Ora2Pg Performances

Gilles Darold

- Consultant @ Dalibo
- Auteur d'Ora2Pg
- Auteur de PgBadger
- Auteur de pgFormatter
- Et autres => http://www.darold.net/

Cas pratique

- Migration d'une base Oracle de 2,5 To vers PostgreSQL
- Base de type DataWareHouse utilisée pour du Marketing analytique
- A l'origine 55 000 tables dont la quasi totalité sont des tables auto-générées par l'application
- 1,5 To de données réparties dans 200 tables dont 3 tables partitionnées

Ressources

- La cible => Serveur PostgreSQL
  - CPU : 2x6 coeurs multithreadés = 24 coeurs
  - RAM : 96 GB
  - Stockage:
    - Carte RAID: Cache: 512 Mo, Batterie: oui
    - RAID 1: 2 x 146 Go SAS 15k (Système)
    - RAID 10: 8 x 600 Go SAS 15k (PGDATA)
    - RAID 1: 2 x 146 Go SAS 15k (XLOG)
  - Système de fichier : EXT4
- La source => Serveur Oracle
  - configuration matérielle identique

“bulk load”

- Règles d'usage du chargement massif dans PostgreSQL
  - Utilisation de COPY
  - Pas d'index
  - Pas de contraintes
  - Pas de triggers
  - Pas de séquences

  } créés en fin de chargement

  - Un maintenance_work_mem élevé
    - pour la création des index et constraintes
  - Augmentation du checkpoint_segments (>64/1GB)
  - Désactivation de l'archivage et de la Streaming Replication
- Dernière version de PostgreSQL - 9.2

Optimisation I/O

- Paramètres de durabilité
  - fsync = off
  - full_page_writes = off
  - synchronous_commit = off
  - WAL sur des disques SSD
- Tuning du Kernel
  - vm.dirty_background_ratio = 1
  - vm.dirty_ratio = 2

PostgreSQL configuration - 1

- shared_buffers = 10GB
- maintenance_work_mem = 2GB
- checkpoint_segments = 64
- checkpoint_completion_target = 0.9
  - 4,5 min => 270s : 1Go/270s = 3,7Mo/s
- wal_level = mimimal
- archive_mode = off
- wal_buffers = 32Mo

PostgreSQL configuration - 2

- Impact de l'autovacuum
  - autovacuum = on => pas de problèmes, INSERT
  - autoanalyze => ne se déclenchera qu'a la fin
- Gestion des traces
  - log_destination & log_directory sur la partition système
  - log_min_duration_statement = -1
  - On ne trace que les erreurs

Ora2Pg : Performances

- Préparer la migration coté Oracle
  - Interruption de production => savoir ce qui est permis.
  - Limiter au possible les données à exporter (purge, archivage de données).
  - Suppression des index inutiles : non-utilisés ou redondants.
  - Ne charger que les données vivantes, le reste pourra être chargé plus tard.
  - Vider la corbeille avant (Recyclebin => BIN$...) ou ajouter BIN\$.* à la directive EXCLUDE

Installation d'Ora2Pg

- Mise en oeuvre d'Ora2Pg
  - Migration initiée depuis le serveur PostgreSQL
  - Installation du client Oracle
  - Installation DBD:Oracle et DBD::Pg
  - Installation Ora2Pg
  - Configuration générique
    - Séparation des définitions de tables, index et contraintes dans des fichiers différents (FILE_PER_*)
    - Paramétrage des connexions à la base Oracle
    - Définition du schéma à exporter

Rapport sur la base Oracle

- Scan de la base Oracle
  - ora2pg -t SHOW_REPORT –dump_as_html
  - Scan du catalog Oracle : + 45 minutes avant pour 55 000 tables, depuis ora2pg 11.0 : 2 minutes
- Rapport HTML sur la base Oracle
  - Taille de la base Oracle 2.5 To
  - 207 tables et 197 index dont 3 tables partitionnées
  - 1116 tables de partition
  - 2989 index de partition

