Bacula et PostgreSQL, optimisation et retour d'expérience

Eric Bollengier / Marc Cousin

Plan 1/2

- Présentation Bacula
  - Présentation
  - Historique
  - Architecture
- Le catalogue des sauvegardes
  - Schéma simplifié du catalogue
  - Problématiques
  - Code d'insertion original
  - Pourquoi cela ne fonctionne pas sous PostgreSQL

Plan 2/2

- Solutions proposées
  - Procédures stockées
  - Solution dite « Mode Batch »
- Situation actuelle
  - Quelques chiffres sympas
  - Performances sur la restauration
- Pourquoi préférer PostgreSQL ?

Présentation

- Bacula : solution de sauvegarde réseau *BSD, Linux, Mac OS X, Unix, Windows.
- Projet initial :
  - Sauvegarde du Palm au Mainframe
  - Fonctions “Enterprise”
  - Relecture des données pour une période de 30 ans (si materiel adéquat)
  - Scalable
  - Licence libre (GPL v2) - FSFE

Historique 1/2

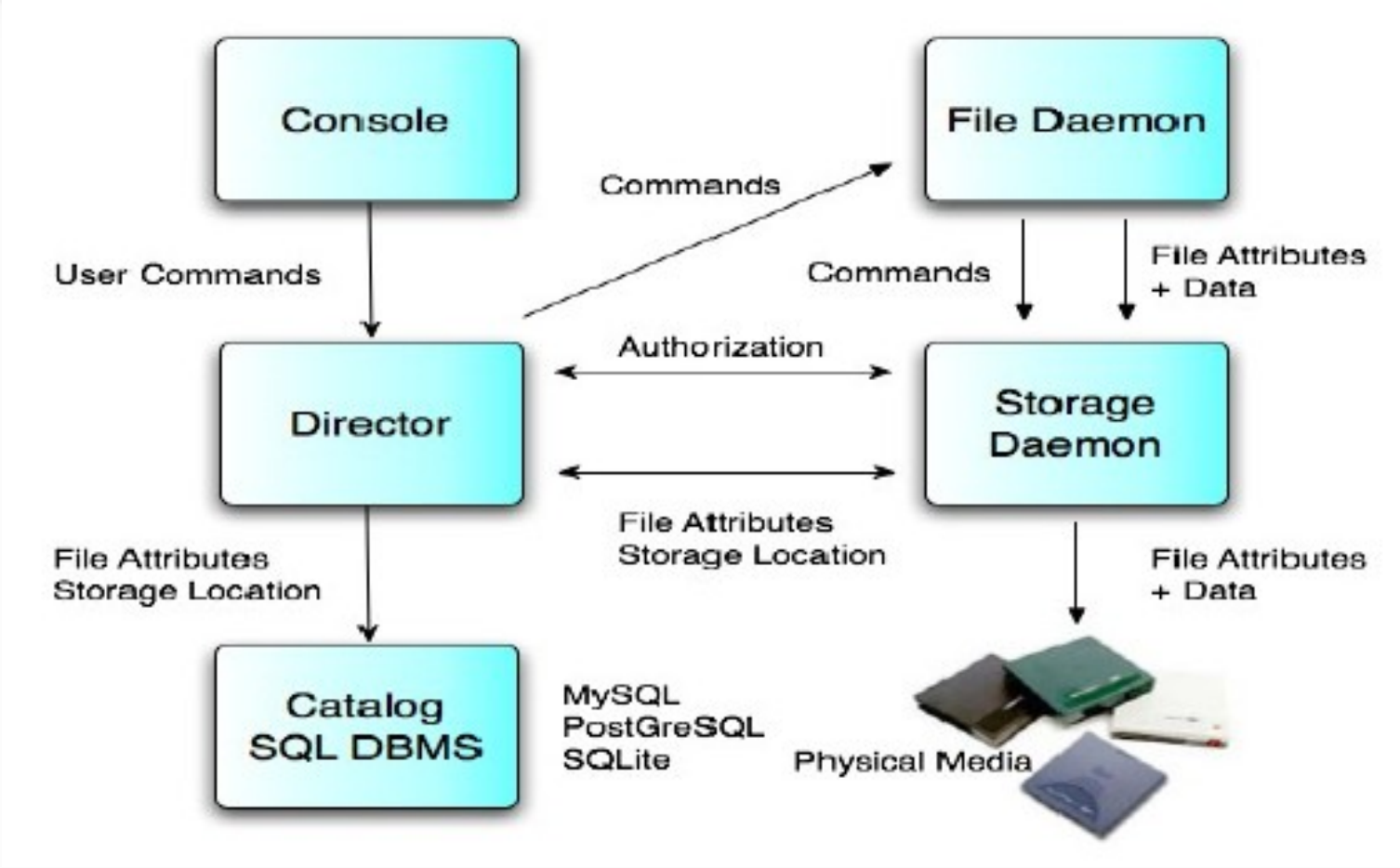
Bacula = Backup + Dracula

- Janvier 2000 – Démarrage du Projet – Kern Sibbald
  - 14 Avril 2002 – 1er version sur Source Forge (version 1.16)
  - 29 Juin 2006 – Version 1.38.11
  - Janvier 2007 – Version 2.0.0 ←
  - Aout 2007 – Version 2.2.0
  - Aout 2008 – Version 2.4.0
  - Avril 2009 – Version 3.0.0 (actuellement 3.0.3)
- + de 1 million de téléchargements (toutes versions)

Historique 2/2

- Catalogue SQL
  - *État de l'art* en 2002 : catalogue *spécialisé*
  - Nombreuses critiques sur l'utilisation de SQL
  - Mysql, SQLite puis PostgreSQL
  - Une connexion SQL pour l'ensemble des threads

