Comparaison de tables à distance

Rapport de recherche A/375/CRI

Fabien Coelho – fabien@coelho.net

composé avec LaTeX, révision 463 du document

Fabien Coelho

job enseignant-chercheur à MINES ParisTech

ingénieur civil 1993, docteur en informatique 1996

recherche compilation pour architectures parallèles, un peu cryptographie

enseignement java, db, réseaux, responsable d'option...

- écoles d'ingénieurs, MS et Badges CGE
- formations continues : PostgreSQL, SVN, perl

base de données cours avec PostgreSQL

- pratique : correction automatique en ligne
- supports en français `http://www.coelho.net/cours/si/`

Contributions (mineures) à PostgreSQL

localisation des erreurs de syntaxe

```
fabien=# SELECT 2 - FROM users;
ERROR:  syntax error at or near "FROM"
LINE 1: SELECT 2 - FROM users;
                   ^
```

pgxs développement d'extensions infrastructure `make` de PostgreSQL

```
SCRIPTS    = pg_comparator
MODULES    = checksum casts
DATA_built = checksum.sql casts.sql
...

PGXS       := $(shell pg_config --pgxs)
include $(PGXS)
```

warning si erreur de type sur clefs étrangères

etc autres petits machins (*e.g.* ALSO)

Autres contributions (mineures) à des logiciels libres...

svn subversion, gestion de sources

- traduction français
- entretien de la *complétion bash*,
- quelques outils `rcs2svn`, `svn-merge-repos`...

apache serveur web

- module `mod_macro`

salix qualité des schémas relationnels. exemples :

- deux fois plus de tables dans un projet PostgreSQL vs MySQL
- 11% des tables MySQL et 21% des tables PgSQL sans PK
- 91% des bases MySQL et 57% des schémas PgSQL sans FK

comparator comparaison de tables

Problème abordé

– comparaison des données de deux tables relationnelles

  tuples ajoutés `INSERT`, retirés `DELETE`, modifiés `UPDATE`

– à distance : minimisation des communications

– dans deux bases de données relationnelles

  pas forcément les mêmes ! PostgreSQL, Oracle, MySQL...

Problème *NON* abordé

– synchronisation de *schémas* de base de données `ALTER`

Motivation

La confiance n’exclut pas le contrôle !

rapprochement vérification des données *set reconciliation problem*

- après transfert de données (dump/restore)
- réplication asynchrone
- réplication synchrone

recherche algorithmique !

- je peux faire mieux que ce que j’ai vu

Comparaisons. . .

delta compression sur données locales

sur textes : commande `diff`

à distance pas d'accès

données ordonnées ? fichiers ?

G. Maxia article dans *sysadmin*

arbre de hash, mais algorithmeS asymétriques

Algorithme de la commande `rsync`

- Andrew Trigell, P MacKerras 1996
- synchronisation à distance de fichiers, d'arborescence
- algorithme fondamentalement asymétrique
- communication : un seul retour-aller, volume non optimal
- succès plus lié au côté pratique ?

Contraintes et hypothèses

– algorithme relationnel !

  opérations disponibles dans une base relationnelle

  fonction de hash, aggrégations, tri

– identification des tuples **clef** !

– pas d'hypothèse sur les valeurs clefs, tuples, répartition

– petit nombre de différences

  sinon, autant tout transférer pour comparer. . .