Ora2Pg - Database Migration Report

Version	Oracle Database 10g Enterprise Edition Release 10.2.0.4.0
Schema	DBEXEMPLE
Size	2531166.00 MB

Object	Number	Invalid	Comments
DATABASE LINK	2	0	Database links will not be exported. You may try the dblink perl contrib module or use the SQL/MED PostgreSQL features with the different Foreign Data Wrapper (FDW) extentions.
INDEX	197	0	5 function based b-tree index(es) 154 b-tree index(es) 159 index(es) are concerned by the export, others are automatically generated and will do so on PostgreSQL. Bitmap index(es) will be exported as b-tree index(es) if any. Cluster, domain, bitmap join and IOT indexes will not be exported at all. Reverse indexes are not exported too, you may use a trigram-based index (see pg_trgm) or a reverse() function based index and search. Use 'varchar_pattern_ops', 'text_pattern_ops' or 'bpchar_pattern_ops' operators in your indexes to improve search with the LIKE operator respectively into varchar, text or char columns.
INDEX PARTITION	2989	0	
JOB	2	0	Job are not exported. You may set external cron job with them.
MATERIALIZED VIEW	2	0	All materialized view will be exported as snapshot materialized views, they are only updated when fully refreshed.
PACKAGE BODY	2	0	Total size of package code: 174109 bytes. Number of procedures and functions found inside those packages: 49.
SEQUENCE	25	0	Sequences are fully supported, but all call to sequence_name.NEXTVAL or sequence_name.CURRVAL will be transformed into NEXTVAL('sequence_name') or CURRVAL('sequence_name').
TABLE	207	0	239 check constraint(s). **Total number of rows: 6,582,909,430** Top 10 of tables sorted by number of rows: table1 has 4,307,820,577 rows table2 has 1,095,565,281 rows table3 has 885,863,083 rows table4 has 87,656,224 rows table5 has 62,338,986 rows table6 has 22,185,337 rows table7 has 21,793,251 rows table8 has 17,407,464 rows table9 has 8,620,518 rows table10 has 7,305,792 rows Top 10 of largest tables: table6: 2120 MB (22185337 rows) table20: 1080 MB (3691726 rows) table21: 1010 MB (21793251 rows) table22: 970 MB (3722155 rows) table23: 850 MB (4471196 rows) table24: 770 MB (4471196 rows) table25: 610 MB (7296623 rows) table10: 570 MB (7305792 rows) table9: 530 MB (8620518 rows) table26: 240 MB (730652 rows)
TABLE PARTITION	1116	0	1116 range partitions Partitions are exported using table inheritance and check constraint. Hash partitions are not supported by PostgreSQL and will not be exported.
TRIGGER	1	0	Total size of trigger code: 394 bytes.
Total	4543	0	

Tables volumineuses

- Nombre de lignes
  - 1 table de 4,3 milliards de lignes (partitionnée)
  - 1 table de 1 milliard de lignes (partitionnée)
  - 1 table de 885 millions de lignes (partitionnée)
  - 4 tables entre 20 et 100 millions de lignes
- Volume de la table
  - 1 table de 2Go
  - 3 tables de 1Go
  - 6 tables entre 500Mo et 1Go

Import du schéma

- Adaptation du schéma
  - NUMBER sans précision => bigint, decimal ou numeric
  - Champs monétaires => numeric
  - Timestamp => date (si on a que l'année)
  - Identification des mots réservés dans les noms de table et de colonne
  - Séparation des index, contraintes et séquence
  - Identification des index avec fonctions et contraintes CHECK, reprise éventuelle du code SQL
- Chargement du schéma
  - psql -h ip_adresse -U owner dbmig < tables.sql