Architecture



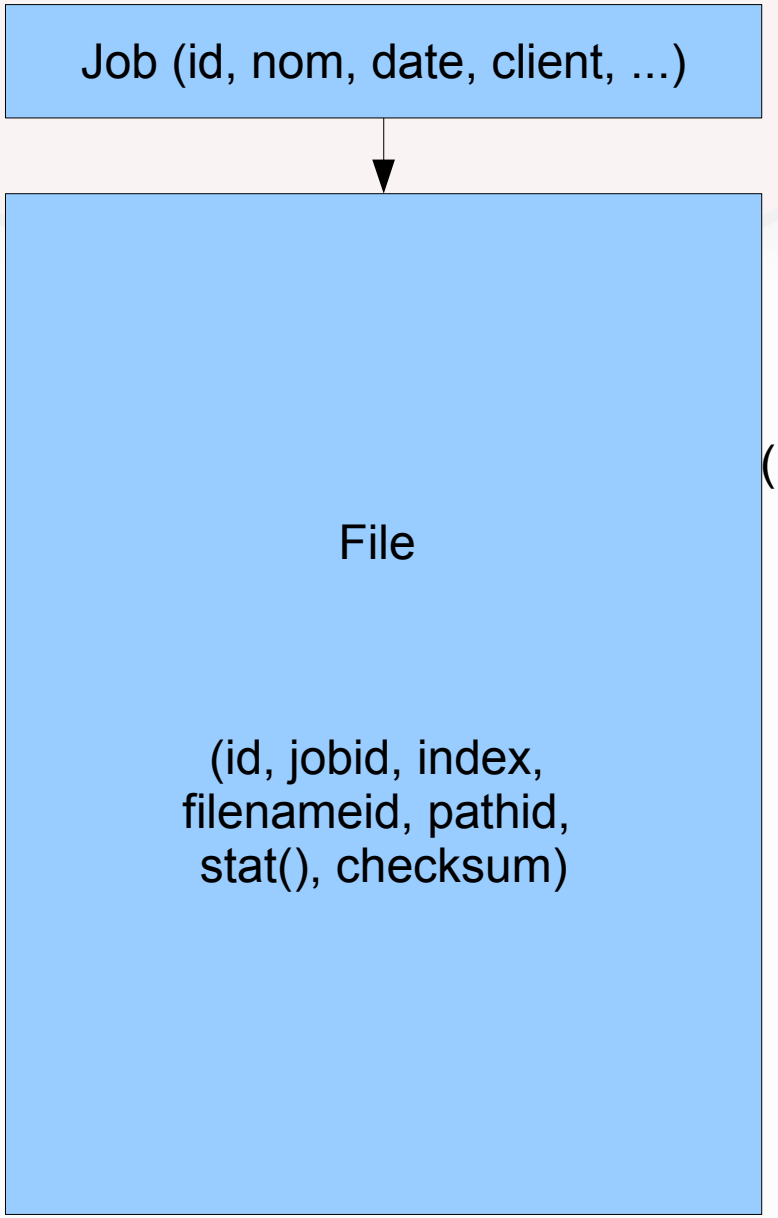
Chaque élément peut se trouver sur un serveur différent

Schéma simplifié du catalogue

Le catalogue des sauvegardes

Job (id, nom, date, client, ...) (1 tuple par sauvegarde)

