si se tiene a dos aplicaciones transaccionales que requieren protección de alguna falla, sería sencillo tener sistemas adicionales de replicación para que cada sitio sea normalmente primario para una aplicación, y donde la destrucción de un sitio podría abordarse mediante la consolidación de los servicios en el sitio restante.

¿Qué es un clúster de Slony-I?

Un clúster de Slony es una agrupación básica de instancias de bases de datos donde ocurre la replicación. Si el clúster se llama ***micluster***, entonces Slony-I creará al momento de inicializar cada nodo un esquema con el nombre ***micluster***, dentro del cual se crearán tablas que almacenan información de configuración y replicación.

Requisitos

Los siguientes son los requisitos más importantes a tener en cuenta antes de configurar la replicación:

- Cualquier plataforma que pueda ejecutar Postgres puede correr en principio Slony-I.
- Utilizar exactamente la misma versión de Slony-I en todos los nodos.
- Versión 8.3 o superior de Postgres para utilizar Slony-I.
- Codificación similar de la base de datos (UNICODE, SQL ASCII, LATIN1).
- Sincronización de la hora en los sistemas operativos donde estén las bases.
- Tener una red estable.
- Tablas a replicar con llaves primarias (asegurar una forma única de identificar filas).
- tcpip socket = true en el archivo *postgresql.conf*.
- Accesos correctos en *pg_hba.conf*.
- Utilizar un ambiente de pruebas antes de realizarlo en producción.

Limitaciones actuales

Slony no replica automáticamente:

- Cambios en los objetos grandes (BLOB).
- Los cambios realizados por comandos DDL.
- Cambios en los usuarios y roles.

La razón principal de estas limitaciones es que Slony-I recopila actualizaciones mediante disparadores, y ni los cambios de esquema, ni las operaciones de objetos grandes son capturados por los disparadores. Slony-I tiene una capacidad de propagar los cambios de DDL en particular si se los envía como scripts a través de la operación slonik ***SLONIK EXECUTE SCRIPT***. Eso no ocurre automáticamente, por lo que se debe construir un script SQL DDL y enviarlo a través de ***SLONIK EXECUTE SCRIPT***.[25]

[25] http://slony.info/adminguide/2.1/doc/adminguide/slony.pdf

2.5.3.2 PgPool-II

Qué es PgPool-II?

PgPool-II es un middleware[26] que se encuentra entre los servidores Postgres y un cliente de una base de datos Postgres. Ofrece las siguientes características:

- ***Pooling de conexiones***: Mantiene las conexiones establecidas con los servidores Postgres, y las reutiliza cuando una nueva conexión con las mismas propiedades (por ejemplo: nombre de usuario, base de datos, versión del protocolo) entra en juego, reduciendo la actividad de conexiones, y mejorando el rendimiento global del sistema.

- ***Replicación:*** Puede administrar múltiples servidores Postgres. La activación de la característica de replicación permite crear una copia de seguridad en tiempo real de dos o más clústeres de Postgres, de manera que el servicio pueda continuar sin interrupción si algún clúster falla.

- ***Balanceo de carga***: Si una base de datos es replicada, la realización de una consulta SELECT en cualquier servidor devolverá el mismo resultado. PgPool-II aprovecha la función de replicación a fin de reducir la carga de cada servidor de Postgres. Lo hace distribuyendo las consultas SELECT entre los servidores disponibles, mejorando el rendimiento global del sistema. En un escenario ideal, el rendimiento de lectura podría mejorar proporcionalmente al número de servidores Postgres. El balanceo de carga funciona mejor en un escenario donde hay una gran cantidad de usuarios que ejecutan muchas consultas de sólo lectura al mismo tiempo.

- ***Límite de conexiones excedentes***: Hay un límite en el número máximo de conexiones simultáneas con Postgres, y las nuevas conexiones son rechazadas cuando se alcanza este número. Al aumentar el número máximo de conexiones, aumenta el consumo de recursos y tiene un impacto negativo en el rendimiento general del sistema. PgPool-II también

[26] Middleware: Es un software que asiste a una aplicación para interactuar o comunicarse con otras aplicaciones, software, redes, hardware y/o sistemas operativos.

cuenta con un límite para el número máximo de conexiones, pero las conexiones adicionales se pondrán en cola en lugar de devolver un mensaje de error.

- ***Consulta paralela***: Mediante la función consulta en paralelo, los datos pueden ser divididos entre varios servidores, por lo que una consulta se puede ejecutar en todos los servidores simultáneamente, reduciendo el tiempo de ejecución total. La consulta paralela funciona mejor en la búsqueda de datos a gran escala.

PgPool-II utiliza el protocolo backend y frontend de Postgres, y transmite mensajes entre un backend y un frontend. Por lo tanto, una aplicación de base de datos (*frontend*) piensa que PgPool-II es el servidor real Postgres, y el servidor (*backend*) ve a PgPool-II como uno de sus clientes. Debido a que PgPool-II es transparente para el cliente y el servidor, una aplicación de base de datos existente se puede utilizar con PgPool-II sin cambiar demasiado su código fuente.

Plataformas soportadas

PgPool-II funciona en Linux, Solaris, FreeBSD[27], y la mayoría de las arquitecturas de tipo UNIX, no es compatible con Windows. Las versiones de servidores Postgres soportadas son 6.4 o superiores. Para utilizar la función de consulta en paralelo, se debe utilizar la versión 7.4 o superiores.

Si se utiliza Postgres 7.3 o versiones más avanzadas, algunas características de PgPool-II no estarán disponibles, de todos modos no se debería utilizar una versión tan antigua.

Se debe asegurar de que todos los servidores Postgres estén utilizando la misma versión de Postgres, además no es recomendable mezclar diferentes instalaciones de Postgres con distintas opciones de construcción, como por ejemplo incluir soporte SSL o no, de usar diferentes tamaños de bloque, esto puede afectar la funcionalidad de PgPool-II.

---

[27] FreeBSD: Es un avanzado sistema operativo para arquitecturas x86 compatibles, amd64 compatibles, UltraSPARC®, IA-64, PC-98 y ARM.

Modo de Agrupación de Conexiones

En el modo de agrupación de conexiones (*pool de conexiones*), se pueden usar todas las funciones en modo Raw[28] y la función de pool de conexiones. Para activar este modo es necesario activar el parámetro *connection_cache*. Los siguientes parámetros tienen efecto sobre el pool de conexiones:

- ***max_pool***: Es el número máximo de conexiones en caché de los procesos hijos en PgPool-II, el cual reutiliza la conexión en caché si una conexión entrante se conecta a la misma base de datos con el mismo nombre de usuario, si no crea una nueva conexión con el servidor. Si el número de conexiones en caché excede el parámetro ***max_pool***, la conexión más antigua será descartada, y se utilizará el slot para la nueva conexión. El valor predeterminado es 4. Se debe tener en cuenta que el número de conexiones de procesos PgPool-II para los backends puede alcanzar el valor dado por ***num_init_children * max_pool***, este parámetro sólo se puede configurar en el arranque del servidor.

- ***connection_life_time:*** Las conexiones almacenadas en caché tienen un tiempo de expiración en segundos. Una conexión en caché que haya expirado será desconectada. El valor predeterminado es 0, lo que significa que las conexiones de caché no serán desconectadas.

- ***reset_query_list:*** Especifica los comandos SQL que son enviados para restablecer la conexión con el servidor cuando se sale de una sesión. Se pueden especificar múltiples comandos delimitando cada uno por "**;**". El valor predeterminado es ***reset_query_list = 'ABORT; DISCARD ALL'***.

[28] Raw: Este modo es útil simplemente para limitar el exceso de conexiones a los servidores, o para habilitar la recuperación de fallos con varios servidores.

Modo de Replicación

Este modo permite la replicación de datos entre los backends. Los siguientes parámetros de configuración se deben configurar junto con los parámetros anteriores, a continuación se detallan los más importantes:

- ***replication_mode***: Si este parámetro se establece como true, se activa el modo de replicación. El valor predeterminado es false.

- ***load_balance_mode:*** Cuando se establece en true, las consultas SELECT serán distribuidas a cada backends para balancear la carga. El valor predeterminado es false. Este parámetro sólo se puede configurar en el arranque del servidor.

- ***replication_stop_on_mismatch:*** Cuando este valor se establece en true, si todos los backends no devuelven la misma clase de paquete, los backends que se diferencian al conjunto de resultados más frecuentes se degeneran. Un típico caso de uso es una sentencia SELECT que es parte de una transacción, se establece *replicate_select* en true, y SELECT devuelve un número diferente de filas entre los backends.

  Se debe tener en cuenta que PgPool-II no examina el contenido de los registros devueltos por la sentencia SELECT. Si se establece en false, la sesión se termina y los backends no se degeneran. El valor predeterminado es false.

- ***replicate_select:*** Cuando este parámetro se establece en true, PgPool-II replica las consultas SELECT en el modo de replicación. Si es falso, sólo las envía a la base de datos principal. El valor predeterminado es false. Si una consulta SELECT está dentro de un bloque de transacción explícita, *replicate_select* y *load_balance_mode* afectarán el funcionamiento de la replicación.

- ***recovery_user:*** Este parámetro especifica un nombre de usuario Postgres para la recuperación en línea, puede cambiarse sin necesidad de reiniciar el sistema.

- ***recovery_password:*** Este parámetro especifica una contraseña Postgres para la recuperación en línea, se puede cambiar sin necesidad de reiniciar el sistema.

Modo Master Slave

Este modo se utiliza para unir PgPool-II con otro software de replicación maestro/esclavo (como Slony-I y replicación por Streaming), que es responsable de realizar la replicación de datos real. Se debe configurar la información en los nodos de las bases de datos (*backend_hostname, backend_port, backend_weight, backend_flag* y *backend_data_directory*) si se necesita la funcionalidad de recuperación en línea, de la misma manera que en el modo de replicación. Además se debe configurar *master_slave_mode* y *load_balance_mode* en true.

PgPool-II entonces enviará las consultas que necesitan ser replicadas en la base de datos maestra, y si es posible con las otras consultas se realizará un balanceo de carga. Las consultas enviadas a la base de datos maestra porque no pueden ser balanceadas son valoradas en el algoritmo de balanceo de carga.

En el modo *maestro/esclavo*, las sentencias DDL y DML[29] de tabla temporal se pueden ejecutar sólo en el nodo principal. La instrucción SELECT puede ser ejecutada de manera forzada en el maestro, pero para eso es necesario poner como comentario / * NO LOAD BALANCE * / antes de la instrucción SELECT.

En el modo *maestro/esclavo*, el parámetro *replication_mode* debe configurarse en false y el parámetro *master_slave_mode* en true, este modo tiene un modo *master_slave_sub* que por defecto es *Slony*. También puede configurarse a Stream, si se desea trabajar con el sistema replicación integrado de Postgres (*Streaming Replication*). Se debe reiniciar PgPool-II si es que se cambian algunos de los parámetros anteriores.

Es posible configurar *white_function_list* y *black_function_list* para controlar el balanceo de carga en modo *maestro/esclavo*.

---

[29] DML (*Data Manipulation Language*): El lenguaje de manipulación de datos sirve para llevar a cabo las transacciones en la base de datos, entendiéndose por transacciones a los procesos de inserción, actualización, eliminación y selección.

Modo Paralelo

En este modo se activa la ejecución paralela de consultas, las tablas pueden ser divididas, y los datos distribuidos a cada nodo, por otra parte las funciones de replicación y de balanceo de carga se pueden utilizar al mismo tiempo. En el modo paralelo, los parámetros *replication_mode* y *load_balance_mode* se establecen en true en ***pgpool.conf***, *master_slave* se establece en false y *parallel_mode* se establece como true. Cuando este parámetro es cambiado PgPool-II debe ser reiniciado.

Arrancando/Parando PgPool-II

Para iniciar o detener PgPool-II:

- ***Iniciar PgPool-II***: Todos los backends y el sistema de base de datos se deben iniciar si es necesario antes de arrancar PgPool-II. La sentencia es *pgpool [-c][-f config_file][-a hba_file][-F pcp_config_file][-n][-D][-d]*

-c	--clear-cache	Borra la caché de consultas.
-f config_file	--config-file config-file	Especifica *pgpool.conf.*
-a hba_file	--hba-file hba_file	Especifica *pool_hba.conf.*
-F pcp_config_file	--pcp-password-file	Especifica *pcp.conf.*
-n	--no-daemon	Modo no daemon (no se desconecta la terminal).
-D	--discard-status	Descartar el archivo *pgpool_status* y no restaura el estado anterior.
-C	--clear-oidmaps	Descartar mapas OID en *memqcache_oiddir* de el caché de consultas en

		memoria.
-d	--debug	Modo debug.

- ***Detener PgPool-II:*** Hay dos maneras de detener PgPool-II. Una es utilizando un comando de PCP[30], y la otra con un comando PgPool-II. A continuación se muestra un ejemplo del comando PgPool-II. El comando es *pgpool [-f config_file][-F pcp_config_file] [-m {s[mart]|f[ast]|i[mmediate]}] stop*

-m s[mart]	--mode s[mart]	Espera a que los clientes se desconecten y apaga PgPool-II (predeterminado)
-m f[ast]	--mode f[ast]	No espera a que los clientes se desconecten y apaga PgPool-II de inmediato
-m i[mmediate]	--mode i[mmediate]	Lo mismo que '-m f '

Recargando la configuración de archivos PgPool-II

PgPool-II puede volver a cargar los archivos de configuración sin reiniciar el sistema con la sentencia *pgpool [-c][-f config_file][-a hba_file][-F pcp_config_file] reload.*

-f config_file	--config-file config-file	Especifica *pgpool.conf.*
-a hba_file	--hba-file hba_file	Especifica *pool_hba.conf.*
-F pcp_config_file	--pcp-password-file	Especifica *pcp.conf.*

[30] 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.

Se debe tener en cuenta que algunos elementos de la configuración no se pueden cambiar mediante la recarga, la nueva configuración tiene efecto después de un cambio a nuevas sesiones.

Mostrando Comandos (Show)

PgPool-II proporciona información a través del comando SHOW, que es una instrucción SQL real, pero PgPool-II intercepta este comando cuando se solicita información específica a PgPool-II. Las opciones disponibles son:

- ***pool_status***: Se usa para obtener la configuración.
- ***pool_nodes***: Se utiliza para obtener la información de los nodos.
- ***pool_processes***: Es usado para obtener la información de los procesos PgPool-II.
- ***pool_pools***: Se usa para obtener información sobre PgPool-II.
- ***pool_version***: Se utiliza para obtener la configuración.

Backup

Para realizar una copia de seguridad (*Backup*) de la base de datos del sistema, es posible hacerla por medio de un backup físico, un backup lógico (*pg_dump, pg_dumpall*) y el método PITR del mismo modo que Postgres. Se debe tener presente que el uso de un backup lógico y PITR debe ser realizado en el directorio de Postgres, en lugar de PgPool-II para evitar errores causados por *load_balance_mode* y *replicate_select*.

Recuperación en línea

Mientras que PgPool-II funciona en el modo de replicación, puede sincronizar una base de datos y conectar un nodo al mismo tiempo que presta servicios a clientes, esto se conoce como recuperación en línea. Un nodo objetivo de recuperación debe estar en el estado independiente antes de efectuar la recuperación en línea, si se desea agregar dinámicamente un servidor Postgres, se

debe poner el *backend_hostname* y sus parámetros asociados y volver a cargar *pgpool.conf* , de esta forma PgPool-II registra este nuevo nodo como un nodo independiente.

Se debe tener la precaución de detener si es que corresponde el *autovacuum* en el nodo principal, el autovacuum puede cambiar el contenido de la base de datos y puede causar inconsistencias si se está ejecutando la recuperación en línea. Esto sólo se aplica si se está recuperando de un mecanismo de copia simple, pero no se aplica si se está utilizando el mecanismo PITR de Postgres.

Si se ha iniciado el servidor Postgres destino, es necesario apagarlo, PgPool-II realiza la recuperación en línea en dos fases separadas. El cliente deberá esperar algunos segundos o minutos para conectarse a PgPool-II, mientras que un nodo de recuperación sincroniza la base de datos. Se deben seguir los siguientes pasos:

- Punto de control (Checkpoint).
- Primera etapa de la recuperación en línea.
- Esperar hasta que todos los clientes se hayan desconectado.
- Punto de control (Checkpoint).
- Segunda etapa de recuperación en línea.
- Iniciar el postmaster (Ejecutar *pgpool_remote_start*).
- Agregar un nodo.

El primer paso de la sincronización de datos se llama ***First Stage***, en donde los datos son sincronizados. En esta etapa, los datos pueden ser actualizados o recuperados de cualquier tabla de forma simultánea. Se puede especificar la ejecución de un script durante esta primera etapa, en donde PgPool-II le pasa tres argumentos al script:

- La ruta de acceso al clúster de base de datos de un nodo maestro.
- El nombre del host de un nodo objetivo de recuperación.
- La ruta de acceso al clúster de base de datos de un nodo objetivo de recuperación.

La sincronización de datos finaliza durante lo que se llama ***Second Stage***. Antes de entrar en la segunda etapa, PgPool-II espera a que todos los clientes se desconecten, se bloquea cualquier nueva conexión entrante hasta que la segunda etapa se termine. Después de que todas las conexiones han terminado, PgPool-II combina datos actualizados entre la primera etapa y la segunda etapa, este es el paso final de la sincronización de datos.

Es muy importante tener en cuenta que existe una restricción sobre la recuperación en línea, si PgPool-II está instalado en varios hosts, la recuperación en línea no funciona correctamente, porque PgPool -II tiene que parar a todos los clientes durante la segunda etapa de recuperación en línea. Si hay varios hosts PgPool-II, sólo uno de ellos habrá recibido el comando de recuperación en línea y bloqueará las demás conexiones.

Solución de Problemas

A continuación se describen los problemas y sus soluciones temporales mientras se utiliza PgPool-II:

- ***Verificación de fallas de estado***: La característica de comprobación de estado de PgPool-II detecta fallas en los nodos de base de datos.

```
2010-07-23 16:42:57 ERROR: pid 20031: health check failed. 1 th host foo at port 5432 is dow
n
2010-07-23 16:42:57 LOG:   pid 20031: set 1 th backend down status
2010-07-23 16:42:57 LOG:   pid 20031: starting degeneration. shutdown host foo(5432)
2010-07-23 16:42:58 LOG:   pid 20031: failover_handler: set new master node: 0
2010-07-23 16:42:58 LOG:   pid 20031: failover done. shutdown host foo(5432)
```

  El registro muestra que el nodo de base de datos 1 (*host foo*) se cae y se desconecta de PgPool-II, y posteriormente el nodo de base de datos 0 se convierte en maestro. Se debe comprobar el nodo base de datos 1 y eliminar la posible causa de fallas, luego si es posible realizar una recuperación en línea con la base de datos nodo 1.

- ***Error de lectura desde la interfaz***: Este registro indica que el programa de la aplicación no se desconecta correctamente desde PgPool-II. Las posibles causas pueden ser errores de aplicaciones cliente, terminación forzada (*kill*) de una aplicación cliente o falla temporal de la red. Este tipo de eventos no llevan a una destrucción de la base de datos o a un problema de la consistencia de los datos, es sólo una advertencia acerca de una

violación del protocolo. Es recomendable comprobar las aplicaciones y redes si el mensaje sigue apareciendo.

```
2010-07-26 18:43:24 LOG:   pid 24161: ProcessFrontendResponse: failed to read kind from frontend. fron
tend. frontend abnormally exited
```

- ***Errores por diferencias de tipo***: Es posible recibir este tipo de error cuando PgPool-II funciona en modo de replicación.

```
2010-07-22 14:18:32 ERROR: pid 9966: kind mismatch among backends. Possible last query wa
s: "FETCH ALL FROM c;" kind details are: 0[T] 1[E: cursor "c" does not exist]
```

PgPool-II espera las respuestas de los nodos de base de datos luego de enviarles un comando SQL, este mensaje indica que no todos los nodos de base de datos devuelven el mismo tipo de respuesta. Obtendremos la sentencia SQL que posiblemente provocó el error después de "*Possible last query was*". Luego sigue el tipo de respuesta, si la respuesta indica un error, se mostrará el mensaje de error de Postgres. Se pueden observar las respuestas que muestra el nodo base de datos: "***0 [T]***" (comenzando a enviar una descripción de fila), y "***1 [E***" indica que el nodo de base de datos **1** devuelve un error con el mensaje "*cursor **c** no existe* ", mientras que el nodo de base de datos **0** envía una descripción de la fila.

Por otro lado este error se podrá ver también cuando se trabaje también en modo *maestro/esclavo*. Por ejemplo, incluso en este modo, será enviado un conjunto de comandos a todos los nodos de base de datos para mantener a todos los nodos de base de datos en el mismo estado. Se deberán comprobar las bases de datos y volver a sincronizarlas por medio de la recuperación en línea si se observa que no están sincronizadas.

- ***PgPool-II detecta una diferencia en el número de tuplas insertadas, actualizadas o eliminadas***: En el modo de replicación, PgPool-II detecta un número diferente de filas INSERT / UPDATE / DELETE en los nodos afectados.

```
2010-07-22 11:49:28 ERROR: pid 30710: pgpool detected difference of the number of inserted,
updated or deleted tuples. Possible last query was: "update t1 set i = 1;"
2010-07-22 11:49:28 LOG:   pid 30710: ReadyForQuery: Degenerate backends: 1
2010-07-22 11:49:28 LOG:   pid 30710: ReadyForQuery: Affected tuples are: 0 1
```

En el ejemplo anterior, el número devuelto de filas actualizadas por ***update t1 set i = 1*** es diferente entre los nodos de base de datos, la siguiente línea indica que la base de datos ***1*** quedó desconectada como consecuencia de ello, y que el número de filas afectadas para el nodo de base de datos ***0*** es 0, mientras que para el nodo de base de datos ***1*** que es 1. En este caso se debe parar el nodo de base de datos que tiene mal los datos y hacer una recuperación en línea.