Réglage d'Ora2Pg

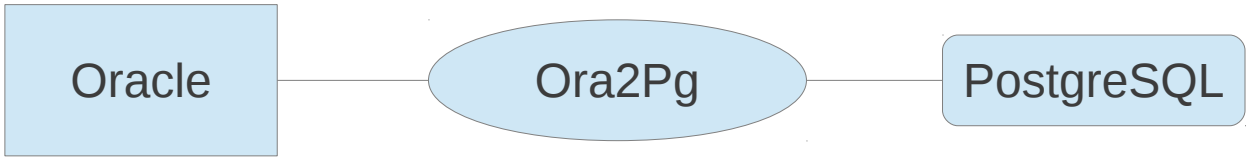
- Configuration de la connexion à PostgreSQL
  - PG_DSN, PG_USER et PG_PWD
- Test de chargement pour déterminer la vitesse et vérifier les types
  - Ajustement du DATA_LIMIT par défaut (10000)
- ora2pg -t COPY -c ora2pg.conf -a TABLE6

[==========>] 28 917 895 of 29 505 996 rows
(98.0%) table TABLE6 (61277.7 recs/sec)

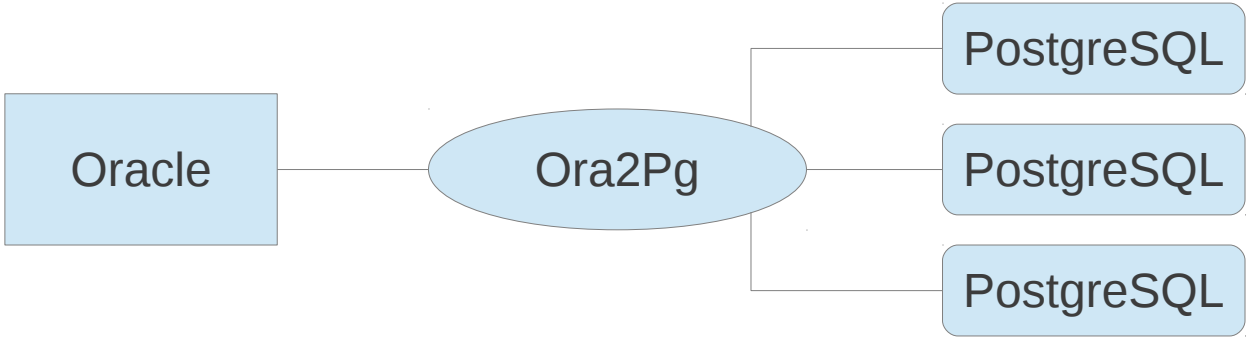
- DATA_LIMIT = 60000

Chargement à grande vitesse - 1

- Avant la 11.0, limitation à 25 000 rec/sec

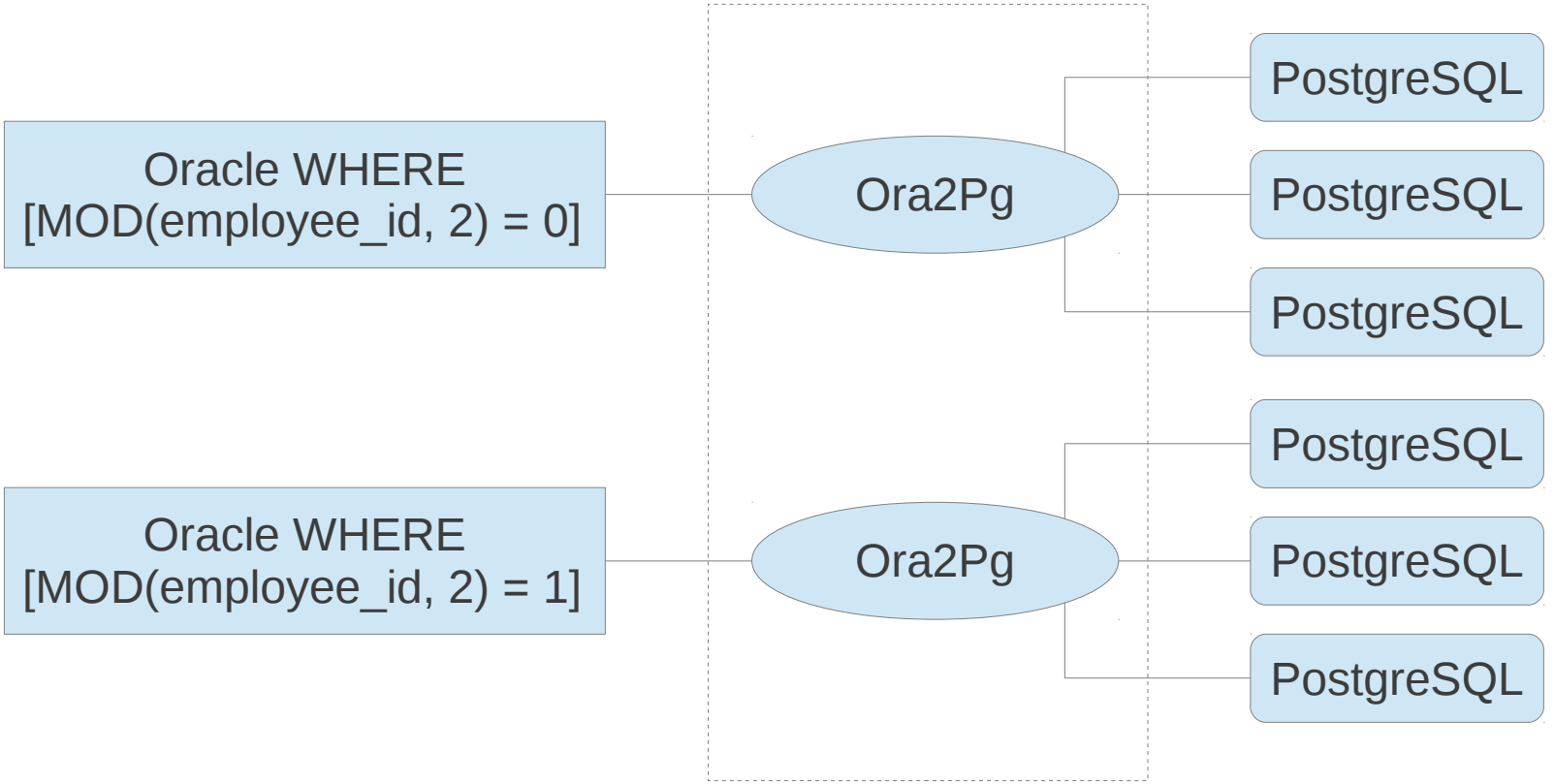


- Avec l'option -j 8, limitation à 160 000 rec/sec



- Avec l'option -J 8 -j 3, limitation à 260 000 rec/sec
  - DEFINED_PKEY EMPLOYEES:employee_id
  - SELECT * FROM EMPLOYEES WHERE MOD(employee_id, 2) = N

Chargement à grande vitesse - 2



Chargement à grande vitesse - 3

- Identification des différentes valeurs à -j et -J
  - table avec 2 entiers : > 900 000 tuples/sec
  - table moyenne : 140 000 tuples/sec
  - table avec BLOB : 15 à 1000 tuples/sec
- Attention : -j x -J = Nombre de coeurs utilisés
- Option -L pour modifier le DATA_LIMIT
- SHOW_TABLE pour traitement particulier
  - sortie tables avec + de lignes
  - sortie tables les plus volumineuses
  - Tables avec BLOB/RAW/CLOB => SHOW_COLUM

Chargement des partitions

- Cas des partitions
  - Insertion dans la table principale
  - Déplacement dans la table partitionnée par trigger ou rule.
  - Performances de chargement catastrophiques
- Avec Ora2Pg 11.1 l'import se fait directement dans la partition cible
  - Pas de triggers mais présence des contraintes CHECK
- ALLOW_PARTITION
  - Permet de limiter l'import à certaines tables de la partition uniquement