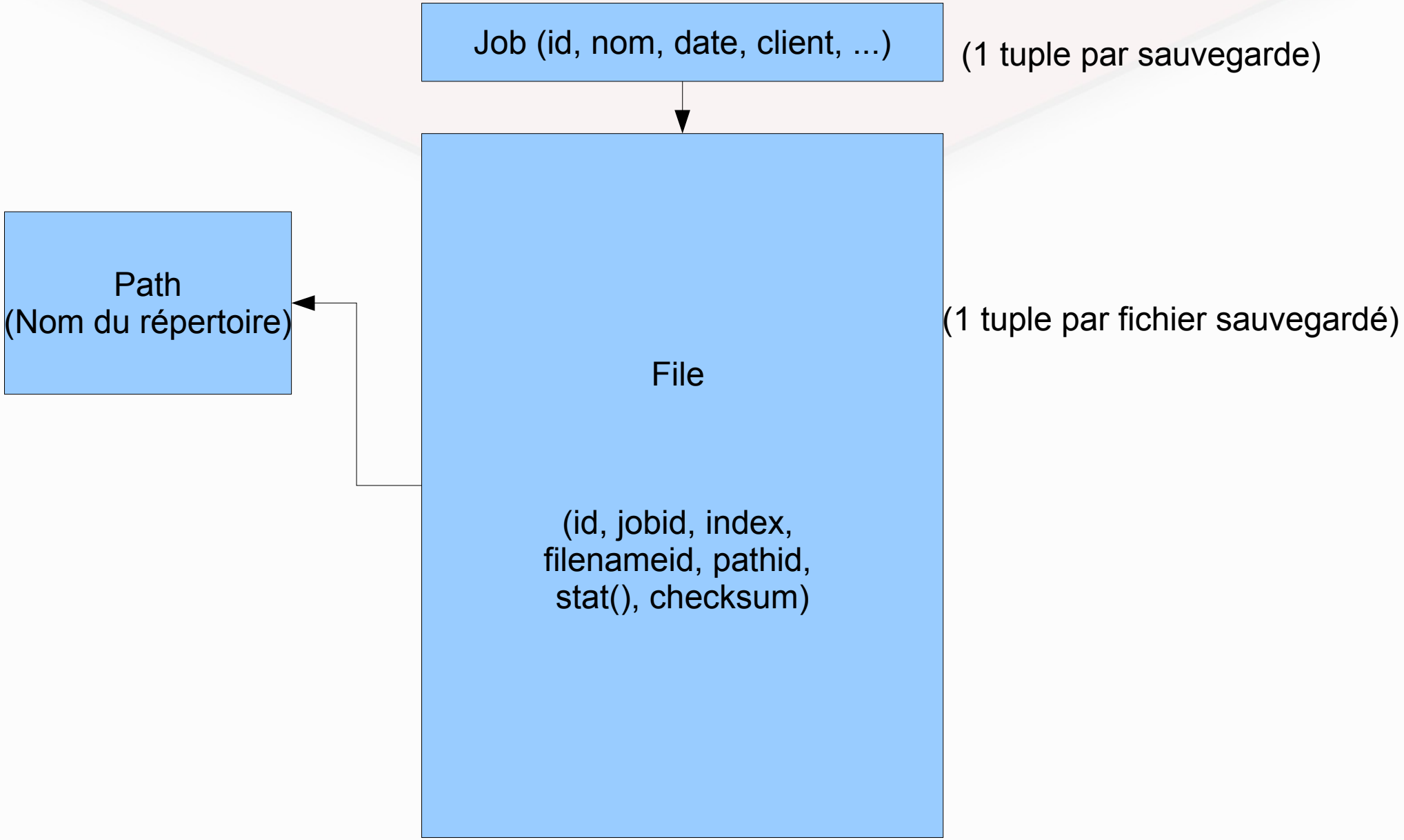
Le catalogue des sauvegardes



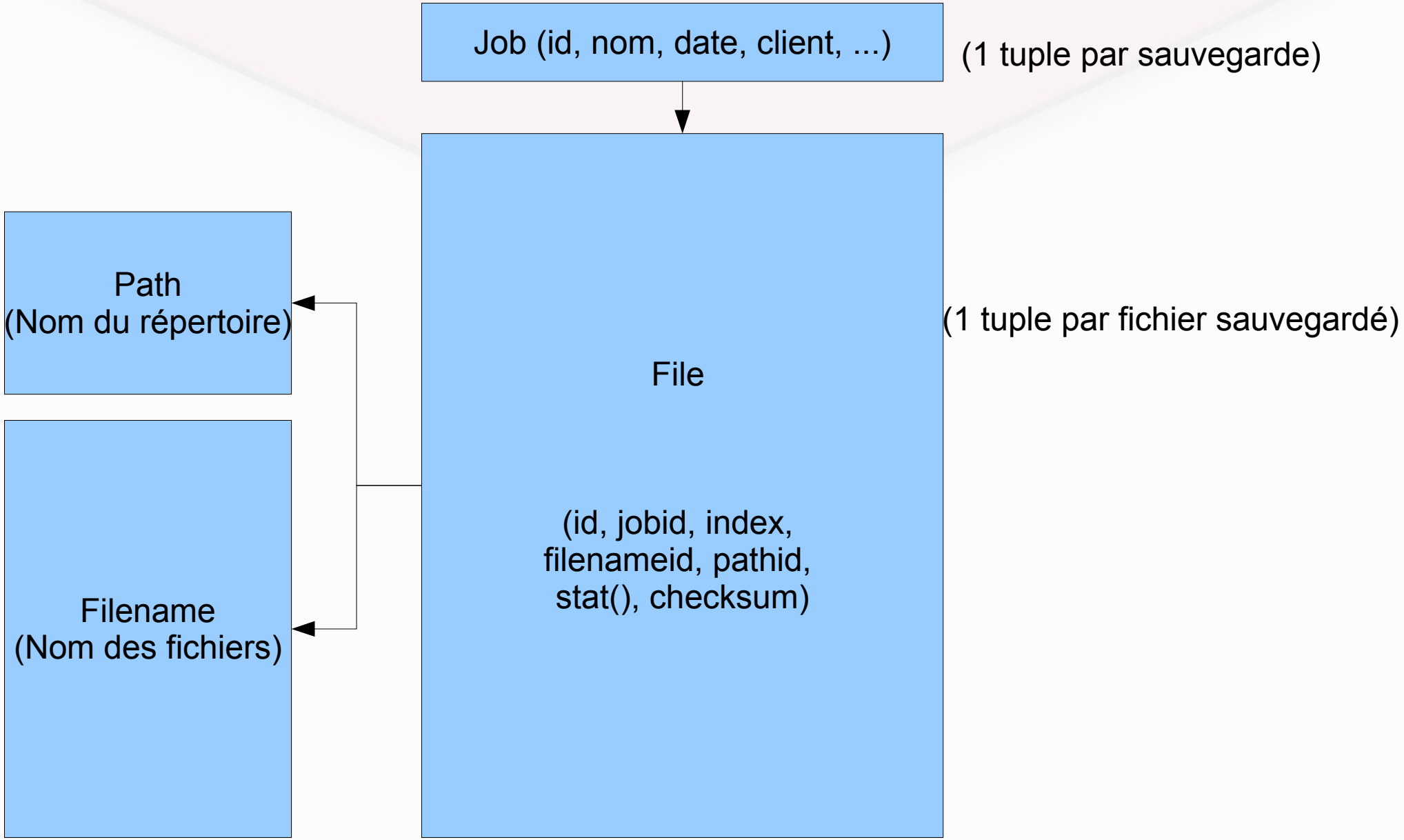
(1 tuple par sauvegarde)

(1 tuple par fichier sauvegardé)

