Londiste, la réplication vue par Skype

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.

Londiste est un autre système de réplication asynchrone basé sur les triggers. Son installation est un peu particulière, mais pas désagréable du tout. Elle est même bien plus simple que l'installation de Slony. Créé par Skype en Python pour ses besoins propres, il est disponible depuis peu de temps mais progresse bien et mérite sa place dans ce hors-série.

Installation de Londiste

Londiste faisant parti des Skytools, il faut installer ce paquet. Comme nous l'avons vu dans l'article sur le Log Shipping, nous allons utiliser le paquet disponible pour squeeze (première version Debian à avoir ce paquet). Skytools est constitué de deux paquets sous Debian : skytools et skytools-modules. Ce dernier est spécifique à la version installée de PostgreSQL. Il n'existe pas encore à ma connaissance de modules pour 8.4. Nous allons donc utiliser un PostgreSQL version 8.3, et utiliser les modules Skytools de cette version.

```
debian1:~$ wget -q http://ftp.fr.debian.org/debian/pool/main/s/skytools/skytools_2.1.8-2_i386.deb
debian1:~$ wget -q http://ftp.fr.debian.org/debian/pool/main/s/skytools/skytools-modules-8.3_2.1.8-2_i386.deb
debian1:~$ aptitude install python-psycopg2
[… différents messages de progression …]
debian1:~$ dpkg -i skytools_2.1.8-2_i386.deb skytools-modules-8.3_2.1.8-2_i386.deb
[… différents messages de progression …]
```

Pour que la réplication fonctionne, il n'est pas nécessaire de les installer sur le serveur debian2. Cependant, l'un des buts d'un système de réplication étant de pouvoir basculer sur l'esclave si nécessaire, il peut être intéressant d'installer immédiatement les outils sur l'esclave.

Configuration de PostgreSQL

Les démons pgqadm et londiste vont être exécutés en tant qu'utilisateur Unix postgres et vont se connecter à la base de données base1, en tant qu'utilisateur PostgreSQL postgres, à partir du serveur debian1 vers les serveurs debian1 et debian2. Il nous faut donc configurer PostgreSQL pour qu'il accepte les connexions sur l'interface réseau de debian1 et de debian2 et pour qu'il accepte l'authentification de l'utilisateur postgres. Le premier fichier à configurer est postgresql.conf dans le répertoire /etc/postgresql/8.3/main. Il faut changer le paramètre listen_addresses pour que sa valeur soit '*'. Le deuxième fichier à modifier est pg_hba.conf pour permettre les connexions à partir de debian1 et de debian2. Il nous faut donc ajouter les deux lignes suivantes:

```
host    all         postgres    192.168.0.8/32        md5
host    all         postgres    192.168.0.10/32       md5
```

Nous pouvons redémarrer PostgreSQL:

```
debian1:~# /etc/init.d/postgresql-8.3 restart
Restarting PostgreSQL 8.3 database server: main.
```

Nous devons ensuite donner un mot de passe à l'utilisateur PostgreSQL postgres:

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

Enfin, nous allons créer le fichier .pgpass dans le répertoire personnel de l'utilisateur postgres afin que ce dernier n'ait pas à saisir son mot de passe pour la connexion à la base base1 de debian1 et de debian2.

```
postgres@debian1:~$ cat <<_EOF_ >>~/.pgpass
> debian1:5432:base1:postgres:postgres
> debian2:5432:base1:postgres:postgres
> _EOF_
postgres@debian1:~$ chmod 600 ~/.pgpass
```

Effectuez toute la configuration de ce chapitre pour le serveur debian2.

Testez la connexion depuis debian1 au serveur debian1 et debian2. Faites de même à partir de debian2. Si tout se passe bien, vous devriez pouvoir vous connecter à la base base1 des deux serveurs sans avoir à saisir de mot de passe. Si ce n'est pas le cas, vous devez corriger le problème avant de continuer.

Mise en place du PGQ

PGQ est l'acronyme de PostGresql Queue, un système de gestion de queue créé par Skype. Ce système va récupérer les modifications effectuées sur une base dans un journal (une table d'un schéma). Donc un équivalent des tables sl_log_1 et sl_log_2 de Slony.

Commençons par son paramétrage. Nous allons copier le fichier de configuration exemple dans le répertoire de configuration de PostgreSQL:

```
debian1:~$ cp /usr/share/doc/skytools/conf/pgqadm.ini /etc/postgresql/8.3/main
```

Puis nous allons le modifier pour obtenir ce fichier:

```
[pgqadm]

# should be globally unique
job_name = pgqadm_replication

db = dbname=base1 host=debian1

# how often to run maintenance [minutes]
maint_delay_min = 5
```

```
# how often to check for activity [secs]
loop_delay = 0.1

logfile = ~/log/%(job_name)s.log
pidfile = ~/pid/%(job_name)s.pid

use_skylog = 0
```

Le paramètre job_name peut être configuré suivant votre convenance. Le paramètre db indique dans quelle base sera créé le système de gestion de queues. Cela veut dire l'ajout du langage PL/pgsql, l'ajout du schéma pgqadm et de ces tables et séquences. Remarquez que nous n'allons pas utiliser un utilisateur particulier pour les connexions. Il pourrait être intéressant de le faire. Ce n'est ni recommandé ni déconseillé, donc nous ne le ferons pas, mais sachez néanmoins que c'est possible.

