Optimizaciones de PostgreSQL

Francisco de Borja Lopez Rio

wu@e-shell.org

En el artículo anterior de la serie (http://alf.dyndns.ws/numero1/articulo02.php; PostgreSQL, instalación y primeros pasos) vimos como instalar un servidor PostgreSQL minimamente funcional en OpenBSD y FreeBSD. Ahora es el momento de ver algunas de las opciones de las que dispone PostgreSQL para optimizar nuestro sistema y sacarle el máximo rendimiento posible.

Veremos que hay algunas opciones que podremos cambiar sobre la marcha con simples sentencias SQL, y veremos que hay otras configuraciones que requieren que le enviemos una señal SIGHUP al servidor. Las opciones que requieren que reiniciemos el servicio las vamos a establecer en el fichero $PGDATA/postgresql.conf, siendo $PGDATA la ruta al directorio donde se almacena la info de la base de datos.

*Wu repite con nosotros con la continuación de su excelente serie sobre PostgreSQL en *BSD. Esperamos que su relación con este ezine no se quede ahí y podamos disfrutar de más textos suyos en siguientes ediciones.*

Tabla de contenidos

1. Cambios dinámicos - pg_settings

Mediante la vista pg_settings, tenemos acceso a varios parámetros de configuración interesantes de PostgreSQL, algunos de ellos podremos cambiarlos sobre la marcha con el server corriendo.

```
template1=# \d pg_settings
View "pg_catalog.pg_settings"
 Column  | Type | Modifiers
---------+------+-----------
 name    | text |
 setting | text |
View definition: SELECT a.name, a.setting FROM pg_show_all_settings() \
a(name text, setting text);
Rules: pg_settings_u,
       pg_settings_n
template1=#
```

La vista pg_settings se compone de dos campos, name y settings. El primero es el nombre del parámetro, mientras el segundo es el valor que tendrá asignado . Para los que estamos acostumbrados a trabajar con sistemas unix, sobre todo a nivel de configuración del kernel, nos puede parecer similar al sistema al utilizado en las sysctl de los kernels unix-like (Linux, BSD, Solaris, etc).

Para una explicación detallada de todas las opciones configurables dinámicamente y lo que hace cada una, lo mejor es echarle un ojo a la documentación oficial del administrador de bases de datos PostgreSQL, en el apartado http://www.postgresql.org/docs/7.3/static/runtime.html Server Run-time Environment.

Para cambiar un valor con posibilidad de actualización dinámica, lo haremos a través de una sentencia SQL UPDATE, en el siguiente ejemplo, voy a actualizar el valor del parametro effective_cache_size, que sirve para establecer el tamaño efectivo del cache de disco que usara PostgreSQL para guardar datos en espacio de kernel, el valor por defecto es 1000:

```
template1=# select * from pg_settings where name='effective_cache_size';
         name         | setting
----------------------+---------
 effective_cache_size | 1000
(1 row)
template1=#
```

Para actualizarlo, con un simple update:

```
template1=# update pg_settings set setting=1024 where \
name='effective_cache_size';
 set_config
------------
 1024
(1 row)
template1=#
```

Ahora podemos ver que se ha realizado la actualización de forma correcta siendo el valor del parametro 1024 en lugar de 1000:

```
template1=# select * from pg_settings where name='effective_cache_size';
         name         | setting
----------------------+---------
 effective_cache_size | 1024
(1 row)
template1=#
```

Esto es aplicable para casi todos los parámetros de configuración, lo que nos permite cambiar al instante determinados parámetros dependiendo del uso que PostgreSQL está haciendo del servidor; es decir, memoria, tiempo de CPU, etc que esta utilizando PostgreSQL.

De todas formas, veremos ahora que hay parametros como max_connections, que no permiten cambios con el servidor corriendo.