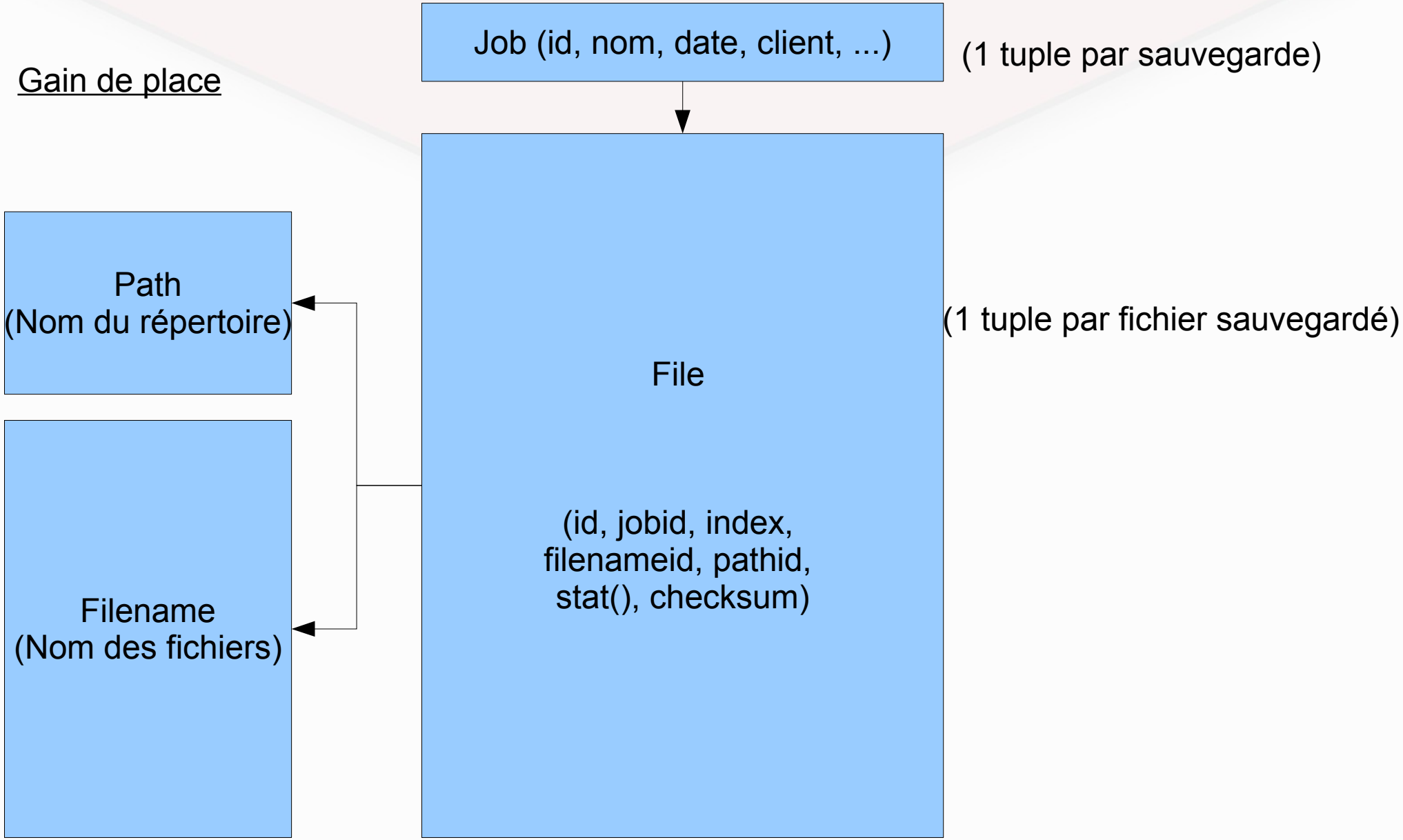
Le catalogue des sauvegardes



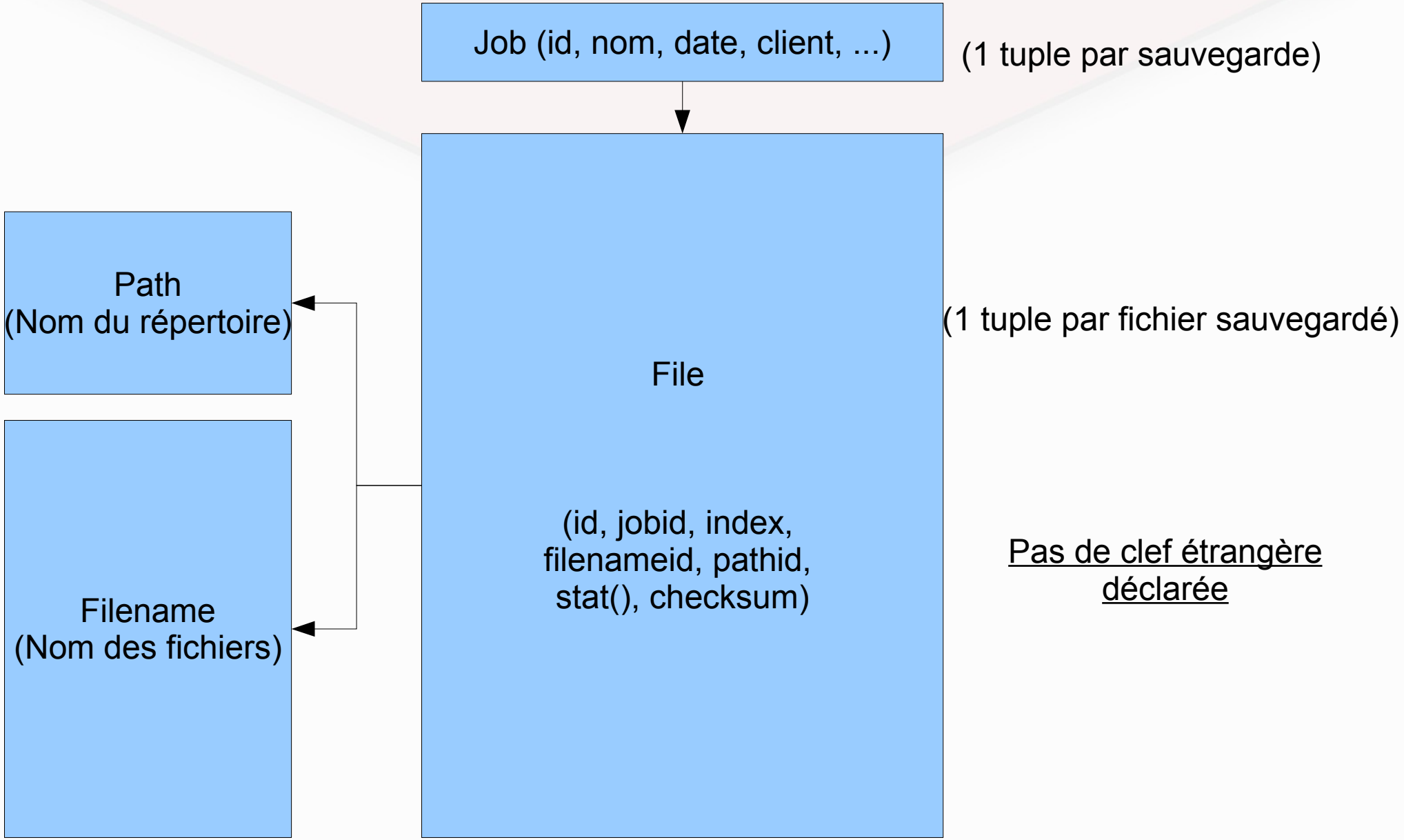
Le catalogue des sauvegardes



Le catalogue des sauvegardes



Le catalogue des sauvegardes



Procédure historique

Ancienne procédure 1/6

Fichier	/etc/passwd	Index	Lstat	Checksum

Lock de « la » connexion SQL
pour toute la procédure

Ancienne procédure 2/6

/etc/passwd	Index	Lstat	Checksum

Cherche « passwd » dans la table Filename

Filename

Ancienne procédure 3/6

/etc/passwd	Index	Lstat	Checksum

Cherche « passwd » dans la table Filename

Si absent, insert

Filename

Ancienne procédure 4/6

/etc/passwd	Index	Lstat	Checksum

Cherche « passwd » dans la table Filename

→ Si absent, insert

Filename

Cherche « /etc/ » dans la table Path

Path

Ancienne procédure 5/6

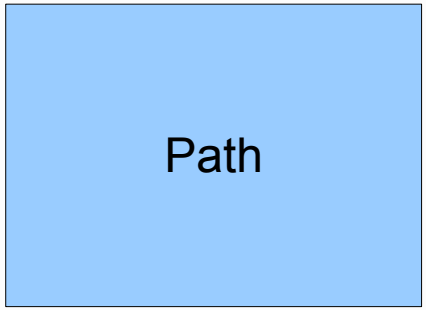
/etc/passwd	Index	Lstat	Checksum

Cherche « passwd » dans la table Filename

→ Si absent, insert

Cherche « /etc/ » dans la table Path

→ Si absent, insert



Ancienne procédure 6/6

/etc/passwd	Index	Lstat	Checksum

Cherche « passwd » dans la table Filename

→ Si absent, insert

Filename

Cherche « /etc/ » dans la table Path

→ Si absent, insert

Path

Insert (PathId, FilenameId, Index, Lstat, Checksum)

File

PostgreSQL est lent !

- Une transaction par opération (autocommit!)
- Une seule session à la base
- Temps d'aller et retour entre le director et la base pour chaque opération
- Jusqu'à 5 requêtes pour chaque fichier
- Reparsing de chaque requête
- Perfs acceptables pour MyIsam et SQLite 2, pas pour PostgreSQL, InnoDB, SQLite 3

Première tentative d'optimisation

- Communauté => fsync = off
- Solution «instinctive» pour un DBA : procédure stockée ! Réduit les problèmes de dialogue entre SGBD et director, et pas de parsing
- Gain : X2, peu intrusif au niveau du code

Rejeté : propriétaire à Postgres, ne s'attaque qu'à une partie des problèmes

Procédure actuelle « Mode Batch »