Algorithme de comparaison : *arbre de hash*

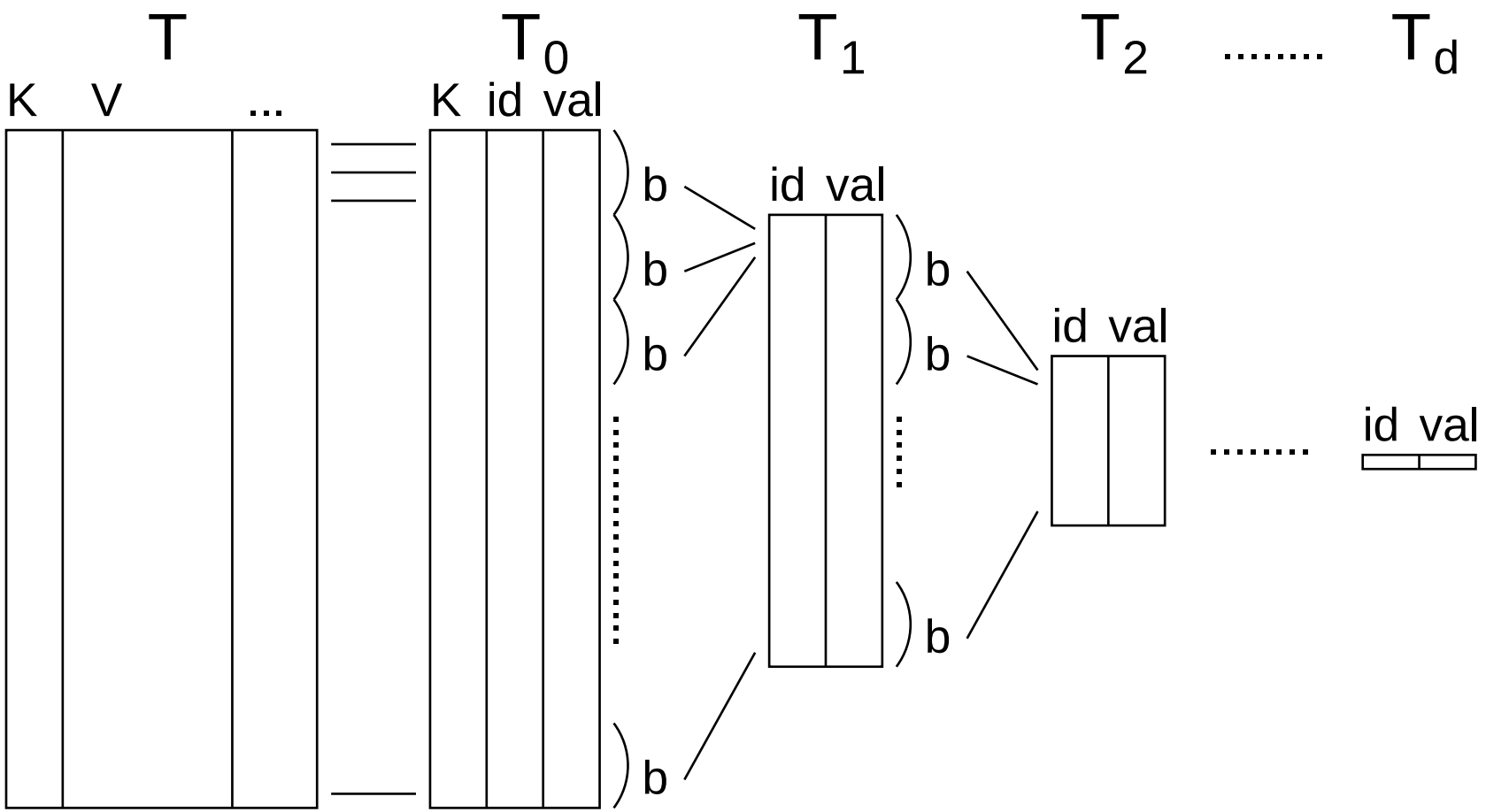
construction de résumé identiques des deux côtés

- *randomisation* des valeurs avec fonctions de hash
- structure hiérarchique arborescente : blocs de taille b
- combinaison des signatures : XOR (commutatif associatif)

réconciliation fusion descendante

- identification des clefs des tuples ajoutées/retirées/modifiées
- niveau par niveau

Construction des tables de résumés



Requêtes de construction

– m masques pour le regroupement **hiérarchique**

m_1 partie commune grande ...$m_d = 0$ un seul groupe

– h fonction de hashage pour randomisation

– W condition pour comparaison partielle

```
CREATE TEMP TABLE T_0 AS
SELECT
  K AS key,
  h(K) AS id,
  h(K,V) AS val
FROM T WHERE W;
```

```
CREATE TEMP TABLE T_i AS
SELECT
  id & m_i AS id,
  XOR(val) AS val
FROM T_{i-1}
GROUP BY id & m_i;
```

Quelques remarques sur la construction

parcourt complet nécessaire. . .