Ora2Pg versus ETL

- Ora2Pg et ETL
  - Ora2pg -t KETTLE -j 16 -a TABLE8
  - Génération d'un fichier TABLE8.ktr de définition de transformation pour Kettle (http://kettle.pentaho.com)
- Permet le chargement en ligne de commande :
  - JAVAMAXMEM=4096 ./pan.sh -file TABLE8.ktr -level Detailed
- Attention aux OOM

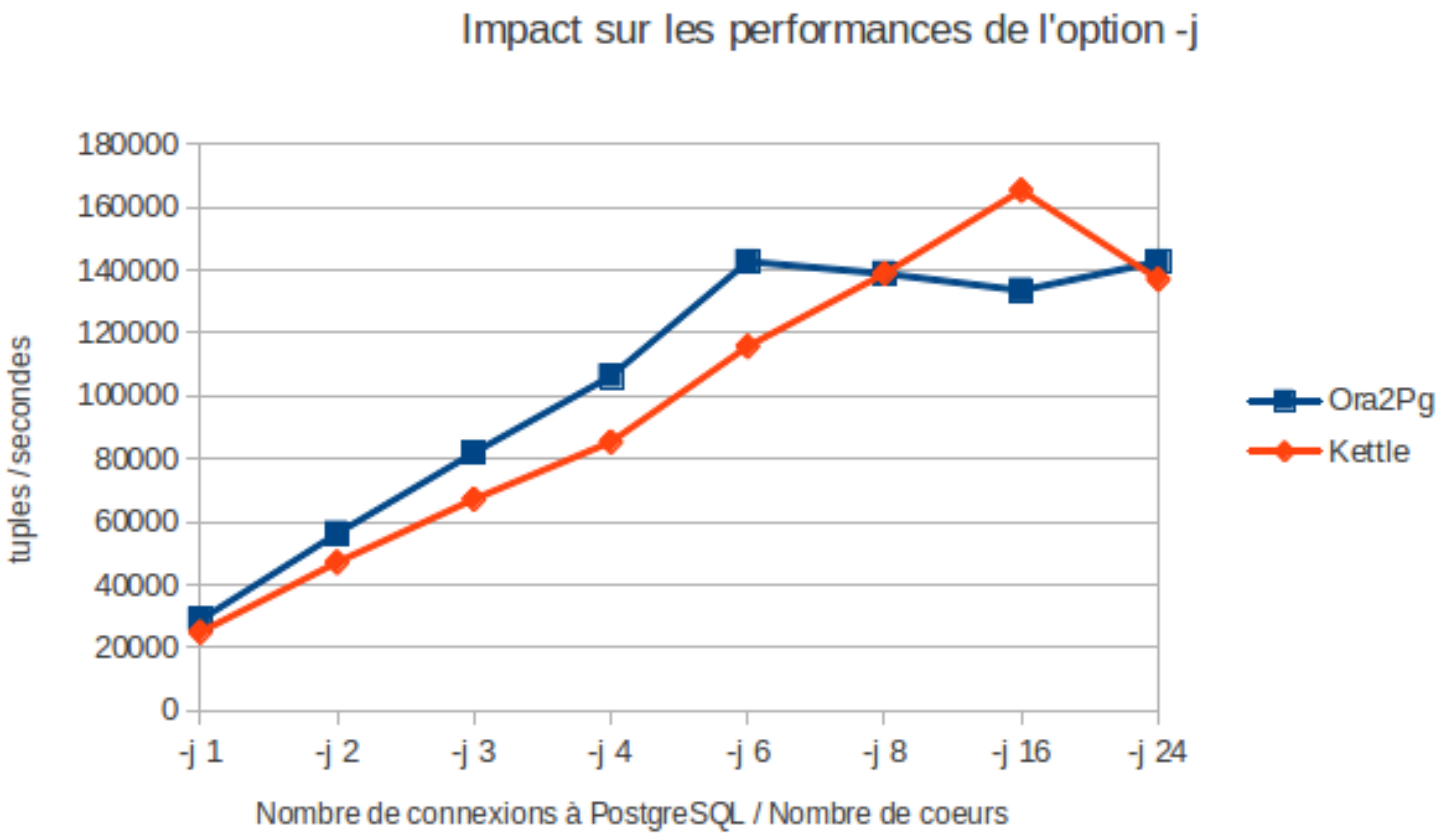
Benchmark table générique