Restricciones

A continuación se enumeran las principales restricciones que se pueden encontrar al utilizar PgPool-II:

- ***Funcionalidad de Postgres***: Si se utiliza la función *pg_terminate_backend()* para detener un motor, esto desencadenará una recuperación de fallos. Al día de hoy no existe una solución alternativa, por lo que no se debe usar esta función.

- ***Objetos grandes***: La versión de PgPool-II 2.3.2 o sus versiones posteriores admiten la replicación de objetos grandes, si el servidor Postgres es 8.1 o una versión posterior, para ello es necesario habilitar la directiva *lobj_lock_table* en *pgpool.conf*. La replicación de objetos grandes con la función *lo_import* no está soportada.

- ***Tablas temporarias en modo maestro/esclavo:*** La creación, inserción, actualización y/o eliminación de tablas temporales siempre se ejecuta en el nodo maestro. Con PgPool-II 3.0 o en versiones posteriores, la sentencia SELECT en tablas temporarias también se ejecuta en el nodo maestro. Sin embargo, si el nombre de la tabla temporal es utilizado como un literal en una sentencia SELECT, no hay manera de detectarlo, y será balanceada la carga de la sentencia SELECT, esto disparará un error "*No se encuentra la tabla*" o se encontrará otra tabla con el mismo nombre, para evitar este problema se debe usar el comentario SQL / * NO LOAD BALANCE * /.

```
Sample SELECT which causes a problem:
SELECT 't1'::regclass::oid;
```

El comando de *psql\d* utiliza nombres de tabla literales, la versión de Pgpool-II 3.0 o versiones posteriores comprueban si la sentencia SELECT incluye el acceso a los

catálogos del sistema y envía esas consultas al maestro, de esta manera se evita el problema.

- ***Consultas***: A continuación se hace referencia a las principales consultas que no pueden ser procesados por PgPool-II:

  - ***Insert (en modo paralelo):*** No se puede usar DEFAULT con la clave de particionamiento de columna, por ejemplo si la columna ***x*** en la tabla ***t*** es la clave de columna de particionamiento, esto no es válido, además las funciones no se pueden utilizar para este valor.

```
INSERT INTO t(x) VALUES (DEFAULT);
```

```
INSERT INTO t(x) VALUES (func());
```

    Los valores constantes se deben utilizar con INSERT con la clave de particionamiento. SELECT INTO e INSERT INTO ... SELECT no están soportadas.

  - ***Update (en modo paralelo):*** La consistencia de datos entre los backend se puede perder si los valores de las claves de columnas de particionamiento son actualizados. Una operación no se puede deshacer si una consulta ha provocado un error en algunos backend debido a una violación de una restricción. Si se llama a una función en la cláusula WHERE, esa consulta no puede ser ejecutada correctamente, por ejemplo:

```
UPDATE branches set bid = 100 where bid = (select max(bid) from beances);
```

  - ***Select … for Update (en modo paralelo):*** Si una función es llamada en la cláusula WHERE, esa consulta no puede ser ejecutada correctamente. Por ejemplo:

```
SELECT * FROM  branches where bid = (select max(bid) from beances) FOR UPDATE;
```

  - ***Copy (en modo paralelo):*** La función COPY BINARY no es soportada, la copia desde archivos tampoco es soportada, sólo COPY FROM STDIN y COPY TO STDOUT son soportadas.

- ***Alter/Create Table (en modo paralelo):*** Para actualizar las reglas de particionamiento, se debe reiniciar PgPool-II, con el fin de leer las reglas desde la base de datos del sistema.

- ***Transaction (en modo paralelo):*** Sentencias SELECT ejecutadas dentro de un bloque de transacción serán ejecutadas en una transacción independiente, por ejemplo:

```
BEGIN;
INSERT INTO t(a) VALUES (1);
SELECT * FROM t ORDER BY a; <-- INSERT above is not visible from this SELECT statement
END;
```

- ***Schemas (en modo paralelo):*** Los objetos de un esquema que no sean públicos deben estar totalmente descriptos como *schema.object.* PgPool-II no puede resolver el esquema correcto cuando la ruta se establece como *set search_path = xxx* y el nombre del esquema es omitido en una consulta.

- ***Table Name/Column Name (en modo paralelo):*** Una tabla o un nombre de columna no pueden empezar por *pool_*. Al reescribir las consultas, estos nombres son usados para el procesamiento interno.[31]

[31] http://www.pgpool.net/docs/latest/pgpool-en.html

2.5.3.3 **Bucardo**

El corazón de Bucardo es un demonio Perl que detecta peticiones NOTIFY y las activa mediante la conexión a bases de datos remotas, copiando los datos de una lado a otro. Toda la información que el demonio necesita se almacena en la base de datos principal de Bucardo, incluyendo una lista de todas las bases de datos que participan en la replicación y la forma de llegar a ellas, todas las tablas que se van a replicar, y cómo se va a replicar cada una.

El primer paso en el funcionamiento de Bucardo es agregar dos o más bases de datos en la base de datos principal de Bucardo, luego se agrega información sobre qué tablas se van a replicar, así como cualquier grupo de tablas. A continuación, se añaden las sincronizaciones, estas se denominan acciones de replicación, se copia un grupo específico de tablas de un servidor a otro servidor, o grupo de servidores.

Una vez que Bucardo se ha configurado, los disparadores comienzan a almacenar información sobre las filas que han cambiado en determinadas tablas. Para una sincronización multimaestro, se lleva a cabo el siguiente proceso:

1. Se hace un cambio en la tabla y luego se graba en la tabla *bucardo_delta*.
2. Se envía una notificación al demonio principal de Bucardo para hacerle saber que la tabla ha cambiado.
3. El demonio notifica al controlador para que se sincronice y vuelve a escuchar.
4. El controlador crea un *kid*[32] para manejar la replicación, o señales de una ya existente.
5. El kid comienza una nueva transacción y deshabilita los disparadores y reglas en las tablas correspondientes.

---

[32] Kid: Un programa creado por el controlador a cargo de hacer el trabajo real, por lo general creado con un mandato específico, como por ejemplo "replicar de A hacia B para sincronización X". Sólo habla con el controlador que lo creó, también configurable en cuanto a si sale después de hacer su trabajo, o se cuelga a esperar a otro puesto de trabajo desde el controlador.

6. A continuación, se recopila una lista de las filas que han cambiado desde la última replicación, y luego las compara para determinar que debe hacerse.
7. Si se produce un conflicto, entonces el controlador de conflictos estándar, o uno personalizado se ejecuta para resolver el conflicto.
8. Los disparadores y las reglas se habilitan nuevamente y se confirma la transacción.
9. Si falla la transacción, entonces se ejecutan los controladores de excepciones personalizados.
10. El programa *kid* le indica al controlador de que ha finalizado.

Qué es Bucardo?

Bucardo es un programa de replicación para dos o más bases de datos Postgres. Se trata específicamente, de un sistema de replicación asíncrono, multi-maestro, maestro-esclavo y basado en tabla. Está desarrollado en Perl, y utiliza un amplio uso de disparadores, *Pl/ Pgsql*, y *Pl/PerlU*.

Requerimientos

Bucardo es un script desarrollado en Perl que requiere que los siguientes módulos sean instalados antes de que se pueda ejecutar:

- ***DBI*:** El DBI es el módulo de interfaz de base de datos estándar de Perl, en el que se definen un conjunto de métodos, variables y convenciones que proporcionan una interfaz de base de datos consistente independiente de la base de datos real que se utiliza.
- ***DBD::Pg*:** Es un módulo de Perl que trabaja con el módulo DBI para proveer acceso a bases de datos PostgreSQL.
- ***Sys::Hostname*:** Es un módulo que suministra una sola función, que es la de obtener el nombre del host.

- **Sys::Syslog:** Este módulo facilita el envío de mensajes al *daemon* de *syslog* . Es una interfaz para el programa UNIX *syslog ()*.

- ***Boolean:*** La mayoría de los lenguajes de programación tienen un tipo de datos Boolean nativo. Perl cuenta con un módulo que proporciona soporte booleano básico, mediante la definición de dos objetos especiales que son true y false.

- ***DBIx::Safe (1.2.4):*** Es un módulo de Perl que permite un acceso seguro y controlado para manejar la DBI. Se utiliza por Bucardo para asegurarse de que el código personalizado no interfiera con el funcionamiento normal.

Bucardo requiere de una base de datos para instalar su esquema principal, esta base de datos debe ser de la versión de Postgres 8.1 o una versión superior, y debe tener disponible los lenguajes PL/pgSQL y PL/PerlU. Además, para instalar el script de instalación se requiere hacerlo con un superusuario, se recomienda crear un nuevo usuario llamado *Bucardo* para este propósito. Las bases de datos que participan en la replicación deben ser de la versión de Postgres 8.1 o versiones superiores.

Debe estar disponible el lenguaje PL/pgSQL, y si la base de datos sólo se utiliza como un objetivo para sincronizaciones de *fullcopy*[33], en cuyo caso el único requisito es la versión de Postgres 8.1.

Bucardo requiere un sistema de tipo Unix, en la actualidad sólo se ha probado en variantes de Linux, pero no hay razón para sospechar que no debe trabajar en BSD, Solaris y otros sistemas similares. Bucardo no funciona actualmente en sistemas Win32, pero probablemente no sea tan difícil de lograrlo en este momento.

---

[33] Fullcopy: Es un tipo de sincronización de Bucardo que copia toda una tabla de una base de datos que funciona como maestro, a una o más bases de datos esclavas. Las filas de las bases esclavas se eliminan, y las tablas se rellenan con el comando COPY. Esta es la única sincronización que se puede utilizar para las tablas que no tienen ninguna clave primaria y sin un único índice.

Replicación en Bucardo con modo maestro/esclavo

Mientras que con Bucardo se puede hacer la replicación *maestro/maestro*, la mayoría de las personas lo utilizan sólo para la replicación *maestro/esclavo*, en donde una base de datos maestra envía cambios a una o varias bases de datos de esclavas.

Replicar entre más de dos maestros

Actualmente Bucardo no pude replicar entre más de dos nodos maestros, sólo se admite de maestro a maestro, como así también de un maestro a muchos esclavos.

Bucardo necesita ejecutarse en la base de datos que se está replicando

El programa Bucardo se puede ejecutar en cualquier lugar, y no tiene que estar en cualquiera de los servidores implicados en la replicación. La ventaja principal de tener que ejecutarse en el mismo servidor es que se reduce el tiempo de red.

Por qué Bucardo da advertencias al iniciar acerca de las secuencias desiguales

Esto es consecuencia de que el usuario *Bucardo* tiene diferentes rutas de búsqueda en diferentes servidores de Postgres, esto se ha corregido en la versión 4.5.0.

Qué tan rápido se producirá la replicación

No hay una respuesta simple a esta pregunta, ya que depende de la cantidad de tablas que se replican en una sincronización, la velocidad de la red, lo ocupada que este la base de datos, etc. Como regla general, los cambios llegan a las otras bases de datos en cuestión de solamente uno o dos segundos.

Replicar DDL con Bucardo

Bucardo no puede replicar DDL, Bucardo está basado en triggers y Postgres no proporciona triggers en las tablas del sistema.[34]

[34] http://bucardo.org

2.5.3.4 PgCluster II

Qué es PgCluster?

PGCluster II es un sistema de replicación sincrónico multimaestro para PostgreSQL. La replicación está basada en consultas, en donde los nodos independientes de base de datos pueden ser replicados. No cuenta con un único punto de fallo ya que posee con un balanceador de carga, un servidor de replicación y un clúster de base de datos. El proceso de restauración debe hacerse manualmente. Es posible añadir un clúster de base de datos y un servidor de replicación sobre la marcha, PGCluster II cuenta con la siguiente estructura:

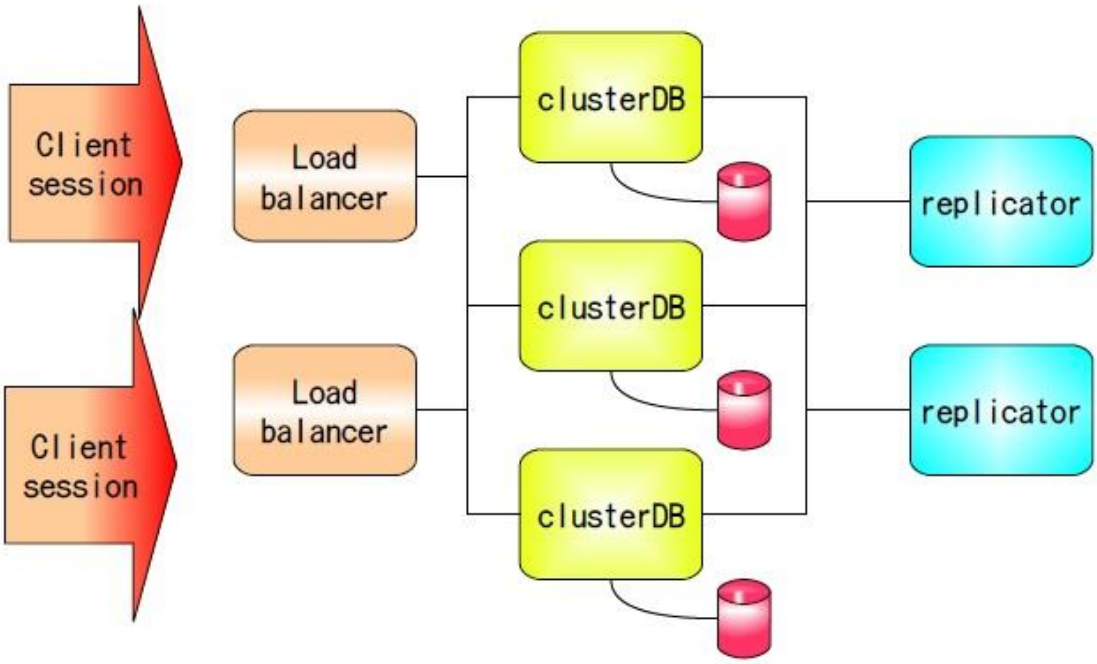

Imagen 4 – Estructura de PGCluster II - www.pgcon.org/2007/schedule/events/6.en.html -Atsushi Mitani.

Datos compartidos del sistema de clustering

Para el almacenamiento de datos compartidos en discos compartidos, se pueden encontrar entre otros los distintos protocolos y sistemas de almacenamiento:

- NFS[35], GFS[36], GPFS[37], etc.
- NAS[38].

Estado del bloqueo y de la caché compartida con IPC Virtual, en la siguiente figura se puede observar la estructura de datos compartidos.

---

[35] NFS (*Network File System*): El sistema de archivos de red es un protocolo de nivel de aplicación, según el Modelo OSI. Es utilizado para sistemas de archivos distribuidos en un entorno de red de computadoras de área local. Posibilita que distintos sistemas conectados a una misma red accedan a ficheros remotos como si se tratara de ficheros locales.

[36] GFS (*Google File System*): El Sistema de Archivos Google, es un sistema de archivos distribuido propietario desarrollado por Google Inc. Está especialmente diseñado para proveer eficiencia, fiabilidad de acceso a datos usando sistemas masivos de clúster de procesamiento en paralelo.

[37] GPFS (*General Parallel File System*): Es un sistema de ficheros distribuido de alto rendimiento desarrollado por IBM. GPFS proporciona un acceso concurrente de alta velocidad a aplicaciones que se encuentran ejecutando en múltiples nodos de un clúster dando una visión de un disco compartido entre todos ellos.

[38] NAS (*Network Attached Storage*): Es el nombre dado a una tecnología de almacenamiento dedicada a compartir la capacidad de almacenamiento de un servidor con computadoras personales o servidores clientes a través de una red que normalmente es TCP/IP.

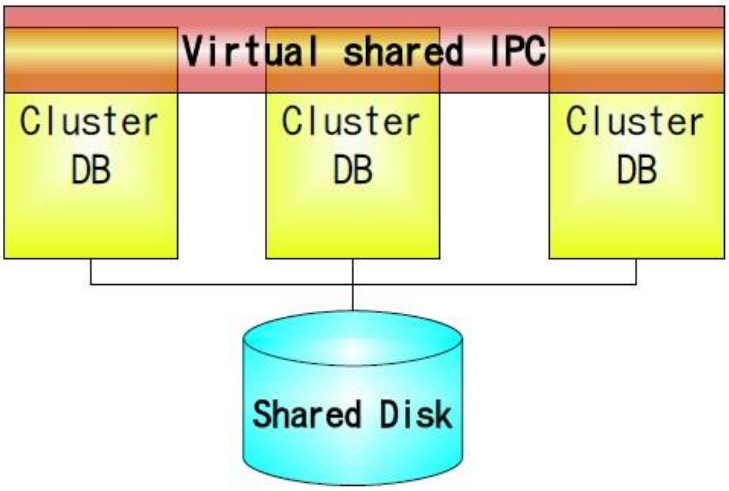

Imagen 5 – Concepto de Datos Compartidos - www.pgcon.org/2007/schedule/events/6.en.html - Atsushi Mitani.

IPC Virtual

Se trata de compartir semáforos y memoria compartida entre los nodos de base de datos, donde la escritura en los nodos remotos es a través de procesos del clúster y la lectura es desde el directorio del nodo local.

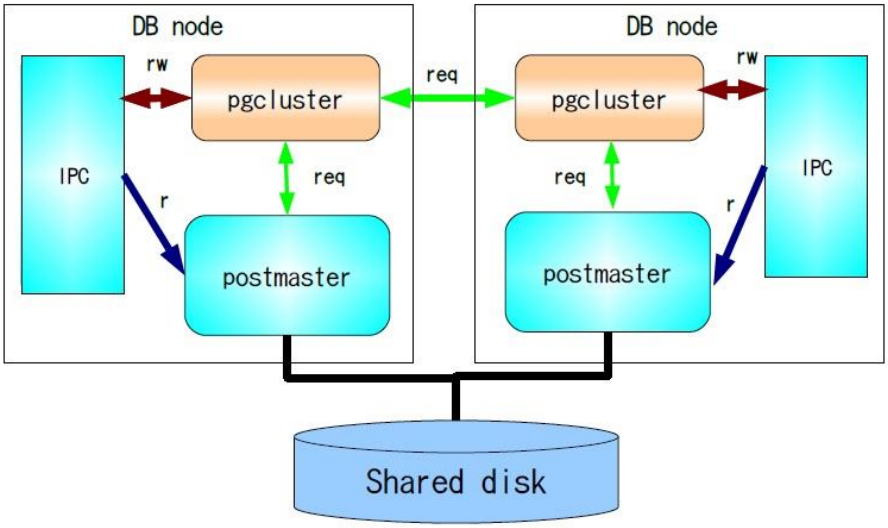

Imagen 6 – Memoria compartida - www.pgcon.org/2007/schedule/events/6.en.html -Atsushi Mitani.

Semáforos

Se utilizan para bloquear el control, la cantidad de semáforos que se utilizan depende del parámetro de configuración *max-connections*. De forma predeterminada, se usan 7 x 16 semáforos, se requiere de una tabla de mapeo.

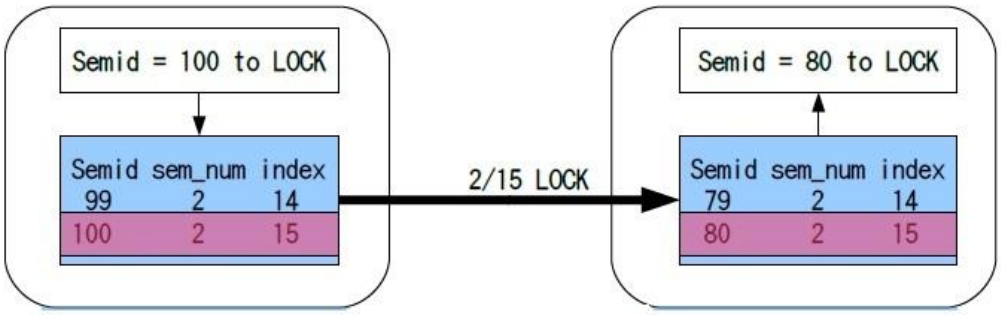

Imagen 7 – Tabla de mapeo - www.pgcon.org/2007/schedule/events/6.en.html -Atsushi Mitani.

Memoria Compartida

La memoria compartida se comunica durante cada proceso de backend. Almacena los datos de los registros, caches, buffers, etc. Se asigna una única memoria compartida, la cual se divide en una serie de celdas. Existen más de 100 entradas de punteros.