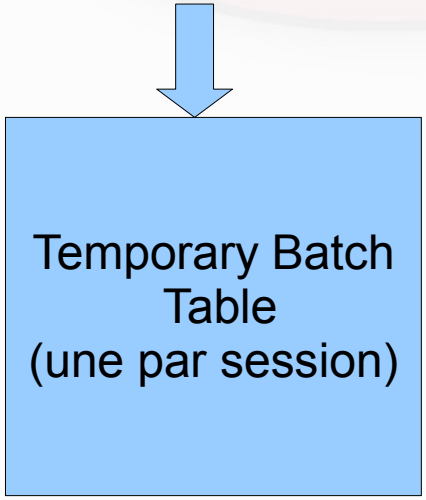
Insertion « Batch » 1/5

Fichier	/etc/passwd	Index	Lstat	Checksum

Insertion « Batch » 2/5

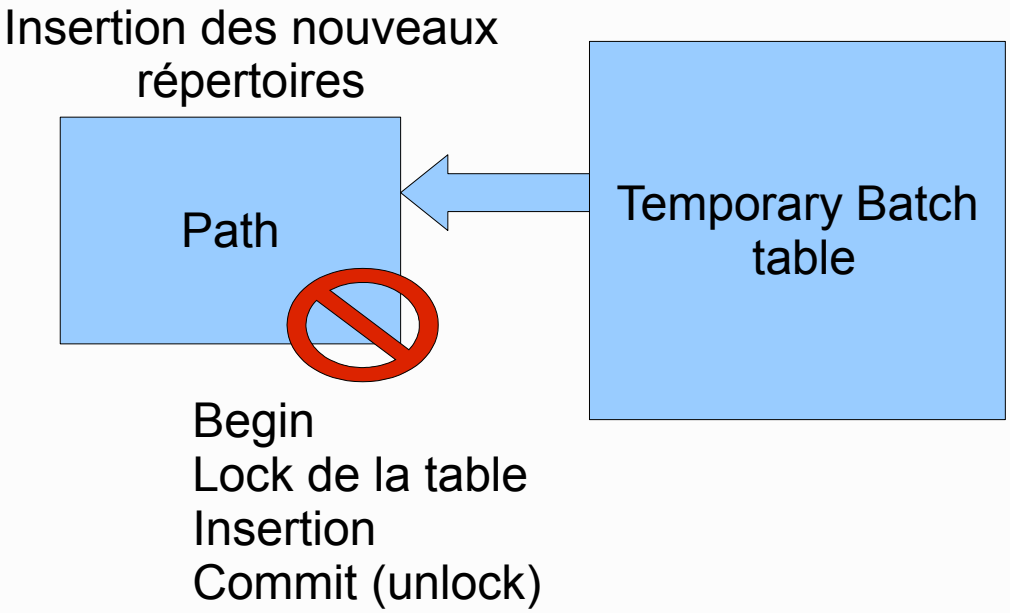
/etc/passwd	Index	Lstat	Checksum

insert dans une table temporaire
Via une connexion dédiée + COPY sous Postgres

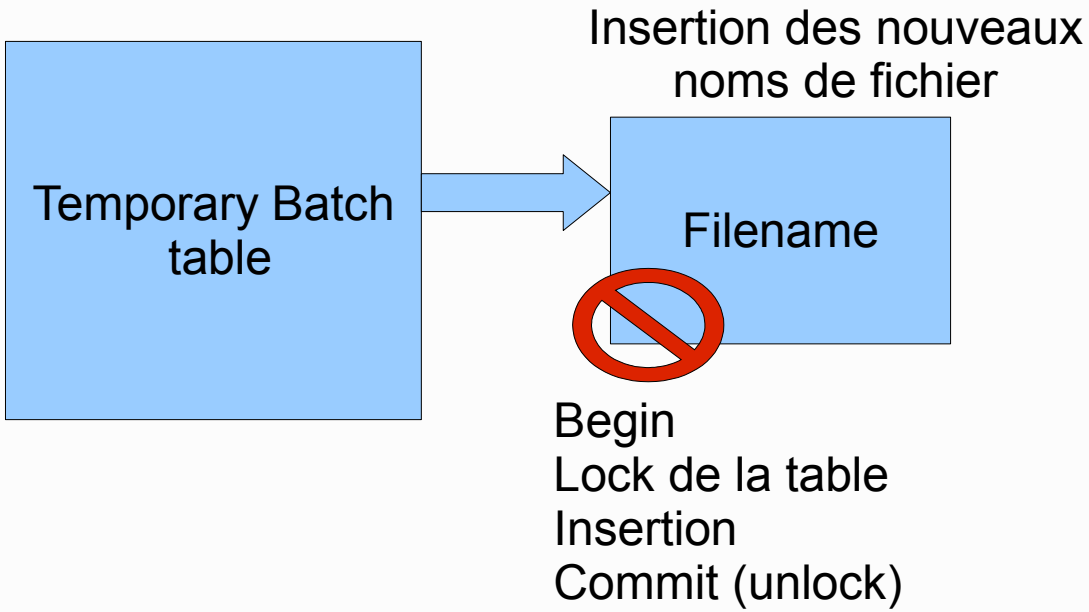


Insertion « Batch » 3/5

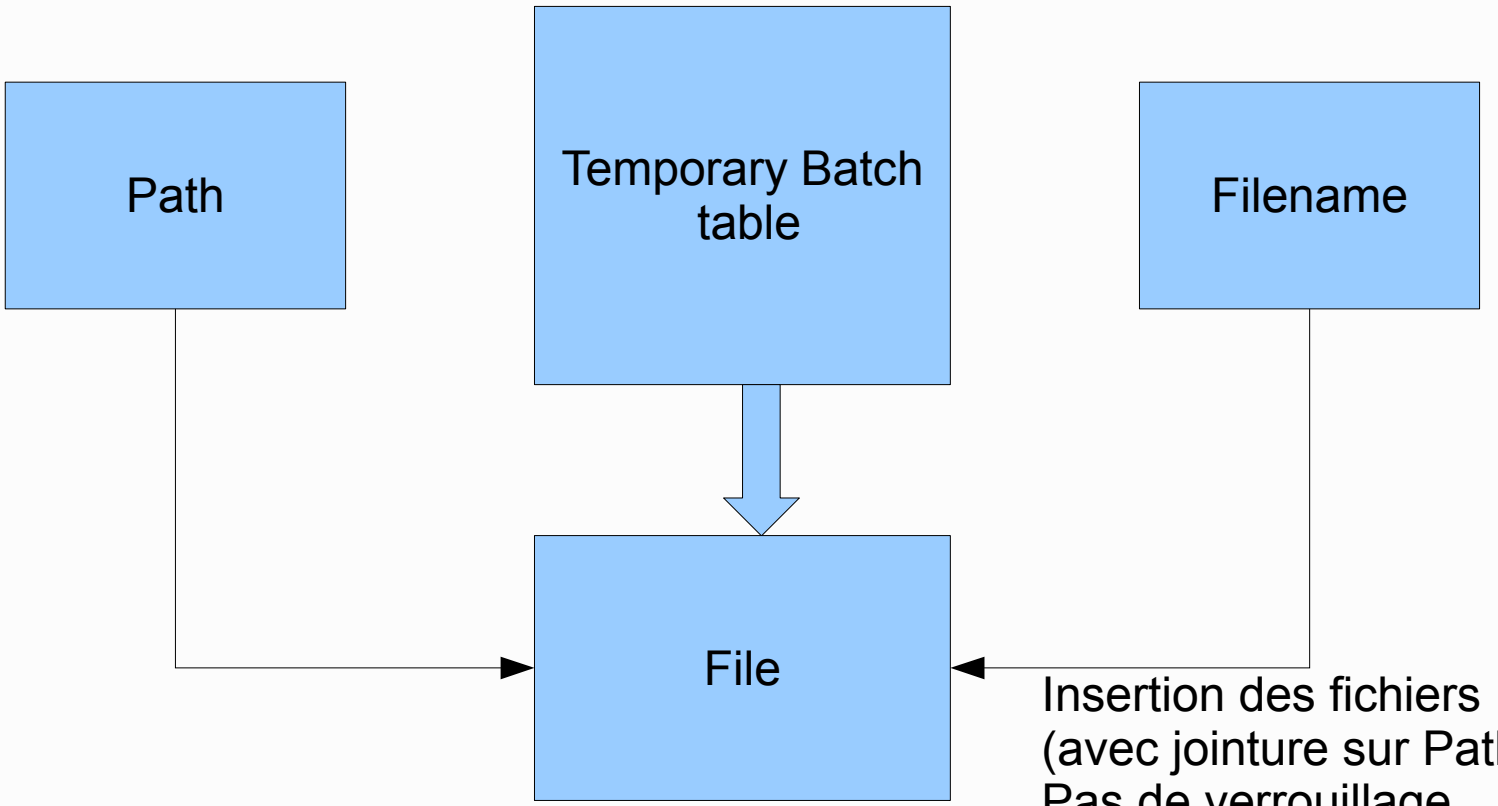
Backup terminé, ou bien limite atteinte.