- Structure PostgreSQL de la table utilisée pour le test :

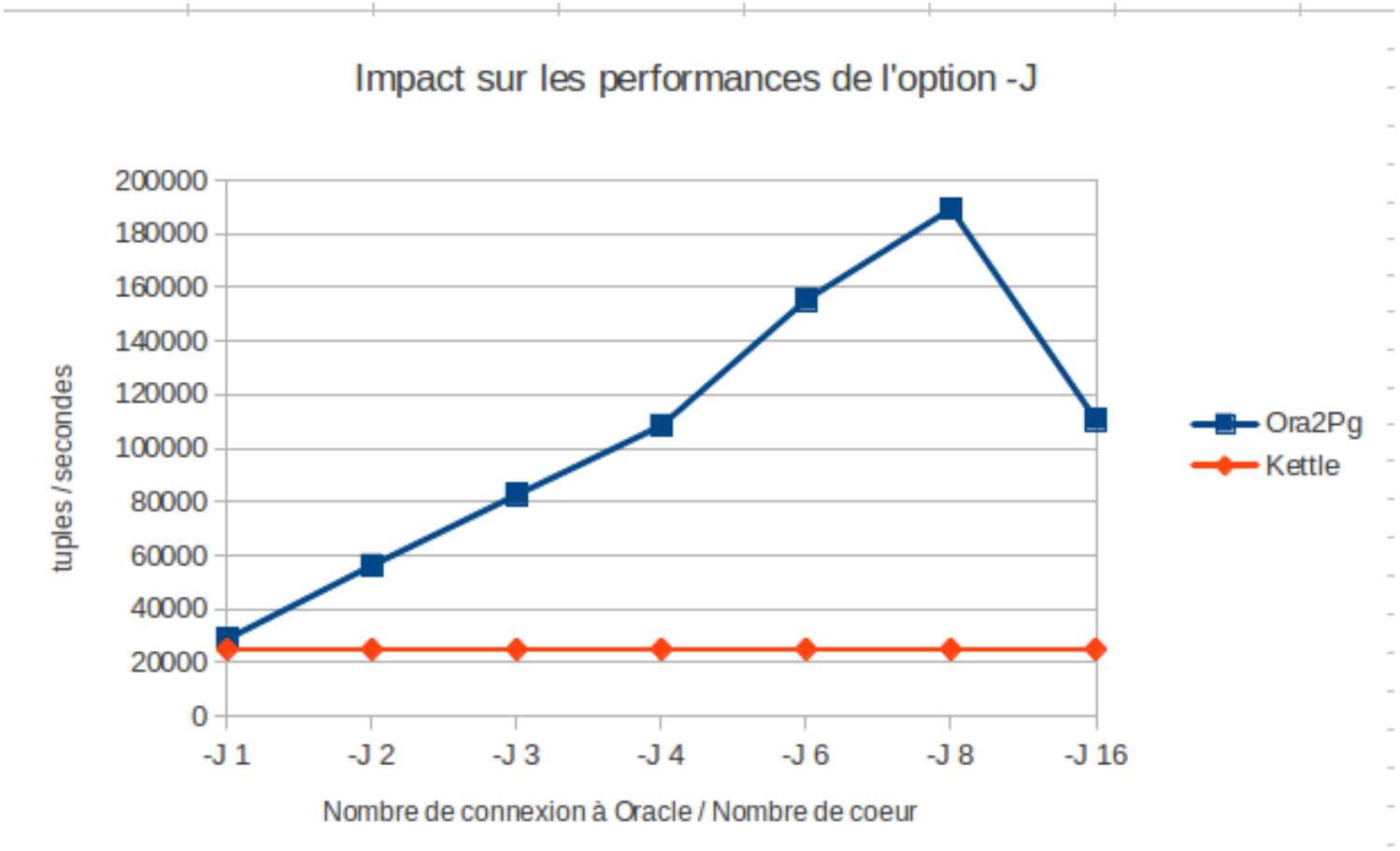
```
CREATE TABLE log_job (
    id_log bigint DEFAULT 0 NOT NULL,
    id_job bigint DEFAULT 0 NOT NULL,
    id_error bigint DEFAULT 0 NOT NULL,
    error_type integer DEFAULT 0 NOT NULL,
    error_message character varying(512),
    nom_object character varying(255),
    date_error timestamp without time zone
);
```