Vu que le démon va écrire les fichiers de trace dans le sous-répertoire log et le fichier PID dans le sous-répertoire pid du répertoire personnel de l'utilisateur postgres, il nous faut créer ces deux répertoires.

```
debian1:~# su – postgres
postgres@debian1:~$ mkdir ~/log ~/pid
```

À cause d'un bug du paquet Debian (voir http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=534310 pour les détails), il nous faut modifier le fichier /usr/lib/python2.5/site-packages/skytools/installer_config.py. En effet, ce fichier Python cherche des scripts SQL dans /usr/share/skytools alors qu'ils se trouvent dans /usr/share/postgresql/8.3/contrib. J'avoue ne pas avoir trouvé d'autres moyens que la modification de ce fichier.

```
postgres@debian1:~$ exit
logout
debian1:~$ cat <<_EOF_ >/usr/lib/python2.5/site-packages/skytools/installer_config.py
>
> sql_locations = [
>    "/usr/share/postgresql/8.3/contrib",
> ]
> _EOF_
```

Maintenant, nous pouvons installer le système PGQ:

```
debian1:~$ su – postgres
postgres@debian1:~$ pgqadm /etc/postgresql/8.3/main/pgqadm.ini install
2009-08-14 19:35:28,836 5200 INFO Installing plpgsql
2009-08-14 19:35:28,855 5200 INFO txid_current_snapshot is installed
2009-08-14 19:35:28,858 5200 INFO Installing pgq
2009-08-14 19:35:28,858 5200 INFO   Reading from /usr/share/postgresql/8.3/contrib/pgq.sql
```

L'action install de pgqadm commence par activer le langage PL/pgsql si ce dernier ne l'est pas déjà. Il vérifie que la fonction txid_current_snapshot existe, et l'installe dans le cas contraire. Enfin, il exécute le script pgq.sql pour créer le schéma pgq et ces différentes tables et séquences.

Il ne nous reste plus qu'à lancer le démon avec la commande suivante:

```
postgres@debian1:~$ pgqadm /etc/postgresql/8.3/main/pgqadm.ini ticker -d
```

Mise en place de londiste

Là-aussi, nous allons commencer par son paramétrage. Nous allons copier le fichier de configuration exemple dans le répertoire de configuration de PostgreSQL:

```
debian1:~$ cp /usr/share/doc/skytools/conf/londiste.ini /etc/postgresql/8.3/main
```

Nous allons ensuite le modifier pour obtenir ceci:

```
[londiste]

# should be unique
job_name = replication_base1

# source queue location
provider_db = dbname=base1 host=debian1

# target database - it's preferable to run "londiste replay"
# on same machine and use unix-socket or localhost to connect
subscriber_db = dbname=base1 host=debian2

# source queue name
pgq_queue_name = londiste.replication

logfile = ~/log/%(job_name)s.log
pidfile = ~/pid/%(job_name)s.pid

# how often to poll event from provider
#loop_delay = 1

# max locking time on provider (in seconds, float)
#lock_timeout = 10.0
```

Le paramètre job_name peut être configuré suivant votre convenance. Le paramètre provider_db indique la base à répliquer alors que le paramètre subscriber_db précise où la répliquer.

Nous pouvons dès maintenant installer londiste sur le fournisseur et sur l'abonné:

```
debian1:~$ su – postgres
postgres@debian1:~$ londiste /etc/postgresql/8.3/main/londiste.ini provider install
2009-08-15 01:49:18,000 2184 INFO plpgsql is installed
2009-08-15 01:49:18,006 2184 INFO txid_current_snapshot is installed
2009-08-15 01:49:18,008 2184 INFO pgq is installed
2009-08-15 01:49:18,009 2184 INFO Installing londiste
2009-08-15 01:49:18,016 2184 INFO   Reading from /usr/share/postgresql/8.3/contrib/londiste.sql
postgres@debian1:~$ londiste /etc/postgresql/8.3/main/londiste.ini subscriber install
2009-08-15 01:49:41,430 2188 INFO Installing plpgsql
2009-08-15 01:49:41,538 2188 INFO Installing londiste
2009-08-15 01:49:41,539 2188 INFO   Reading from /usr/share/postgresql/8.3/contrib/londiste.sql
```

Ça ressemble beaucoup à l'installation de PGQ. Le script SQL est différent, mais ces actions correspondent aussi en gros: création d'un schéma cette fois appelé londiste, ajout de tables et de séquences.

Il est possible de lancer le démon londiste dès maintenant:

```
postgres@debian1:~$ londiste /etc/postgresql/8.3/main/londiste.ini replay -d
postgres@debian1:~$ ps -ef | grep [l]ondiste
postgres  2197     1  0 01:51 ?        00:00:00 /usr/bin/python /usr/bin/londiste /etc/postgresql/8.3/main/londiste.ini replay -d
```

Il est bien en cours d'exécution.