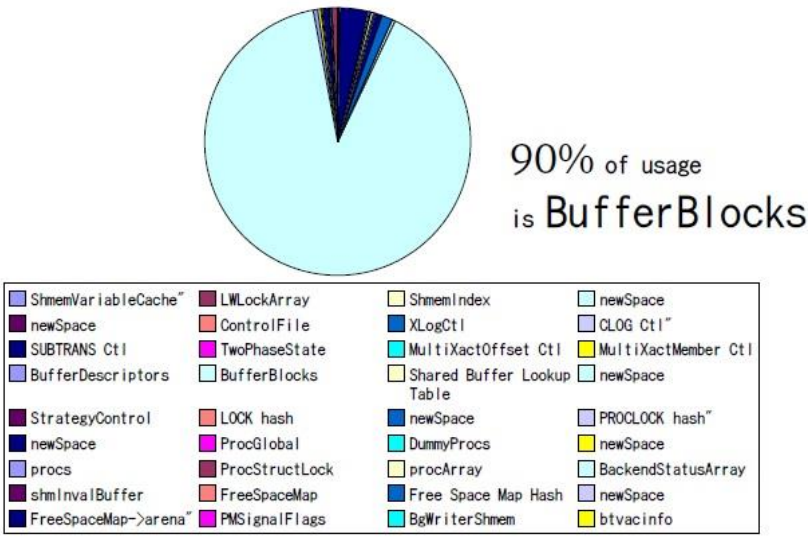

Imagen 8 – Memoria Compartida Utilizada - www.pgcon.org/2007/schedule/events/6.en.html - Atsushi Mitani.

Secuencia de Procesos

A continuación se muestran una serie de figuras en las cuales se pueden observar las distintas secuencias de inicio, de IPC sync y de parada.

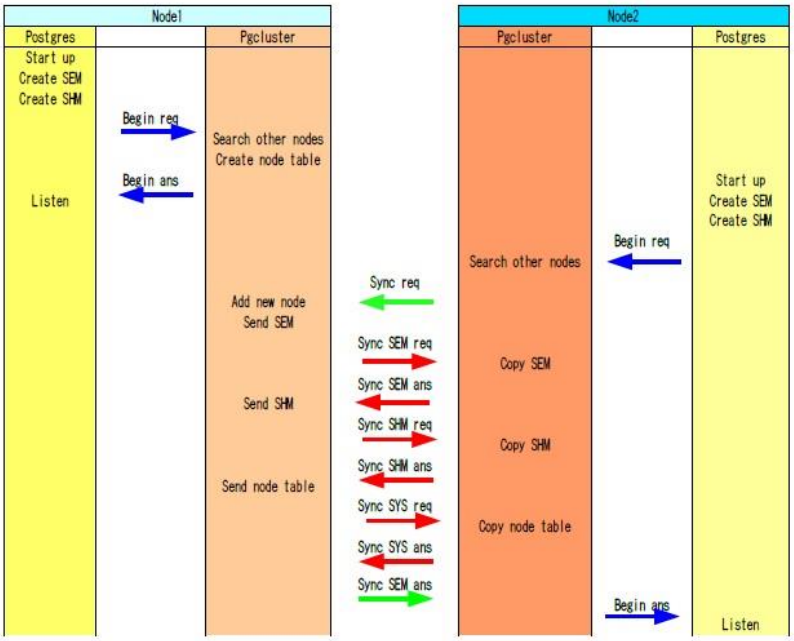

Imagen 9 – Secuencia de inicio - www.pgcon.org/2007/schedule/events/6.en.html -Atsushi Mitani.

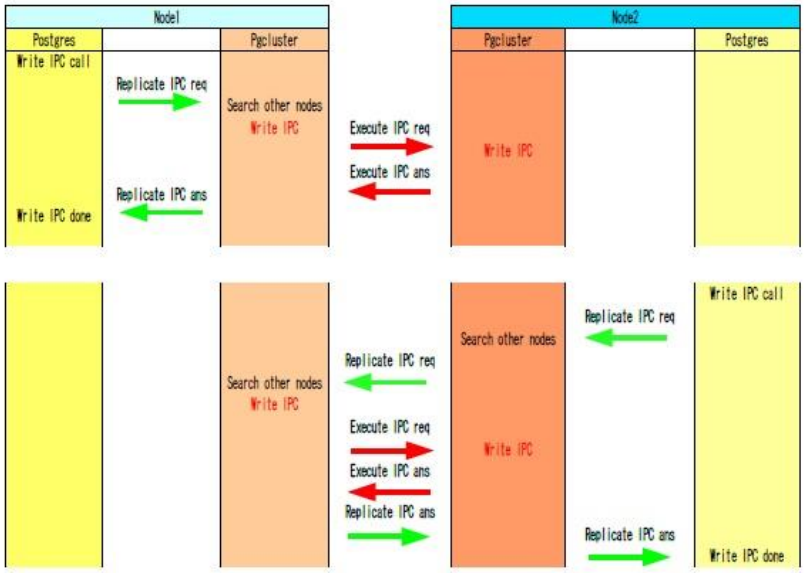

Imagen 10 – Secuencia IPC sync - www.pgcon.org/2007/schedule/events/6.en.html -Atsushi Mitani.

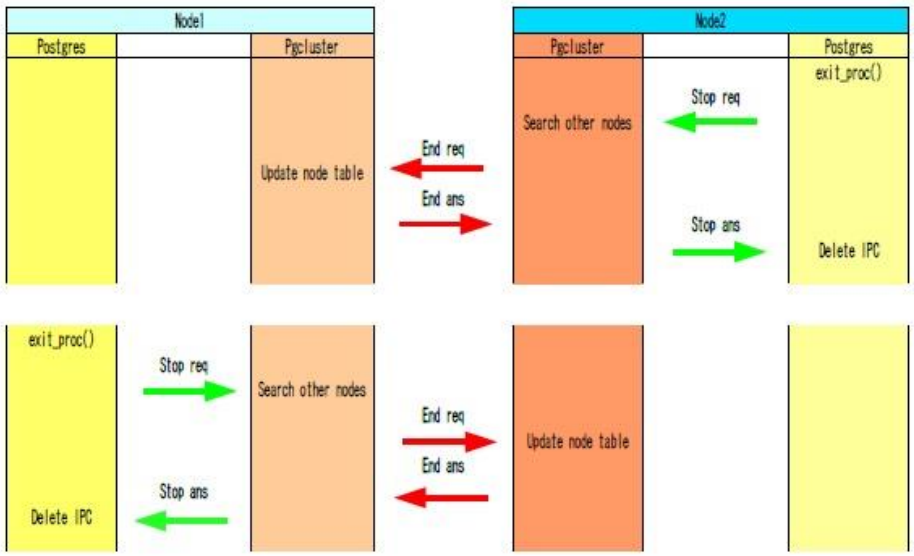

Imagen 11 – Secuencia de parada - www.pgcon.org/2007/schedule/events/6.en.html -Atsushi Mitani.

Ventajas y desventajas de PGCluster II

Dentro de las ventajas se puede decir que PGCluster II posee suficiente *HA* (High Availability), lo que significa que tiene una alta disponibilidad. Otra ventaja es su buen rendimiento para la carga de datos de lectura, y en cuanto a su costo solo se requieren servidores normales tipo PC y cuenta con licencias de software BSD[39].

En las desventajas que tiene PGCluster II se destacan problemas de rendimiento, es bastante malo para la carga de datos de escritura, además tiene problemas de mantenimiento y poca documentación.

[39] BSD (*Berkeley Software Distribution*): Es un sistema operativo derivado del sistema Unix nacido a partir de los aportes realizados a ese sistema por la Universidad de California en Berkeley.

Coexistencia entre HA (High Availability) y HP (High Performance)

HA y *HP* entran en conflicto entre sí, dado que *HA* necesita redundancia y *HP* requiere una respuesta rápida.

Desde el punto de vista de rendimiento posee escalas de replicación para las operaciones de lectura de datos y no para la escritura de los mismos. La consulta en paralelo tiene efecto tanto en *HA* como en *HP*, sin embargo no es sencillo añadir redundancia con *HA*. El clustering de datos compartidos cuenta también con escalas para ambos. Sin embargo no es aconsejable para datos de gran tamaño, y además el disco compartido necesita redundancia.

Disco compartido

Cada nodo comparte todos los clúster de base de datos: base/, global/, pg_clog/, pg_multixact/, pg_subtrans/, pg_tblspc/, pg_twophase/, pg_xlog/.

Cada nodo tiene sus propios archivos de configuración: pg_hba.conf, pg_ident.conf, postgresql.conf y pgcluster.conf.

Cada nodo debe tener los mismos valores de configuración ya sea en las conexiones (max_connections) o con el uso de recursos (memoria, espacio libre en el mapa).

pgcluster.conf

En la tabla de descripción de PGCluster II se puede indicar el nombre de host/IP y el puerto, pueden describirse múltiples servidores y el mejor de ellos puede ser designado como el maestro. En el nodo de auto descripción podemos configurar el nombre de host/IP, el puerto y sólo puede describirse un nodo.

Solución conveniente para HA y HP

La siguiente figura muestra un solución conveniente para *HA* y para *HP*, la cual es medida por dos parámetros como la disponibilidad y el desempeño.

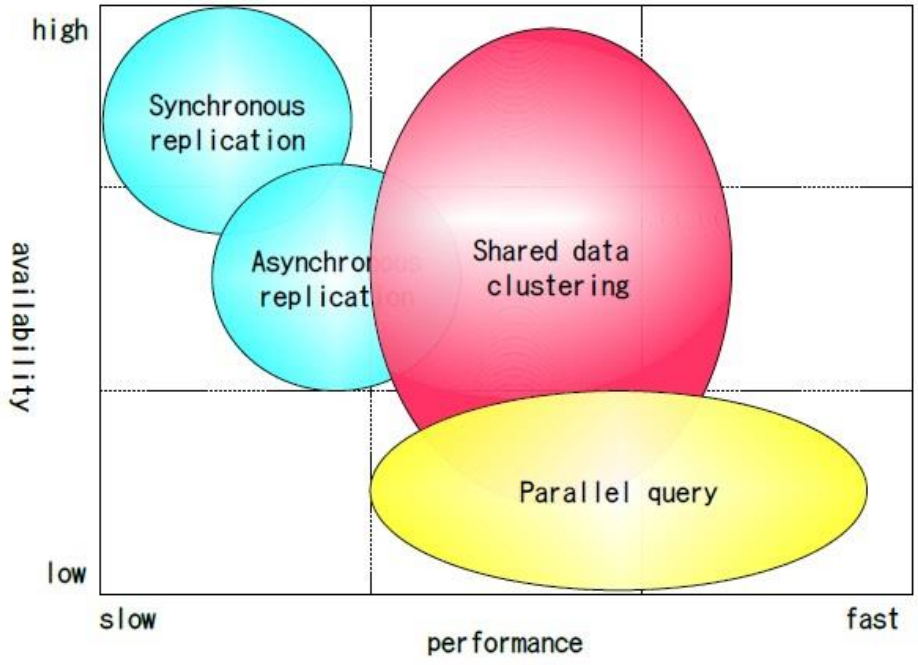

Imagen 12 – Solución conveniente para HA y HP - www.pgcon.org/2007/schedule/events/6.en.html -Atsushi Mitani.

Conclusión

El rendimiento debería mejorar aún más, determinados tipos de escrituras de datos (y borrados) en la memoria no necesitan sincronizarse. Los métodos de conversión (desde el offset al direccionamiento local) deberían mejorar. Se tendría que liberar el código fuente y la documentación lo antes posible.[40]

[40] http://www.pgcon.org/2007/schedule/attachments/26-pgcluster2pgcon.pdf

2.5.3.5 PyReplica

Motivación - Objetivos de PyReplica

Los objetivos principales que se pueden destacar por los cuales se desarrolló el sistema PyReplica son:

- Fácil instalación (scripts, sin compilación).
- Fácil administración (sin comandos).
- Fácil adaptación (filtrar y/o transformar).
- Fácil mantenimiento.
- Eficiencia (memoria, red, sin polling).
- Multiplataforma (Windows/Linux).

Características de PyReplica

En el sistema de replicación PyReplica, el cual se encuentra desarrollado en el lenguaje Python[41] se destacan las siguientes características:

- Asincrónico.
- Maestro/Esclavo y Multimaestro limitado.
- Replicación Condicional.

---

[41] Python: Es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis muy limpia y además que favorezca un código legible. Es un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional.

- Detección de conflictos.
- Notificaciones vía email.
- Monitoreo de las conexiones.
- Conexiones directas a los backends.
- Sin protocolos especiales (*SQL textual*).
- Si herramientas externas (*rsync*).
- Protegido con transacciones en dos fases.

PyReplica puede soportar la replicación de valores devueltos por funciones de fecha, aleatorias, secuenciales, etc. Pero como sucede con el sistema Slony-I y otros replicadores basados en disparadores, PyReplica no soporta:

- Replicación de DDL automática (pero puede usarse para propagar órdenes DDL a varios servidores).
- Replicación de secuencias.
- Replicación sincrónica (simultánea).
- Resolución de conflictos (se pueden evitar con *reglas/disparadores* y/o detectarlos).

Usos de PyReplica

PyReplica puede ser usado de dos modos distintos, los cuales son mencionados a continuación:

- ***Maestro/Esclavo:*** En donde el esclavo es de solo lectura.

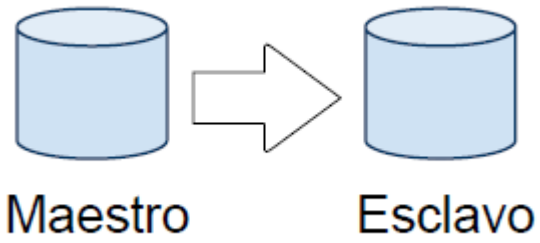

Imagen 13 - https://code.google.com/p/pyreplica/, página 5 – 2004-2010.

- Respaldo / Failover.
- Balanceo de carga.
- Datawarehouse / Consolidación.

- ***Multimaestro***: En donde ambos son maestro y esclavo (lectura/escritura)

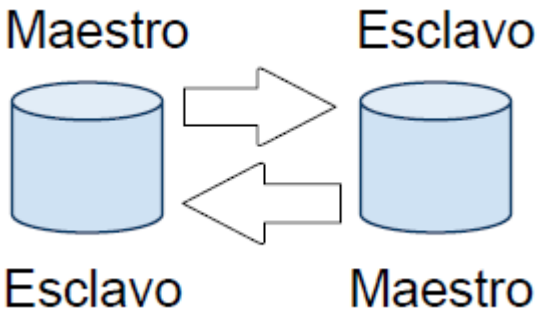

Imagen 14 - https://code.google.com/p/pyreplica/, página 5 – 2004-2010.

- Redundancia / Alta Disponibilidad.
- Servidores Remotos (Ej. puntos de venta).
- Servidores Móviles (Ej. vendedores).
- Requiere particionado lógico para evitar conflictos de escritura.

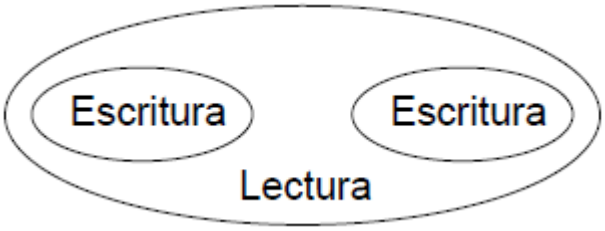

Imagen 15 - https://code.google.com/p/pyreplica/, página 5 – 2004-2010.

Programación de PyReplica

Dentro de PyReplica podemos encontrar las siguientes características en cuanto a su programación:

- Lenguaje Python:
  - Código compacto, simple y claro.
  - Baterías incluidas: email, hilos, configuración.
- Es un sistema relativamente simple de programar:
  - Posee 3 archivos fuentes principales, 500 líneas de código aproximadamente en total.
  - Menos de 150 líneas para el disparador y el replicador.
  - Aproximadamente es 1/8 del tamaño de Slony-I (disparador).
  - Desarrollo para Windows y Linux.
  - Sin compilaciones.
  - Conjunto de pruebas automatizadas.

Estructura de PyReplica

En PyReplica podemos encontrarnos con la siguiente estructura:

- ***py_log_trigger:***
  - Es el disparador de registro.
  - Detecta y almacena sentencias de replicación.
- ***pyreplica.py:***
  - Es el cliente de la replicación.
  - Ejecuta las sentencias de replicación.
- ***daemon.py***:
  - Es el demonio residente (configuración, hilos, etc.).

Disparador de registro py_log_replica

PyReplica cuenta con un disparador de registro, en el cual se pueden encontrar las siguientes características:

- Conversión de datos Python – PostgreSQL.
- Detecta cambios y genera sentencias SQL:
  - Inserciones (INSERT).
  - Modificaciones (UPDATE y SELECT).
  - Eliminaciones (DELETE).
- Verifica condiciones.
- Almacena el SQL en el registro (*replica_log*).
- Notifica a las réplicas.

Replicador (cliente y demonio)

PyReplica cuenta con un replicador, el cual está diseñado para realizar entre otras las siguientes tareas:

- Se conecta a ambas bases de datos.
- Escucha notificaciones.
- Ejecuta las sentencias SQL registradas.
- Maneja las transacciones.
- Maneja los hilos (por cada *maestro/esclavo*).
- Envía email en caso de conflictos.
- Registra archivos de logs para seguimiento.[42]

[42] https://code.google.com/p/pyreplica/

2.5.3.6 Streaming Replication (*Replicación en flujo*)

Dos de las características más importantes incluidas a partir de la versión 9.0 de PostgreSQL son Hot Standby (*HS*) y Streaming Replication (*SR*), estas dos características implementan en el núcleo de PostgreSQL lo necesario para instalar un sistema de replicación asincrónico maestro-esclavo, en el que los nodos esclavos se pueden utilizar para realizar consultas de solo lectura. Un sistema de replicación de estas características se podrá usar tanto para añadir redundancia a las bases de datos, como para liberar de trabajo al servidor principal en lo referente a consultas de solo lectura.

El Streaming Replication permite transferir asincrónicamente registros WAL sobre la marcha (record-based log shipping) entre un servidor maestro y uno o varios servidores esclavos. El Streaming Replication se configura mediante los parámetros *primary_conninfo* en el fichero *recovery.conf* y *max_wal_senders*, *wal_sender_delay* y *wal_keep_segments* en *postgresql.conf*.

En la práctica un proceso denominado *WAL receiver*, que es el receptor en el servidor esclavo, se conecta mediante una conexión TCP/IP al servidor maestro, en dicho servidor existe otro proceso denominado *WAL sender* que es el encargado de mandar los registros WAL sobre la marcha al servidor esclavo. Se puede observar cómo funciona el Streaming Replication en el siguiente gráfico:

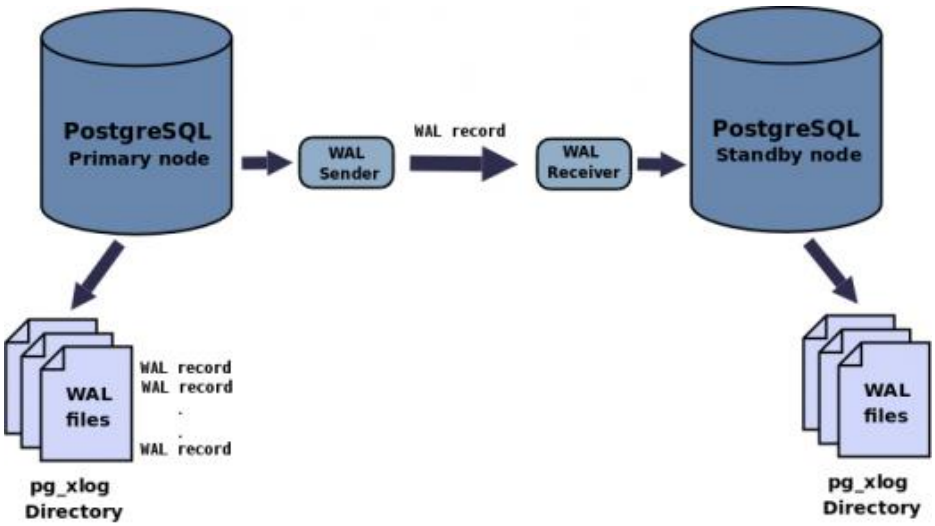

Imagen 16 - Material Capacitación SIU, página 39 – Agosto de 2009.

El Hot Standby permite acceder en modo de solo lectura a todos los datos disponibles en el servidor esclavo en donde se están replicando las bases de datos, se configura mediante los parámetros *hot_standby* y *max_standby_delay* en *postgresql.conf*. A continuación se muestra un gráfico de cómo funciona Hot Standby utilizando Streaming Replication y la transferencia de ficheros WAL:

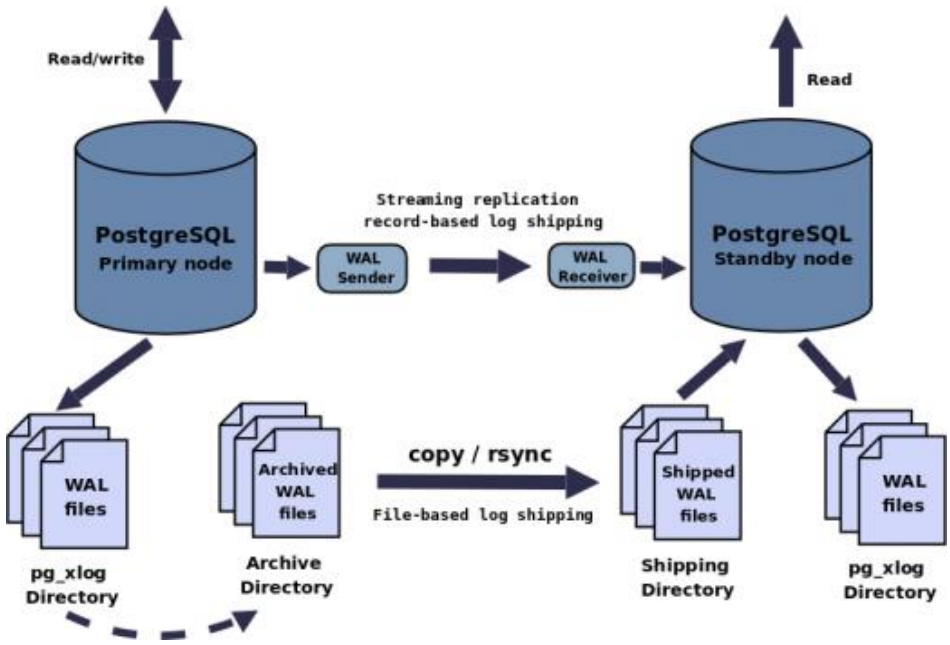

Imagen 17 - Material Capacitación SIU, página 39 – Agosto de 2009.

