−Table des matières

pgbouncer, un pooler simple mais efficace

Cet article, écrit par Guillaume Lelarge, a été publié dans le hors-série 44 du magazine GNU/Linux Magazine France, hors-série dédié à PostgreSQL. Il est disponible maintenant sous licence Creative Commons.

pgbouncer est un autre pooler de connexions. Contrairement à pgPool-II, il se contente d'être ça. Par contre, il va proposer des options uniques, comme une option permettant de préciser le moment où on souhaite changer de connexions. Cela ne sera pas forcément une connexion unique pour une session. Cela pourra être une connexion unique par transaction ou par instruction. Une option unique en son genre, mais qui a ses propres limitations.

En dehors de cette fonctionnalité très particulière, pgbouncer a d'autres qualités. Il est remarquablement léger. Son utilisation de la mémoire est très faible. Le fait de ne pas être aussi un système de réplication et un outil de répartition de charge l'aide à être très petit.

De plus, il n'est pas lié à un seul serveur. La configuration permet d'indiquer le ou les serveurs qui hébergent telle ou telle base.

Prérequis

Comme pgbouncer ne supporte que la version 3 du protocole de communication de PostgreSQL, vous devez utiliser une version 7.4 ou ultérieure. Ce qui tombe très bien vu qu'il s'agit de la version la plus ancienne encore maintenue par les développeurs de PostgreSQL. Donc, si vous avez une version plus ancienne, commencez par mettre à jour PostgreSQL. Puis testez pgPool-II et pgbouncer pour choisir celui qui vous convient.

Installation

pgbouncer dispose d'un paquet pour Debian, mais ce dernier est pour la version Sid. Néanmoins, il est possible de l'installer sur une Etch. pgbouncer utilise libevent et il en faut une version particulièrement à jour. Donc nous utiliserons aussi la version Sid de cette bibliothèque.

```
debian1:~# wget http://ftp.fr.debian.org/debian/pool/main/p/pgbouncer/pgbouncer_1.3.1-1_i386.deb
debian1:~# wget http://ftp.fr.debian.org/debian/pool/main/libe/libevent/libevent-1.4-2_1.4.12-stable-1_i386.deb
debian1:~# dpkg -i libevent-1.4-2_1.4.12-stable-1_i386.deb pgbouncer_1.3.1-1_i386.deb
Sélection du paquet libevent-1.4-2 précédemment désélectionné.
(Lecture de la base de données... 22073 fichiers et répertoires déjà installés.)
Dépaquetage de libevent-1.4-2 (à partir de libevent-1.4-2_1.4.12-stable-1_i386.deb) ...
Sélection du paquet pgbouncer précédemment désélectionné.
Dépaquetage de pgbouncer (à partir de pgbouncer_1.3.1-1_i386.deb) ...
Paramétrage de libevent-1.4-2 (1.4.12-stable-1) ...
Paramétrage de pgbouncer (1.3.1-1) ...
pgbouncer daemon disabled in /etc/default/pgbouncer (warning).
Traitement des actions différées (« triggers ») pour « man-db »...
```

Et voilà, installation terminée. Passons à la configuration.

Configuration de pgbouncer

Le fichier de configuration de pgbouncer se trouve par défaut dans le répertoire /etc/pgbouncer. Il se nomme pgbouncer.ini. Son extension nous indique qu'il se paramètre comme tout autre fichier .ini (un format de fichier de configuration plus connu sous Windows que sous Linux). Donc, pour ceux qui n'en ont jamais vu, un fichier .ini peut comporter des commentaires. Ces derniers commencent toujours avec un point-virgule. Il est possible de déclarer des sections en plaçant un ou plusieurs mots entre crochets.

D'ailleurs, ce fichier comprend deux sections:

- databases, qui sert à déclarer toutes les bases de données qui seront utilisables à partir de pgbouncer;
- pgbouncer, qui permet de configurer des paramètres généraux.

Commençons par les paramètres généraux.

Les trois premiers permettent d'indiquer sur quelles interfaces (listen_addr), sur quel port (listen_port) ainsi que sur quel socket (unix_socket_dir) le serveur pgbouncer va attendre les connexions. Par défaut sur Debian, pgbouncer écoute sur le port 6432 de l'interface locale et sur la socket stockée dans le répertoire /var/run/postgresql. Contrairement à pgPool-II, la méthode d'authentification est fixe. Ce sera soit any, soit trust, soit plain, soit crypt soit md5.