Insertion « Batch » 4/5



Insertion « Batch » 5/5



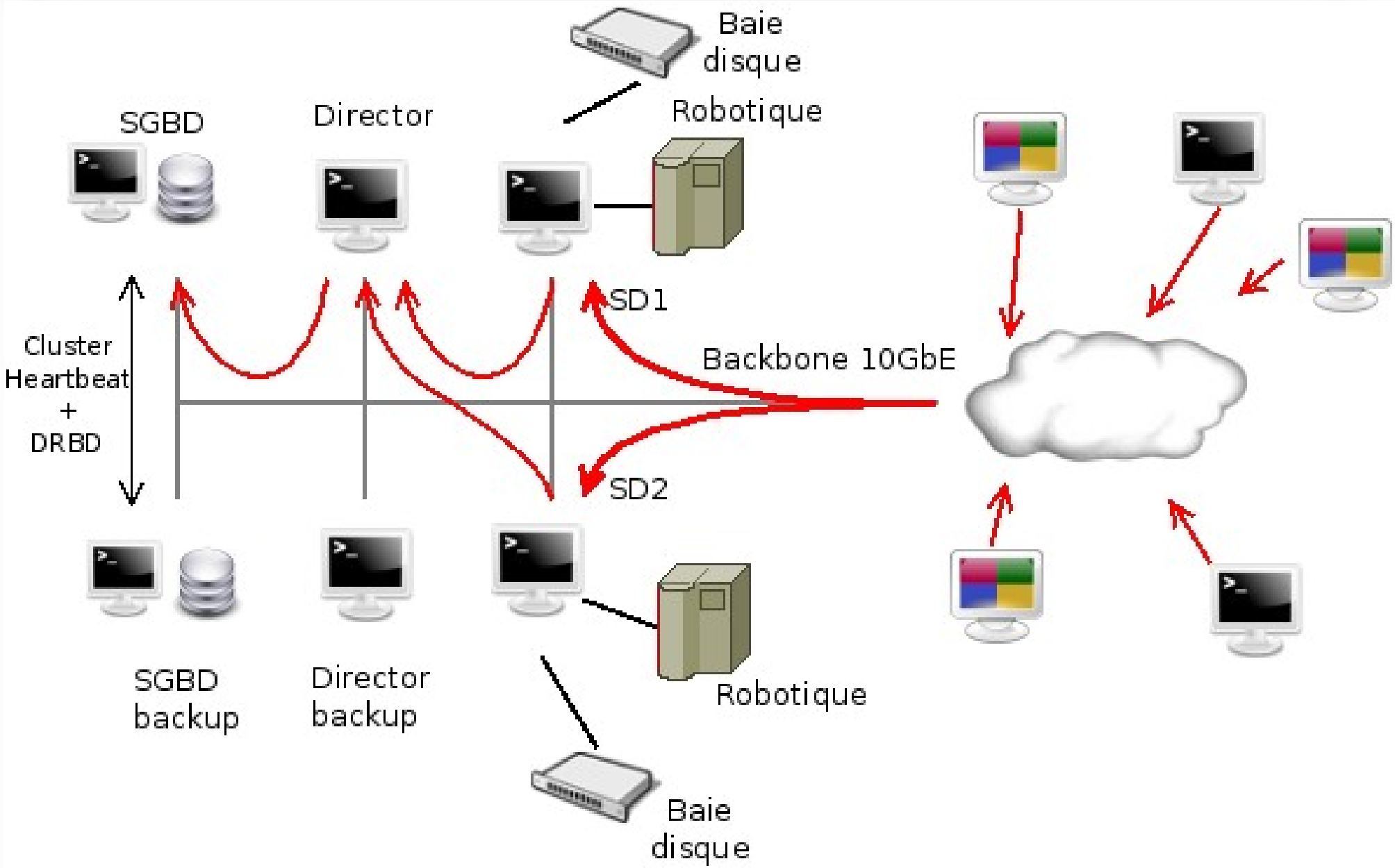
Insertion des fichiers
(avec jointure sur Path et Filename)
Pas de verrouillage

En production : X 10

En production…

- File :
  - 1,1 milliard de tuples
  - Heap : 150 Go
  - Index : PK : 30 Go, (jobid, pathid, filenameid) : 50 G
- Filename :
  - 100 millions de tuples
  - Heap : 7Go
  - Index : 10 Go
- Path :
  - 23 millions de tuples
  - Heap : 3 Go
  - Index : 3 Go

En production…



En production…

Côté sauvegardes :

- Chaque dimanche :

  50 millions de fichiers, 10 To, 373 jobs

- Week-end entier :