Si vous regardez les traces générées par le démon, vous verrez qu'il ne fait pas grand chose actuellement. En effet, nous n'avons déclaré aucune table à répliquer.

Créons une table et demandons sa réplication:

```
postgres@debian1:~$ psql -q base1
```

```
base1=# CREATE TABLE t1 (id serial primary key, texte text);
NOTICE:  CREATE TABLE créera des séquences implicites « t1_id_seq » pour la colonne serial « t1.id »
NOTICE:  CREATE TABLE / PRIMARY KEY créera un index implicite « t1_pkey » pour la table « t1 »
base1=# INSERT INTO t1 (texte) SELECT 'Ligne '||i::text FROM generate_series(1, 10000) AS i;
base1=# \q
postgres@debian1:~$ londiste /etc/postgresql/8.3/main/londiste.ini provider add public.t1
2009-08-15 02:01:36,075 2272 INFO Adding public.t1
```

Remarquez que nous indiquons le schéma dans la dénomination de la table. Si vous indiquez seulement le nom de la table, Londiste ajoute automatiquement le schéma public devant. Il ne prend donc pas en considération la valeur de search_path pour trouver dans quel schéma est réellement stockée la table.

Tout ce que cette commande fait est que les modifications de cette table sont prêtes à être journalisées dès qu'un abonné sera présent. Nous allons donc abonner debian2 à cette journalisation. Dans un premier temps, nous devons créer la même table sur base1 de debian2:

```
postgres@debian1:~$ psql -q -h debian2 base1
base1=# CREATE TABLE t1 (id serial primary key, texte text);
NOTICE:  CREATE TABLE créera des séquences implicites « t1_id_seq » pour la colonne serial « t1.id »
NOTICE:  CREATE TABLE / PRIMARY KEY créera un index implicite « t1_pkey » pour la table « t1 »
base1=# INSERT INTO t1 (texte) SELECT 'Ligne '||i::text FROM generate_series(1, 10000) AS i;
base1=# \q
postgres@debian1:~$ londiste /etc/postgresql/8.3/main/londiste.ini subscriber add public.t1
2009-08-15 02:05:16,861 2291 INFO Checking public.t1
2009-08-15 02:05:16,894 2291 INFO Adding public.t1
```

Cela ne veut pas dire que les données sont déjà présentes sur debian2. Il faudra un petit laps de temps.

```
postgres@debian1:~$ londiste /etc/postgresql/8.3/main/londiste.ini subscriber tables
Table                          State
public.t1                          -
```

Le transfert ici n'a toujours pas commencé.

```
postgres@debian1:~$ londiste /etc/postgresql/8.3/main/londiste.ini subscriber tables
Table                          State
public.t1                      in-copy
```

La copie est en cours.

```
postgres@debian1:~$ londiste /etc/postgresql/8.3/main/londiste.ini subscriber tables
Table                          State
public.t1                         ok
```

La copie est terminée. Ce qu'indiquent d'ailleurs les traces:

```
2009-08-15 02:05:16,861 2291 INFO Checking public.t1
2009-08-15 02:05:16,894 2291 INFO Adding public.t1
2009-08-15 02:05:52,819 2197 INFO storing state of public.t1: copy:0 new_state:in-copy
2009-08-15 02:05:52,832 2197 INFO Table public.t1 status changed to 'in-copy'
2009-08-15 02:05:52,833 2197 INFO Launching copy process
2009-08-15 02:05:53,130 2297 INFO Resetting queue tracking on dst side
2009-08-15 02:06:52,493 2297 INFO Starting full copy of public.t1
2009-08-15 02:06:52,571 2297 INFO Dropping t1_pkey
2009-08-15 02:06:52,577 2297 INFO public.t1: truncating
2009-08-15 02:06:52,593 2297 INFO public.t1: start copy
2009-08-15 02:06:52,765 2297 INFO public.t1: copy finished: 157788 bytes, 10000 rows
2009-08-15 02:06:52,767 2297 INFO Creating t1_pkey
2009-08-15 02:06:52,826 2297 INFO storing state of public.t1: copy:1 new_state:catching-up
2009-08-15 02:06:56,994 2297 INFO storing state of public.t1: copy:1 new_state:wanna-sync:44
2009-08-15 02:06:56,998 2297 INFO Table public.t1 status changed to 'wanna-sync:44'
2009-08-15 02:06:59,625 2197 INFO storing state of public.t1: copy:0 new_state:do-sync:45
2009-08-15 02:06:59,637 2197 INFO Table public.t1 status changed to 'do-sync:45'
2009-08-15 02:07:00,085 2297 INFO storing state of public.t1: copy:1 new_state:ok
2009-08-15 02:07:00,089 2297 INFO Table public.t1 status changed to 'ok'
2009-08-15 02:07:00,120 2297 INFO Resetting queue tracking on dst side
2009-08-15 02:07:00,129 2297 INFO got SystemExit(0), exiting
```

Ce qui est somme toute assez compréhensible. N'hésitez pas à jeter un œil aux traces de Slony pour voir la différence, c'est malheureusement le jour et la nuit.