Notre conseil est plutôt d'utiliser md5 bien que le défaut est trust. Any est spécifique à pgbouncer: dans ce cas, même le nom de l'utilisateur n'est pas vérifié. Toujours pour l'authentification, il est possible de fournir une liste d'utilisateurs et de mots de passe dans un fichier dont le chemin complet sera indiqué avec le paramètre auth_file.

pgbouncer permet d'accéder à une base de données virtuelle. Son but est de fournir des informations sur le paramétrage actuel mais aussi des statistiques sur les accès passés et présents. Évidemment, l'accès à cette base est protégée. Le paramètre admin_users permet de préciser les utilisateurs qui sont autorisées à lire et à modifier, alors que le paramètre stats_users indique les utilisateurs qui ne peuvent que consulter les statistiques.

pool_mode précise le mode de gestion des connexions. pgbouncer est capable de faire en sorte qu'une session se passe entièrement sur une seule connexion (mode session). Le mode transaction permet de changer de connexion à chaque nouvelle transaction, implicite ou explicite. Enfin, le mode statement change de connexion à chaque instruction, ce qui interdit l'usage des transactions explicites.

Il est possible de préciser une ou plusieurs requêtes à exécuter avant de changer de connexion. Ces requêtes doivent être enregistrées dans le paramètre server_reset_query. La configuration des limites de connexion est un peu particulière avec pgbouncer. max_client_conn indique le nombre maximum de connexions possibles via pgbouncer, quelque soit les bases de données ou le serveur PostgreSQL concerné. pgbouncer ne gèrera pas plus que ce nombre de connexions clients/pgbouncer. Si vous essayez de dépasser ce nombre, vous obtiendrez le message suivant:

```
ERROR:  no more connections allowed
```

default_pool_size est le nombre maximum de connexions pgbouncer/PostgreSQL pour un pool (nous verrons plus tard qu'un pool est lié une base de données côté pgbouncer). Il est à savoir que ce nombre est configurable en général s'il se trouve dans la section pgbouncer. Il est possible de personnaliser sa configuration pool par pool avec la section databases.

pgbouncer utilise des descripteurs de fichiers pour les connexions. Le nombre de descripteurs peut être bien plus important que ce que n'autorise par défaut le système d'exploitation. Le maximum théorique est de:

```
max_client_conn + (max_pool_size * nombre_de_bases * nombre_d_utilisateurs)
```

Le cas échéant, ulimit vous permettra de configurer le nombre de descripteurs disponibles sur votre système d'exploitation Linux.

Le paramètre logfile indique le fichier où sont enregistrées toutes les traces de pgbouncer. Vous pouvez demander de tracer les connexions (paramètre log_connections), les déconnexions (log_disconnections) et enfin demander l'envoi des erreurs du pooler au client (log_pooler_errors).

La configuration de certains délais est possible:

- server_lifetime permet de terminer une connexion au serveur PostgreSQL si elle a duré plus que ce temps en secondes (1200 par défaut, soit 20 minutes);
- server_idle_timeout termine une connexion au serveur si elle n'a pas été utilisée pendant cette période de temps (60 secondes par défaut);
- server_connect_timeout annule une tentative de connexion si le serveur n'a pas répondu durant ce laps de temps (15 secondes par défaut);
- server_login_retry ajoute un délai après un échec de connexion (là-aussi, 15 secondes par défaut);
- query_timeout annule une requête si cette dernière ne renvoie pas le résultat pendant ce temps en seconde (fonctionnalité désactivée par défaut car considérée comme dangereuse);
- client_idle_timeout ferme une connexion s'il n'y a aucune activité sur cette connexion pendant ce délai (fonctionnalité désactivée par défaut car considérée comme dangereuse);
- client_login_timeout déconnecte un client qui n'a pas réussi à s'authentifier après s'être connecté pendant ce laps de temps en seconde (60 par défaut).