- construction des structures des **deux** côtés
- avec les mêmes paramètres ! *masques, fonctions, regroupements*

table racine T_0 aussi grande que la table initiale

- clef présente deux fois : valeur, et hash
- maintenable en permanence ? procédures automatiques `TRIGGER`

  donc pas nécessairement temporaires
- éventuellement attributs ajoutés à la table initiale ?

autres tables recalcul nécessaire ? !

T_d : id $= 0$ et val $= h(T)$

deux hashs clef seule (randomisation), clef+valeur (important !)

Réconciliation : fusion des tables par niveau

– part du dernier niveau T_d : signature de toute la table

– parcourt parallèle des clefs aggrégées **id** des deux côtés

`SELECT` ...

– valeur existe d'un seul côté ?

groupe de `INSERT`/`DELETE`

sans doute beaucoup de différences...

– valeur existe des deux côtés : comparaison des **id**

hash du contenu de la table, y compris les clefs

identique parfait ! tout le bloc est bon

différent investigation de **ce bloc** au niveau suivant

– dernier niveau T_0 : identification des clefs

Analyse de l'algorithme

paramètres n tuples, k différences, b taille bloc, c taille hash

profondeur $d = \lceil \frac{\log_2 n}{\log_2 b} \rceil$

nombre de requêtes effectuées $\mathcal{O}(d)$ b grand

construction $d + 1$

réconciliation 1 (pas de modication) à $d + 1$

attention à la taille des requêtes : nombre de modification !

communication selon nombre de différences

k **petit** $\mathcal{O}(kcbd)$ c et b petits ?

k **grand** $k \approx n$, récupère tout T_0 : $\mathcal{O}(cn)$

Implémentation *expérimentale*

`pg_comparator`

script perl *Proof of Concept* version 1.4.4, juin 2008

auto documenté

extensions utiles, installées avec **pgxs**

- petits hashs (suffisants) `INT2 INT4 INT8`
- casts `VARBIT` de/vers `BYTEA` refusé dans *PostgreSQL core*
- aggrégation `XOR` refusé dans *PostgreSQL core*

résultat rapporte les différences trouvées

free logiciel libre disponible `http://www.coelho.net/`

NE FAIT RIEN D'UTILE :-)

Syntaxe : `pg_comparator src dst`

– identification des données

`login:pass@host:5432/base/schema.table?id:attx,atty`

valeurs par défaut...

– très paramétrable via options...

threads (= coredump) ; verbose ; stats ; checksum ; cutoff ;

– 90% du temps passé dans la construction de T_0

```
sh> pg_comparator \
   localhost/family/calvin?id:data \
   sablons/family/calvin
INSERT 12
DELETE 25
UPDATE 17
...
```

Test en local

```
sql> SELECT pg_size_pretty(pg_relation_size('foo1')), COUNT(*)
     FROM foo1;
211 MB, 999999

vailly> time pg_comparator --verbose --stats \
      /fabien/foo1?id:c1,c2 \
      /fabien/foo2?id:c3,c4

UPDATE 200000
INSERT 818181
UPDATE 666666
DELETE 333333
UPDATE 200001
UPDATE 125000

real    0m55.389s
...
```

