Comment j'ai optimisé un postgres

Déjà, il faut commencer par trouver le fichier de config de ce bidule qui peut se situer ici /etc/postgresql/9.3/main/postgresql.conf ou là /var/lib/pgsql/9.3/data/postgresql.conf.

Le plus important à suivre dans ce document est la signification des différents paramètres modifiés, et non des valeurs.
Les valeurs vont varier d'un environnement à l'autre.

La machine sur laquelle je fais ces optimisations est une Redhat 6, avec postgres 9.3, 32G de RAM, et des disques SATA.

Quelques instruments importants qui pourront aider à trouver les points de lenteurs :
- explain, et explain analyze.
- analyze des bloats.

1- explain :

Explain, suivi d'une requête, explique comment la requête va être traitée (c'est à dire, le plan d'exécution de la requête), et vous donne une prévision du coût de la requête.
Plus vous arrivez à baisser ce coût, et moins votre requête prendra du temps pour s'exécuter.
Il est à noter que la requête n'est pas vraiment exécutée ; on peut donc faire des explain sur des requêtes INSERT ou DELETE.

Exemple :

```
explain select * from pg_stat_activity where query_start > '2016-04-18';

                                  QUERY PLAN
-------------------------------------------------------------------------------------------------
 Nested Loop  (cost=1.20..2.80 rows=1 width=337)
   ->  Hash Join  (cost=1.07..2.45 rows=1 width=273)
         Hash Cond: (s.usesysid = u.oid)
         ->  Function Scan on pg_stat_get_activity s  (cost=0.00..1.25 rows=33 width=209)
               Filter: (query_start > '2016-04-18 00:00:00+00'::timestamp with time zone)
         ->  Hash  (cost=1.03..1.03 rows=3 width=68)
               ->  Seq Scan on pg_authid u  (cost=0.00..1.03 rows=3 width=68)
   ->  Index Scan using pg_database_oid_index on pg_database d  (cost=0.13..0.33 rows=1 width=68)
         Index Cond: (oid = s.datid)
```

2- Explain analyze :

Différents de explain dans le sens qu'il exécute réellement la requête, et vous donne le temps réel d'exécution de celle-ci.
Ne pas jouer avec sur des requêtes INSERT, UPDATE, DELETE à moins de les encadrer dans une transaction, puis de faire un rollback, ou à moins de savoir ce que vous faîtes.

Exemple :

```
explain analyze select * from uid where id like '%ab%' order by id;

                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------
-
  Sort  (cost=132705.83..133513.82 rows=323196 width=213) (actual time=2844.340..2914.540 rows=313119 loops=1)
  Sort Key: id
  Sort Method: quicksort  Memory: 82907kB
  ->  Seq Scan on uid  (cost=0.00..103130.09 rows=323196 width=213) (actual time=0.014..1205.534 rows=313119 loops=1)
        Filter: ((id)::text ~~ '%ab%'::text)
        Rows Removed by Filter: 2589223
 Total runtime: 2936.588 ms
```

3- **Analyse des bloats** :

Une bonne explication de ce que c'est que le bloat se trouve ici : https://www.keithf4.com/checking-for-postgresql-bloat/
En gros, c'est lorsque les informations d'une table ou d'un index sont dispersées sur le disque (un peu comme la fragmentation d'un système de fichiers).
Clairement, ça complique les recherches pour servir une requête.
Les "vacuum full" nous permettent de venir à bout de ces bloats.
Les bloats arrivent avec DELETE et UPDATE, pas avec les INSERT.