Comme vous pouvez le voir, pgPool-II et pgbouncer partagent un bon nombre d'options. Leur nom change parfois mais il est quand même assez facile de s'y retrouver. La vraie grosse différence tient dans la section des bases de données.

En effet, alors que pgPool-II permet d'indiquer un serveur sur lequel tous les clients iront se connecter via le démon pgpool, pgbouncer va permettre de spécifier des bases de données. Ces dernières pourront faire partie du même serveur PostgreSQL ou de plusieurs serveurs PostgreSQL. Cela donne des possibilités très intéressantes.

La section databases permet de définir un ensemble de bases de données. Le nom du paramètre sera le nom de la base de données que le client devra utiliser pour se connecter. Cela correspondra ou non au nom de la base de données réelle. La valeur de ce paramètre sera un ensemble de paramètres indiquant où se connecter: adresse IP du serveur, port TCP/IP, nom de la base (optionnel), nom de l'utilisateur (optionnel). Il est aussi possible d'y ajouter quelques paramètres de configuration du pool. Voici la liste des paramètres possibles :

- host, adresse IP du serveur PostgreSQL;
- port, port TCP/IP pour la connexion au serveur PostgreSQL;
- dbname, nom de la base de données;
- user, nom de l'utilisateur;
- password, mot de passe de l'utilisateur;
- pool_size, taille du pool pour cette base de données;
- connect_query, requête à exécuter au moment de la connexion;
- client_encoding, datestyle, timezone, paramètres PostgreSQL pouvant être surchargés par pgbouncer.

Voici un exemple de configuration:

```
d1db1 = host=debian1 dbname=db1
d1db2 = host=debian1 dbname=db2 user=u1
d2db1 = host=debian2 dbname=db1
```

Avec cette configuration, si un utilisateur se connecte sur pgbouncer en demandant une connexion à la « base » d1db1, il sera mis en relation avec la base db1 sur le serveur debian1. S'il demande une connexion à la « base » d1db2, il sera connecté sur la base db2 du serveur debian1 en tant qu'utilisateur u1. Enfin, s'il demande une connexion à la « base » d2db1, il sera en fait connecté à la base db1 du serveur debian2.

L'idée comme quoi le client indique un nom de base qui lui permettra de se connecter sur différents serveurs PostgreSQL, différentes bases de données, avec un paramétrage qui peut être personnalisé est très puissante. Elle peut aussi être source de confusion pour les nouveaux utilisateurs.

Maintenant, attaquons-nous à la configuration de pgbouncer et de PostgreSQL pour montrer l'utilisation de ce pooler de connexions.

Configuration des utilisateurs

Avant de lancer la connexion vers PostgreSQL, pgbouncer vérifie le couple utilisateur/mot de passe. Pour cela, la configuration permet d'indiquer le chemin complet vers un fichier contenant une liste des utilisateurs et de leurs mots de passe. Le format est simple: le nom de l'utilisateur suivi d'un espace, suivi du mot de passe. “user” “mot de passe”

Étrangement, les guillemets sont nécessaires. Sans eux, le fichier est déclaré invalide.

Vu que les mots de passe sont directement dans ce fichier, il est conseillé, voire requis, de modifier le propriétaire et les droits du fichier. Sous Debian, nous nous trouvons avec un fichier appartenant à root (pourquoi pas…) mais avec les droits 644, ce qui fait que tout le monde peut lire ce fichier. Pensez donc toujours à vérifier le propriétaire et les droits de ce fichier, et modifiez-les le cas échéant, d'autant plus si vous utilisez le mode trust pour lequel les mots de passe apparaissent en clair:

```
debian1:~# chown postgres:postgres /etc/pgbouncer/userlist.txt
debian1:~# chmod 600 /etc/pgbouncer/userlist.txt
```

Il est à savoir qu'il est aussi possible d'utiliser le fichier des utilisateurs de PostgreSQL. Ce fichier n'est jamais modifié manuellement. PostgreSQL le met à jour quand il reçoit une requête d'ajout d'un utilisateur ou de modification de mot de passe. Évidemment, cela suppose que le serveur PostgreSQL se trouve sur le même serveur physique que pgbouncer. L'intérêt est que tout nouvel utilisateur ajouté dans PostgreSQL se trouve immédiatement déclaré au niveau de pgbouncer.