Expériences

- temps complet de la comparaison

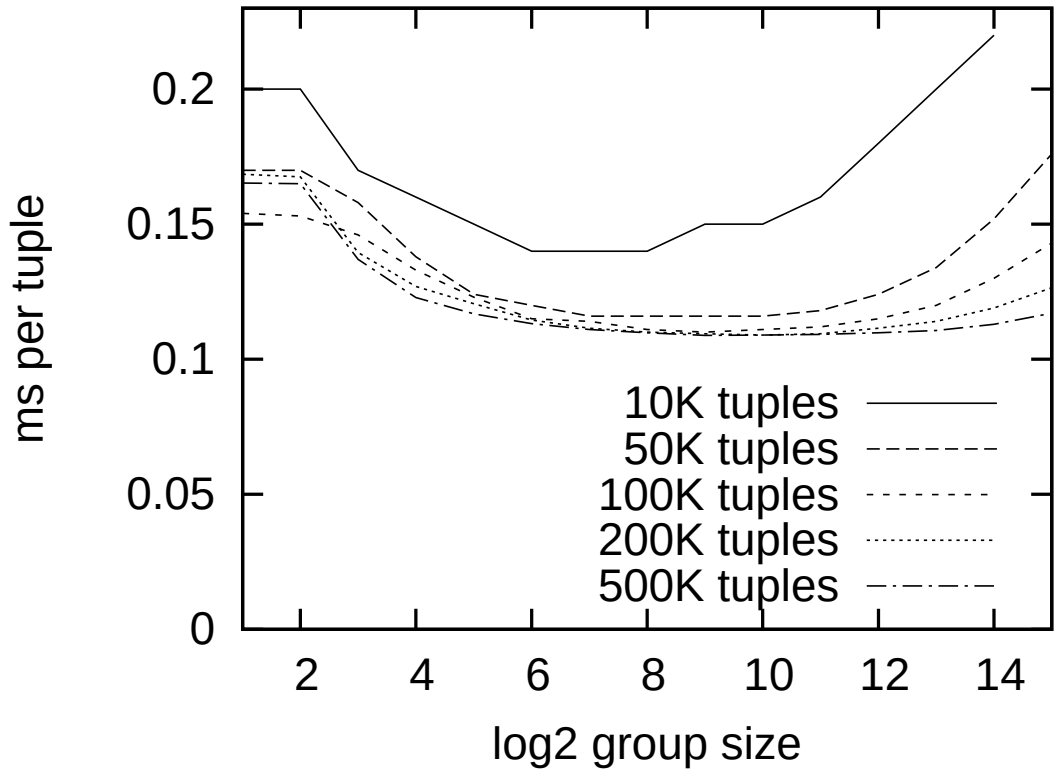
  construction, requêtes, fusion...

  éventuellement normalisé par tuple
- tailles de blocs de regroupement : $2 \ldots 2^{15}$

  puissance de deux car utilise des masques
- tailles de tables : 10K à 500K tuples
- nombre de différences à trouver : 3 à 100 à 2400
- bande passante réseau `tc` : LAN, avg, low

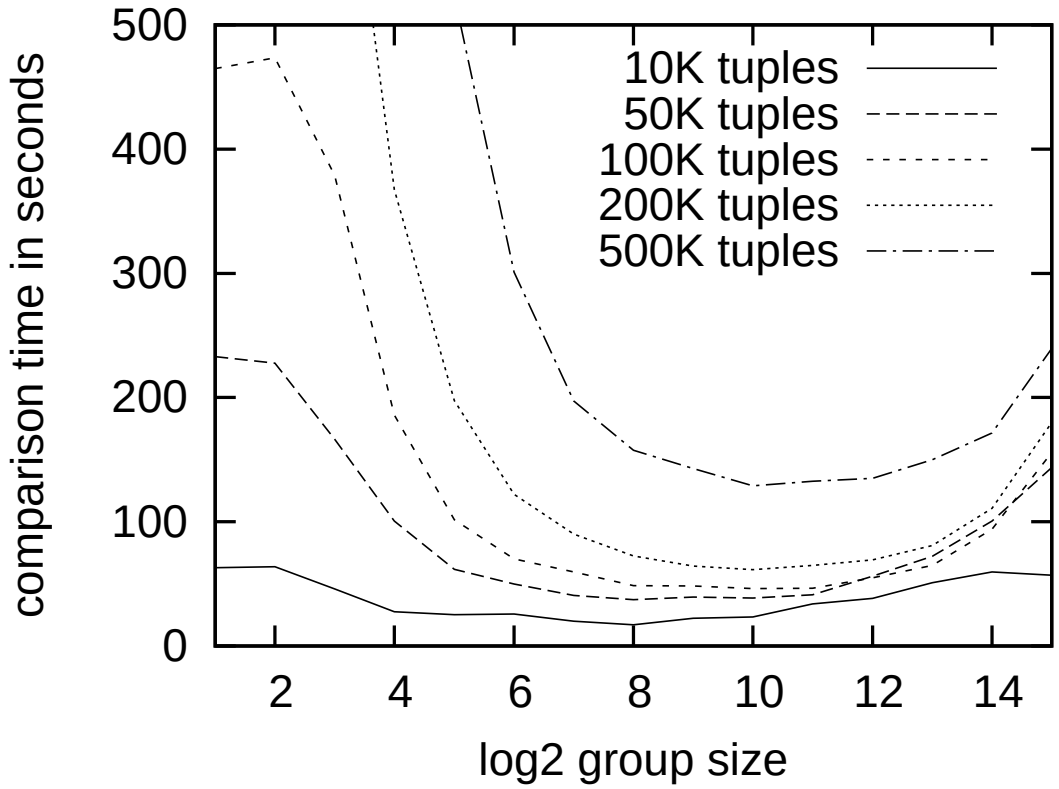
ms/tuple selon taille de groupe et de tables