Il existe plusieurs requêtes sur Internet pour détecter les bloats, j'en utilise 2 :

```
SELECT
  current_database(), schemaname, tablename, /*reltuples::bigint, relpages::bigint, otta,*/
  ROUND((CASE WHEN otta=0 THEN 0.0 ELSE sml.relpages::FLOAT/otta END)::NUMERIC,1) AS tbloat,
  CASE WHEN relpages < otta THEN 0 ELSE bs*(sml.relpages-otta)::BIGINT END AS wastedbytes,
  iname, /*ituples::bigint, ipages::bigint, iotta,*/
      ROUND((CASE WHEN iotta=0 OR ipages=0 THEN 0.0 ELSE ipages::FLOAT/iotta END)::NUMERIC,1) AS ibloat,
  CASE WHEN ipages < iotta THEN 0 ELSE bs*(ipages-iotta) END AS wastedibytes
FROM (
  SELECT
    schemaname, tablename, cc.reltuples, cc.relpages, bs,
    CEIL((cc.reltuples*((datahdr+ma-
          (CASE WHEN datahdr%ma=0 THEN ma ELSE datahdr%ma END))+nullhdr2+4))/(bs-20::FLOAT)) AS otta,
              COALESCE(c2.relname,'?') AS iname, COALESCE(c2.reltuples,0) AS ituples, COALESCE(c2.relpages,0) AS ipages,
        COALESCE(CEIL((c2.reltuples*(datahdr-12))/(bs-20::FLOAT)),0) AS iotta -- very rough approximation, assumes all cols
```

```
  FROM (
    SELECT
      ma,bs,schemaname,tablename,
      (datawidth+(hdr+ma-(CASE WHEN hdr%ma=0 THEN ma ELSE hdr%ma END)))::NUMERIC AS
datahdr,
       (maxfracsum*(nullhdr+ma-(CASE WHEN nullhdr%ma=0 THEN ma ELSE nullhdr%ma END)))
AS nullhdr2
    FROM (
      SELECT
        schemaname, tablename, hdr, ma, bs,
        SUM((1-null_frac)*avg_width) AS datawidth,
        MAX(null_frac) AS maxfracsum,
        hdr+(
          SELECT 1+COUNT(*)/8
          FROM pg_stats s2
          WHERE null_frac<>0 AND s2.schemaname = s.schemaname AND s2.tablename = s.tablename
        ) AS nullhdr
      FROM pg_stats s, (
        SELECT
          (SELECT current_setting('block_size')::NUMERIC) AS bs,
          CASE WHEN SUBSTRING(v,12,3) IN ('8.0','8.1','8.2') THEN 27 ELSE 23 END AS hdr,
          CASE WHEN v ~ 'mingw32' THEN 8 ELSE 4 END AS ma
        FROM (SELECT version() AS v) AS foo
      ) AS constants
      GROUP BY 1,2,3,4,5
    ) AS foo
  ) AS rs
  JOIN pg_class cc ON cc.relname = rs.tablename
    JOIN pg_namespace nn ON cc.relnamespace = nn.oid AND nn.nspname = rs.schemaname AND
nn.nspname <> 'information_schema'
  LEFT JOIN pg_index i ON indrelid = cc.oid
  LEFT JOIN pg_class c2 ON c2.oid = i.indexrelid
) AS sml
ORDER BY wastedibytes DESC;

 SELECT
         current_database(), schemaname, tablename, reltuples::bigint, relpages::bigint, otta,
         ROUND(CASE WHEN otta=0 THEN 0.0 ELSE sml.relpages/otta::numeric END,1) AS tbloat,
         CASE WHEN relpages < otta THEN 0 ELSE relpages::bigint - otta END AS wastedpages,
                 CASE WHEN relpages < otta THEN 0 ELSE bs*(sml.relpages-otta)::bigint END AS
wastedbytes,
           CASE WHEN relpages < otta THEN '0 bytes'::text ELSE (bs*(relpages-otta))::bigint || ' bytes'
END AS wastedsize,
         iname, ituples::bigint, ipages::bigint, iotta,
         ROUND(CASE WHEN iotta=0 OR ipages=0 THEN 0.0 ELSE ipages/iotta::numeric END,1) AS
ibloat,
         CASE WHEN ipages < iotta THEN 0 ELSE ipages::bigint - iotta END AS wastedipages,
         CASE WHEN ipages < iotta THEN 0 ELSE bs*(ipages-iotta) END AS wastedibytes,
          CASE WHEN ipages < iotta THEN '0 bytes' ELSE (bs*(ipages-iotta))::bigint || ' bytes' END AS
```

```
wastedisize
        FROM (
          SELECT
            schemaname, tablename, cc.reltuples, cc.relpages, bs,
            CEIL((cc.reltuples*((datahdr+ma-
                  (CASE WHEN datahdr%ma=0 THEN ma ELSE datahdr%ma END))+nullhdr2+4))/(bs-
20::float)) AS otta,
                        COALESCE(c2.relname,'?') AS iname, COALESCE(c2.reltuples,0) AS ituples,
COALESCE(c2.relpages,0) AS ipages,
                   COALESCE(CEIL((c2.reltuples*(datahdr-12))/(bs-20::float)),0) AS iotta -- very rough
approximation, assumes all cols
          FROM (
            SELECT
              ma,bs,schemaname,tablename,
                (datawidth+(hdr+ma-(case when hdr%ma=0 THEN ma ELSE hdr%ma END)))::numeric AS
datahdr,
               (maxfracsum*(nullhdr+ma-(case when nullhdr%ma=0 THEN ma ELSE nullhdr%ma END)))
AS nullhdr2
            FROM (
              SELECT
                schemaname, tablename, hdr, ma, bs,
                SUM((1-null_frac)*avg_width) AS datawidth,
                MAX(null_frac) AS maxfracsum,
                hdr+(
                  SELECT 1+count(*)/8
                  FROM pg_stats s2
                        WHERE null_frac<>0 AND s2.schemaname = s.schemaname AND s2.tablename =
s.tablename
                ) AS nullhdr
              FROM pg_stats s, (
                SELECT
                  (SELECT current_setting('block_size')::numeric) AS bs,
                  CASE WHEN substring(v,12,3) IN ('8.0','8.1','8.2') THEN 27 ELSE 23 END AS hdr,
                  CASE WHEN v ~ 'mingw32' THEN 8 ELSE 4 END AS ma
                FROM (SELECT version() AS v) AS foo
              ) AS constants
              GROUP BY 1,2,3,4,5
            ) AS foo
          ) AS rs
          JOIN pg_class cc ON cc.relname = rs.tablename
           JOIN pg_namespace nn ON cc.relnamespace = nn.oid AND nn.nspname = rs.schemaname AND
nn.nspname <> 'information_schema'
          LEFT JOIN pg_index i ON indrelid = cc.oid
          LEFT JOIN pg_class c2 ON c2.oid = i.indexrelid
        ) AS sml
        WHERE sml.relpages - otta > 0 OR ipages - iotta > 10 ORDER BY wastedbytes DESC
```