Configuration rapide de pgbouncer

Nous allons autoriser la connexion à deux bases de données, sur deux serveurs différents. Voici le contenu de la section databases:

```
[databases]
base1 = host=debian1 dbname=b1
base2 = host=debian2 dbname=b3
```

Dans la section pgbouncer, nous allons simplement ajouter un utilisateur dans la liste des administrateurs:

```
admin_users = postgres
```

En ce qui concernent les utilisateurs, il nous faut ajouter l'utilisateur postgres dans le fichier `/etc/pgbouncer/userlist.txt`:

```
"postgres" "postgres"
```

Ensuite, il faut ajouter le mot de passe à cet utilisateur:

```
debian1:~# su - postgres
postgres@debian1:~$ psql -c "ALTER USER postgres PASSWORD 'postgres'"
```

Configuration de PostgreSQL

Pour les besoins du test, nous allons créer trois bases de données sur debian1, de nom b1, b2 et b3:

```
debian1:~# su - postgres
postgres@debian1:~$ createdb b1
postgres@debian1:~$ createdb b2
postgres@debian1:~$ createdb b3
```

Nous n'allons en créer qu'une sur debian2:

```
debian2:~# su - postgres
postgres@debian2:~$ createdb b3
```

Comme pgbouncer va se connecter en TCP/IP sur les deux serveurs PostgreSQL, nous devons modifier le fichier de configuration pg_hba.conf pour autoriser les accès. Ce dernier devra contenir les deux lignes suivantes:

```
host    all         all         192.168.10.66/32      trust
host    all         all         192.168.10.67/32      trust
```

Il ne nous reste plus qu'à modifier le fichier postgresql.conf pour que le serveur PostgreSQL écoute sur toutes les interfaces réseau:

```
listen_addresses = '*'
```

Maintenant, nous pouvons redémarrer PostgreSQL:

```
debian1:/etc/postgresql/8.4/main# /etc/init.d/postgresql-8.4 restart
Restarting PostgreSQL 8.4 database server: main.
```

Lancement

Debian dispose d'un script de démarrage. Pour que ce dernier s'exécute au démarrage du serveur, il est nécessaire de faire une légère modification dans le fichier /etc/default/pgbouncer. Il faut passer la variable START à 1:

```
START=1
```

Pour les autres, il est possible de démarrer pgbouncer manuellement de cette façon:

```
debian1:~# pgbouncer -d /etc/pgbouncer/pgbouncer.ini
```

L'option -d indique à pgbouncer de fonctionner en mode démon. Le fichier indiqué est le fichier de configuration avec son chemin complet.

Utilisation

PostgreSQL lancé mais pgbouncer arrêté, nous avons le même soucis qu'avec pgPool-II:

```
debian1:~# su - postgres
postgres@debian1:~$ psql -l
                             Liste des bases de données
   Nom    | Propriétaire | Encodage |     Tri     | Type caract. |    Droits d'accès
----------+--------------+----------+-------------+--------------+-----------------------
 b1       | postgres     | UTF8     | fr_FR.UTF-8 | fr_FR.UTF-8  |
 b2       | postgres     | UTF8     | fr_FR.UTF-8 | fr_FR.UTF-8  |
 b3       | postgres     | UTF8     | fr_FR.UTF-8 | fr_FR.UTF-8  |
 postgres  | postgres     | UTF8     | fr_FR.UTF-8 | fr_FR.UTF-8  |
 template0 | postgres     | UTF8     | fr_FR.UTF-8 | fr_FR.UTF-8  | =c/postgres
                                                                 : postgres=CTc/postgres
 template1 | postgres     | UTF8     | fr_FR.UTF-8 | fr_FR.UTF-8  | =c/postgres
                                                                 : postgres=CTc/postgres
(6 lignes)

postgres@debian1:~$ psql -p 6432 base1
psql: n'a pas pu se connecter au serveur : Aucun fichier ou répertoire de ce type
        Le serveur est-il actif localement et accepte-t-il les connexions sur la
        socket Unix « /var/run/postgresql/.s.PGSQL.6432 » ?
```

