Un pas de plus vers l'OLAP

Nouveautés de PostgreSQL 10

2017.12.14

Dalibo - http://dalibo.com

http://www.dalibo.org/conferences

Nouveautés de PostgreSQL 10

Un pas de plus vers l'OLAP

TITRE : Nouveautés de PostgreSQL 10
SOUS-TITRE : Un pas de plus vers l'OLAP

REVISION: 2017.12.14

Contents

INTRODUCTION

UN PEU D'HISTOIRE ET QUELQUES CHIFFRES

- Plus de 20 ans d'existence
- Une nouvelle version par an...
- ...développée par une communauté mondiale de bénévoles
- Sous licence libre (dérivée de BSD)
- Plus de 1,4 million de lignes de code C

CHANGEMENTS DANS POSTGRESQL 10

NUMÉROTATION DES VERSIONS

Ancienne numérotation exprimée sur 3 nombres :

```
9 . 6 . 3
Majeure1 . Majeure2 . Mineure
```

Nouvelle numérotation exprimée sur 2 nombres uniquement :

```
10 . 2
Majeure . Mineure
```

NOMMAGE

- Au niveau des répertoires
  - `pg_xlog` -> `pg_wal`
  - `pg_clog` -> `pg_xact`
- Au niveau des fonctions
  - `xlog` -> `wal`
  - `location` -> `lsn`
- Au niveau des outils

```
- xlog   wal
```

CONFIGURATION

- Réplication activée par défaut :
  - pour 10 connexions
  - en local
- Parallélisme activé par défaut :
  - 2 workers maximum par exécution
  - 4 exécutions parallèles concurrentes maximum

UN PAS DE PLUS VERS L'OLAP

POURQUOI ?

- PostgreSQL très bon en transactionnel (OLTP)
- Moins bon en Business Intelligence
- Améliorations depuis quelques années

FONCTIONNALITÉS APPORTÉES PAR LA VERSION 9.5

- `GROUPING SETS` (et `ROLLUP`, `CUBE`) : plusieurs aggrégats dans un même GROUP BY
- `INSERT ... ON CONLICT ...` : merge et plus

FONCTIONNALITÉS APPORTÉES PAR LA VERSION 9.6

- Début du parallélisme :
  - Scan Séquentiel
  - Jointures (Nested Loop et Hash Join)
  - Aggrégats

FONCTIONNALITÉS APPORTÉES PAR LA VERSION 10

- Parallelisme : Index Scan, Jointure (Merge Join), Sous-requêtes non corrélées
- Partitionnement déclaratif
- Réplication logique

PARALLÉLISME

FONCTIONNEMENT

- Pour les requêtes en lecture
- Selon la taille des tables et index mis en jeu
- Lancement de processus supplémentaires pour partager le travail :
  - Si le coût est intéressant pour le planner
  - Dans la limite de la configuration

CONFIGURATION

- `max_worker_processes = 8` : nombre max. background workers (parallèles ou non)
- `max_parallel_workers = 8` : nombre max. bg worker pour le parallèlisme pour l'instance
- `max_parallel_workers_per_gather = 2` : nombre max. bg worker pour une requête
- `parallel_tuple_cost` et `parallel_setup_cost` : configuration du planner
- `min_parallel_table_scan_size` et `min_parallel_index_scan_size` : seuil d'utilisation selon la taille des objets

---

LIMITES

- Paralléliser est parfois plus lent !
  - `CREATE TABLE test (i int);`
  - `INSERT INTO test (i) SELECT generate_series(1,100000000);`
- Requête : `SELECT * FROM test WHERE i = 1;`
- Résultat sur un portable avec SSD :
  - 1 worker : 2 minutes
  - 2 workers : 1 minute 41 sec
  - 4 workers : 1 minute 35 sec (100% de CPU, 28% d'IO wait)
  - 1 index btree sur la colonne i : **82 millisecondes**

PARTITIONNEMENT

AVANT DE PARTITIONNER UNE TABLE

- Réponse à un problème de performance, plutôt que de volumétrie
- Quand l'utiliser ?
  - Volumes tellement importants que les index ne suffisent plus
  - Lorsqu'on utilise des prédicats très filtrants
  - Cas particuliers : queues / optimisation des suppressions
- Alternatives :
  - index partiels
  - index couvrants
  - index BRIN

AVANT POSTGRESQL 10

- Le partitionnement par héritage se base sur
  - Une table mère vide
  - Un trigger qui oriente les écritures dans les partitions
  - Des tables filles créées par héritage avec contrainte CHECK
- Disponible depuis longtemps

SUR POSTGRESQL 10

- Mise en place et administration simplifiées car intégrées au moteur
- Plus de trigger
  - insertions plus rapides
  - routage des données insérées dans la bonne partition
  - erreur si aucune partition destinataire
- Partitions
  - attacher/détacher une partition
  - contrainte implicite de partitionnement
  - expression possible pour la clé de partitionnement
  - sous-partitions possibles
- Changement du catalogue système