Nombre de tuples dans la table : 10 000 000

N connexions à PostgreSQL – 1 connexion à Oracle



N connexions à Oracle – 1 connexion à PostgreSQL



Meilleurs performances

- Ora2Pg : 8 connexions à Oracle (-J 8) et 3 connexions à PostgreSQL (-j 3)
  - Utilisation des 24 coeurs
  - 260271,8 tuples / sec
  - L'inverse (-j 8 et -J 3) : 236610,8 tuples/sec
- Kettle obtient ses meilleurs performances avec 2 connexions à Oracle (-J 2) et 16 à PostgreSQL (-j 16)

Données binaires (champs BLOB)

- Test sur une table de 1,2 Go et 260 000 enregistrements
- Ora2pg avec une et quatre connexions à PostgreSQL :
  - Ora2Pg -j 1 -L 1000 : 614 tuples / sec
  - Ora2Pg -j 4 -L 1000 : 614 tuples / sec
- Kettle avec 1 connexion à PostgreSQL :
  - 922 tuples / sec
- Kettle avec 2 connexions à PostgreSQL :
  - 1229 tuples / sec
- Kettle est plus performant dans les deux cas

Limitation des données

- Soit par exclusion de tables, dans le cas de données obsolètes ou temporaires
  - EXCLUDE HIST_.* TMP_.*
- Soit par exclusion temporaire le temps de la migration
  - WHERE TABLE1[DATE_CREATE >= '2013-01-01']
- le restant est importé lorsque la base est de nouveau en production
  - WHERE TABLE1[DATE_CREATE < '2013-01-01']
- Soit les deux.

Index et contraintes

- Chargement des index, contraintes et séquences
  - psql -h ip_address -U owner migdb < INDEXES_tables.sql
  - psql -h ip_address -U owner migdb < CONSTRAINTS_tables.sql
  - psql -h ip_address -U owner migdb < SEQUENCES_tables.sql
- Pas de parallélisme pour leur création
- Ils représentent souvent de 1 à 2 tiers du volume total de la base
- Parallélisme avec dispatcher_pg
  - https://github.com/marco44/dispatcher_pg
  - Permet de paralléliser des requêtes sur PostgreSQL

Conclusion

- Vous venez de voir les dernières évolutions d'Ora2Pg
- N'oubliez pas de remettre fsync et full_pages_write à on !!!
- Ordre d'idée sur les temps de migration :
  - Migration 1 To avant la v11 => 20-25 heures
  - Migration 1 To après la v11 => 10-15 heures

  } dépendant du schéma et streaming_replication

- Remerciement pour leur implication dans les versions 11.x
  - Marc Cousin
  - Photobox
  - Bouygues Telecom
  - Dalibo

Ora2Pg : Performances

Merci de votre attention.

Des questions ?

Version : 11.4 (2013-06-06)

Site officiel : http://ora2pg.darold.net/

Code : https://github.com/darold/ora2pg