Il n'est pas possible de se connecter à base1. Lançons maintenant pgbouncer:

```
postgres@debian1:~$ /usr/sbin/pgbouncer /etc/pgbouncer/pgbouncer.ini
2009-08-29 12:10:50.469 2343 LOG File descriptor limit: 1024 (H:1024), max_client_conn: 100, max fds possible: 110
2009-08-29 12:10:50.480 2343 LOG listening on 127.0.0.1:6432
2009-08-29 12:10:50.483 2343 LOG listening on unix:/var/run/postgresql/.s.PGSQL.6432
```

Attention, tout comme PostgreSQL, pgbouncer refuse de s'exécuter sous le compte root. Le mieux dans ce cas est d'utiliser le compte postgres.

```
postgres@debian1:~$ psql -p 6432 base1
psql (8.4.0)
Saisissez « help » pour l'aide.

base1=#
```

La connexion s'est bien faite. Psql affiche le nom base1 car il utilise ce qui lui a été fourni en ligne de commande, mais en fait nous ne sommes pas connectés à base1 vu que cette base n'existe pas:

```
base1=# \l
                             Liste des bases de données
   Nom    | Propriétaire | Encodage |     Tri     | Type caract. |    Droits d'accès
----------+--------------+----------+-------------+--------------+-----------------------
 b1       | postgres     | UTF8     | fr_FR.UTF-8 | fr_FR.UTF-8  |
 b2       | postgres     | UTF8     | fr_FR.UTF-8 | fr_FR.UTF-8  |
 b3       | postgres     | UTF8     | fr_FR.UTF-8 | fr_FR.UTF-8  |
 postgres  | postgres     | UTF8     | fr_FR.UTF-8 | fr_FR.UTF-8  |
 template0 | postgres     | UTF8     | fr_FR.UTF-8 | fr_FR.UTF-8  | =c/postgres
                                                                 : postgres=CTc/postgres
 template1 | postgres     | UTF8     | fr_FR.UTF-8 | fr_FR.UTF-8  | =c/postgres
                                                                 : postgres=CTc/postgres
(6 lignes)
```

La base base1 n'existe que dans le fichier de configuration de pgbouncer, qui nous a re-routé vers la base db1 de debian1, comme le montre le résultat de la requête suivante:

```
base1=# SELECT datname, procpid, current_query FROM pg_stat_activity ;
 datname | procpid |                       current_query
---------+---------+-------------------------------------------------------------
 b1      |    2416 | select datname, procpid, current_query from pg_stat_activity ;
(1 ligne)
```

Le fichier de trace de pgbouncer l'indique aussi:

```
2009-08-29 12:19:11.687 2414 LOG C-0x95e68e0: base1/postgres@unix:6432 login successful: db=base1 user=postgres
2009-08-29 12:19:11.687 2414 LOG S-0x9603b18: base1/postgres@127.0.0.1:5432 new connection to server
```

La commande ps nous le confirme:

```
postgres@debian1:~$ ps -ef | grep "[b]1"
postgres  2416  2260  0 12:19 ?        00:00:00 postgres: postgres b1 127.0.0.1(38834) idle
```

Nous pouvons lancer toutes les requêtes SQL que nous voulons, elles seront exécutées sur cette base b1:

```
postgres@debian1:~$ psql -p 6432 base1
psql (8.4.0)
Saisissez « help » pour l'aide.

base1=# CREATE TABLE t1 (id integer);
CREATE TABLE
base1=# INSERT INTO t1 SELECT i FROM generate_series(1, 1000) AS i;
INSERT 0 1000
base1=# SELECT * FROM t1 WHERE id BETWEEN 200 AND 205;
 id
-----
 200
 201
```

```
 202
 203
 204
 205
(6 lignes)
```

Remarquez aussi que, si on quitte psql, la connexion est maintenue par pgbouncer:

```
base1=# \q
postgres@debian1:~$ ps -ef | grep "[b]1"
postgres  2416  2260  0 12:19 ?        00:00:00 postgres: postgres b1 127.0.0.1(38834) idle
```

Et si nous nous reconnectons après coup, nous nous retrouvons sur la même connexion:

```
postgres@debian1:~$ psql -p 6432 base1
psql (8.4.0)
Saisissez « help » pour l'aide.

base1=# SELECT datname, procpid, current_query FROM pg_stat_activity ;
 datname | procpid |                       current_query
---------+---------+------------------------------------------------------------------
 b1      |    2416 | select datname, procpid, current_query from pg_stat_activity ;
(1 ligne)
```