2. Cambios estáticos - postgresql.conf y parámetros de postmaster

2.1. postgresql.conf

En algunos casos, los parámetros de configuración de PostgreSQL no pueden cambiarse haciendo un UPDATE en la vista pg_settings, si no que hemos de cambiarlo en el fichero postgresql.conf, y lanzarle un SIGHUP al servidor, para reiniciarlo y que releea los ficheros de configuración. El fichero postgresql.conf tiene un formato sencillo, del estilo:

```
nombre = valor
```

Siguiendo el mismo estilo que en la vista pg_settings, y similar tambien al fichero `/etc/sysctl.conf` de OpenBSD o FreeBSD. El archivo es como cualquier fichero de configuración de unix, o shell script, de forma que todo lo que aparezca en una línea a partir del caracter # es considerado un comentario.Un ejemplo de valor que debemos de establecer en el fichero postgresql.conf es el max_connections, que tiene un valor por defecto de 32 conexiones simultáneas:

```
max_connections = 32
```

Podríamos cambiarlo facilmente al numero que quisieramos (Con algunas limitaciones como veremos más adelante):

```
max_connections = 1024
```

Una vez cambiado el parámetro, tendríamos que enviarle un SIGHUP al server, que en nuestro caso es sencillo gracias al script rc.pgsql que hemos visto en el artículo anterior, para reiniciar el server:

```
rc.pgsql restart
```

2.2. Pasándole opciones a postmaster

Otra opción para asignar valores a estos parámetros es en el momento de lanzar el postmaster para arrancar el servidor, invocandolo con la opción -c parametro=valor establecemos ese valor para ese parámetro de configuración. Esta forma es especialmente útil en tests y pruebas del server, para ir probando los valores rápidamente, sin tener que estar editando el `postgresql.conf` y lanzando señales SIGHUP. Para ver una descripción detallada de las opciones de llamada de postmaster, basta con mirar la pagina man de postmaster(1).

3. Parámetros de configuración interesantes

Vamos a ver ahora algunos parámetros que de base serán interesantes, cada caso ha de ser estudiado luego aparte del resto, para encontrar una configuración óptima para él.

authentication_timeout:

Tiempo máximo que esperaremos una autenticación correcta del usuario, el valor por defecto es 60 segúndos. Si pasados esos 60 segúndos el usuario no se autentica correctamente el server cierra la conexión. Un valor adecuado serían entre 15 y 30 segúndos, de esa forma evitaríamos usuarios que establecen conexiones sin autenticarse y que ocupan conexiones que no podrían ser usadas por otros usuarios (un DoS).

client_min_messages, server_min_messages:

Ambos indican el nivel de información que se sacará del servidor, el primero indica la cantidad de información que le es enviada a un cliente conectado al sistema mientras que el segundo indica la información que se va ha enviar al log del sistema. Los posibles valores y sus implicaciones podemos verlos en el apartado http://www.postgresql.org/docs/7.3/static/runtime-config.html Run-time Configuration de la documentación de PostgreSQL.

fsync:

Con esta opción = on, PostgreSQL llama a fsync() para asegurarse de que los datos son grabados en el disco físicamente después de cada commit por transacción. Esto tiene ventajas e inconvenientes ya que en caso de un fallo en el sistema, si la base de datos se cierra inapropiadamente y los datos se han guardado físicamente a disco no será necesaria una reparación de la base de datos, con lo que el rearranque del sistema se agiliza muchísimo, aparte de que la perdida de datos se reduce casi a 0.

El incoveniente más grande es que dependiendo del disco que tengamos, el proceso de fsync() puede ralentizar al sistema demasiado, ya que cada vez que se llame a fsync() ha de esperar a que se escriba en el disco la información. fsync() ha sido mejorada mucho a partir de las versiones 7.1 de PostgreSQL, por lo que la diferencia de velocidad con la opción activada o no puede no ser realmente ventajosa en un entorno en el que el sistema se reinicie con frequencia.

geqo:

O Generic Query Optimization, con esta opción activamos o desactivamos el algoritmo que utiliza PostgreSQL para optimizar los querys al sistema. El valor por defecto es on (activado) y es más que aconsejable dejarlo activado. De todas formas, disponemos de opciones relacionadas con geqo, de forma que se pueden optimizar determinados parametros.