El tipo de replicación incluido en el núcleo de PostgreSQL está basado en la transferencia de registros WAL entre servidores. Esta transferencia se puede realizar registro a registro (***record-based log shipping***) o en ficheros WAL completos (***file-based log shipping***).

Si se usa solamente *SR* se tendrá que tener cuidado en que los ficheros WAL en el servidor maestro no sean reciclados antes de ser transferidos al servidor esclavo. Y si se transfieren solamente ficheros WAL sin utilizar *SR*, se perderán las últimas transacciones registradas en el servidor maestro en caso de caída del mismo. Es por esto que para tener un sistema más robusto, se suelen usar los dos métodos conjuntamente.

Una replicación basada en la transferencia de registros WAL significa que se replicaran absolutamente todas las bases de datos y cambios que realicemos en el servidor maestro. Si lo que

se necesita es replicar solamente algunas de las bases de datos existentes en el maestro o algunas tablas, se tendrá que usar otro tipo de replicación.

Para configurar un sistema *HS* usando *SR* y transferencia de registros WAL, habrá que realizar lo siguiente:

- Configurar el acceso automático mediante llaves SSH entre el servidor maestro y el esclavo.
- Configurar uno de los servidores como maestro.
- Activar el archivo de ficheros WAL en el servidor maestro.
- Activar la transferencia al servidor esclavo de ficheros WAL archivados en el maestro.
- Configurar el acceso necesario en el maestro para que el servidor esclavo pueda conectarse vía *SR.*
- Arrancar el servidor maestro.
- Realizar una copia de seguridad *'base'*, mediante el mismo procedimiento que se utiliza con *PITR.*
- Restaurar en el servidor esclavo la copia de seguridad *'base'* realizada en el maestro.
- Configurar el servidor esclavo en recuperación continua de registros WAL.
- Configurar el servidor esclavo como nodo Hot Standby.
- Crear un fichero recovery.conf en el esclavo.
- Activar el uso de *SR* en el esclavo.
- Activar el uso de ficheros WAL transferidos, en el proceso de restauración.
- Arrancar el servidor esclavo.

Todo este proceso es más fácil de lo que parece.[43]

[43] http://www.postgresql.org.es/node/483

Ventajas y Desventajas

Al utilizar Streaming Replication se pueden encontrar ciertas características las cuales pueden aportar tanto ventajas como desventajas, dentro de las características a favor se encuentran:

- Sencillo de implementar.
- Todo lo que se haga en el servidor principal, incluyendo sentencias DLL, se replica en el secundario.
- Puede usarse para aligerar la carga del servidor principal.

En cuanto a las desventajas que se obtienen al utilizar Streaming Replication, se pueden mencionar las siguientes:

- No es posible especificar que bases de datos o tablas se quieren replicar.
- No se pueden hacer cambios en el esquema en el servidor esclavo (por ejemplo, una indexación distinta).
- Las dos máquinas deben tener arquitecturas (32 o 64 bits) y versiones similares de PostgreSQL.

Autenticación

Es muy importante que los privilegios de acceso para la replicación se establezcan para que sólo los usuarios confiables puedan leer la secuencia WAL, ya que es fácil de extraer información confidencial de la misma. Los servidores Standby deben autenticarse con el servidor principal como superusuario o con una cuenta que tenga los privilegios para la replicación. Es recomendable crear una cuenta de usuario dedicada a la replicación y con privilegios de acceso para la replicación. Si bien al tener privilegios de replicación se otorgan permisos muy altos, estos no permiten al usuario modificar los datos en el servidor primario, lo que se hace con privilegios de usuario root. La autenticación del cliente para la replicación es controlada por un registro *pg_hba.conf* especificando la replicación en el campo de la base de datos. Por ejemplo si el servidor Standby se está ejecutando en el host con dirección IP 192.168.1.100 y el nombre de la cuenta para la replicación es *usdba*, el administrador puede añadir en el servidor principal la siguiente línea en el fichero *pg_hba.conf*:

 # Allow the user "usdba" from host 192.168.1.100 to connect to the primary

 # as a replication standby if the user's password is correctly supplied.

 # TYPE DATABASE USER ADDRESS METHOD

host replication usdba 192.168.1.100/32 md5

El nombre del host y el número del puerto del servidor principal, nombre de usuario de la conexión, y la contraseña se especifican en el archivo *recovery.conf*. La contraseña también se puede establecer en el servidor standby en el archivo ~ /.pgpass. Por ejemplo si el servidor principal se está ejecutando en el host con dirección IP 192.168.1.50, puerto 5432, el nombre de cuenta para la replicación es *usdba*, y la contraseña es *usdbapass*, el administrador puede añadir en el servidor standby la siguiente línea al archivo *recovery.conf*:

 # The standby connects to the primary that is running on host 192.168.1.50

 # and port 5432 as the user "usdba" whose password is "usdbapass".

primary_conninfo = 'host=192.168.1.50 port=5432 user=usdba password=usdbapass'

Monitoreo

Un indicador importante de la salud de la Replicación por Streaming es la cantidad de registros WAL generado en el servidor principal, pero que todavía no se aplica en el servidor de Standby. Se puede calcular este retraso comparando la posición actual de escritura WAL en el servidor principal con la última ubicación WAL recibida por el servidor Standby. Estos pueden ser recuperados utilizando *pg_current_xlog_location* en el servidor principal y *pg_last_xlog_receive_location* en el servidor Standby respectivamente.

Se puede obtener una lista de los procesos WAL emisores a través de *pg_stat_replication*. Las grandes diferencias entre *pg_current_xlog_location* y *sent_location* podrían indicar que el servidor maestro está sobrecargado, mientras que las diferencias entre *sent_location* y

*pg_last_xlog_receive_loc*ation en el servidor Standby podrían mostar un retraso de la red, o que el servidor Standby está sobrecargado.[44]

[44] http://www.postgresql.org/files/documentation/pdf/9.2/postgresql-9.2-US.pdf

2.5.4 Balanceo de carga

2.5.4.1 SQL Relay

Qué es SQL Relay?

SQL Relay es un poderoso middleware de acceso a bases de datos que consta de un pool de conexiones basado en servidores y API[45] de clientes para usarlo.

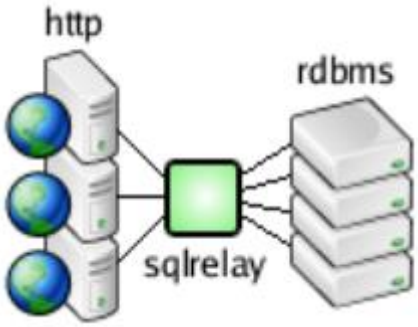

Imagen 18 – Esquema de SQL Relay - http://sqlrelay.sourceforge.net/documentation.html

Con SQL Relay, se puede acelerar y mejorar la escalabilidad de las aplicaciones con bases de datos basadas en la Web, bases de datos de acceso a las plataformas compatibles, distribución de acceso a bases de datos en clúster o replicadas, consultas de rutas y migración de aplicaciones de una base de datos a otra.

Características de SQL Relay

- ✓ ***Pooling de Conexión Base de datos Persistente***

  Un sistema de Pooling de conexión de bases de datos persistentes mantiene un grupo de conexiones que han iniciado sesión en la base de datos, lo que reduce el tiempo que tarda

[45] API (*Application Programming Interface*): Es el conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

la aplicación para conectarse a la base de datos. El inicio de sesión en algunas bases de datos puede tomar una cantidad notable de tiempo.

Imagen 19 – Conexión sin SQL Relay - http://sqlrelay.sourceforge.net/documentation.html

De hecho, una aplicación puede tardar más en iniciar sesión, que en ejecutar todas las consultas que necesita para funcionar. Esto puede ser especialmente problemático para aplicaciones transitorias como las aplicaciones basadas en la Web que se inician, conectan, consultan y cierran una y otra vez. Iniciar sesión en una pool de conexiones toma un lapso muy corto de tiempo y puesto que el sistema de pooling de conexión mantiene sesión iniciada en las conexiones de base de datos, las bases de datos están disponibles de manera inmediata.

Imagen 20 - Conexión con SQL Relay - http://sqlrelay.sourceforge.net/documentation.html

Un sistema de pooling de conexión puede reducir significativamente la cantidad de tiempo que tarda una aplicación en empezar a ejecutar las consultas. Muchos sistemas de pool de conexiones están integrados en servidores de aplicaciones o se ejecutan dentro de un servidor web, pero sólo pueden ser utilizados por aplicaciones que se ejecutan dentro de ese servidor y por lo general sólo por las aplicaciones escritas en un lenguaje en particular. SQL Relay funciona de manera autónoma y puede ser utilizado por cualquier aplicación, incluyendo aplicaciones que se ejecutan en otros equipos de la red. Esto es especialmente útil si se cuenta con una variedad de aplicaciones escritas en diferentes idiomas, que se ejecutan en distintas plataformas, y que es necesario que se comuniquen con la misma base de datos.

✓ ***Uso de proxy***

El acceso a bases de datos desde plataformas no compatibles como Oracle por ejemplo, no soporta FreeBSD, NetBSD y OpenBSD. Sin embargo, al ejecutar el software de servidor de *Relay SQL* en una plataforma compatible, como Linux y el software de cliente de SQL Relay en la plataforma BSD, la máquina BSD puede acceder a la base de datos Oracle.

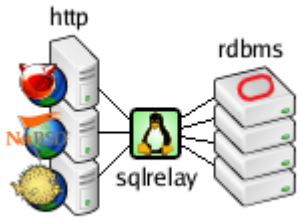

Imagen 21 - Utilizando SQL Relay - http://sqlrelay.sourceforge.net/documentation.html

De hecho, el software cliente de SQL Relay se puede construir y ejecutar en la mayoría de los sistemas Unix, independientes de la arquitectura. Por lo tanto, SQL Relay puede proporcionar acceso a bases de datos a partir de una amplia variedad de plataformas no soportada por el proveedor de base de datos.

✓ ***Limitación (Throttling)***

El Throttling limita el número de conexiones que se pueden iniciar con una determinada base de datos y peticiones de los clientes, en lugar de negar completamente el acceso. "Demasiadas conexiones", es un error común que es observado por los usuarios. Por ejemplo:

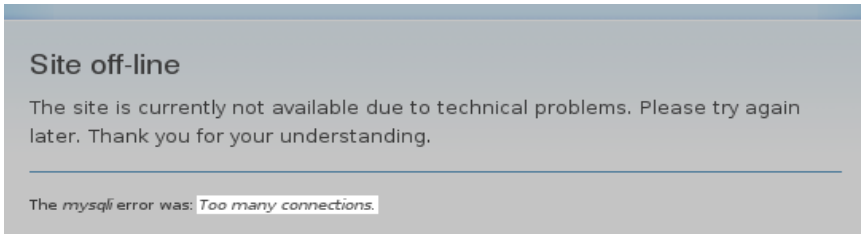

Imagen 22 – Error presente en situaciones de demasiadas conexiones - http://sqlrelay.sourceforge.net/documentation.html

SQL Relay puede ser configurado para mantener un cierto número de conexiones de bases de datos abiertas bajo demanda hasta cierto punto. Si la aplicación está tan ocupada que necesita más conexiones que las que tiene configurada, SQL Relay pondrá a los clientes en colas en lugar de devolver un error. Las conexiones en cola consumen relativamente pocos recursos y pueden ser configuradas para esperar por una cierta cantidad de tiempo antes de que se den por vencidas o de que esperen indefinidamente.

- ✓ ***Balanceo de Carga***

El balanceo de carga distribuye las consultas sobre un grupo de servidores. Por ejemplo, si se tiene un clúster de base de datos o un conjunto de servidores replicados, SQL Relay puede ser configurado para distribuir las consultas sobre ellos.

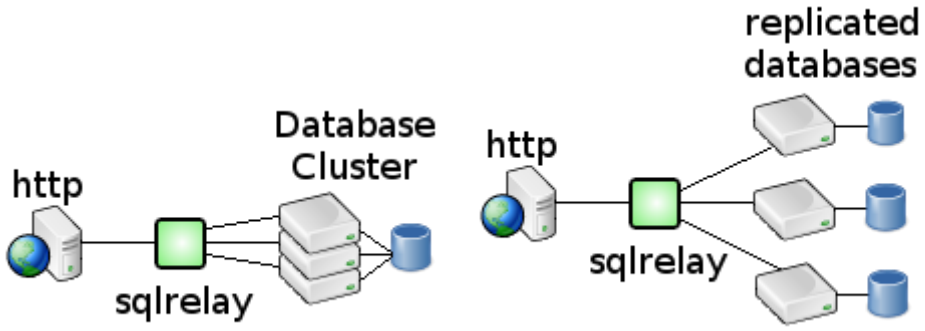

Imagen 23 – Opciones de Balanceo - http://sqlrelay.sourceforge.net/documentation.html

Si algunas de las máquinas de bases de datos son más potentes que otras, la carga se distribuye de manera desproporcionada y lejos de los servidores de bases de datos menos poderosos.

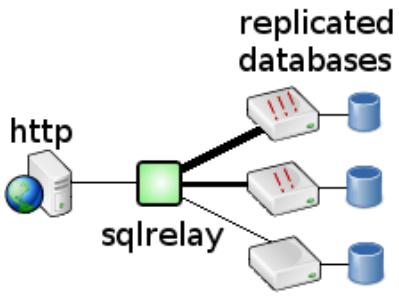

Imagen 24 – Balanceo de Carga - http://sqlrelay.sourceforge.net/documentation.html

Tener en cuenta que SQL Relay no se puede utilizar actualmente para replicar las bases de datos o para mantener bases de datos replicadas sincronizadas. Si se está utilizando SQL Relay para acceder a bases de datos replicadas, entonces se debe asumir que hay algún medio por el cual las bases de datos se mantienen sincronizados que es ajeno a SQL Relay.

- ✓ ***Consulta de enrutamiento y filtrado***

El enrutamiento de consultas envía las consultas que coinciden con un patrón a un conjunto de servidores de base de datos, y las consultas que coinciden con otro patrón las envía a otro conjunto de servidores. Un uso muy común para la consulta de enrutamiento es enrutar las consultas *DML* y *DDL* a un servidor maestro y distribuir las consultas *select* a través de un grupo de esclavos.

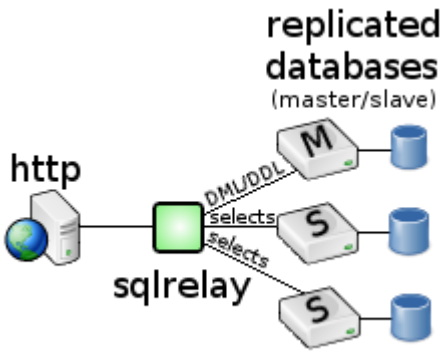

Imagen 25 – Enrutamiento y Filtrado - http://sqlrelay.sourceforge.net/documentation.html

El filtrado de consultas evita que se ejecuten consultas que coinciden con un cierto patrón. Un uso común para el filtrado de consultas es identificar aquellas consultas y evitar que se ejecuten, por ejemplo, *select * from unatabla donde id = 100* podría estar bien.

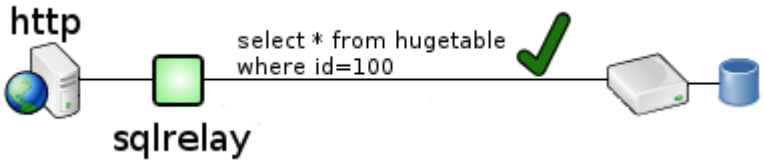

Imagen 26 – Filtrado de Consultas - http://sqlrelay.sourceforge.net/documentation.html

Pero *select * from unatabla* sin una cláusula *where* puede saturar a la base de datos.

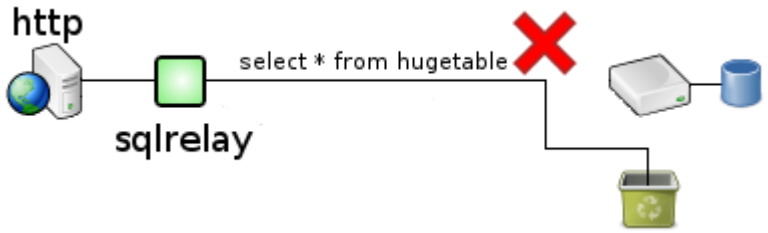

Imagen 27 - Filtrado de Consultas - http://sqlrelay.sourceforge.net/documentation.html

✓ ***Bases de datos soportadas***

SQL Relay soporta Oracle, MySQL, PostgreSQL, Sybase, IBM DB2, Firebird y bases de datos SQLite directamente, utilizando la API de cliente nativo para las bases de datos.

✓ ***Conectar a la Base de Datos utilizando una conexión local***

Uno de los errores más comunes que se cometen al usar SQL Relay es ejecutar SQL Relay en el mismo equipo que la base de datos, pero no utilizando una "conexión local" para la base de datos. Es un error fácil de cometer, pero cuando se repara se puede obtener una mejora sustancial en el rendimiento. La mayoría de las bases de datos soportan "*conexiones remotas*" y "*conexiones locales*". Si se está ejecutando SQL Relay en una máquina que tiene la base de datos en otro servidor, hay que configurarlo para conectarse a la base de datos mediante una conexión remota. Sin embargo, si se está ejecutando Relay SQL en el mismo equipo que la base de datos, se debe configurar para conectarse a la base de datos utilizando si es posible una conexión local.

✓ ***Cuántas conexiones se deben ejecutar***

Una buena regla a seguir es ejecutar tantas conexiones como la base de datos pueda manejar, la mejor manera de determinar cuántas conexiones ejecutar es hacer una suposición, dejar que se ejecuten las aplicaciones, monitorear el rendimiento del servidor de SQL Relay y de bases de datos, y en consecuencia ajustar el número de conexiones.

Se pueden utilizar programas de test de carga como *ApacheBench* o *LoadRunner* para realizar pruebas automatizadas, para ello se deben configurar secuencias de comandos y

páginas para simular el tipo de carga que las aplicaciones pondrán en SQL Relay y en la base de datos, además se puede obtener una idea exacta de cómo se ejecuta una aplicación de extremo a extremo. Es posible dejar que el programa de test de carga se ejecute, supervise el rendimiento del servidor SQL Relay y de bases de datos, y ajuste el número de conexiones en consecuencia.

También hay un programa en el directorio *test/stress* de la distribución SQL Relay llamado *querytest* que inicia sesión en el servidor SQL Relay, ejecuta una serie de consultas y cierra la sesión, una y otra vez, lo más rápido posible. Se pueden ejecutar muchas instancias de *querytest* simultáneamente para simular el tipo de carga que las aplicaciones pueden colocar en SQL Relay y en la base de datos. Es aceptable *querytest* como punto de partida, pero para hacer pruebas más serias se deben modificar las consultas que se ejecutan para poder simular con más precisión las aplicaciones.[46]

[46] http://sqlrelay.sourceforge.net/documentation.html

2.5.4.2 HAProxy

Qué es HAProxy?

HAProxy es una solución gratuita, muy rápida y confiable que ofrece alta disponibilidad, balanceo de carga y proxy para aplicaciones basadas en TCP y HTTP. Es especialmente adecuado para los sitios web que trabajan bajo cargas muy altas. El modo en que funciona hace que su integración en arquitecturas existentes sea muy fácil y sin riesgo, y no deja de ofrecer la posibilidad de no exponer a los servidores web frágiles a la red, como se observa a continuación:

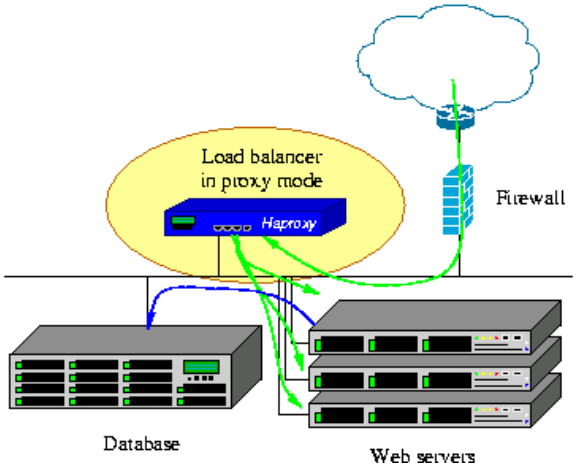

Imagen 28 – HAProxy - http://haproxy.1wt.eu/

Plataformas Soportadas

HAProxy es conocido por ejecutarse correctamente en los siguientes sistemas operativos o plataformas:

- Linux 2.4 en x86, x86_64, Alpha, SPARC, MIPS, PARISC.
- Linux 2.6 en x86, x86_64, ARM (ixp425), PPC64.
- Solaris 8/9 en UltraSPARC 2 y 3.
- Solaris 10 en Opteron y UltraSPARC.