La procédure avec Slony réclame de créer un nouveau set avec la nouvelle table, d'abonner le nœud esclave à ce nouveau set, puis de fusionner l'ancien et le nouveau set. Bref, très lourd. Avec Londiste, rien de tout ça: déclaration de la table au fournisseur, puis à l'abonné… et c'est terminé !

Répliquer facilement une grosse base

Nous allons prendre comme exemple la base du forum fluxbb de PostgreSQL (disponible sur http://forums.postgresql.fr).

L'installation de londiste est identique à ce qui est indiqué ci-dessus, je n'en reparlerai donc pas. La configuration de PostgreSQL est pratiquement identique. Le seul point de différence est le nom de la base de données. Il ne s'agit plus de base1, mais de fluxbb. Le contenu du fichier ~postgres/.pgpass devient donc:

```
debian1:5432:fluxbb:postgres:postgres
debian2:5432:fluxbb:postgres:postgres
```

Nous allons ajouter le schéma de la base fluxbb de debian1 sur la base de même nom sur debian2:

```
postgres@debian1:~$ pg_dump -s -n public fluxbb | psql -h debian2 fluxbb
SET
SET
SET
SET
SET
SET
SET
SET
CREATE TABLE
ALTER TABLE
CREATE SEQUENCE
ALTER TABLE
ALTER SEQUENCE
CREATE TABLE
ALTER TABLE
[… suite des messages de progression ...]
```

La mise en place de PGQ est aussi identique à ce qui est indiqué ci-dessus, je ne reviendrai donc pas là-dessus. Attention à bien renommer la base de données dans le paramètre db.

Par contre, nous allons reprendre toute la mise en place de Londiste. C'est parti.

Commençons par copier le fichier de configuration exemple dans le répertoire de configuration de PostgreSQL en lui donnant un nom spécifique à notre réplication:

```
debian1:~$ cp /usr/share/doc/skytools/conf/londiste.ini /etc/postgresql/8.3/main/londiste_fluxbb.ini
```

Nous allons ensuite le modifier pour obtenir ceci:

```
[londiste]

# should be unique
job_name = replication_fluxbb

# source queue location
provider_db = dbname=fluxbb host=debian1

# target database - it's preferable to run "londiste replay"
# on same machine and use unix-socket or localhost to connect
subscriber_db = dbname=fluxbb host=debian2

# source queue name
pgq_queue_name = londiste.replication_fluxbb

logfile = ~/log/%(job_name)s.log
pidfile = ~/pid/%(job_name)s.pid

# how often to poll event from provider
#loop_delay = 1

# max locking time on provider (in seconds, float)
#lock_timeout = 10.0
```

Nous installons londiste sur le fournisseur et sur l'abonné:

```
postgres@debian1:~$ londiste /etc/postgresql/8.3/main/londiste_fluxbb.ini provider install
2009-08-15 10:13:50,414 4770 INFO plpgsql is installed
2009-08-15 10:13:50,421 4770 INFO txid_current_snapshot is installed
2009-08-15 10:13:50,424 4770 INFO pgq is installed
2009-08-15 10:13:50,426 4770 INFO Installing londiste
2009-08-15 10:13:50,427 4770 INFO   Reading from /usr/share/postgresql/8.3/contrib/londiste.sql
postgres@debian1:~$ londiste /etc/postgresql/8.3/main/londiste_fluxbb.ini subscriber install
2009-08-15 10:13:59,215 4772 INFO Installing plpgsql
2009-08-15 10:13:59,239 4772 INFO Installing londiste
2009-08-15 10:13:59,240 4772 INFO   Reading from /usr/share/postgresql/8.3/contrib/londiste.sql
```

Nous lançons le démon londiste:

```
postgres@debian1:~$ londiste /etc/postgresql/8.3/main/londiste_fluxbb.ini replay -d
```

Nous devons maintenant ajouter toutes les tables de la base fluxbb dans le système de réplication. Plutôt que d'ajouter chaque table une par une, nous allons les récupérer via une requête SQL qui cherchera les informations dans le schéma information_schema (schéma faisant parti du standard SQL) et nous allons passer chaque nom de table à l'outil londiste pour qu'il l'intègre dans la configuration du fournisseur:

```
postgres@debian1:~$ requete="select table_schema||'.'||table_name from information_schema.tables where table_schema NOT IN ('pg_catalog', 'information_schema', 'pgq', 'londiste') ORDER BY 1;"
postgres@debian1:~$ psql -Atc "$requete" fluxbb
public.fbb_bans
public.fbb_categories
public.fbb_censoring
public.fbb_config
public.fbb_forum_perms
public.fbb_forums
public.fbb_groups
public.fbb_online
public.fbb_posts
public.fbb_ranks
public.fbb_reports
public.fbb_search_cache
public.fbb_search_matches
public.fbb_search_words
public.fbb_subscriptions
public.fbb_topics
public.fbb_users
```