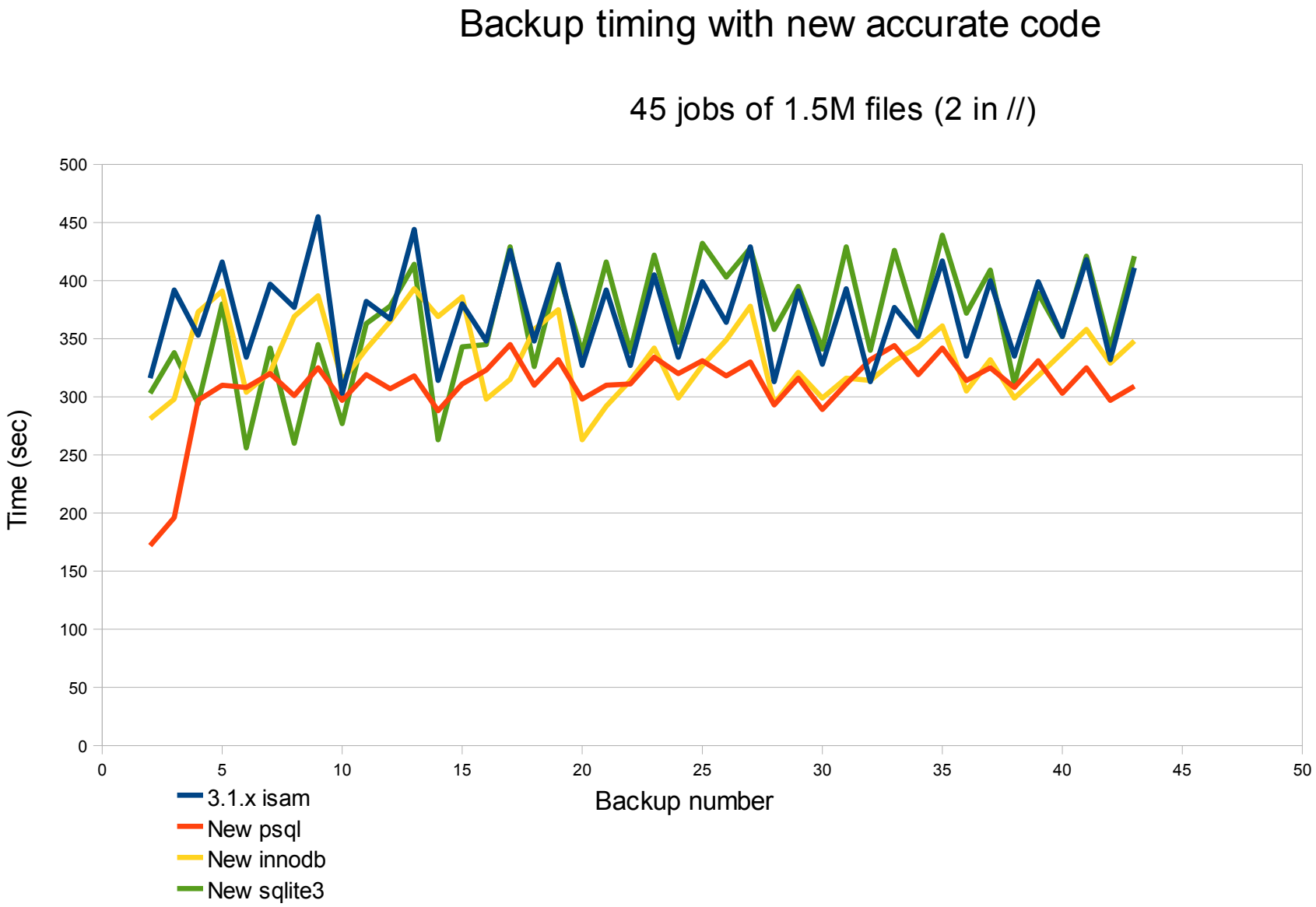
  73 millions de fichiers, 18 To, 670 jobs

- Chaque semaine :

  120 millions de fichiers, 43 To, 1850 jobs

- En ligne dans le système de sauvegardes : 292To/1.1 milliard de fichiers

<troll>Comparaison entre moteur</troll>

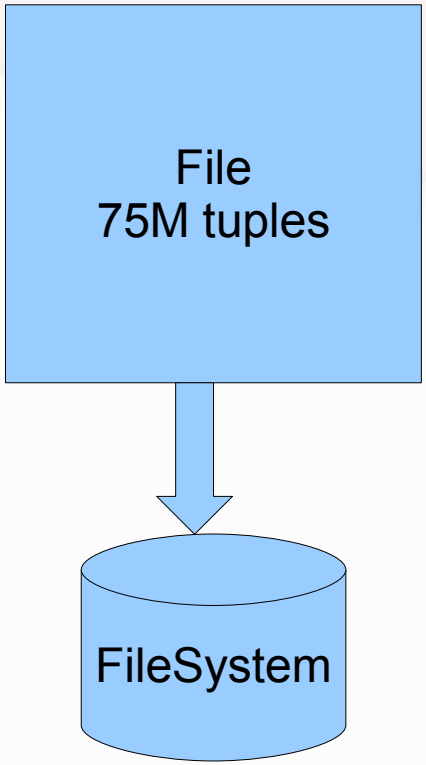


Et la restauration dans tout ça ?

Vitesse Sauvegarde

Temps Restauration

Et la restauration dans tout ça ?



Sélection des fichiers :
1.5M fichiers sur 50 jobs
~ 60s

+

Ecriture du disque (reiserfs)

=

~ 5mins

Sélection des fichiers pour la restauration

```
SELECT Path.Path, Filename.Name, Temp.FileIndex,
       Temp.JobId, Temp.LStat
  FROM (

   SELECT DISTINCT ON (FilenameId, PathId)
          StartTime, JobId, FileId,
          FileIndex, PathId, FilenameId, LStat
     FROM File JOIN Job USING (JobId)
    WHERE JobId IN (1,2,3,4)
    ORDER BY FilenameId, PathId, StartTime DESC

 ) AS Temp
 JOIN Filename ON (Filename.FilenameId = Temp.FilenameId)
 JOIN Path ON (Path.PathId = Temp.PathId)
WHERE Temp.FileIndex > 0
ORDER BY Temp.JobId, Temp.FileIndex ASC
```

Pourquoi préférer PostgreSQL ?

- Administration « maîtrisée »
- Efficacité sur purges
- COPY
- Stabilité de PostgreSQL

Contact et Questions

Marc Cousin
cousinmarc@gmail.com

Eric Bollengier
eric.bollengier@baculasystems.com

PS : Sur youtube, rechercher :
'faroult sql practice' (merci à lui)