Una vez mas podemos encontrar mas información sobre estos parametros en el apartado http://www.postgresql.org/docs/7.3/static/runtime-config.html Run-time Configuration de la documentación de PostgreSQL.

hostname_lookup:

Si queremos que en lugar de la dirección IP desde la que se conecta el usuario, se loguee el nombre del host (si resuelve) deberíamos poner a on esta opción. Es bastante desaconsejable, ya que ralentiza bastante el sistema en general.

lc_*:

Establecidas como vimos en el artículo anterior en el momento de crear el database cluster (el directorio de trabajo de PostgreSQL), podemos cambiarlas ahora si fuese necesario. De todas formas, y como ya hemos visto, es mejor crear el database cluster con las locales adecuadas y olvidarnos de esto.

log_*:

Con estos parámetros configuramos el log del servidor, aquí le decimos si queremos que loguee las conexiones de los usuarios (log_connections), que marque cada línea del log con el pid del proceso y la fecha (log_pid y log_timestamp respectivamente), y si queremos que loguee tambien las sentencias SQL ejecutadas en todo momento en el servidor (log_statement).

max_expr_depth:

Este parámetro establece el número máximo de subsentencias que se van a poder alcanzar, por ejemplo subselects. El valor por defecto es 10000, lo cual en principio y si no tenemos ninguna necesidad fuera de lo normal, es mas que suficiente.

password_encryption:

Si activamos esta opción, al hacer un create user with password, pero sin ponerle encrypted, nos encryptara el password de todas formas, evitando asi el tener passwords sin encriptar en el sistema.

silent_mode:

Con esta opción activada, el servidor no suelta ningún tipo de mensaje ni por stderr ni por stdout. Esta opción es útil si tenemos configurado PostgreSQL para loguear por syslog.

ssl:

Esta opción ha de estar activada para que podamos establecer conexiones encryptadas al servidor (la opción hostssl de pg_hba.conf). Para poder activar esta opción, tendremos que generar un certificado y configurar un par de cosas, eso lo veremos en el tercer artículo de la serie.

statement_timeout:

Con esto podemos definir un número máximo de segúndos que una sentencia puede estar ejecutandose. Por defecto es 0, lo que la desactiva. Es una opción interesante en entornos en los que no nos interese que un usuario pueda ejecutar consultas demasiado largas que bloqueen recursos del servidor demasiado tiempo.

stats_start_collector:

Con esta opción, activamos o desactivamos el statistics collector de PostgreSQL, que sirve para monitorizar numerosos parámetros del servidor. En próximos articulos veremos como beneficiarse de estas estadísticas para ir afinando el servidor a nuestras necesidades.

syslog:

Con esta opción establecemos el tipo de logueo que hará el server:

- 0 - stdout/stderr solamente.
- 1 - Loguea a syslog y por stdout/stderr
- 2 - syslog, aunque algunos errores saldran a stderr a menos que activemos la opcion silent_mode que comentamos antes.

syslog_ident:

El string con el que se identificará ante el syslog (por defecto postgres).

unix_socket_*:

Estos parámetros nos permiten definir en que directorio se va a crear el socket unix mediante el que se hacen las conexiones locales (unix_socket_directory), el grupo dueño de ese fichero (unix_socket_group), y los permisos con los que será creado el fichero (unix_socket_permissions), por defecto 511 (usuario lectura y ejecucion, grupo y otros lectura).

virtual_host:

Con este parámetro podemos definir para que dirección ip o nombre de host escuchará PostgreSQL peticiones por TCP/IP, por defecto escucha en todas las interfaces de las que disponga la máquina.

max_connections y shared_buffers:

Estas dos las he dejado para el final por que son bastante especiales, además de ser dependientes una de la otra. Para cada valor de max_connections (que indica el numero máximo de conexiones simultáneas que manejará el servidor) tenemos que tener al menos el doble para shared_buffers (buffers de memoria utilizados por el servidor).

A la hora de establecer estas opciones, tendremos que tener en cuenta el uso de SYSV semaphores de PostgreSQL; es un tema espinoso, ya que dependiendo de como se encuentren configurados determinados parámetros del kernel de nuestro servidor, veremos que se nos permite un determinado máximo en max_connections o no. Para más información sobre esto, podemos recurrir al apartado http://www.postgresql.org/docs/7.3/static/kernel-resources.html Managing Kernel Resources de la documentación de PostgreSQL.