La requête fonctionne bien. Nous pouvons envoyer le résultat à londiste.

```
postgres@debian1:~$ psql -Atc "$requete" fluxbb | xargs londiste /etc/postgresql/8.3/main/londiste_fluxbb.ini provider add
2009-08-15 10:54:45,089 4901 INFO Adding public.fbb_bans
2009-08-15 10:54:45,120 4901 INFO Adding public.fbb_categories
2009-08-15 10:54:45,130 4901 INFO Adding public.fbb_censoring
2009-08-15 10:54:45,142 4901 INFO Adding public.fbb_config
2009-08-15 10:54:45,153 4901 INFO Adding public.fbb_forum_perms
2009-08-15 10:54:45,166 4901 INFO Adding public.fbb_forums
2009-08-15 10:54:45,178 4901 INFO Adding public.fbb_groups
2009-08-15 10:54:45,189 4901 INFO Adding public.fbb_online
Traceback (most recent call last):
  File "/usr/bin/londiste", line 134, in <module>
    script.start()
  File "/usr/bin/londiste", line 97, in start
    self.script.start()
  File "/usr/lib/python2.5/site-packages/skytools/scripting.py", line 372, in start
    run_single_process(self, self.go_daemon, self.pidfile)
  File "/usr/lib/python2.5/site-packages/skytools/scripting.py", line 96, in run_single_process
    runnable.run()
  File "/usr/lib/python2.5/site-packages/londiste/setup.py", line 73, in run
    self.admin()
  File "/usr/lib/python2.5/site-packages/londiste/setup.py", line 166, in admin
    self.provider_add_tables(self.args[3:])
  File "/usr/lib/python2.5/site-packages/londiste/setup.py", line 247, in provider_add_tables
    self.provider_add_table(tbl)
  File "/usr/lib/python2.5/site-packages/londiste/setup.py", line 271, in provider_add_table
    self.exec_provider(q, [self.pgq_queue_name, tbl])
  File "/usr/lib/python2.5/site-packages/londiste/setup.py", line 303, in exec_provider
    src_curs.execute(sql, args)
  File "/usr/lib/python2.5/site-packages/psycopg2/extras.py", line 88, in execute
    return _cursor.execute(self, query, vars, async)
psycopg2.InternalError: ERREUR:  need key column
CONTEXT:  PL/pgSQL function "provider_add_table" line 2 at RETURN
```

Aïe. Nous avons oublié de vérifier que chaque table disposait d'une clé primaire. Et à priori, la table fbb_online n'en a pas (la table du dernier message « Adding ». Nous allons utiliser une requête SQL pour connaître toutes les tables qui n'ont pas de clé primaire.

```
postgres@debian1:~$ psql -q fluxbb
fluxbb=# SELECT relname FROM pg_class
fluxbb-# WHERE relkind='r' AND NOT relhaspkey
fluxbb-# AND relnamespace=2200;
      relname
--------------------
 fbb_online
 fbb_search_matches
(2 lignes)
```

Le filtre permet de ne récupérer que les tables (« relkind='r' », r pour relation, un mot généralement employé en anglais pour une table) qui n'ont pas de clé primaire (« NOT relhaspkey ») mais qui font partie du schéma public (« relnamespace=2200 »). Nous filtrons sur le schéma public car toutes les tables de cette base se trouvent dans ce schéma. Dans le cas contraire, il aurait fallu ajouter les identifiants des autres schémas à incorporer (voir le catalogue

système pg_namespace).

Voyons les déclarations des tables fbb_online et fbb_search_matches, seules tables n'ayant pas de clé primaire.

```
fluxbb=# \d fbb_online
                Table « public.fbb_online »
 Colonne |          Type          |              Modificateurs
---------+------------------------+-----------------------------------------
 user_id | integer                | not null default 1
 ident   | character varying(200) | not null default ''::character varying
 logged  | integer                | not null default 0
 idle    | smallint               | not null default 0
Index :
    « fbb_online_user_id_idx » btree (user_id)

fluxbb=# \d fbb_search_matches
      Table « public.fbb_search_matches »
    Colonne    |   Type   |   Modificateurs
---------------+----------+--------------------
 post_id       | integer  | not null default 0
 word_id       | integer  | not null default 0
 subject_match | smallint | not null default 0
Index :
    « fbb_search_matches_post_id_idx » btree (post_id)
    « fbb_search_matches_word_id_idx » btree (word_id)
```

Il est donc possible de leur ajouter une colonne id, qu'on déclarera comme clé primaire.

```
fluxbb=# ALTER TABLE fbb_online ADD COLUMN id serial PRIMARY KEY;
NOTICE:  ALTER TABLE créera des séquences implicites « fbb_online_id_seq » pour la colonne serial « fbb_online.id »
NOTICE:  ALTER TABLE / ADD PRIMARY KEY créera un index implicite « fbb_online_pkey » pour la table « fbb_online »
fluxbb=# ALTER TABLE fbb_search_matches ADD COLUMN id serial PRIMARY KEY;
NOTICE:  ALTER TABLE créera des séquences implicites « fbb_search_matches_id_seq » pour la colonne serial « fbb_search_matches.id »
NOTICE:  ALTER TABLE / ADD PRIMARY KEY créera un index implicite « fbb_search_matches_pkey » pour la table « fbb_search_matches »
```