- FreeBSD 4.10 - 8 en x86, amd64, Macppc, Alpha, Sparc64 y VAX.

El mayor rendimiento se puede lograr con las versiones superiores a 1.2.5 de HAProxy, se puede ejecutar en Linux 2.6 o con parches *epoll*[47] del kernel Linux 2.4. El sistema predeterminado de *polling* de la versión 1.1 es *select(),* que es común entre la mayoría de los sistemas operativos, pero puede llegar a ser lento cuando se trata de miles de descriptores de archivos.

En versiones de Linux más recientes, *HAProxy* puede utilizar *splice ()* para reenviar los datos entre las interfaces sin ninguna copia, el rendimiento por encima de 10 Gbps se puede lograr solamente de esa manera.

Desempeño

Diseñar algo confiable desde cero es más difícil en el corto plazo, pero a la larga es más fácil de mantener el código roto. En los programas de procesos individuales el fallo más pequeño provocará que un programa se detenga, entre en un bucle o se congele. No se han encontrado errores en el código ni en la producción durante los últimos 10 años.

HAProxy se ha instalado en los sistemas Linux que sirven a millones de páginas cada día, y que sólo se han reiniciado a los 3 años para una completa actualización del sistema operativo. Estos sistemas no estaban expuestos directamente a Internet. En estos sistemas, el software no puede fallar sin que se note de inmediato. Algunas personas confían tanto que lo utilizan como solución por defecto para resolver problemas sencillos.

HAProxy es realmente ideal, porque los indicadores que devuelve proporcionan una gran cantidad de información valiosa acerca de la salud de la aplicación, el comportamiento y defectos, que se utilizan para que sea aún más fiable. La mayor parte del trabajo es ejecutada por el sistema operativo, por esta razón una gran parte de la fiabilidad implica al sistema operativo en sí. El kernel de Linux necesita por lo menos una mejora todos los meses para corregir un fallo o vulnerabilidad,

---

[47] Epoll: Es un mecanismo de notificación escalable I/O para Linux. Está destinado a sustituir a las antiguas llamadas al sistema POSIX *select()* y *poll()*, para conseguir en aplicaciones más exigentes un mejor rendimiento, donde el número de descriptores de archivos es grande.

por lo que algunas personas prefieren ejecutarlo en Solaris. La fiabilidad puede disminuir significativamente cuando el sistema es empujado a sus límites de funcionamiento. Esta es una razón importante para ajustar finamente la interfaz *sysctls*[48]. Sin embargo, es importante asegurarse de que el sistema nunca se quedará sin memoria y que nunca va a cambiar. Un sistema bien afinado debe ser capaz de funcionar plenamente durante años sin que se caiga y sin reducir la velocidad.

Confiabilidad

HAProxy implica varias técnicas que se encuentran comúnmente en las arquitecturas de los sistemas operativos para lograr el máximo rendimiento:

- Un solo proceso, un modelo orientado a eventos reduce considerablemente el costo de cambio de contexto y el uso de la memoria. Es posible el procesamiento de cientos de tareas en un milisegundo, y el uso de la memoria es del orden de unos pocos kilobytes por sesión mientras que la memoria consumida en modelos como Apache está en el orden de un megabyte por proceso.

- Un solo búfer sin copia de datos entre lecturas y escrituras siempre que sea posible, ahorra una gran cantidad de ciclos de CPU y ancho de banda de memoria útil. Con frecuencia el cuello de botella estará entre los buses de *E/S* de la CPU y las interfaces de red. A 10 Gbps el ancho de banda de la memoria también puede convertirse en un cuello de botella.

- El gestor de memoria MRU, al utilizar conjuntos de memoria de tamaño fijo para la asignación de la memoria inmediata favorece regiones *hot cache*[49] sobre *regiones cold cache*[50], lo que reduce drásticamente el tiempo necesario para crear una nueva sesión.

---

[48] Sysctls: Es una interfaz para examinar y cambiar dinámicamente los parámetros en los sistemas operativos BSD y Linux.

[49] Hot cache: Su contenido está en una de las líneas del cache del CPU.

[50] Cold cache: Su contenido no está mapeado en el cache del CPU.

- La factorización del trabajo, tales como múltiples *accept()* a la vez, y la capacidad de limitar el número de *accept()* por iteración cuando se ejecuta en modo multi-proceso, de manera que la carga se distribuya uniformemente entre los procesos.

- En el análisis optimizado de cabeceras HTTP los encabezados se analizan interpretándolos sobre la marcha, y el análisis se optimiza para evitar una relectura de cualquier área de memoria previamente leída. Los puntos de control se utilizan cuando se llega a un extremo del buffer con una cabecera incompleta, por lo que el análisis no se inicia nuevamente desde el principio. Analizar una petición HTTP en promedio suele durar 2 microsegundos en un Pentium-M 1.7 GHz.

- La cuidadosa reducción del número de llamadas al sistema, la mayor parte del trabajo se realiza en el espacio del usuario por defecto, como el tiempo la lectura, la agregación de buffer, la activación/desactivación del descriptor de archivos.

Todas las micro optimizaciones resultan en un muy bajo uso de la CPU incluso en cargas moderadas. Con cargas muy altas cuando la CPU está saturada, es bastante común observar figuras como 5% de usuario y 95% de sistema, lo que significa que el proceso de HAProxy consume alrededor de 20 veces menos que su sistema homólogo, esto explica que la puesta a punto del sistema operativo es muy importante.

Esto también explica por qué el procesamiento de capa 7 tiene poco impacto en el rendimiento, incluso si se duplica el trabajo en el espacio del usuario, la distribución de la carga se parecerá más a un 10% del usuario y 90% de sistema, lo que significa una pérdida efectiva de sólo el 5% de la potencia de procesamiento. Esta es la razón del porque en sistemas de alto nivel, el rendimiento de HAProxy en capa 7 puede superar fácilmente los balanceadores de carga de hardware en el que el procesamiento complejo que no puede ser realizado por los ASIC[51], tiene que ser realizado por CPU lentos. El siguiente es el resultado de una comparación de rendimiento que se

[51] ASIC (*Circuito Integrado para Aplicaciones Específicas*): Es un circuito integrado hecho a la medida para un uso en particular, en vez de ser concebido para propósitos de uso general. Se usan para una función específica.

realiza sobre HAProxy 1.3.9 en EXOSEC (*http://www.exosec.fr/*) con un Pentium 4 de un solo núcleo y con interfaces PCI-Express:

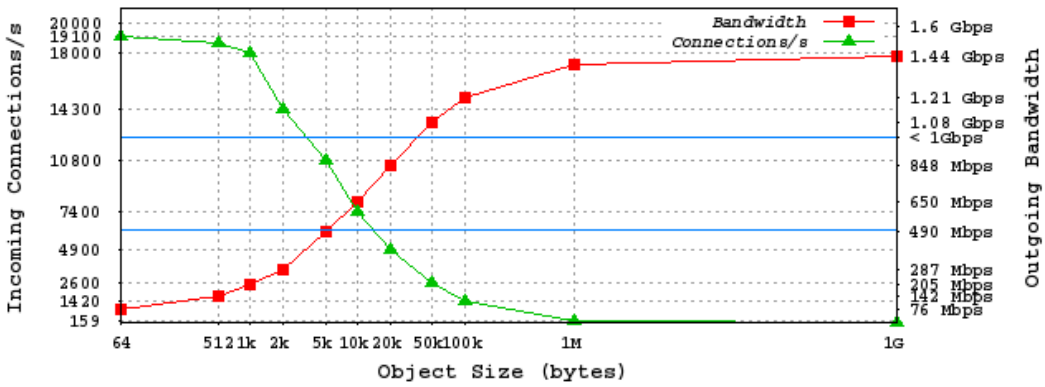

Imagen 29 – Rendimiento de HAProxy - http://haproxy.1wt.eu/

En resumen, una tasa de éxito por encima de 10.000/s se mantiene durante objetos de menos de 6 kB, y en Gigabit/s se mantiene durante objetos mayores que 40 kB.

Los balanceadores de carga de software usan almacenamiento en búfer TCP y no son sensibles a las solicitudes largas y a los tiempos de respuesta altos. Un efecto secundario de almacenamiento en búfer de HTTP es que aumenta la aceptación de la conexión del servidor mediante la reducción de la duración de la sesión, lo que deja espacio para nuevas solicitudes. Dependiendo del hardware, las tasas esperadas son del orden de unas pocas decenas de miles de nuevas conexiones por segundo con decenas de miles de conexiones simultáneas.

Hay 3 factores importantes que se utilizan para medir el rendimiento de un balanceador de carga:

- ***La tasa de sesión***: Este factor es muy importante, ya que determina directamente si el balanceador de carga no es capaz de distribuir todas las solicitudes que recibe. En su mayoría depende de la CPU. Este factor se mide con diferentes tamaños de objetos, los resultados más rápidos por lo general provienen de los objetos vacíos como por ejemplo códigos de respuesta HTTP 302, 304 o 404.

- ***La concurrencia de sesión***: Este factor está ligado al anterior. En general, la tasa de sesión se reducirá cuando el número de sesiones concurrentes aumenta (con excepción del mecanismo de sondeo epoll). Mientras más lento sean los servidores, mayor será el

número de sesiones simultáneas para un mismo tipo de sesión. Si un balanceador de carga recibe 10.000 sesiones por segundo y los servidores responden en 100 ms, el balanceador de carga tendrá 1.000 sesiones concurrentes. Este número está limitado por la cantidad de memoria y la cantidad de descriptores de archivos que el sistema puede manejar. Con un buffer de 8 kB, HAProxy necesitará cerca de 16 kB por sesión, que resulta en alrededor de 60.000 sesiones por GB de RAM. Los balanceadores de carga de capa 4 por lo general manejan millones de sesiones simultáneas dado que no procesan los datos, por lo que no necesitan ningún buffer. Por otra parte, a veces son diseñados para ser utilizados en el modo de servidor de retorno directo, en la que el balanceador de carga sólo ve el tráfico hacia adelante, lo cual obliga a mantener las sesiones por un largo tiempo luego de su fin para evitar el corte de sesiones antes de que se cierren.

- ***La tasa de datos***: Este factor es generalmente opuesto al de la tasa de sesión, se mide en Megabytes/s (*MB/s*) o a veces en Megabits/s (*Mbps*). Las tasas más altas de datos se consiguen con objetos de gran tamaño para minimizar la sobrecarga causada por el establecimiento de la sesión.

Por lo tanto cuando son conocidos los 3 límites, el cliente debe tener en cuenta de que generalmente esta por debajo de todos ellos. Una buena regla en balanceadores de carga de software es considerar un rendimiento promedio en la práctica de la mitad de la sesión máxima y de tasas de objetos de datos de tamaño medio.

Seguridad

La seguridad es una preocupación importante al implementar un software balanceador de carga. Es posible configurar el sistema operativo para limitar el número de puertos abiertos y servicios accesibles, pero queda expuesto el propio balanceador de carga.

La única vulnerabilidad encontrada hasta ahora data de principios del año 2002 y solamente se prolongó durante una semana, se introdujo cuando fueron reelaborados los logs. De todos modos, se tiene mucho cuidado al escribir el código para manipular cabeceras. Las combinaciones imposibles de estado son revisadas y devueltas, y los errores se procesan desde la creación hasta la finalización de una sesión. Algunas personas alrededor del mundo han contribuido con revisar el

código y han sugerido depuraciones y mayor claridad para facilitar la auditoría. Se rechazan los parches que introducen procesamiento sospechoso o en los que no se toma suficiente atención a las condiciones anormales.

Los registros proporcionan una gran cantidad de información para ayudar a mantener un nivel de seguridad satisfactorio. Sólo pueden ser enviados a través de paquetes UDP, porque una vez que se ha hecho chroot, el socket UNIX */dev/log* es inaccesible, y no debe ser posible escribir en un archivo. La siguiente información es especialmente útil:

- El IP de origen y el puerto del solicitante permiten encontrar su origen en los registros del firewall.
- La codificación adecuada de la solicitud garantiza que el solicitante no pueda ocultar caracteres no imprimibles, ni engañar a una terminal.
- La solicitud arbitraria, el encabezado de la respuesta y la captura de cookies ayudan a detectar los ataques de escaneo, servidores proxy y hosts infectados.
- Los temporizadores ayudan a diferenciar las solicitudes escritas a mano desde los navegadores.

HAProxy también proporciona control de encabezado basado en expresiones regulares. Partes de la solicitud, así como también la solicitud y respuesta de los encabezados pueden ser negados, permitidos, eliminados, reescritos o añadidos. Esto es comúnmente utilizado para bloquear las solicitudes o encodings (por ejemplo el exploit Apache Chunk) peligrosos, y para prevenir la fuga accidental de la información desde el servidor al cliente.[52]

[52] http://haproxy.1wt.eu/

2.5.5 Backups

Al igual que con todo lo que contiene datos importantes, las bases de datos Postgres se deben resguardar regularmente. Si bien el procedimiento suele ser sencillo, es importante tener una comprensión clara de las técnicas. Existen tres enfoques fundamentalmente diferentes para realizar copias de seguridad de datos en PostgreSQL:

- SQL Dump.
- File system level backup.
- Continuous archiving.

Cada uno tiene sus propias fortalezas y debilidades, estos enfoques se analizan a su vez en las siguientes secciones.

2.5.5.1 SQL Dump

Lo que se busca con este método de volcado es generar un archivo de texto con los comandos SQL, que cuando es suministrado nuevamente al servidor se vuelve a crear la base de datos en el mismo estado en que se encontraba en el momento del volcado. Postgres proporciona para este propósito el programa *pg_dump*. El uso básico de este comando es el siguiente: *pg_dump dbname > outfile.*

Como puede observarse, *pg_dump* escribe su resultado en la salida estándar, *pg_dump* es una aplicación cliente de Postgres, esto significa que se puede realizar el procedimiento de backup de cualquier servidor remoto que tenga acceso a la base de datos. Se debe tener en cuenta que *pg_dump* no funciona con permisos especiales, y que además se debe tener acceso de lectura a todas las tablas a las que se les desee hacer una copia de seguridad.

Para especificar con qué servidor de base de datos debe ponerse en contacto *pg_dump*, se debe utilizar en la línea de comandos las opciones ***-h*** host y ***-p*** puerto, el host predeterminado es el localhost o lo que especifique la variable de entorno PGHOST. Del mismo modo, el puerto por defecto es el indicado por la variable de entorno PGPORT o, en todo caso, el precompilado por defecto.

Como cualquier otra aplicación cliente de PostgreSQL, *pg_dump* se conecta de forma predeterminada a la base de datos con el nombre de usuario que es igual al del sistema operativo actual, para corregir esto se debe especificar la opción **-U** o establecer la variable de entorno PGUSER.

Pg_dump es también el único método que funciona para transferir una base de datos a una máquina con una arquitectura diferente, como por ejemplo transferir de un servidor de 32 bits a un servidor de 64 bits.

Los volcados creados por *pg_dump* son internamente consistentes, estos representan una instantánea de la base de datos en el momento en que *pg_dump* comienza a correr, *pg_dump* no bloquea otras operaciones en la base de datos durante el trabajo, como excepciones se encuentran las operaciones que necesitan operar con un bloqueo exclusivo, como la mayoría de las formas de ALTER TABLE.

- ***Restaurando el Dump***: Los archivos de texto creados por *pg_dump* están destinados a ser leídos con el programa psql. El formato general del comando para restaurar un volcado es *psql dbuno < infile,* donde *infile* es el archivo de salida del comando *pg_dump*. La base de datos *dbuno* no se crea por este comando, por lo que se debe crear antes de ejecutar *psql,* por ejemplo con *createdb -T template0 dbuno*. El comando *psql* admite opciones similares a *pg_dump* para conectarse al servidor de base de datos y para especificar su nombre de usuario. De forma predeterminada, la secuencia de comandos *psql* se sigue ejecutando después de que se ha detectado un error de SQL, es posible ejecutar *psql* con la variable *ON_ERROR_STOP* establecida para modificar ese comportamiento, y de esta manera obtener una salida *psql* con un estado de salida de 3 si se produce un error SQL: *psql -- set ON_ERROR_STOP dbuno = on < infile.*

- ***Usando pg_dumpall***: pg_dump vuelca una sola base de datos a la vez, y no envía información acerca de los *Roles* o *Tablespaces*, porque son de todo el clúster en lugar una base de datos. Para soportar el volcado de todo el contenido de un clúster de base de datos, se proporciona el programa *pg_dumpall*, este programa respalda cada base de datos en un clúster determinado, y también conserva los datos de todo el clúster, como las definiciones de *Roles* y de *Tablespaces*. El uso básico de este

comando es el siguiente *pg_dumpall > outfile*. El volcado resultante puede ser restaurado con *psql -f infile postgres*. Siempre es necesario tener acceso a la base de datos como superusuario al restaurar un volcado de *pg_dumpall*, ya que es necesario para restaurar la información de *Roles* y de *Tablespaces*.

- ***Manejando grandes Bases de Datos:*** Algunos sistemas operativos tienen límites del tamaño máximo de archivo que causan problemas al crear grandes archivos de salida de *pg_dump*. Afortunadamente, *pg_dump* puede escribir en la salida estándar, por lo que puede utilizar las herramientas estándar de Unix para evitar este posible problema. Hay varios métodos posibles:

  - *Usando dumps comprimidos*: Se puede utilizar un programa de compresión como por ejemplo gzip: *pg_dump dbprueba | gzip > filename.gz*. Recargar con *gunzip -c fileprueba.gz | psql dbprueba o cat fileprueba.gz | gunzip | psql dbprueba.*

  - *Usando división* (***split***): El comando ***split*** permite dividir la salida en archivos más pequeños que son de tamaño aceptables en el sistema de archivos. Por ejemplo, para hacer fragmentos de 1 megabyte: *pg_dump dbprueba | split -b 1m – fileprueba*. Actualizar con: *cat fileprueba * | psql dbprueba.*

  - *Utilizar el formato de volcado de **pg_dump***: El formato de volcado por defecto comprime datos a medida que se escribe en el archivo de salida, esto produce volcados de archivos con un tamaño similar al del comando *gzip*, tiene la ventaja de que las tablas se pueden restaurar selectivamente. El siguiente comando vuelca una base de datos utilizando el formato de volcado por defecto: *pg_dump -Fc dbprueba > fileprueba*. Un volcado con formato por defecto debe ser restaurado con *pg_restore*, como por ejemplo: *pg_restore -d dbprueba fileprueba.*

2.5.5.2 File System Level Backup

Una alternativa de copia de seguridad es copiar directamente los archivos que PostgreSQL usa para almacenar los datos en la base de datos. Se puede utilizar cualquier método que se prefiera para hacer copias de seguridad del sistema de archivos, por ejemplo: *tar -cf backup.tar /usr/local/pgsql/data.*

Existen dos restricciones que hacen que este método no sea práctico, o al menos inferior al del método de *pg_dump*:

1. El servidor de base de datos debe ser apagado con el fin de obtener una copia de seguridad utilizable. Las medidas a medias, como no permitir todas las conexiones no funcionan (en parte porque las herramientas de ***tar*** y similares no toman una instantánea atómica del estado del sistema de archivos). También es necesario apagar el servidor antes de restaurar los datos.

2. Si se ha profundizado en los detalles de la estructura de archivos de la base de datos, es posible tentarse a realizar una copia de seguridad o restaurar sólo algunas tablas individuales o bases de datos de sus respectivos archivos o directorios. Esto no va a funcionar porque la información contenida en estos archivos no se puede utilizar sin los archivos de log de *commit*, *pg_clog / **, que contiene el estado del *commit* de todas las transacciones, un archivo de la tabla sólo se puede usar con esta información. También es posible restaurar sólo una tabla y los datos asociados a *pg_clog*, eso sería hacer a todas las otras tablas en el clúster de base de datos inútiles. Las copias de seguridad del sistema de archivos sólo funcionan para una copia de seguridad completa y una restauración de un clúster de base de datos entero.

Un enfoque alternativo de copia de seguridad del sistema de archivos es hacer una "copia consistente" del directorio de datos, si el sistema de archivos admite esa funcionalidad.

El procedimiento típico es hacer una "*instantánea*" del volumen que contiene la base de datos y, luego copiar todo el directorio de datos de la instantánea en un dispositivo de backup, a continuación liberar la instantánea. Esto funciona incluso cuando el servidor de base de datos está en ejecución. Sin embargo, una copia de seguridad creada de esta manera guarda los archivos de base de datos en un estado como si el servidor de base de datos no se apagara correctamente, por lo

tanto, cuando se inicia el servidor de base de datos, sobre los datos copiados se pensará que la instancia del servidor anterior se estropeo y se volverá a reproducir el registro WAL. Esto no es un problema, se debe asegurar de incluir los archivos WAL en la copia de seguridad. Se puede realizar un control antes de tomar la instantánea para reducir el tiempo de recuperación.

Si la base de datos se distribuye a través de varios sistemas de archivos, puede que no exista ninguna forma de obtener instantáneas exactamente simultáneas de todos los volúmenes. Por ejemplo, si los archivos de datos y registro de WAL están en discos diferentes, o si los espacios de tablas están en diferentes sistemas de archivos, puede que no sea posible utilizar copias de seguridad de instantáneas porque las mismas deben ser simultáneas.

Si las instantáneas simultáneas no son posibles, una opción es apagar el servidor de base de datos el tiempo suficiente para poder establecer todas las instantáneas congeladas. Otra opción es realizarle a la base un backup tipo "*continuous archiving"* (Sección siguiente), ya que dichas copias de seguridad son inmunes a presentar cambios en el sistema durante la copia de seguridad. Para ello es necesario habilitar el "*continuous archiving"* solo durante el proceso de backup, la restauración se realiza mediante la recuperación de archivo continuo.