Maintenant, connectons-nous à base2:

```
postgres@debian1:~$ psql -p 6432 base2
psql (8.4.0)
Saisissez « help » pour l'aide.

base2=# SELECT datname, procpid, current_query FROM pg_stat_activity ;
 datname | procpid |                       current_query
---------+---------+------------------------------------------------------------------
 b3      |    3954 | select datname, procpid, current_query from pg_stat_activity ;
(1 ligne)
```

Les traces de pgbouncer nous indiquent ceci:

```
2009-08-29 12:37:07.159 2414 LOG C-0x95e68e0: base2/postgres@unix:6432 login successful: db=base2 user=postgres
2009-08-29 12:37:07.160 2414 LOG S-0x9603c08: base2/postgres@192.168.10.67:5432 new connection to server
```

Bien que nous nous soyons connecté sans indiquer de serveur, pgbouncer a vu dans sa configuration que base2 correspond à la base de données b3 sur le serveur debian2 et a lancé la connexion adéquate.

Du coup, on ne voit la connexion que sur debian2:

```
debian2:~# ps -ef | grep "[b]3"
postgres  3954  3865  0 12:39 ?        00:00:00 postgres: postgres b3 192.168.10.66(52401) idle
```

Console d'administration de pgbouncer

L'un des points forts de pgbouncer est la disponibilité d'une console. Pour en profiter, il faut se connecter à la base de données virtuelle pgbouncer en tant qu'un des utilisateurs listés dans les paramètres admin_users et stats_users.

```
postgres@debian1:~$ psql -p 6432 pgbouncer
psql (8.4.0, serveur 8.0/bouncer)
ATTENTION : psql version 8.4, version du serveur 8.0.
        Certaines fonctionnalités de psql pourraient ne pas fonctionner.
Saisissez « help » pour l'aide.

pgbouncer=#
```

Le message d'avertissement est normal. pgbouncer agit comme un serveur PostgreSQL lorsque nous nous connectons à la base pgbouncer. La version utilisée par les développeurs de pgbouncer est justement en 8.0. Nous nous connectons donc avec une version 8.4 de psql sur un serveur 8.0, d'où le message d'avertissement de psql que vous pouvez tranquillement ignorée.

Toutes les fonctionnalités accessibles via cette console sont listées par la commande « SHOW HELP »:

```
pgbouncer=# SHOW HELP;
NOTICE:  Console usage
DÉTAIL :
        SHOW HELP|CONFIG|DATABASES|POOLS|CLIENTS|SERVERS|VERSION
        SHOW STATS|FDS|SOCKETS|ACTIVE_SOCKETS|LISTS|MEM
        SET key = arg
        RELOAD
        PAUSE [<db>]
        SUSPEND
        RESUME [<db>]
        SHUTDOWN
SHOW
```

Nous n'allons pas les voir toutes, mais regardons néanmoins les principales. « SHOW DATABASES » indiquent la liste des bases de données avec leur configuration:

```
pgbouncer=# SHOW DATABASES;
   name    |     host      | port | database  | force_user | pool_size | reserve_pool
-----------+---------------+------+-----------+------------+-----------+--------------
 base1     | 127.0.0.1     | 5432 | b1        |            |        20 |            0
 base2     | 192.168.10.67 | 5432 | b3        |            |        20 |            0
 pgbouncer |               | 6432 | pgbouncer | pgbouncer  |         2 |            0
```

```
(3 lignes)
```

On peut ainsi voir qu'il ne peut pas y avoir plus de deux connexions sur la base pgbouncer et que l'utilisateur est forcé à pgbouncer. « SHOW POOLS » va indiquer la liste des pools configurés et quelques autres informations statistiques:

```
pgbouncer=# SHOW POOLS;
 database  |   user    | cl_active | cl_waiting | sv_active | sv_idle | sv_used | sv_tested | sv_login | maxwait
-----------+-----------+-----------+------------+-----------+---------+---------+-----------+----------+---------
 base1     | postgres  |         2 |          0 |         0 |       0 |       1 |         0 |        0 |       0
 base2     | postgres  |         1 |          0 |         0 |       0 |       1 |         0 |        0 |       0
 pgbouncer | pgbouncer |         1 |          0 |         0 |       0 |       0 |         0 |        0 |       0
(3 lignes)
```