Parfait, recommençons l'ajout des tables.

```
postgres@debian1:~$ psql -Atc "$requete" fluxbb | xargs londiste /etc/postgresql/8.3/main/londiste_fluxbb.ini provider add
2009-08-15 18:29:40,173 5941 INFO Adding public.fbb_online
2009-08-15 18:29:40,217 5941 INFO Adding public.fbb_posts
2009-08-15 18:29:40,228 5941 INFO Adding public.fbb_ranks
2009-08-15 18:29:40,242 5941 INFO Adding public.fbb_reports
2009-08-15 18:29:40,253 5941 INFO Adding public.fbb_search_cache
2009-08-15 18:29:40,264 5941 INFO Adding public.fbb_search_matches
2009-08-15 18:29:40,277 5941 INFO Adding public.fbb_search_words
2009-08-15 18:29:40,289 5941 INFO Adding public.fbb_subscriptions
2009-08-15 18:29:40,299 5941 INFO Adding public.fbb_topics
2009-08-15 18:29:40,313 5941 INFO Adding public.fbb_users
```

Remarquez qu'il ne dit rien lorsque nous essayons d'ajouter une table déjà présente. Il s'est contenté d'ajouter les tables manquantes.

Maintenant, abonnons debian2 à toutes les tables dont debian1 est fournisseur. Pour cela, nous allons récupérer toutes les tables que fournit debian1 (argument « provider tables » de londiste), que nous passerons à londiste avec l'argument « subscriber add ».

```
postgres@debian1:~$ londiste /etc/postgresql/8.3/main/londiste_fluxbb.ini provider tables | xargs londiste /etc/postgresql/8.3/main/londiste_fluxbb.ini subscriber add
2009-08-15 18:30:57,050 5953 INFO Checking public.fbb_censoring
2009-08-15 18:30:57,097 5953 INFO Checking public.fbb_categories
2009-08-15 18:30:57,116 5953 INFO Checking public.fbb_subscriptions
2009-08-15 18:30:57,128 5953 INFO Checking public.fbb_search_matches
2009-08-15 18:30:57,151 5953 ERROR subscriber table public.fbb_search_matches has no primary key (vvv)
2009-08-15 18:30:57,152 5953 INFO Checking public.fbb_topics
2009-08-15 18:30:57,167 5953 INFO Checking public.fbb_search_words
2009-08-15 18:30:57,182 5953 INFO Checking public.fbb_forum_perms
2009-08-15 18:30:57,202 5953 INFO Checking public.fbb_reports
2009-08-15 18:30:57,217 5953 INFO Checking public.fbb_online
2009-08-15 18:30:57,235 5953 ERROR subscriber table public.fbb_online has no primary key (vvvv)
2009-08-15 18:30:57,236 5953 INFO Checking public.fbb_bans
2009-08-15 18:30:57,257 5953 INFO Checking public.fbb_users
2009-08-15 18:30:57,282 5953 INFO Checking public.fbb_forums
2009-08-15 18:30:57,313 5953 INFO Checking public.fbb_posts
2009-08-15 18:30:57,335 5953 INFO Checking public.fbb_ranks
2009-08-15 18:30:57,361 5953 INFO Checking public.fbb_config
2009-08-15 18:30:57,378 5953 INFO Checking public.fbb_search_cache
2009-08-15 18:30:57,397 5953 INFO Checking public.fbb_groups
```

Re-aïe. Nous n'avons ajouté les clés primaires que sur debian1. Il faut aussi les mettre sur debian2. En effet, tout comme Slony, Londiste ne propage pas les modifications de schéma. Ajoutons donc ses deux clés sur les tables de la cible:

```
postgres@debian1:~$ psql -q -h debian2 fluxbb
fluxbb=# ALTER TABLE fbb_online ADD COLUMN id serial PRIMARY KEY;
NOTICE:  ALTER TABLE créera des séquences implicites « fbb_online_id_seq » pour la colonne serial « fbb_online.id »
NOTICE:  ALTER TABLE / ADD PRIMARY KEY créera un index implicite « fbb_online_pkey » pour la table « fbb_online »
fluxbb=# ALTER TABLE fbb_search_matches ADD COLUMN id serial PRIMARY KEY;
NOTICE:  ALTER TABLE créera des séquences implicites « fbb_search_matches_id_seq » pour la colonne serial « fbb_search_matches.id »
NOTICE:  ALTER TABLE / ADD PRIMARY KEY créera un index implicite « fbb_search_matches_pkey » pour la table « fbb_search_matches »
```