Otra opción es usar *rsync* para realizar una copia de seguridad del sistema de archivos, esto se hace primero ejecutando *rsync* mientras el servidor de bases de datos está en marcha, luego se debe apagar el servidor de base de datos el tiempo suficiente para hacer un segundo *rsync*. El segundo *rsync* será mucho más rápido que el primero, ya que tiene relativamente pocos datos para transferir, y el resultado final será consistente porque el servidor se encuentra apagado. Este método permite una copia de seguridad del sistema de archivos que se realiza con un mínimo tiempo de inactividad.

Tener en cuenta que una copia de seguridad del sistema de archivos es típicamente más grande que un volcado SQL, sin embargo hacer una copia de seguridad del sistema de archivos puede ser más rápido.

2.5.5.3 Continuous archiving and Point in Time Recovery (PITR)

En todo momento, PostgreSQL mantiene una escritura de log llamada WAL en el subdirectorio *pg_xlog/* del directorio de datos del clúster, el log guarda cada cambio realizado en los archivos de datos de la base de datos. Este log existe principalmente para propósitos de seguridad, si el sistema se bloquea, la base de datos puede ser restaurada mediante la reproducción de las entradas del log realizado desde el último punto de control.

La existencia del log hace que sea posible el uso de una tercera estrategia de copias de seguridad de bases de datos, se puede combinar un backup a nivel de sistema de archivos (*File System Level*) con un backup de los archivos WAL. Si es necesaria una recuperación, se debe restaurar el backup del sistema de archivos y luego reproducirlo desde los archivos de respaldo de la WAL para poder llevar el sistema a un estado actual. Este enfoque es más complejo de administrar que cualquiera de los enfoques anteriores, pero tiene algunas ventajas importantes:

- ✓ No es necesario un backup del sistema de archivos relativamente consistente como punto de partida, cualquier inconsistencia interna en el backup se puede corregir mediante la reproducción del log. Por lo tanto no es necesario un sistema de archivos con capacidad de instantánea, sólo basta con una herramienta *tar* o una de archivado similar.

- ✓ Dado que es posible combinar una secuencia larga de archivos WAL indefinidamente para su reproducción, una copia de seguridad continua puede lograrse simplemente continuando con el almacenamiento de los archivos WAL. Esto es particularmente útil para grandes bases de datos, pero podría no ser conveniente realizar con frecuencia un backup completo.

- ✓ No es necesario reproducir las entradas WAL hasta el final, se puede detener la reproducción en cualquier momento y tener una visión consistente de la base de datos como era en ese momento. Por lo tanto, esta técnica es compatible con PITR: es posible restaurar la base de datos a su estado en cualquier momento desde que fue realizado el backup de la base de datos.

- ✓ Si continuamente se provee de la serie de archivos WAL a otro equipo que haya sido cargado con el mismo archivo de backup de la base, se tendrá un sistema de espera activa, es decir que en cualquier momento se puede abrir la segunda máquina y se va a tener una copia casi actual de la base de datos.

Al igual que con la técnica de backup del sistema de archivos, este método sólo puede soportar la restauración de un clúster de base de datos completa, no un subgrupo. Asimismo, se requiere una gran cantidad de almacenamiento de archivos dado que el backup de la base puede ser voluminoso y un sistema ocupado generará muchos megabytes de tráfico WAL que tienen que ser archivados. Sin embargo, es la técnica preferida de copia de seguridad en muchas situaciones donde es necesaria una alta fiabilidad.

Para lograr una recuperación exitosa utilizando *continuous archiving* (también llamado "*online backup*"), se necesita una secuencia continua de archivos WAL almacenados que se retrotrae al menos hasta la hora de inicio de la copia de seguridad. Así que para comenzar se deben establecer y probar los procedimientos para almacenar archivos WAL antes de tomar el primer backup de la base. Como consecuencia, a continuación se describen los mecanismos de archivado de registros WAL.

- ***Configuración de Archivado de WAL***

  En cierto sentido cuando un sistema de PostgreSQL se ejecuta produce indefinidamente una larga secuencia de registros WAL. El sistema divide físicamente esta secuencia en archivos de segmentos WAL, que son normalmente de un tamaño de 16 MB cada uno. Los archivos del segmento reciben nombres numéricos que reflejan su posición en la secuencia de registros WAL. Cuando no se utiliza el archivado del registro WAL, normalmente el sistema crea algunos archivos del segmento y luego los recicla renombrando los archivos del segmento necesarios, se supone que dichos archivos cuyos contenidos preceden al penúltimo punto de control no son de interés y pueden ser reciclados. Al archivar datos WAL, se debe capturar el contenido de cada archivo de segmento una vez que se encuentre completo, y guardar los datos en algún lugar antes de que el archivo de segmentos sea reciclado para su reutilización. Dependiendo de la aplicación y del hardware disponible, podrían haber distintas maneras de guardar los datos en alguna parte, como por ejemplo se podrían copiar los archivos del segmento a un directorio de archivos de red (*NFS - Network File System*) montado en otra máquina, o juntarlos por lotes y grabarlos en DVD. PostgreSQL permite al administrador especificar un comando que se ejecute para copiar un archivo del segmento completo a donde tenga que ir, el comando puede ser tan simple como un *cp*, o se puede invocar una secuencia de comandos más compleja.

Para habilitar el archivado de registros WAL, se debe establecer el parámetro de configuración para archivar *wal_level* (o *hot_standby*), *archive_mode* en *on*, y especificar el comando shell a utilizar en el parámetro de configuración *archive_command*. En la práctica siempre se hacen estos ajustes en el archivo *postgresql.conf*. En *archive_command*, *%p* se sustituye por el nombre de la ruta del archivo, y *%f* se reemplaza por el nombre del archivo.

Después de que los parámetros *%f* y *%p* se han reemplazados, el comando real que es ejecutado podría tener este aspecto:

test ! -f /mnt/server/archivedir/00000001000000A900000065 && cp pg_xlog/00000001000000A900000065 /mnt/server/archivedir/00000001000000A900000065

Un comando similar se genera por cada nuevo archivo que va a ser archivado. Es importante que el archivo de comando retorne un estado de salida cero sólo si tiene éxito, al conseguir un estado de salida cero, PostgreSQL asume que el archivo ha sido guardado correctamente y luego lo podrá eliminar o reciclar. Sin embargo, un estado de salida distinto de cero indica que el archivo no ha sido almacenado, sino que lo seguirá intentando periódicamente hasta que lo consiga. El comando de archivo generalmente debe estar diseñado para no sobrescribir cualquier archivo de almacenamiento pre-existente, es una importante medida de seguridad para preservar la integridad de un archivo en caso de un error del administrador, es recomendable analizar el comando de archivo para garantizar que no sobrescriba a un archivo existente, y que devuelva el estado distinto de cero en este caso. Mientras se diseña la configuración del archivado, se debe tener en cuenta lo que ocurrirá si el archivo de comando falla repetidamente debido a cierto aspecto y que eso requiera la intervención del operador o que el archivo se queda sin espacio. Por ejemplo, esto podría ocurrir si se escribe en una cinta sin un cambiador automático, cuando la cinta se llena, no se puede archivar nada más hasta que se cambie la cinta. Cualquier condición de error debe ser comunicada a un operador de modo que la situación se puede resolver razonablemente rápido.

El directorio pg_xlog/ se seguirá llenando con los archivos WAL hasta que se resuelva la situación. Si el sistema de archivos que contiene pg_xlog/ se llena, PostgreSQL hará una parada de pánico. Las transacciones no confirmadas se pierden y la base de datos permanecerá desconectada hasta que se libere algo de espacio. La velocidad del comando

de archivado no es importante, siempre y cuando pueda continuar con la tasa media con la que el servidor genera los datos WAL. El funcionamiento normal continúa incluso si el proceso de archivado sufre un retraso. Si el proceso de archivado se retrasa significativamente, esto aumentará la cantidad de datos que se pueden perder en el caso de un desastre, esto también significa que el directorio *pg_xlog/* contendrá un gran número de segmentos de archivos que aún no han sido archivados, lo que eventualmente podría exceder el espacio disponible en disco. Se recomienda supervisar el proceso de archivado para asegurarse de que está funcionando correctamente.

Al escribir el comando de archivo, se debe asumir que los nombres de archivo del proceso de archivado pueden ser de hasta 64 caracteres de longitud y pueden contener cualquier combinación de letras ASCII, dígitos y puntos. No es necesario preservar la ruta relativa original (*% p*), pero es necesario preservar el nombre del archivo (*% f*). Tener en cuenta que aunque el proceso de archivado de WAL permita restaurar las modificaciones realizadas en los registros de la base de datos PostgreSQL, esto no restaurará los cambios hechos sobre los archivos de configuración como por ejemplo *postgresql.conf*, *pg_hba.conf* y *pg_ident.conf*, ya que son editados manualmente por medio de las operaciones de SQL. Es posible resguardar los archivos de configuración utilizando procedimientos normales de copia de seguridad del sistema de archivos.

El comando de archivo sólo se invoca en segmentos WAL completados, por lo tanto si el servidor genera poco tráfico WAL, o tiene períodos de inactividad en donde genera poco tráfico, podría haber un retraso entre la realización de una transacción y el almacenamiento seguro de archivos. Para poner un límite en cuanto a la antigüedad de los archivos de datos que pueden ser archivados, se puede establecer el parámetro *archive_timeout* para forzar al servidor cambiar a un nuevo segmento de archivo WAL más a menudo. Por lo tanto, es aconsejable establecer un tiempo muy corto en *archive_timeout*, la configuración de *archive_timeout* en un minuto más o menos suele ser un tiempo razonable.

Además, se puede forzar un cambio de segmento manualmente con *pg_switch_xlog* si nos queremos asegurar de que cuando una transacción finalice sea archivada tan pronto como sea posible.

Cuando el parámetro de *wal_level* es mínimo, algunos comandos SQL se optimizan para evitar el registro de archivos WAL, si el archivado o la replicación por *Streaming* se activan durante la ejecución de uno de estos estados, los registros WAL no contienen información suficiente para la recuperación de archivos. Por esta razón, *wal_level* sólo se puede cambiar en el arranque del servidor, sin embargo, *archive_command* se puede cambiar al recargar un archivo de configuración. Si se desea detener temporalmente el archivado, una forma de hacerlo es establecer en *archive_command* la cadena vacía ('). Esto hará que los archivos WAL se acumulen en *pg_xlog/* hasta que una orden de *archive_command* los reestablezca.

- ***Hacer un Backup de la Base***

  La manera más fácil realizar un backup de la base es utilizar la herramienta *pg_basebackup*. Se puede crear un backup de la base, ya sea como un archivo normal o como un archivo *tar*, si se requiere más flexibilidad que *pg_basebackup* se puede hacer una copia de seguridad de la base utilizando la API de bajo nivel. No es necesario preocuparse por la cantidad de tiempo que se necesita para hacer una backup de la base. Para hacer el backup se tendrán que guardar todos los archivos de segmentos WAL generados durante y después del backup del sistema de archivos, para facilitar esta tarea el proceso de backup de la base crea un archivo del historial del backup que se almacena inmediatamente en el área de archivos de la WAL.

  Este archivo es llamado por el primer segmento de archivos WAL que se necesita para el backup del sistema de archivos, por ejemplo, si el archivo WAL inicial es *000000100001234000055CD* el archivo histórico de copia de seguridad se llamará por ejemplo como *000000010000123400055CD.007C9330.backup,* la segunda parte del nombre del archivo es sinónimo de una posición exacta dentro del archivo WAL, y normalmente puede ser ignorada. Una vez que se ha archivado de forma segura el backup del sistema de archivos y los archivos de segmentos de la WAL utilizados durante el backup, todos los segmentos de la WAL archivados con nombres numéricamente inferiores ya no son necesarios para recuperar el backup del sistema de archivos y pueden eliminarse. No obstante, se debe considerar mantener varios conjuntos de backup que brinden absoluta seguridad de que se pueden recuperar los datos. El archivo histórico de backup es sólo un pequeño archivo de texto, contiene la cadena de texto que se le dio a

pg_basebackup, así como las horas de inicio y finalización y segmentos WAL del backup. Si se ha utilizado la etiqueta para identificar el archivo del *dump* asociado, el archivo histórico de archivado es suficiente para decidir que archivo dump se desea restaurar. También se debe considerar el tiempo que se está dispuesto a gastar en la recuperación, si la recuperación es necesaria, el sistema tendrá que reproducir todos los segmentos WAL, y eso podría tomar un tiempo considerable si ha pasado mucho tiempo desde el último backup de la base.

- ***Recuperación utilizando un Backup de Archivo Continuo* (*Continuous Archive*)**

  El siguiente es el procedimiento de lo que se necesita para recuperar una copia de seguridad:

  1. Detener el servidor, si se está ejecutando.
  2. Si se dispone de espacio, copiar todo el directorio de datos del clúster y los *tablespaces* en una ubicación temporal en caso de que se necesiten más tarde, si no hay suficiente espacio, al menos se debe guardar el contenido de *pg_xlog* del subdirectorio del clúster, ya que podría contener registros que no fueron archivados antes de que el sistema se corrompiera.
  3. Eliminar todos los archivos y subdirectorios existentes en el directorio de datos del clúster y en los directorios raíz de los *tablespaces* que se utilizan.
  4. Restaurar los archivos de la base de datos desde la copia de seguridad del sistema de archivos, asegurarse de que se restauran con permisos de propietario, específicamente con los del usuario del sistema de base de datos, no con los del usuario *root* y con los permisos adecuados. Si se utilizan *tablespaces* se deben verificar que los enlaces simbólicos en *pg_tblspc/* se restauraron correctamente.
  5. Eliminar todos los archivos presentes en *pg_xlog/,* los cuales provienen del backup del sistema de archivos y por lo tanto son probablemente obsoletos.
  6. Si se tiene que desarchivar los archivos de segmentos WAL que se guardaron en el paso 2, copiarlos en *pg_xlog /,* lo mejor es copiarlos, no moverlos, por lo que aún

tienen los archivos no modificados si se produce un problema y hay que empezar de nuevo.

7. Crear un comando de recuperación de archivo *recovery.conf* en el directorio de datos del clúster. También puede ser que se desee modificar temporalmente *pg_hba.conf* para evitar que se conecten usuarios comunes hasta que la recuperación se haya realizado correctamente.

8. Iniciar el servidor, éste se pondrá en modo de recuperación y luego leerá los archivos WAL archivados que necesita. Si la recuperación se terminó debido a un error externo, el servidor simplemente se puede reiniciar y seguirá con la recuperación. Una vez finalizado el proceso de recuperación, el servidor cambiará el nombre de *recovery.conf* a *recovery.done* para evitar entrar accidentalmente en el modo de recuperación y luego comenzará con las operaciones normales de la base de datos.

9. Examinar el contenido de la base de datos para asegurarse de que se haya recuperado al estado deseado, si no es así, volver al paso 1. Si todo está bien, permitir que los usuarios se conecten mediante la restauración en *pg_hba.conf*.

La clave de todo esto es crear un archivo de configuración de recuperación que describa cómo se desea recuperar y en qué medida debe ejecutarse la recuperación. Se puede utilizar *recovery.conf.sample,* normalmente ubicado como un prototipo en el directorio de instalación *share/*. Lo único que se necesita especificar en *recovery.conf* es el parámetro *restore_command*, que le dice a PostgreSQL cómo recuperar segmentos de archivos WAL archivados. Al igual que en *archive_command*, puede contener los caracteres *% f* que se reemplazan por el nombre del archivo de log deseado y los caracteres *p%*, que se reemplazan por el nombre de la ruta para copiar el archivo del log. El nombre de la ruta es relativa al directorio de trabajo actual, es decir al directorio de datos del clúster. Se debe escribir % si se necesita insertar un carácter % real en el comando. El comando más utilizado es similar a:

restore_command = 'cp /mnt/server/archivedir/%f %p'

El cual copiará segmentos de archivos WAL previamente almacenados desde el directorio /*mnt /server /archivedir*. Es importante que el comando devuelva el código de

salida distinto de cero en caso de fallo, el comando va a ser llamado solicitando archivos que no están presentes en el segmento de archivos WAL, debiendo retornar un valor diferente de cero cuando fuera solicitado, esto no es una condición de error. No todos los archivos solicitados serán los segmentos de archivos WAL, también se deben esperar peticiones de archivos con un sufijo de *.backup* o *.history*. Además tener en cuenta que el nombre del path *p%* de la base será diferente de *%f*, no se debe esperar que sean intercambiables. Los segmentos WAL que no se pueden encontrar en el archivo WAL, se buscarán en *pg_xlog/*, esto permite el uso de segmentos recientes no archivados. Sin embargo, los segmentos que están disponibles en el archivo serán utilizados en preferencia a los archivos en *pg_xlog/*.

Normalmente, la recuperación va a continuar con todos los segmentos WAL disponibles, restaurando así la base de datos a un punto actualizado en el tiempo. Por lo tanto, una recuperación normal finalizará con un mensaje de "*archivo no encontrado*", el texto exacto del mensaje de error depende de su elección en *restore_command*. También puede aparecer un mensaje de error como *00000001.history* al comienzo de la recuperación de un archivo, esto también es normal y no indica un problema en situaciones simples de recuperación. Si se desea recuperar a un punto anterior en el tiempo, solo basta con especificar el punto de detención deseado en el archivo *recovery.conf*. Se puede especificar el punto de detención, conocido como el "*objetivo de recuperación*", ya sea por ejemplo por *fecha/hora* o por el nombre de un punto de restauración.

Los puntos de parada deben ser luego de la hora de finalización del backup de la base de datos, es decir la hora de finalización de *pg_stop_backup*. No se puede utilizar un backup de la base de datos para recuperar a un tiempo en que el backup se estaba realizando. Para recuperar a un cierto momento, se debe regresar a la copia de seguridad de la base de datos anterior y avanzar desde allí.

Si la recuperación encuentra dañados los datos de la WAL se detendrá en ese punto y el servidor no se iniciará, en tal caso el proceso de recuperación se puede volver a ejecutar desde el principio especificando un "*objetivo de recuperación*" antes del punto de corrupción para que la recuperación se pueda completar normalmente. Si la recuperación falla por una razón externa, tal como un fallo del sistema o si el archivo de la WAL se encuentra inaccesible, entonces se puede volver a iniciar la recuperación y ésta se

reiniciará prácticamente desde el punto donde ha ocurrido la falla. El reinicio de la recuperación funciona como si fueran puntos de control durante el funcionamiento normal, el servidor fuerza periódicamente todo el estado en el disco, y luego actualiza el archivo *pg_control* para indicar que los datos que ya han sido procesados de la WAL no tienen que ser escaneados nuevamente.[53]

[53] http://www.postgresql.org/files/documentation/pdf/9.2/postgresql-9.2-US.pdf

2.6 RESULTADOS ESPERADOS

En la actualidad en el Centro Informático y Tecnológico de la UNCuyo los distintos sistemas utilizan bases de datos Postgres. Para el alojamiento de los distintos sistemas y sus respectivas bases de datos se cuenta con dos servidores "IBM BladeCenter H Chassis" que poseen las siguientes especificaciones: dos microprocesadores Intel Cuad Core, 14 bahías blade, 32 Gb de memoria RAM por bahía, dos "IBM System Storage Array DS3515" con doce discos Serial Attached SCSI (SAS) que suman una capacidad total de 13 Tb. Para la refrigeración de la sala donde se alojan los equipos se disponen de dos equipos de aire acondicionado con alimentación trifásica de 9000 frigorías cada uno.

La alimentación eléctrica de los IBM BladeCenter se realiza mediante dos UPS de 10 Kva cada una, además se cuenta con un grupo electrógeno trifásico con motor diésel que brinda una capacidad de 53 Kva para respaldo en caso de cortes de luz prolongados.

En base a la tecnología disponible y luego de llevar a cabo el proceso de investigación, el cual abarcó una búsqueda, y un profundo análisis de la información, se puede concluir que para satisfacer los objetivos que implican tener confiabilidad, recuperación de fallos y optimización de bases de datos con Postgres en servidores con sistemas operativos Linux utilizando sistemas de replicación se opta por la siguiente configuración:

- El sistema de replicación asincrónica maestro-esclavo elegido, en el que los nodos esclavos se pueden utilizar para realizar consultas de solo lectura es "***Streaming Replication***". Se selecciona el método de "Streaming Replication" por sobre los otros métodos por lo siguiente:

    1) Se busca principalmente la replicación de la totalidad de las bases de datos.

    2) Se dispone de una arquitectura idéntica de software y de hardware en los servidores en los cuales se implementará.

    3) Es sencillo de implementar y de mantener.

    4) Está incluido en el núcleo de Postgres y cuenta con la robustez y el soporte de Postgres.

5) Las actualizaciones van de la mano con las actualizaciones de las versiones de Postgres.

6) Se cuenta con manuales e información que facilitan su implementación y mantenimiento.

- El método de balanceo de carga seleccionado para manejar las distintas consultas que se realizan en los servidores Postgres es "***SQL Relay***". Lo que se busca con *SQL Relay* es poder acelerar y mejorar la escalabilidad de las aplicaciones con bases de datos basadas en la Web y la distribución de acceso a bases de datos en clúster o replicadas. Además SQL Relay proporciona un balanceo de carga que distribuye las consultas sobre un grupo de servidores.

