Taille des bases PostgreSQL

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 28 mai 2009

https://www.bortzmeyer.org/postgresql-taille-bases.html

Quand on apprend les SGBD et SQL (par exemple en lisant la Guide des bases de données en manga <https://www.bortzmeyer.org/manga-guide-databases.html>), on ne travaille qu'avec des bases minuscules et on ne remplit jamais le disque. Mais les SGBD sont typiquement utiisés pour stocker de **grandes** quantités de données et, là, de nouveaux problèmes surgissent pour le programmeur et le DBA. Le développement du système DNSmezzo <http://www.dnsmezzo.net/> est l'occasion de tester PostgreSQL avec de grandes bases.

Pour l'instant, les chiffres n'ont pas de quoi impressionner, la base commence juste à se remplir. On y reviendra dans quelques temps. Mais on peut déjà de se préparer.

Les fonctions utiles au DBA sont bien documentées <http://www.postgresql.org/docs/current/interactive/functions-admin.html>. Par exemple, pour connaître la taille d'une base, on ne peut pas utiliser `df` ou `ls` car la base n'est pas stockée dans un fichier unique. Mais PostgreSQL fournit ce qu'il faut :

```
dnsmezzo2=> SELECT pg_database_size('dnsmezzo2');
 pg_database_size
------------------
      34512910768
(1 row)
```

C'est une taille en octets. Pas très lisible. Mais il y a des fonctions plus jolies :

```
dnsmezzo2=> SELECT pg_size_pretty(pg_database_size('dnsmezzo2'));
 pg_size_pretty
----------------
 32 GB
(1 row)
```

C'est mieux. Et si on veut voir la taille par **table** et non plus pour une base entière ? PostgreSQL le permet aussi :

```
dnsmezzo2=> SELECT pg_size_pretty(pg_relation_size('DNS_packets'));
 pg_size_pretty
----------------
 3537 MB
(1 row)
```

Cette taille n'inclus pas les index et autres données auxiliaires. Pour les voir :

```
dnsmezzo2=> SELECT pg_size_pretty(pg_total_relation_size('DNS_packets'));
 pg_size_pretty
----------------
 8219 MB
(1 row)
```

Si on a une base qui se remplit automatiquement, il est prudent de surveiller automatiquement les disques, ce que permettent tous les moniteurs (par exemple mon <http://www.kernel.org/software/mon/>). Mais le script check_postgres <http://bucardo.org/check_postgres/>, écrit par Greg Sabino Mullane, permet de faire mieux en surveillant divers paramètres comme la taille des tables :

```
% sudo -u postgres /usr/local/sbin/check_postgres.pl  --action disk_space
POSTGRES_DISK_SPACE OK:  FS /dev/sda2 mounted on /var is using 53.97 GB of 73.61 GB (78%) * FS /dev/sda4 mou
```

Son format de sortie n'est pas conçu pour les humains mais pour des programmes comme Nagios ou Munin. Ici, la surveillance des tables :

```
% check_postgres.pl -c 10G --dbname=dnsmezzo2 --action table_size
POSTGRES_TABLE_SIZE OK: DB "dnsmezzo2" largest table is "public.dns_packets": 3537 MB | time=0.09  informati
```

On indique la taille maximale de la table (ici, dix gigaoctets) avec l'option `-c`. Le programme alertera si une table dépasse cette taille et un programme comme mon ou Nagios pourra alors agir :

```
% check_postgres.pl -c 3G --dbname=dnsmezzo2 --action table_size
POSTGRES_TABLE_SIZE CRITICAL: DB "dnsmezzo2" largest table is "public.dns_packets": 3537 MB [...]
```

Examiner la taille des tables et des bases, ou bien l'occupation disque avec la commande `df`, peut parfois donner un résultat surprenant : si on détruit des tuples avec la commande SQL `DELETE`, la taille des tables ne diminue pas. Ce point est bien expliqué dans la documentation <http://www.postgresql.org/docs/current/interactive/routine-vacuuming.html> : le SGBD ne récupère pas automatiquement l'espace libre, car celui-ci peut resservir pour de nouvelles données. Si on veut vraiment retrouver ses gigaoctets, on peut se servir de `VACUUM` et `VACUUM FULL` :

```
dnsmezzo2=#  SELECT pg_size_pretty(pg_total_relation_size('DNS_packets'));
 pg_size_pretty
----------------
 8219 MB
(1 row)

dnsmezzo2=# vacuum full;
VACUUM

dnsmezzo2=#  SELECT pg_size_pretty(pg_total_relation_size('DNS_packets'));
 pg_size_pretty
----------------
 8020 MB
(1 row)
```

Dans ce cas, le gain a été très faible car peu de suppressions ont eu lieu. Sur une base très vivante, le gain peut être bien plus spectaculaire. Mais attention ! `VACUUM FULL` bloque complètement la base, parfois pendant de longues périodes (voir la documentation de Postgresql <http://www.postgresql.org/docs/current/interactive/routine-vacuuming.html> pour des solutions alternatives).

Bon, une fois qu'on peut voir quelle place prennent les données, peut-on les mettre à un autre endroit, sur un autre disque ? Par défaut, PostgreSQL crée ses bases dans un répertoire qui dépend des options de lancement du serveur (sur Debian, par défaut, c'est `/var/lib/postgresql/$VERSION/main/base`) mais on peut demander à ce qu'elles soient mises ailleurs (et cela peut être intéressant de répartir les bases sur plusieurs contrôleurs disque, pour des raisons de performance). Cela se fait avec les "*tablespaces*" <http://www.postgresql.org/docs/current/interactive/manage-ag-tablespaces.html>. On crée une "*tablespace*" sur le disque de son choix :

```
% psql -c "CREATE tablespace Compta location '/big/disk/compta/db'"
```

et on peut l'utiliser, par exemple au moment de la création de la base :

```
% createdb --tablespace compta comptabilite
```

On peut ensuite voir les "*tablespaces*" et les bases qui les utilisent avec le catalogue de PostgreSQL <http://www.postgresql.org/docs/current/interactive/catalog-pg-database.html> :

```
=> SELECT datname AS database, spcname AS tablespace, spclocation AS directory
          FROM pg_database INNER JOIN pg_tablespace
          ON pg_tablespace.oid = pg_database.dattablespace;
     database      |     tablespace     |   directory
-------------------+--------------------+---------------
 template1         | pg_default         |
 essais            | pg_default         |
 ucd               | pg_default         |
 dnswitness-ipv6   | pg_default         |
 dnsmezzo2         | databasespartition | /databases/db
 comptabilite      | compta             | /big/disk/compta/db
```

Avec ce système, une base de données est toute entière dans un répertoire Unix, celui du "*tablespace*". Si on veut répartir une base, c'est possible avec les "*tablespaces*", puisque des commandes comme `CREATE TABLE` peuvent prendre un "*tablespace*" comme argument. Si on veut par contre répartir une table sur plusieurs endroits, il faut utiliser le partitionnement <http://www.postgresql.org/docs/current/interactive/ddl-partitioning.html> mais c'est nettement plus compliqué et je n'ai pas encore d'expérience pratique avec.

Si vous voulez la liste des table/index par taille, ainsi que des informations sur le "*tablespace*" utilisé, voyez ce joli code <https://gist.github.com/cquest/8331683>. Si on a mis une base sur un "*tablespace*" qui, à l'expérience, ne convient pas, on peut changer ensuite <https://www.bortzmeyer.org/postgresql-changer-tablespace.html>.