Maintenant, c'est bon. Pour savoir si le précédent appel à londiste a ajouté déjà quelques tables, nous allons utiliser l'argument « subscriber missing » de londiste pour connaître toutes les tables manquantes.

```
postgres@debian1:~$ londiste /etc/postgresql/8.3/main/londiste_fluxbb.ini subscriber missing
Table: public.fbb_bans
Table: public.fbb_categories
Table: public.fbb_censoring
Table: public.fbb_config
Table: public.fbb_forum_perms
Table: public.fbb_forums
Table: public.fbb_groups
Table: public.fbb_online
Table: public.fbb_posts
Table: public.fbb_ranks
Table: public.fbb_reports
Table: public.fbb_search_cache
Table: public.fbb_search_matches
Table: public.fbb_search_words
Table: public.fbb_subscriptions
Table: public.fbb_topics
Table: public.fbb_users
```

Autrement dit, aucune n'a été ajouté à l'abonnement. Nous allons recommencer:

```
postgres@debian1:~$ londiste /etc/postgresql/8.3/main/londiste_fluxbb.ini provider tables | xargs londiste /etc/postgresql/8.3/main/londiste_fluxbb.ini subscriber add
2009-08-15 18:33:23,667 5967 INFO Checking public.fbb_censoring
2009-08-15 18:33:23,697 5967 INFO Checking public.fbb_categories
2009-08-15 18:33:23,714 5967 INFO Checking public.fbb_subscriptions
2009-08-15 18:33:23,731 5967 INFO Checking public.fbb_search_matches
2009-08-15 18:33:23,750 5967 INFO Checking public.fbb_topics
2009-08-15 18:33:23,768 5967 INFO Checking public.fbb_search_words
```

```
2009-08-15 18:33:23,785 5967 INFO Checking public.fbb_forum_perms
2009-08-15 18:33:23,800 5967 INFO Checking public.fbb_reports
2009-08-15 18:33:23,823 5967 INFO Checking public.fbb_online
2009-08-15 18:33:23,841 5967 INFO Checking public.fbb_bans
2009-08-15 18:33:23,857 5967 INFO Checking public.fbb_users
2009-08-15 18:33:23,881 5967 INFO Checking public.fbb_forums
2009-08-15 18:33:23,897 5967 INFO Checking public.fbb_posts
2009-08-15 18:33:23,914 5967 INFO Checking public.fbb_ranks
2009-08-15 18:33:23,930 5967 INFO Checking public.fbb_config
2009-08-15 18:33:23,947 5967 INFO Checking public.fbb_search_cache
2009-08-15 18:33:23,975 5967 INFO Checking public.fbb_groups
2009-08-15 18:33:24,001 5967 INFO Adding public.fbb_censoring
2009-08-15 18:33:24,022 5967 INFO Adding public.fbb_categories
2009-08-15 18:33:24,037 5967 INFO Adding public.fbb_subscriptions
2009-08-15 18:33:24,059 5967 INFO Adding public.fbb_search_matches
2009-08-15 18:33:24,077 5967 INFO Adding public.fbb_topics
2009-08-15 18:33:24,088 5967 INFO Adding public.fbb_search_words
2009-08-15 18:33:24,097 5967 INFO Adding public.fbb_forum_perms
2009-08-15 18:33:24,112 5967 INFO Adding public.fbb_reports
2009-08-15 18:33:24,122 5967 INFO Adding public.fbb_online
2009-08-15 18:33:24,131 5967 INFO Adding public.fbb_bans
2009-08-15 18:33:24,145 5967 INFO Adding public.fbb_users
2009-08-15 18:33:24,155 5967 INFO Adding public.fbb_forums
2009-08-15 18:33:24,164 5967 INFO Adding public.fbb_posts
2009-08-15 18:33:24,173 5967 INFO Adding public.fbb_ranks
2009-08-15 18:33:24,185 5967 INFO Adding public.fbb_config
2009-08-15 18:33:24,195 5967 INFO Adding public.fbb_search_cache
2009-08-15 18:33:24,204 5967 INFO Adding public.fbb_groups
```

Peu après, le transfert commence pour la première table:

```
postgres@debian1:~$ londiste /etc/postgresql/8.3/main/londiste_fluxbb.ini subscriber tables
Table                           State
public.fbb_censoring            in-copy
public.fbb_categories           -
public.fbb_subscriptions        -
public.fbb_search_matches       -
public.fbb_topics               -
public.fbb_search_words         -
public.fbb_forum_perms          -
public.fbb_reports              -
public.fbb_online               -
public.fbb_bans                 -
public.fbb_users                -
public.fbb_forums               -
public.fbb_posts                -
public.fbb_ranks                -
public.fbb_config               -
public.fbb_search_cache         -
public.fbb_groups               -
```

Un peu plus tard, nous pouvons vérifier la progression:

```
postgres@debian1:~$ londiste /etc/postgresql/8.3/main/londiste_fluxbb.ini subscriber tables
Table                           State
public.fbb_censoring            ok
public.fbb_categories           ok
public.fbb_subscriptions        ok
public.fbb_search_matches       catching-up
public.fbb_topics               -
public.fbb_search_words         -
public.fbb_forum_perms          -
public.fbb_reports              -
public.fbb_online               -
public.fbb_bans                 -
public.fbb_users                -
public.fbb_forums               -
public.fbb_posts                -
public.fbb_ranks                -
public.fbb_config               -
public.fbb_search_cache         -
public.fbb_groups               -
```