On peut aussi jeter un coup d’œil sur les tailles des tables dans la BD :

Les 50 tables les plus grosses dans la BD:

```
SELECT
table_name,
   pg_size_pretty(table_size) AS table_size,
   pg_size_pretty(indexes_size) AS indexes_size,
   pg_size_pretty(total_size) AS total_size
   FROM (
         SELECT
         table_name,
         pg_table_size(table_name) AS table_size,
         pg_indexes_size(table_name) AS indexes_size,
         pg_total_relation_size(table_name) AS total_size
         FROM (
            SELECT ('"' || table_schema || '"."' || table_name || '"') AS table_name
            FROM information_schema.tables
            ) AS all_tables
         ORDER BY total_size DESC
       ) AS pretty_sizes limit 50;
```

Les bloats étant détectés, on peut lancer des vacuum full dessus pour récrire complètement les tables. Mais attention, un vacuum full va locker complètement la table pendant tout le temps de l'opération. Pour faire un vacuum full, il faut donc prévoir un temps d'arrêt de l'application, et ce n'est pas une opération à faire régulièrement.

Exemple de vacuum full :
vacuum full uid;

On peut avoir aussi besoin de refaire l'index de la table avec :
reindex table uid;

Prendre note qu'il est très important de lancer la commande ANALYZE contre la table qui vient de subir un vacuum full pour que postgres mette à jour ses statistiques de recherches sur la table.