Les statistiques indiquent notamment le nombre de clients actifs et en attente, le nombre de connexions serveur actives, en attente, utilisées, et ainsi de suite.

```
pgbouncer=# SHOW CLIENTS;
 type |   user   | database  | state  | addr | port | local_addr | local_port |    connect_time     |    request_time     |    ptr    | link
------+----------+-----------+--------+------+------+------------+------------+---------------------+---------------------+-----------+------
 C    | postgres | base1     | active | unix | 6432 | unix       |       6432 | 2009-08-29 13:34:58 | 2009-08-29 13:34:58 | 0x970f900 |
 C    | postgres | base1     | active | unix | 6432 | unix       |       6432 | 2009-08-29 13:35:15 | 2009-08-29 13:35:15 | 0x970f9f0 |
 C    | postgres | base2     | active | unix | 6432 | unix       |       6432 | 2009-08-29 13:35:29 | 2009-08-29 13:35:29 | 0x970fae0 |
 C    | postgres | pgbouncer | active | unix | 6432 | unix       |       6432 | 2009-08-29 13:35:47 | 2009-08-29 13:39:45 | 0x970fbd0 |
(4 lignes)
```

La commande « SHOW CLIENTS » affichent la liste des clients connectés à pgbouncer. La dernière que nous allons montrer est « SHOW STATS »:

```
pgbouncer=# SHOW STATS;
 database  | total_requests | total_received | total_sent | total_query_time | avg_req | avg_recv | avg_sent | avg_query
-----------+----------------+----------------+------------+------------------+---------+----------+----------+-----------
 base1     |              0 |              0 |          0 |                0 |       0 |        0 |        0 |         0
 base2     |              2 |             34 |        125 |            53994 |       0 |        0 |        1 |     26997
 pgbouncer |              1 |              0 |          0 |                0 |       0 |        0 |        0 |         0
(3 lignes)
```

Elle affiche des statisques sur les différentes bases: nombre de requêtes, temps d'exécution, etc. Cela peut fournir des informations capitales sur l'utilisation des connexions via pgbouncer.

Sachez qu'il est aussi possible de contrôler pgbouncer à partir de cette console. Vous avez accès aux actions suivantes:

- PAUSE, pour que pgbouncer tente une déconnexion de tous les serveurs PostgreSQL, une fois que les requêtes ont terminées leur exécution;
- SUSPEND, pour vider tous les tampons des sockets et pour que pgbouncer arrête d'écouter les sockets;
- RESUME, pour recommencer le travail après une commande PAUSE ou SUSPEND;
- SHUTDOWN, pour arrêter tout simplement le processus pgbouncer;
- RELOAD, pour demander au processus de recharger son fichier de configuration.

Quelques astuces

Contrairement à pgPool-II, pgbouncer ne propose pas de répartition de charge. Le seul moyen d'en avoir un est d'utiliser un répartiteur de charge de connexion TCP, comme LVS ou HAProxy.

De même, il est impossible de se connecter en SSL avec pgbouncer. Néanmoins, l'outil Stunnel gère le protocole PostgreSQL depuis la version 4.27. N'hésitez pas à le tester.

Conclusion

Bien que pgbouncer fasse beaucoup moins de choses que pgPool-II, cela ne l'empêche pas d'être un outil très intéressant. Sa console d'administration permet d'obtenir de nombreuses statistiques très intéressantes, autorisant du coup une surveillance très poussée de cet outil.

Le fait qu'il soit capable de gérer des connexions vers différents serveurs est un plus très agréable. Cela se révèle très utile quand on dispose de deux serveurs PostgreSQL, une moitié des bases dont le maître est le serveur 1 et l'autre moitié le serveur 2. Le switchover et le failover s'en trouvent facilités.

Encore une fois, la documentation est très restreinte, mais il faut bien avouer que pour un produit comme pgbouncer, cela a relativement peu d'importance.