Et, au bout d'un moment, tout le transfert est terminé:

```
postgres@debian1:~$ londiste /etc/postgresql/8.3/main/londiste_fluxbb.ini subscriber tables
Table                           State
public.fbb_censoring            ok
public.fbb_categories           ok
public.fbb_subscriptions        ok
public.fbb_search_matches       ok
public.fbb_topics               ok
public.fbb_search_words         ok
public.fbb_forum_perms          ok
public.fbb_reports              ok
public.fbb_online               ok
public.fbb_bans                 ok
public.fbb_users                ok
public.fbb_forums               ok
public.fbb_posts                ok
public.fbb_ranks                ok
public.fbb_config               ok
public.fbb_search_cache         ok
public.fbb_groups               ok
```

Voilà, la réplication est terminée. Toute modification sur le premier serveur se verra renvoyée sur le serveur abonné.

Surveiller londiste

La surveillance de londiste se fait en deux points : surveillance du lag de données entre les deux serveurs, et surveillance du schéma de données.

Le premier point peut se faire grâce à la requête suivante:

```
postgres@debian1:~$ psql -q fluxbb
fluxbb=# SELECT queue_name, consumer_name, lag, last_seen
fluxbb-#   FROM pgq.get_consumer_info();
        queue_name         |   consumer_name   |       lag       |   last_seen
---------------------------+-------------------+-----------------+---------------
 londiste.replication_fluxbb | replication_fluxbb | 00:00:27.567343 | 00:00:27.29114
(1 ligne)
```

Le lag est généralement inférieur à 1 minute. S'il est bien supérieur, il est temps de se poser quelques questions.

Quant au schéma, il vous est toujours possible d'utiliser le script Perl check_postgres.pl avec son action same_schema mais vous disposez aussi de quelques outils du côté de la commande londiste. Nous avons déjà parlé de « subscriber missing », capable de détecter les tables déclarées par le

fournisseur mais manquantes sur l'abonné. Vous pouvez aussi vérifier les différences entre deux structures avec « subscriber check ».

Vous pouvez même vérifier les différences de données grâce à « compare ». Pour résoudre ce type de problème, vous pouvez passer par un « subscriber resync » avec le nom de la table fautive pour ne re-synchroniser que cette table.

Les schémas Skytools

PGQ et Londiste, comme Slony dans l'article précédent, ont créé un schéma pour stocker la configuration.

Voici la liste des tables importantes pour le schéma pgq:

Table	Commentaires
consumer	liste des abonnés
queue	liste des queues déclarées
retry_queue	
subscription	liste des abonnements
tick	

Et voici la liste des tables importantes pour le schéma londiste:

Table	Commentaires
completed	
link	
provider_seq	liste des séquences enregistrées pour le fournisseur
provider_table	liste des tables enregistrées pour le fournisseur
subscriber_pending_fkeys	Liste des clés étrangères dont l'activité est suspendue
subscriber_pending_triggers	Liste des triggers dont l'activité est suspendue
subscriber_seq	Liste des séquences auxquelles est abonné l'esclave
subscriber_table	Liste des tables auxquelles est abonné l'esclave

Ce qui est prévu pour les Skytools 3

L'idée est de conserver le bon (pratiquement tout) et d'ajouter de nouvelles fonctionnalités. On peut notamment citer parmi les fonctionnalités ajoutées:

- la copie parallélisée (permet d'accélérer la mise en place de la réplication sur un gros serveur);
- la commande EXECUTE pour exécuter un script SQL sur les deux serveurs (permet de faciliter la mise à jour du schéma d'une base de données répliquée);
- la création automatique des tables et index lors de l'abonnement de leurs modifications à l'esclave (facilite le travail de mise en place d'une réplication);
- le support de la cascade de serveurs (permet de gérer des cas bien plus complexes de réplication).

Certaines opérations sont facilitées, comme le changement de maître, la pause dans la réplication, etc.

Petit récapitulatif

Avantages majeurs

- très simple à mettre en place
- très grande granularité
- pas de configuration compliquée
- les esclaves en lecture seule
- mise à jour de PostgreSQL rapide et avec le moins possible d'arrêt de production

Inconvénients majeurs

- pas de support de l'exécution de scripts SQL sur le maître et l'esclave (corrigé en version 3)
- pas de copie parallélisée (corrigé en version 3)
- pas de serveurs en cascade (corrigé en version 3)
- pas de réplication automatique des objets
- pas de réplication du TRUNCATE avant la 8.4
- très peu de documentation

Conclusion

Skype nous fournit un excellent produit, qui a profité de beaucoup plus d'attentions aux détails que Slony. Exactement ce qu'il manque à Slony en fait. Cependant, la documentation est pire, étant donné qu'elle n'existe pas. Le wiki de PostgreSQL propose un guide pratique pour aider à l'installation, mais cela en reste là. Autant dire bien peu de choses.

Rien que le fait qu'il n'y ait pas d'outils aidant à la mise en place est un excellent indice pour comprendre le sérieux de produit.