- Los backups de las distintas bases de datos Postgres se llevaran a cabo mediante dos procedimientos que consisten en un backup de los segmentos WAL, el cual resguarda todas las bases de datos y los almacena en el servidor secundario, a este procedimiento se lo conoce como "***Hot Standby***", y el segundo procedimiento consiste en la realización de un backup de las bases de datos mediante la ejecución de un "***Script***" que funciona con el comando ***pg_dump***, el cual se puede ejecutar periódicamente. Además con ***pg_dump*** es posible migrar una base de datos a un servidor con una arquitectura diferente, ya sea transferir de un servidor de 32 bits a un servidor de 64 bits.

2.6.1 Implementación de los métodos de Replicación, Balanceo de Carga y Backup

Se parte de la base de que se cuentan con dos servidores con las mismas características de hardware y sistemas operativos ***Debian GNU Linux 7.0*** y motor de bases de datos ***Postgres* 9.1**. El servidor maestro y el servidor esclavo son llamados "***debían_01***" y "***debían_02***" respectivamente y tienen asignadas las direcciones IP ***192.168.0.10*** para el maestro y ***192.168.0.15*** para el servidor esclavo. El servidor "***debían_01***" será instalado en el IBM BladeCenter principal que a su vez contendrá en otra bahía independiente al servidor "***debían_00***" con dirección IP ***192.168.0.5*** de ***SQL Relay***, y el servidor "***debían_02***" en el IBM BladeCenter secundario.

2.6.1.1 Configuración de Streaming Replication

Lo primero que tenemos que hacer en el servidor maestro (debian_01 - 192.168.0.10) y en el esclavo (debian_02 - 192.168.0.15) es crear los directorios necesarios que se van a utilizar en el sistema de replicación, la manera de crearlos es:

```
root@debian_01:/# mkdir -p /var/pgsql/data
root@debian_01:/# chown -R postgres:postgres /var/pgsql/data
root@debian_01:/# chmod -R 0700 /var/pgsql/data
```

Imagen 30 – Creando los directorios necesarios.

Después se debe configurar el acceso mediante llaves SSH entre el maestro y el esclavo. Primero se tienen que generar las claves pública y privada en el servidor maestro y en el esclavo:

```
root@debian_01:/# su postgres
postgres@debian_01:/$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/var/lib/postgresql/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /var/lib/postgresql/.ssh/id_rsa.
Your public key has been saved in /var/lib/postgresql/.ssh/id_rsa.pub.
The key fingerprint is:
ec:67:1b:6b:27:e1:9a:33:74:90:29:da:ce:27:6d:cc postgres@debian_01
The key's randomart image is:
+--[ RSA 2048]----+
|                 |
|                 |
|         o       |
|      ..+        |
|     o .S.       |
|    . ... o      |
|     o =.o+.     |
|      + E+++.    |
|       +o=oo     |
+-----------------+
postgres@debian_01:/$
```

Imagen 31 – Generación de claves en el servidor principal.

Luego generar las claves en el servidor esclavo.

```
root@debian_02:/# su postgres
postgres@debian_02:/$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/var/lib/postgresql/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /var/lib/postgresql/.ssh/id_rsa.
Your public key has been saved in /var/lib/postgresql/.ssh/id_rsa.pub.
The key fingerprint is:
c8:54:64:cc:a0:94:3d:0f:e2:b0:8f:cf:69:4e:ea:45 postgres@debian_02
The key's randomart image is:
+--[ RSA 2048]----+
|     .o.=+       |
|  ..o.+oo        |
|   +...+         |
|  . .o ..        |
|   oE o S        |
|  ...            |
|   oo.           |
|   +=            |
| .oo.            |
+-----------------+
postgres@debian_02:/$
```

Imagen 32 – Generación de claves en el servidor secundario.

A continuación se debe crear el fichero */var/lib/postgresql/.ssh/authorized_keys* en el servidor maestro con el contenido del fichero */var/lib/postgresql/.ssh/id_rsa.pub* del servidor esclavo, para después crear el fichero */var/lib/postgresql/.ssh/authorized_keys* en el servidor esclavo con el contenido del fichero */var/lib/postgresql/.ssh/id_rsa.pub* del servidor maestro.

En el paso posterior se deben actualizar en el fichero ***postgresql.conf*** los siguientes parámetros:

- ***listen_addresses*** **= '*'** # Define la IP por la que se accede vía TCP/IP a Postgres.
- ***wal_level = hot_standby*** # Define cuanta información se grabará en los ficheros WAL generados.
- ***archive_mode = on*** # Se active el archivado de ficheros WAL en el servidor maestro.
- ***archive_command =***

  'scp /var/lib/postgresql/9.1/main/%p 192.168.0.15:/var/pgsql/data/%f'

Con el comando scp (*Secure CoPy*) se copia de manera encriptada desde el servidor maestro los ficheros WAL archivados en el directorio `/var/lib/postgresql/9.1/main/` al directorio `/var/pgsql/data/` del servidor esclavo.

- ***max_wal_senders = 1*** # Define el número máximo de conexiones que se pueden realizar desde servidores esclavos al servidor maestro vía *Streaming Replication* (1 por servidor esclavo).

- ***wal_keep_segments = 100*** # Define el número máximo de ficheros WAL que se mantendrán sin reciclar en el servidor maestro en el caso de que el *Streaming Replication* se retrase en la replicación de datos. Si además de *Streaming Replication* se utiliza la transferencia de ficheros WAL, este parámetro no es tan importante de configurar.

Como se va a utilizar *Streaming Replication*, se tiene que definir también en el fichero ***pg_hba.conf*** del servidor maestro una línea que permita el acceso del proceso receptor *WAL receiver* al servidor maestro. La última línea de la siguiente imagen le da acceso al proceso del servidor esclavo:

```
  GNU nano 2.2.6         Fichero: /etc/postgresql/9.1/main/pg_hba.conf

# IPv4 local connections:
host    all             all             127.0.0.1/32            md5
# IPv6 local connections:
host    all             all             ::1/128                 md5
# Allow replication connections from localhost, by a user with the
# replication privilege.
#local   replication     postgres                                peer

#host    replication     postgres        ::1/128                 md5
host     replication     all             192.168.0.15/32         md5
```

Imagen 33 – Modificación del archivo *pg_hba.conf*.

Luego de haber finalizado las configuraciones en el servidor maestro se debe reiniciar para que tome los cambios:

```
root@debian_01:/home/marcelo# /etc/init.d/postgresql restart
[ ok ] Restarting PostgreSQL 9.1 database server: main.
root@debian_01:/home/marcelo#
```

Imagen 34 – Reiniciando Postgres en servidor principal.

Una vez arrancado el servidor maestro se tiene que realizar el backup inicial físico por medio del mismo procedimiento que se utiliza con *PITR*. A continuación, se procede a restaurar esa copia de seguridad en el servidor esclavo utilizando los comandos *pg_start_backup*, *rsync* y *pg_stop_backup*:

```
root@debian_01:/# su postgres
postgres@debian_01:/$ psql postgres -c "SELECT pg_start_backup('replica');"
 pg_start_backup
-----------------
 0/24000020
(1 fila)

postgres@debian_01:/$
```

Imagen 35 – Realizando backup inicial – Paso 1.

```
postgres@debian_01:/$ rsync -av --exclude pg_xlog --exclude postgresql.conf
--exclude postgresql.pid --exclude postmaster.pid
--exclude postmaster.opts /var/lib/postgresql/9.1/main/ 192.168.0.15:/var/lib/postgresql/9.1/main/
sending incremental file list
./
backup_label
backup_label.old
postmaster.opts
postmaster.pid
base/1/
base/1/pg_internal.init
base/11918/
global/
global/pg_control
global/pg_internal.init
pg_clog/0000
pg_multixact/offsets/0000
pg_notify/
pg_notify/0000
pg_stat_tmp/
pg_stat_tmp/pgstat.stat
pg_subtrans/0000
replica/

sent 330387 bytes  received 952 bytes  73630.89 bytes/sec
total size is 45241435  speedup is 136.54
```

Imagen 36 – Realizando backup inicial – Paso 2.

```
postgres@debian_01:~/$ psql postgres -c "select pg_stop_backup(), current_timestamp";
NOTICE:  finalización de pg_stop_backup completa, esperando que se archiven los segmentos WAL
requeridos
WARNING:  pg_stop_backup todavía espera que todos los segmentos WAL requeridos sean archivados
(han pasado 60 segundos)
SUGERENCIA:  Verifique que su archive_command se esté ejecutando con normalidad.  pg_stop_backup
puede ser abortado confiablemente, pero el respaldo de la base de datos no será utilizable
a menos que disponga de todos los segmentos de WAL.
NOTICE:  pg_stop_backup completado, todos los segmentos de WAL requeridos han sido archivados
 pg_stop_backup |              now
----------------+-------------------------------
 0/19000DC4     | 2013-11-24 14:59:04.51856-03
(1 fila)

postgres@debian_01:~/$
```

Imagen 37 – Realizando backup inicial – Paso 3.

Para finalizar, resta modificar el fichero ***postgresql.conf*** y además crear el fichero ***recovery.conf*** en el servidor esclavo.

En el archivo ***postgresql.conf*** se tiene que definir los parámetros siguientes:

- ***listen_addresses = '192.168.0.15'*** # Define la IP por la que se accede vía TCP/IP a Postgres.
- ***hot_standby = on*** # Define que el servidor esclavo se podrá utilizar para realizar consultas de solo lectura.

Y en el archivo */var/lib/postgresql/9.1/main/recovery.conf* se definen los parámetros siguientes:

- ***standby_mode = 'on'*** # Especifica que el servidor se inicie en modo '*Standby'*. En Streaming Replication este parámetro debe ser establecido en '***on***'.
- ***primary_conninfo = 'host=192.168.0.10 port=5432 user=postgres'*** # Especifica una cadena de conexión que se utiliza para que el servidor secundario se conecte con el servidor primario.
- ***trigger_file = '/tmp/postgresql.trigger.5432'*** # En este parámetro se define un fichero que en caso de existir sacará al servidor esclavo del modo "*Hot Standby*" y de recuperación continua.
- ***restore_command = 'cp –f /var/pgsql/data/%f /var/lib/postgresql/9.1/main/%p'*** # Este parámetro restaura los ficheros almacenados en */var/pgsql/data/* al directorio de datos, no es necesario si *wal_keep_segments* es lo suficientemente grande para mantener los archivos WAL requeridos por el servidor esclavo.

Una vez efectuada dichas modificaciones, se procede a reiniciar el servidor esclavo para que tome la nueva configuración:

```
root@debian_02:/home/marcelo# /etc/init.d/postgresql restart
[ ok ] Restarting PostgreSQL 9.1 database server: main.
root@debian_02:/home/marcelo#
```

Imagen 38 – Reiniciando Postgres en servidor secundario.

Para corroborar que tanto el servidor maestro, como el servidor esclavo contienen los mismos datos se ejecutan los siguientes comandos en el servidor maestro:

```
postgres@debian_01:/$ psql
psql (9.1.9)
Digite «help» para obtener ayuda.

postgres=# \l
                                 Listado de base de datos
  Nombre   |  Dueño   | Codificación |   Collate   |    Ctype    |      Privilegios
-----------+----------+--------------+-------------+-------------+-----------------------
 Bases     | postgres | UTF8         | es_AR.UTF-8 | es_AR.UTF-8 |
 Becas     | postgres | UTF8         | es_AR.UTF-8 | es_AR.UTF-8 |
 postgres  | postgres | UTF8         | es_AR.UTF-8 | es_AR.UTF-8 |
 template0 | postgres | UTF8         | es_AR.UTF-8 | es_AR.UTF-8 | =c/postgres          +
           |          |              |             |             | postgres=CTc/postgres
 template1 | postgres | UTF8         | es_AR.UTF-8 | es_AR.UTF-8 | =c/postgres          +
           |          |              |             |             | postgres=CTc/postgres
 test001   | postgres | UTF8         | es_AR.UTF-8 | es_AR.UTF-8 |
 toba_2_3  | postgres | UTF8         | es_AR.UTF-8 | es_AR.UTF-8 |
(7 filas)

postgres=#
```

Imagen 39 – Listando bases de datos en el servidor principal.

Y se hace lo mismo en el servidor esclavo:

```
postgres@debian_02:/$ psql
psql (9.1.9)
Digite «help» para obtener ayuda.

postgres=# \l
                                 Listado de base de datos
  Nombre   |  Dueño   | Codificación |   Collate   |    Ctype    |      Privilegios
-----------+----------+--------------+-------------+-------------+-----------------------
 Bases     | postgres | UTF8         | es_AR.UTF-8 | es_AR.UTF-8 |
 Becas     | postgres | UTF8         | es_AR.UTF-8 | es_AR.UTF-8 |
 postgres  | postgres | UTF8         | es_AR.UTF-8 | es_AR.UTF-8 |
 template0 | postgres | UTF8         | es_AR.UTF-8 | es_AR.UTF-8 | =c/postgres          +
           |          |              |             |             | postgres=CTc/postgres
 template1 | postgres | UTF8         | es_AR.UTF-8 | es_AR.UTF-8 | =c/postgres          +
           |          |              |             |             | postgres=CTc/postgres
 test001   | postgres | UTF8         | es_AR.UTF-8 | es_AR.UTF-8 |
 toba_2_3  | postgres | UTF8         | es_AR.UTF-8 | es_AR.UTF-8 |
(7 filas)

postgres=#
```

Imagen 40 – Listando bases de datos en el servidor secundario.

Para confirmar que los cambios son replicados automáticamente al servidor esclavo se crea una base de datos de prueba en el servidor principal de la siguiente manera:

```
postgres@debian_01:/$ psql
psql (9.1.9)
Digite «help» para obtener ayuda.

postgres=# CREATE DATABASE prueba01;
CREATE DATABASE
postgres=# \c prueba01
Ahora está conectado a la base de datos «prueba01» con el usuario «postgres».
prueba01=# CREATE TABLE prueba1 (id bigint, value bigint, primary key (id));
NOTICE:  CREATE TABLE / PRIMARY KEY creará el índice implícito «prueba1_pkey» para la tabla
 «prueba1»
CREATE TABLE
prueba01=# INSERT INTO prueba1 (id, value) VALUES (1,1);
INSERT 0 1
prueba01=#
```

Imagen 41 – Creando base de prueba en servidor principal.

Luego se verifica que la base de datos se encuentra efectivamente creada en el servidor principal:

```
postgres=# \l
                                 Listado de base de datos
   Nombre    |  Dueño   | Codificación |   Collate   |    Ctype    |      Privilegios
-------------+----------+--------------+-------------+-------------+-----------------------
 Bases       | postgres | UTF8         | es_AR.UTF-8 | es_AR.UTF-8 |
 Becas       | postgres | UTF8         | es_AR.UTF-8 | es_AR.UTF-8 |
 postgres    | postgres | UTF8         | es_AR.UTF-8 | es_AR.UTF-8 |
 prueba01    | postgres | UTF8         | es_AR.UTF-8 | es_AR.UTF-8 |
 template0   | postgres | UTF8         | es_AR.UTF-8 | es_AR.UTF-8 | =c/postgres          +
             |          |              |             |             | postgres=CTc/postgres
 template1   | postgres | UTF8         | es_AR.UTF-8 | es_AR.UTF-8 | =c/postgres          +
             |          |              |             |             | postgres=CTc/postgres
 test001     | postgres | UTF8         | es_AR.UTF-8 | es_AR.UTF-8 |
 toba_2_3    | postgres | UTF8         | es_AR.UTF-8 | es_AR.UTF-8 |
(8 filas)

postgres=#
```

Imagen 42 – Listando bases de datos en el servidor principal.

Ahora es tiempo de verificar que la "*Streaming Replication*" se desarrolló de manera satisfactoria y se comprueba en el servidor esclavo de la manera siguiente:

```
postgres@debian_02:/$ psql
psql (9.1.9)
Digite «help» para obtener ayuda.

postgres=# \l
                                 Listado de base de datos
  Nombre   |  Dueño   | Codificación |   Collate   |    Ctype    |      Privilegios
-----------+----------+--------------+-------------+-------------+-----------------------
 Bases     | postgres | UTF8         | es_AR.UTF-8 | es_AR.UTF-8 |
 Becas     | postgres | UTF8         | es_AR.UTF-8 | es_AR.UTF-8 |
 postgres  | postgres | UTF8         | es_AR.UTF-8 | es_AR.UTF-8 |
 prueba01  | postgres | UTF8         | es_AR.UTF-8 | es_AR.UTF-8 |
 template0 | postgres | UTF8         | es_AR.UTF-8 | es_AR.UTF-8 | =c/postgres          +
           |          |              |             |             | postgres=CTc/postgres
 template1 | postgres | UTF8         | es_AR.UTF-8 | es_AR.UTF-8 | =c/postgres          +
           |          |              |             |             | postgres=CTc/postgres
 test001   | postgres | UTF8         | es_AR.UTF-8 | es_AR.UTF-8 |
 toba_2_3  | postgres | UTF8         | es_AR.UTF-8 | es_AR.UTF-8 |
(8 filas)

postgres=# \c prueba01
Ahora está conectado a la base de datos «prueba01» con el usuario «postgres».
prueba01=# \d prueba1
      Tabla «public.prueba1»
 Columna |  Tipo  | Modificadores
---------+--------+---------------
 id      | bigint | not null
 value   | bigint |
Índices:
    "prueba1_pkey" PRIMARY KEY, btree (id)

prueba01=#
```

Imagen 43 – Verificando replicación en servidor secundario.

Una vez finalizada la configuración del "Streaming Replication" tanto en el servidor principal como en el servidor esclavo, es indispensable realizar tareas de mantenimiento para el caso de que se registren fallas en el servidor maestro. Las principales tareas a llevar a cabo son:

- En el servidor maestro se debe limpiar el directorio donde se archivan los ficheros WAL, borrando los ficheros WAL antiguos que no son necesarios.
- En el servidor esclavo limpiar el directorio donde son transferidos los ficheros WAL, borrando los ficheros WAL antiguos que no se necesiten.
- En caso de que falle el servidor maestro, proceder a activar el servidor esclavo como nuevo servidor maestro.
- Monitorizar el estado de la replicación para saber el retraso del servidor esclavo en relación al maestro.

2.6.1.2 Configuración de SQL Relay

Instalación de SQL Relay

Los requerimientos necesarios para poder instalar y configurar SQL Relay son en primer lugar la descarga de la distribución de código fuente más actualizada de la librería Rudiments y de SQL Relay. En este caso tratándose de una distribución *Linux Debian Wh*eezy se deben descargar los archivos con extensión *tar.gz* a un directorio con permisos de lectura y escritura.

Para extraer la distribución del código fuente de *Rudiments Library*, se debe posicionar en el directorio donde se ha descargado el archivo correspondiente y ejecutar lo siguiente:

gunzip rudiments-0.44.1.tar.gz

tar xf rudiments-0.44.1.tar

Luego se deben ejecutar los siguientes comandos:

./configure

make

make install

Para el caso de la distribución del código fuente de *SQL Relay*, se debe posicionar en el directorio donde se ha descargado el archivo correspondiente y ejecutar:

gunzip sqlrelay-0.53.1.tar.gz

tar xf sqlrelay-0.53.1.tar

Y luego se deben ejecutar los siguientes comandos para su instalación:

./configure

make

make install

A menos que se modifique el directorio de instalación, por defecto se instalará en el directorio */usr/local/firstworks/*. Las bibliotecas se instalan en */usr/local/firstworks/lib*, los archivos include se instalan en /usr/local/firstworks/include. Los archivos binarios y secuencias de comandos se instalan en */usr/local/firstworks/bin*. Un archivo de configuración de ejemplo se instalará en el directorio */usr/local/firstworks/etc*. Por otro lado, dentro del directorio */etc/* se encuentra el archivo *sqlrelay*, el cual debe contener el nombre de cada instancia declarada en el archivo *sqlrelay.conf* con el objeto de que se ejecuten automáticamente cada vez que el servidor se encienda o se reinicie.

Filtrado y ruteo de consultas (Query Routing and Filtering)

La función de filtrado y enrutamiento de consultas de SQL Relay permite enviar una serie de consultas al servidor principal de base de datos, y otro conjunto de consultas al servidor secundario, y filtrar las consultas completamente para que no se envían a cualquier servidor. Por medio de esta característica es posible enviar *insert*, *update* y *delete* a la base de datos maestra y distribuir las consultas *select* al servidor secundario, como así también filtrar las consultas que carezcan de algún criterio como por ejemplo una cláusula *where*.

Normalmente, SQL Relay mantendrá un conjunto de conexiones persistentes a las bases de datos y distribuirá las consultas sobre esas conexiones. Las bases de datos que se conectarán son las que se definen en el archivo *sqlrelay.conf*.

Configuración General

Dentro de la etiqueta *instances* se encuentran las instancias que mantendrán conexiones con las bases de datos, estas instancias se definen mediante la etiqueta *instance*, y además las etiquetas *users* y *connections* que definen los parámetros de usuarios y de las conexiones respectivamente. Cada instancia de SQL Relay que será utilizada como un *router* debe omitir la etiqueta de *connections* e incluir una sola etiqueta *router*.

La etiqueta *router* debe contener un conjunto de etiquetas *route*. Cada etiqueta *route* define una instancia de SQL Relay que determina a que bases de datos dirigir las consultas, y contiene un conjunto de etiquetas *query pattern* que define que consultas deben ir hacia el servidor principal y cuales al servidor secundario. A continuación se detalla el archivo de configuración *sqlrelay.conf*.