Exemple : Analyze uid;

4- <u>Modifications de paramètres postgres</u> :

# Optimisation du nombre max de connexions simultanées :
max_connections = 160

# Cache des requêtes (25% de la mémoire _disponible_)
shared_buffers = 8GB

# Faire des tris en RAM
work_mem = 256MB

```
# Mémoire utilisée pour faire le vacuum
maintenance_work_mem = 1GB

# Accélérer les écritures
# J'ai dû jouer avec pour avoir l'optimum
wal_buffers = 16MB
checkpoint_segments = 64
checkpoint_completion_target=0.8

# Cache des données disque en RAM (>50% de la taille de la RAM)
effective_cache_size = 20GB

# Utiliser 1000*300 lignes dans une table pour les statistiques
# Quand on lance analyze, ca va se baser sur 300000 ligne pour updater les stats.
# J'ai des tables de ++millions d'enregistrements.
default_statistics_target = 1000

# Log toutes les requêtes et leur temps d'exécution (0 = tout, 400 = juste pour les requêtes >= 400ms)
log_min_duration_statement = 0

# Le vacuum automatique à on
autovacuum = on

# Log tous les autovacuum
log_autovacuum_min_duration = 0

# Exécuter les autovacuum plus souvent
autovacuum_vacuum_threshold = 20
autovacuum_vacuum_scale_factor = 0.1
autovacuum_analyze_threshold = 10
autovacuum_analyze_scale_factor = 0.05
autovacuum_max_workers = 6
autovacuum_naptime = 20s
```

5- **Petite automatisation du vacuum full** :

Petit script qui utilise le fichier liste_vacuum dans lequel il y a une liste de tables à vacuumiser.

```
#!/bin/bash
for table in `cat liste_vacuum`; do
   echo "Demarrage du vacuum full sur la table $table" | mail -r expediteur@email.com -s "Debut vacuum $table sur Base de données " destinataire1@email.com destinataire2@email.com

  echo "vacuum full verbose $table;" | psql NomdelaBD > /tmp/vacuum_full 2>&1
  echo "reindex table $table;" | psql NomdelaBD >> /tmp/vacuum_full 2>&1
  echo "analyze verbose $table;" | psql NomdelaBD >> /tmp/vacuum_full 2>&1

   cat /tmp/vacuum_full | mail -r expediteur@email.com -s "Fin vacuum $table sur Base de données"
```

destinataire@email.com destinataire2@email.com
done

rm /tmp/vacuum_full
sleep 5
echo "Fin de tous les vacuums" | mail -r expediteur@email.com -s "Fin vacuum" destinataire@email.com destinataire2@email.com

6- Agir sur les autovacuum par table :

```
ALTER TABLE nom_table SET (autovacuum_enabled = true, toast.autovacuum_enabled = true);
ALTER TABLE nom_table SET (autovacuum_vacuum_scale_factor = 0.05,
autovacuum_vacuum_threshold = 50);
ALTER TABLE nom_table SET (autovacuum_analyze_scale_factor = 0.01);
ALTER TABLE nom_table SET (autovacuum_vacuum_threshold = 50);
```

7- Conclusion :

Rendez les autovacuum aussi agressifs que possible sur les tables qui en ont besoin.
Optimiser la config avec les paramètres donnés ci-haut. Cela nécessite de faire des tests pour trouver les bons pour son environnement. (EXPLAIN est votre ami).
Faire des vacuum full seulement quand on n'a plus le choix => downtime, avertir les utilisateurs/clients, ...

Sources :

http://www.revsys.com/writings/postgresql-performance.html
https://wiki.postgresql.org/wiki/Tuning_Your_PostgreSQL_Server
https://wiki.postgresql.org/wiki/Performance_Optimization
https://feeding.cloud.geek.nz/posts/troubleshooting-postgres-performance/
http://blog.pilotsystems.net/2011/juin/quelques-outils-pour-localiser-un-probleme-de-performances
http://fr.slideshare.net/pgconf/five-steps-perform2009
https://wiki.postgresql.org/wiki/Lock_Monitoring
http://www.yerbynet.com/Cours/comprendre_explain.pdf
http://blog.pilotsystems.net/2011/aout/cas-pratique-doptimisation-de-postgresql
https://doc.nuxeo.com/display/ADMINDOC710/PostgreSQL

© Juin 2016

Roger YERBANGA
www.yerbynet.com