- nouvelles colonnes dans `pg_class`
- nouveau catalogue `pg_partitioned_table`

PARTITIONNEMENT PAR LISTE

- Créer une table partitionnée :

```
CREATE TABLE t1(c1 integer, c2 text) PARTITION BY LIST (c1);
```

- Ajouter une partition :

```
CREATE TABLE t1_a PARTITION OF t1 FOR VALUES IN (1, 2, 3);
```

- Détacher la partition :

```
ALTER TABLE t1 DETACH PARTITION t1_a;
```

- Attacher la partition :

```
ALTER TABLE t1 ATTACH PARTITION t1_a FOR VALUES IN (1, 2, 3);
```

PARTITIONNEMENT PAR INTERVALLES

- Créer une table partitionnée :

```
CREATE TABLE t2(c1 integer, c2 text) PARTITION BY RANGE (c1);
```

- Ajouter une partition :

```
CREATE TABLE t2_1 PARTITION OF t2 FOR VALUES FROM (1) TO (100);
```

- Détacher une partition :

```
ALTER TABLE t2 DETACH PARTITION t2_1;
```

CONTROLE D'ERREUR

- S'il n'y a pas de partition :

```
postgres=# INSERT INTO t2 VALUES (0);
ERROR:  no PARTITION OF relation "t2" found for row
DETAIL:  Partition key of the failing row contains (c1) = (0).
```

- Utiliser `MINVALUE` et `MAXVALUE` pour - infini et + infini

VITESSE D'INSERTION

Table non partitionnée

```
INSERT INTO t1 SELECT i, 'toto'
  FROM generate_series(0, 9999999) i;
Time: 10097.098 ms (00:10.097)
```

Nouveau partitionnement

```
INSERT INTO t2 SELECT i, 'toto'
  FROM generate_series(0, 9999999) i;
Time: 11448.867 ms (00:11.449)
```

Ancien partitionnement

```
INSERT INTO t3 SELECT i, 'toto'
  FROM generate_series(0, 9999999) i;
Time: 125351.918 ms (02:05.352)
```

LIMITES

- La table mère ne peut pas avoir de données
- La table mère ne peut pas avoir d'index
  - ni PK, ni UK, ni FK pointant vers elle
- Pas de colonnes additionnelles dans les partitions
- L'héritage multiple n'est pas permis
- Valeurs nulles acceptées dans les partitions uniquement si la table partitionnée le permet
- Partitions distantes pour l'instant pas supportées
- En cas d'attachement d'une partition
  - vérification du respect de la contrainte (verrou bloquant sur la partition)
  - sauf si ajout au préalable d'une contrainte `CHECK` identique
- Ne fonctionne que pour les prédicats simples / Pas de changement du planner

---

REPLICATION LOGIQUE

FONCTIONNEMENT

- A partir de l'infrastructure existante (travaux débutés en 9.4)
  - réplication en flux
  - slots de réplication
- Réplique les changements sur une seule base de données
  - d'un ensemble de tables défini
- Uniquement INSERT / UPDATE / DELETE
  - pas les DDL, ni les TRUNCATE

FONCTIONNEMENT

- Une publication est créée sur le serveur éditeur
- L'abonné souscrit à cette publication, c'est un « souscripteur »
- Un processus spécial est lancé : le « bgworker logical replication ». Il se connecte à un slot de réplication sur le serveur éditeur
- Le serveur éditeur procéde à un décodage logique des journaux de transaction pour extraire les résultats des ordres SQL
- Le flux logique est transmis à l'abonné qui les applique sur les tables

UTILISATION SUR L'ÉDITEUR

- Définir `wal_level` à `logical`
- Exporter le schéma de la base sans données
- Créer une publication pour toutes les tables

```
CREATE PUBLICATION ma_publication FOR ALL TABLES;
```

- Créer une publication pour une table

```
CREATE PUBLICATION ma_publication FOR TABLE t1;
```

UTILISATION SUR LES ABONNÉS

- Initialiser une base de données et importer son schéma
- Créer l'abonnement :

```
CREATE SUBSCRIPTION ma_souscription
CONNECTION 'host=127.0.0.1 port=5433 user=repliuser dbname=bench'
PUBLICATION ma_publication;
```

SUPERVISION

- Sur l'éditeur
  - `pg_stat_replication` pour l'état de la réplication
  - `pg_replication_slots` pour la définition des slots de réplication
  - `pg_publication` pour la définition des publications
  - `pg_publication_tables` pour la liste des tables publiées par publication
- Sur l'abonné
  - `pg_subscription` pour la définition des abonnements
  - `pg_replication_origin_status` pour l'état de la réplication

LIMITES

- Non répliqué :
  - Schéma
  - Séquences
  - *Large objects*
- Pas de publication des tables parents du partitionnement
- Ne convient pas comme fail-over
- Contrainte d'unicité nécessaire pour `UPDATE` et `DELETE`

MERCI

- Nicolas Thauvin - nicolas.thauvin@dalibo.com
- http://dalibo.com - http://github.com/dalibo
- twitter: @orgrim
- blog : http://orgrim.net