Archivo de Configuración sqlrelay.conf

```
<?xml version="1.0"?>

<!DOCTYPE instances SYSTEM "sqlrelay.dtd">

<instances>

<instance id="postgres_debian02" port="9000" socket="/tmp/postgres_db02.socket"
dbase="postgresql" connections="2" maxconnections="15" maxqueuelength="5" growby="1"
ttl="60" endofsession="commit" sessiontimeout="600" runasuser="root" runasgroup="root"
cursors="5" authtier="connection" handoff="reconnect" deniedips="" allowedips=""
debug="none">

        <users>

                <user user="postgres" password="xxxxx"/>

        </users>

        <connections>

                <connection connectionid="db01" string="user=postgres; password=xxxxx;
                db=Becas;host=192.168.0.10;port=5432;typemangling=lookup" metric="1"/>

        </connections>

</instance>

<instance id="postgres_debian03" port="9001" socket="/tmp/postgres_db03.socket"
dbase="postgresql" connections="2" maxconnections="15" maxqueuelength="5" growby="1"
ttl="60" endofsession="commit" sessiontimeout="600" runasuser="root" runasgroup="root"
cursors="5" authtier="connection" handoff="reconnect" deniedips="" allowedips=""
debug="none">
```

```
        <users>
            <user user="postgres" password="xxxxx"/>
        </users>
        <connections>
            <connection connectionid="db2" string="user=postgres; password=xxxxx;
            db=Becas;host=192.168.0.15;port=5432;typemangling=lookup" metric="1"/>
        </connections>
</instance>
<instance id="router" port="9002" socket="/tmp/router.socket" dbase="router" connections="3"
maxconnections="15" maxqueuelength="5" growby="1" ttl="60" endofsession="commit"
sessiontimeout="600" runasuser="root" runasgroup="root" cursors="5" deniedips=""
allowedips="" debug="none" maxquerysize="65536" maxstringbindvaluelength="4000"
maxlobbindvaluelength="71680" idleclienttimeout="-1">
        <users>
            <user user="postgres" password="xxxxx"/>
        </users>
<router>
        <!—Envía todas las consultas DML/DDL al servidor "principal"  -->
        <route host="" port="" socket="/tmp/postgres_db02.socket" user="postgres"
        password="xxxxx">
                <query pattern="^drop "/>
                <query pattern="^create "/>
                <query pattern="^insert "/>
                <query pattern="^update "/>
```

```
                <query pattern="^delete "/>
        </route>
        <!—Envía las demás consultas al servidor “esclavo” -->
        <route host="" port="" socket="/tmp/postgres_db03.socket" user="postgres"
        password="xxxxx">
                <query pattern=".*"/>
        </route>
</router>
</instance>
</instances>
```

Descripción de los atributos utilizados en la configuración

- ***instance***

  - **Id:** El *ID* de la instancia.
  - **Port:** El puerto por donde escucha el oyente.
  - **Socket**: El socket unix por donde escucha el oyente.
  - **Dbase:** El tipo de base de datos al que debe conectarse el demonio de conexión.
  - **Connections**: El número de demonios *sqlr-connection* para iniciar una conexión cuando se utiliza *sqlr-start*.
  - **Maxconnections:** El número máximo de demonios *sqlr-connection* que se pueden crear.
  - **Maxqueuelength:** El tamaño que tiene que crecer la cola de espera antes de que se generen más conexiones.
  - **Growby**: El número de conexiones que se pondrán en marcha en el momento en que se generen nuevas conexiones.
  - **Ttl:** La cantidad de tiempo que una conexión inactiva se mantendrá con vida después de haber sido generada de forma dinámica (no se aplica a las conexiones generadas por *sqlr-start*).
  - **Endofsession:** El comando para emitir cuando un cliente termina su sesión o muere, debería ser commit o rollback.
  - **Sessiontimeout:** Si un cliente deja una sesión abierta para otro cliente, pero ningún cliente la recoge, la sesión expirará después de este intervalo.
  - **Runasuser**: Es el usuario que ejecutará los demonios *sqlr-listener*, *sqlr-connections* y *sqlr-scaler*. Tener en cuenta que el archivo *sqlrelay.conf* debe ser legible por este

usuario y los distintos directorios "*temp*" (normalmente en */usr/local/firstworks/var /sqlrelay*) deben tener permisos de escritura para este usuario.

- **Runasgroup**: Es el grupo para ejecutar los demonios *sqlr-listener* y *sqlr -connections* y *sqlr-scaler*. Tener en cuenta que el archivo *sqlrelay.conf* debe ser legible por este grupo y los distintos directorios "*temp*" (normalmente en /usr/local/firstworks/var/sqlrelay) deben tener permisos de escritura para este grupo.
- **Cursors:** Es el número de cursores de base de datos que cada demonio *sqlr-connection* abrirá y mantendrá abierto. Se pueden abrir más cursores si es necesario, pero el conjunto de cursores se reducirá de nuevo al tamaño establecido al final de cada sesión de cliente.
- **Authtier:** El cliente cuando se conecta con el demonio *sqlr-connection* le enviará su usuario y su contraseña. Si el atributo *authtier* se establece en "*connection*", el demonio *sqlr-connection* comparará el usuario y contraseña con la lista de usuarios y contraseñas almacenadas en el archivo *sqlrelay.conf* y aceptará o rechazará la conexión del cliente.
  Si el atributo authtier se establece en " *database*", el demonio *sqlr-connection* autenticará al usuario contra la propia base de datos en lugar de la lista de usuarios y contraseñas del archivo *sqlrelay.conf*.
- **Handoff:** Cuando un cliente de SQL Relay necesita comunicarse con la base de datos, se conecta a un proceso de escucha que se pone en cola hasta que el demonio de conexión de la base de datos se encuentre disponible. Cuando el demonio está disponible, el cliente se "*transfiere*" al demonio.
- **deniedips:** Una expresión regular que indica a que direcciones IP se les denegará el acceso.
- **Allowedips**: Una expresión regular que indica a que direcciones IP se les permitirá el acceso.
- **Maxquerysize:** Establece la longitud máxima que aceptará el servidor SQL Relay de una consulta, si un cliente intenta enviar una consulta más larga, el servidor cerrará la conexión (por defecto es de 64 kbytes).
- **maxstringbindvaluelength:** Establece la longitud máxima de una cadena de valores bind que el servidor SQL Relay aceptará, si el cliente intenta enviar una cadena de valores bind más larga, el servidor cerrará la conexión (por defecto es de 32 kbytes).
- **maxlobbindvaluelength:** Establece la longitud máxima de un valor *LOB*[54]*/CLOB*[55] que aceptará el servidor SQL Relay, si el cliente intenta enviar un valor LOB/CLOB más largo, el servidor cerrará la conexión (por defecto es de 70 kbytes).

---

[54] LOB (*Large Objects*): Son tipos de datos para almacenar objetos grandes, consisten en largas hileras de bits de longitud variable, que pueden interpretarse como caracteres o simplemente como bits.

[55] CLOB (*Character Large Object*): Es un tipo de objeto LOB que es utilizado para almacenar texto.

- **idleclienttimeout:** Establece los segundos que un cliente puede estar inactivo mientras que ha iniciado sesión en el servidor SQL Relay antes de que sea desconectado, por defecto es ***-1*** que quiere decir para siempre.

- **user**

  - **user:** El nombre de usuario requerido para conectarse con el proceso oyente.
  - **Password:** La contraseña requerida para conectarse con el proceso oyente.

- **connection**

  - **Connectionid:** El ID de la conexión.
  - **String:** La cadena de conexión de la base de datos que el demonio debe utilizar.

Para bases de datos ***postgresql***, los parámetros de la cadena de conexión son los siguientes:

  - ✓ **user**: Es el usuario que se debe utilizar para iniciar sesión en la base de datos.
  - ✓ **password**: Es la contraseña que se debe utilizar para iniciar sesión en la base de datos.
  - ✓ **db**: Es la base de datos que se debe utilizar para iniciar sesión.
  - ✓ **host**: El host que se utiliza para conectarse.
  - ✓ **port**: El puerto para conectarse al host, por defecto es el puerto 5432.
  - ✓ **typemangling**: Si se establece en "*yes*", entonces los tipos de columnas se convierten a los tipos estándar. Si se establece en "*lookup*", entonces la tabla *pg_type* se consulta en el arranque y los tipos de nombres de columna se devuelven como aparecen en dicha tabla.

- **metric:** Un número que influye en cuántas conexiones se deben iniciar.
- **route**

  - **Host:** El host del servidor al que le serán enviadas las consultas.
  - **Port:** El puerto del servidor al que le serán enviadas las consultas.
  - **Socket:** El socket del servidor al cual le serán enviadas las consultas.
  - **User:** Es el usuario que se debe utilizar para iniciar sesión en la base de datos.
  - **Password:** Es la contraseña que se debe utilizar para iniciar sesión en la base de datos.

- **query**

  - **Pattern:** Es una expresión regular, cualquier consulta que coincida con esta expresión regular se enrutará al servidor especificado en la etiqueta *route*.

Monitorear el estado de SQL Relay

El programa ***sqlr-status*** muestra estadísticas acerca de cómo está funcionando determinada instancia de SQL Relay. Se puede ejecutar sustituyendo la palabra *instancia* con el nombre de la instancia de SQL Relay de la cual se requiere conocer el estado. El comando para ejecutarla es el siguiente: ***./sqlr-status -id nombre_instancia***

Para generar las salidas se utilizaron tres scripts desarrollados en PHP, un script que realiza una operación consulta (*Select*), otro que efectúa una actualización de un registro determinado (*Update*) y el tercer script que contiene ambas operaciones (*Select* y *Update*), las respectivas salidas se pueden observar en las siguientes imágenes:

```
root@debian:/usr/local/firstworks/bin# ./sqlr-status -id postgres_debian02
  Open   Database Connections:  2
  Opened Database Connections:  2

  Open   Database Cursors:      10
  Opened Database Cursors:      16

  Open   Client Connections:    0
  Opened Client Connections:    0

  Times  New Cursor Used:       0
  Times  Cursor Reused:         0

  Total  Queries:               0
  Total  Errors:                0
```

Imagen 44 – Estado inicial de *SQL Relay* de la instancia *postgres_debian02*.

```
root@debian:/usr/local/firstworks/bin# ./sqlr-status -id postgres_debian03
  Open   Database Connections:  2
  Opened Database Connections:  2

  Open   Database Cursors:      10
  Opened Database Cursors:      16

  Open   Client Connections:    0
  Opened Client Connections:    0

  Times  New Cursor Used:       0
  Times  Cursor Reused:         0

  Total  Queries:               0
  Total  Errors:                0
```

Imagen 45 – Estado inicial de *SQL Relay* de la instancia *postgres_debian03*.

```
root@debian:/usr/local/firstworks/bin# ./sqlr-status -id router
  Open   Database Connections:  3
  Opened Database Connections:  3

  Open   Database Cursors:      15
  Opened Database Cursors:      15

  Open   Client Connections:    0
  Opened Client Connections:    0

  Times  New Cursor Used:       0
  Times  Cursor Reused:         0

  Total  Queries:               0
  Total  Errors:                0
```

Imagen 46 – Estado inicial de *SQL Relay* de la instancia *router*.

Al ejecutar el script PHP con la operación *Update* se obtienen los estados que se muestran a continuación:

```
root@debian:/usr/local/firstworks/bin# ./sqlr-status -id postgres_debian02
  Open   Database Connections:  2
  Opened Database Connections:  2

  Open   Database Cursors:      10
  Opened Database Cursors:      17

  Open   Client Connections:    0
  Opened Client Connections:    1

  Times  New Cursor Used:       1
  Times  Cursor Reused:         0

  Total  Queries:               1
  Total  Errors:                0
```

Imagen 47 – Estado de la instancia *postgres_debian02* luego de un *UPDATE*.

```
root@debian:/usr/local/firstworks/bin# ./sqlr-status -id postgres_debian03
  Open   Database Connections:  2
  Opened Database Connections:  2

  Open   Database Cursors:      10
  Opened Database Cursors:      17

  Open   Client Connections:    0
  Opened Client Connections:    1

  Times  New Cursor Used:       0
  Times  Cursor Reused:         0

  Total  Queries:               0
  Total  Errors:                0
```

Imagen 48 – Estado de la instancia *postgres_debian03* luego de un *UPDATE*.

```
root@debian:/usr/local/firstworks/bin# ./sqlr-status -id router
  Open   Database Connections:  3
  Opened Database Connections:  3

  Open   Database Cursors:      15
  Opened Database Cursors:      15

  Open   Client Connections:    0
  Opened Client Connections:    1

  Times  New Cursor Used:       1
  Times  Cursor Reused:         1

  Total  Queries:               1
  Total  Errors:                0
```

Imagen 49 – Estado de la instancia *router* luego de un *UPDATE*.

Al ejecutar el script PHP con la operación *Select* se obtienen los estados que se muestran a continuación:

```
root@debian:/usr/local/firstworks/bin# ./sqlr-status -id postgres_debian02
  Open   Database Connections:  2
  Opened Database Connections:  2

  Open   Database Cursors:      10
  Opened Database Cursors:      17

  Open   Client Connections:    0
  Opened Client Connections:    2

  Times  New Cursor Used:       1
  Times  Cursor Reused:         0

  Total  Queries:               1
  Total  Errors:                0
```

Imagen 50 – Estado de la instancia *postgres_debian02* luego de un *SELECT*.

```
root@debian:/usr/local/firstworks/bin# ./sqlr-status -id postgres_debian03
  Open   Database Connections:  2
  Opened Database Connections:  2

  Open   Database Cursors:      10
  Opened Database Cursors:      17

  Open   Client Connections:    0
  Opened Client Connections:    2

  Times  New Cursor Used:       1
  Times  Cursor Reused:         1

  Total  Queries:               1
  Total  Errors:                0
```

Imagen 51 – Estado de la instancia *postgres_debian03* luego de un *SELECT*.

```
root@debian:/usr/local/firstworks/bin# ./sqlr-status -id router
  Open   Database Connections:  3
  Opened Database Connections:  3

  Open   Database Cursors:      15
  Opened Database Cursors:      15

  Open   Client Connections:    0
  Opened Client Connections:    2

  Times  New Cursor Used:       2
  Times  Cursor Reused:         2

  Total  Queries:               2
  Total  Errors:                0
```

Imagen 52 – Estado de la instancia *router* luego de un *SELECT*.

Si en cambio se ejecuta un script PHP con ambas operaciones *Select* y *Update* se obtienen los estados:

```
root@debian:/usr/local/firstworks/bin# ./sqlr-status -id postgres_debian02
  Open   Database Connections:  2
  Opened Database Connections:  2

  Open   Database Cursors:      10
  Opened Database Cursors:      18

  Open   Client Connections:    0
  Opened Client Connections:    3

  Times  New Cursor Used:       2
  Times  Cursor Reused:         0

  Total  Queries:               2
  Total  Errors:                0
```

Imagen 53 – Estado de la instancia *postgres_debian02* luego de un *SELECT Y UPDATE*.

```
root@debian:/usr/local/firstworks/bin# ./sqlr-status -id postgres_debian03
  Open   Database Connections:  2
  Opened Database Connections:  2

  Open   Database Cursors:      10
  Opened Database Cursors:      18

  Open   Client Connections:    0
  Opened Client Connections:    3

  Times  New Cursor Used:       2
  Times  Cursor Reused:         2

  Total  Queries:               2
  Total  Errors:                0
```

Imagen 54 – Estado de la instancia *postgres_debian03* luego de un *SELECT Y UPDATE*.

```
root@debian:/usr/local/firstworks/bin# ./sqlr-status -id router
  Open   Database Connections:  3
  Opened Database Connections:  3

  Open   Database Cursors:      15
  Opened Database Cursors:      15

  Open   Client Connections:    0
  Opened Client Connections:    3

  Times  New Cursor Used:       4
  Times  Cursor Reused:         4

  Total  Queries:               4
  Total  Errors:                0
```

Imagen 55 – Estado de la instancia *router* luego de un *SELECT Y UPDATE*.

A continuación se describe el significado de cómo se deben interpretar las descripciones de las estadísticas:

- *Open Database Connections*: El número de conexiones de bases de datos que están actualmente abiertas.
- *Opened Database Connections*: El número total de conexiones de base de datos que se han abierto desde que se inició la instancia.
- *Open Database Cursors*: El número de cursores[56] de bases de datos que están actualmente abiertos.
- *Opened Database Cursors*: El número total de los cursores de base de datos que se han abierto desde que se inició la instancia.
- *Open Client Connections*: El número de clientes SQL Relay que están actualmente conectados al servidor SQL Relay.
- *Opened Client Connections*: El número total de clientes SQL Relay que se han conectado a una instancia del servidor SQL Relay desde que se inició la misma.
- *Times New Cursor Used*: El número de veces que un cursor no puede ser reutilizado.
- *Times Cursor Reused*: El número de veces que un cursor podría ser reutilizado.
- *Total Queries*: El número total de consultas que se han ejecutado a través de la instancia.
- *Total Errors*: El número total de consultas que generaron errores.

[56] Cursores en Bases de Datos: Se refiere a una estructura de control utilizada para el recorrido y potencial procesamiento de los registros del resultado de una consulta.

2.6.1.3 Configuración de Backups

Para la realización de los backups se han elegido dos mecanismos, el primero es mediante la característica que proporciona Postgres que se llama "*Hot Streaming*" cuya implementación se detalló en el apartado 2.6.1.1. El otro mecanismo para la realización diaria de backups de determinadas bases de datos o de todas las bases de datos se puede utilizar el siguiente script, el cual automatizará la confección de dichos backups:

```
#!/bin/bash

#postgres-backup.sh

# Para borrar los backups anteriores a 7 dias

find /usr/backup/postgresql/ -mtime +7 -exec rm -f {} \;

# Ruta en la que guardar los backups

backup_dir="/usr/backup/postgresql/"


# Parámetros del Usuario

username="postgres"

export PGPASSWORD="postgres"


### Archivo de log llamado pgsql.log que informa lo que acontece con pg_dump

FECHA=$(date)

echo "Generado" > $backup_dir/pgsql.log

echo $FECHA >> $backup_dir/pgsql.log
```

Backup individual por determinadas bases de datos

pg_dump -h localhost -U postgres -c Becas > $backup_dir/Becas.sql

pg_dump -h localhost -U postgres -sv Becas -O > $backup_dir/Becas_schema.sql

pg_dump -h localhost -U postgres -Fc -f $backup_dir/Becas_data.sql -a --disable-triggers Becas

Backup general de todas las bases de datos

pg_dumpall -h localhost -U postgres > $backup_dir/todas_las_bases.out

Para que tome la variable ***PGPASSWORD*** del script se debe crear un archivo con el nombre .pgpass dentro del directorio ***/usr/backup/postgres*** que es donde se van a hacer los backup, debe contener la línea ***127.0.0.1:5432::usurio-postgres:clave-usuario***. Luego se deben cambiar los permisos de ejecución para el usuario root de la manera siguiente: ***#chmod 700 postgres-backup.sh***.

Es posible automatizarlo en el cron del sistema operativo Linux editando el archivo crontab ***# crontab -e*** y agregando la línea siguiente: ***30 6 * * 1-5 /etc/init.d/postgres-backup.sh > /usr/backup/postgresql/pgsql.log***, esto ejecutará el script de lunes a viernes a las 6:30 de la mañana.

3- TRANSFERENCIA Y BENEFICIARIOS

Como principales beneficiarios se pretende llegar en forma directa a los administradores de bases de datos (DBA) que son, los que tienen que solucionar los problemas derivados al mal funcionamiento de una base de datos.

Como beneficiarios indirectos de la implementación de la replicación se encuentran los usuarios de los sistemas.

4- BIBLIOGRAFÍA.

- **Bucardo** (2011) - http://bucardo.org
- **HAProxy** (2010) - Willy Tarreau - http://haproxy.1wt.eu/
- **Manual del usuario de PostgreSQL** (1996 - 1999) - Thomas Lockhart.
- **Material Capacitación SIU** - (Agosto de 2009) - http://www.siu.edu.ar/
- **PGCluster-II** (2007) - Atsushi Mitani - http://www.pgcon.org/2007/schedule/events/6.en.html

  http://www.pgcon.org/2007/schedule/attachments/26-pgcluster2pgcon.pdf
- **PgPool-II** (2003 - 2013) - pgpool Global Development Group - http://www.pgpool.net/docs/latest/pgpool-en.html
- **PostgreSQL-es** (2010) – Rafael Martínez Guerrero- http://www.postgresql.org.es/node/483
- **PostgreSQL 9 Administration Cookbook** (2010), Simon Riggs - Hannu Krosing, Packt Publishing - ISBN 978-1-849510-28-8
- **PostgreSQL 9.2.5 Documentation** (1996 - 2013) - The PostgreSQL Global Development Group - http://www.postgresql.org/files/documentation/pdf/9.2/postgresql-9.2-US.pdf
- **Postgres-R** (1996 - 2010) -Postgres Global Development Group - http://www.postgres-r.org/documentation/terms
- **PyReplica** (2004 - 2010) – Mariano Reingart - https://code.google.com/p/pyreplica/
- **Slony - I** (2013) - Christopher Browne - http://slony.info/adminguide/2.1/doc/adminguide/slony.pdf
- **SQL Relay** (2011) - David Muse - http://sqlrelay.sourceforge.net/documentation.html
- **Transaction Processing in PostgreSQL** (2000) - Tom Lane - http://www.postgresql.org/files/developer/transactions.pdf
- **Tutorial de PostgreSQL** (1996 - 1999) - Thomas Lockhart.