bande passante LAN 100Mb/s

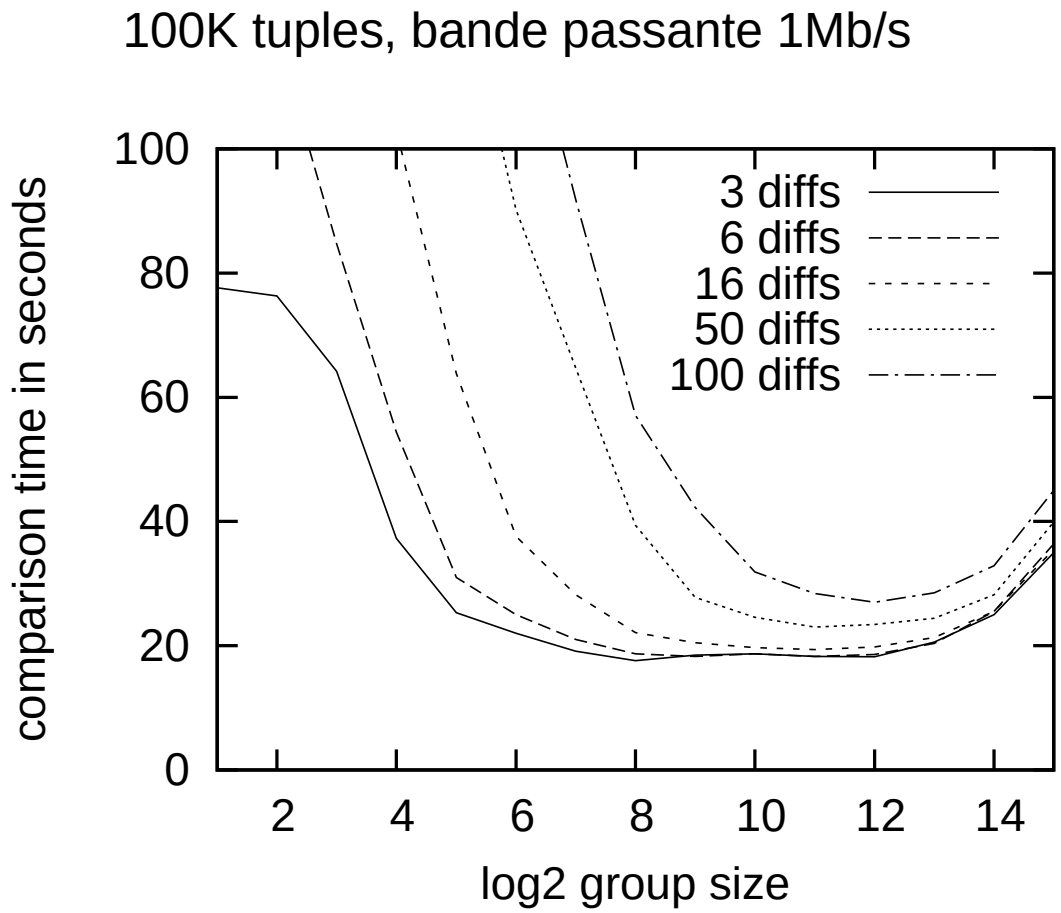


Temps selon taille de groupe et de tables

bande passante 64Kb/s

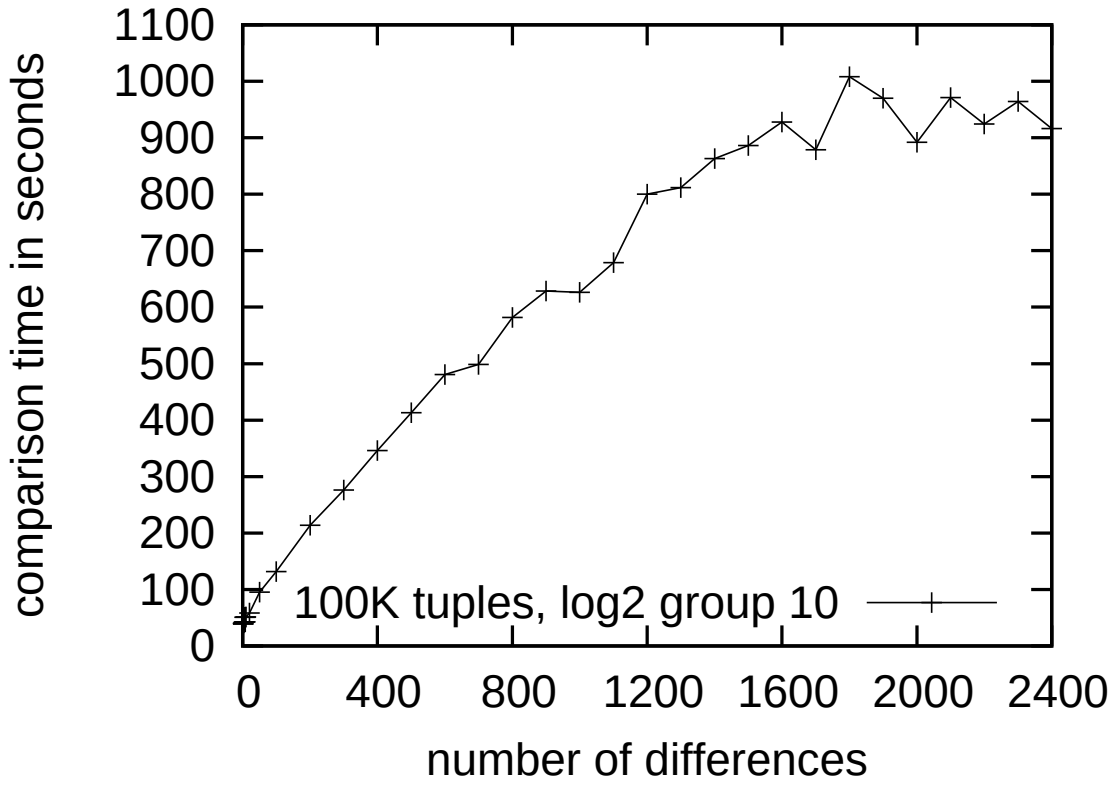


Temps selon taille de groupe et nombre de différence



Temps selon le nombre de différences

linéaire, puis saturation



Remarques

– taille assez grande des blocs $2^{10...12}$ donc $d \leq 3$ en pratique

un peu surprenant, mais bon !

multi niveau, mais pas beaucoup de niveaux :-)

Un utilisateur : Erik Aronesty, *Moon Costumes*

- utilise comme système de réplication asynchrone

  a ajouté une option `exec` à sa version
- synchronisation de deux tables toutes les dix minutes

  23039 lignes, 284 octets/tuples environ 6.2 Mo

  3.3 secondes de synchronisation sur réseau 3 Mb/s
- sauvegarde complète (dump) toutes les 24 heures

Nouvelle implémentation : Michael Nacos

`http://pgdba.net/pg51g/`

- a lu le rapport et réimplémenté l'algorithme
- C, `TRIGGER`, `MD5`...
- une seule table, niveau en attribut

Limitations de l'algorithme

- efficace si peu de modifications
- inefficace si beaucoup de modifications ?
- ne profite pas les groupements natifs (indexés ?)

Conclusion

- un algorithme élégant
- un lecteur de rapport :-)
- un utilisateur satisfait :-)

Évaluation anonyme en ligne

`http://2009.pgday.eu/feedback`

List of Slides

11 Construction des tables de résumés

12 Requêtes de construction

13 Quelques remarques sur la construction

14 Réconciliation : fusion des tables par niveau

15 Analyse de l’algorithme

16 Implémentation *expérimentale*

17 Syntaxe : `pg_comparator src dst`

18 Test en local

19 Expériences

20 ms/tuple selon taille de groupe et de tables

21 Temps selon taille de groupe et de tables

22 Temps selon taille de groupe et nombre de différence

23 Temps selon le nombre de différences