PostgreSQL Sessions #3 : Keynote

1 À propos de l'auteur

» Auteur : Jean-Paul Argudo
» Société : DALIBO
» Date : Février 2012
» URL : https://support.dalibo.com/kb/conferences/keynote_pgsessions_3/

2 Licence

- Licence Creative Common BY-NC-SA
- 3 contraintes de partage :
  - Citer la source (dalibo)
  - Pas d'utilisation commerciale
  - Partager sous licence BY-NC-SA

Cette conférence (diapositives et diapositives commentées) est sous licence **CC-BY-NC-SA**.

Vous êtes libre de redistribuer et/ou modifier cette création selon les conditions suivantes :

- Paternité
- Pas d'utilisation commerciale
- Partage des conditions initiales à l'identique

Vous devez citer le nom de l'auteur original de la manière indiquée par l'auteur de l'œuvre ou le titulaire des droits qui vous confère cette autorisation (mais pas d'une manière qui suggérerait qu'ils vous soutiennent ou approuvent votre utilisation de l'œuvre).

Vous n'avez pas le droit d'utiliser cette création à des fins commerciales.

Si vous modifiez, transformez ou adaptez cette création, vous n'avez le droit de distribuer la création qui en résulte que sous un contrat identique à celui-ci.

Ceci est un résumé explicatif du Code Juridique. La version intégrale du contrat est disponible ici : http://creativecommons.org/licenses/by-nc-sa/2.0/fr/legalcode

3 Plan

- Partie 1 : Tentative de classification des projets
- Partie 2 : Ce que PostgreSQL propose
- Partie 3 : Votre cluster: comment procéder ?

4 Première partie

Introduction
Tentative de classification des projets

5 Répliquer? Pourquoi?

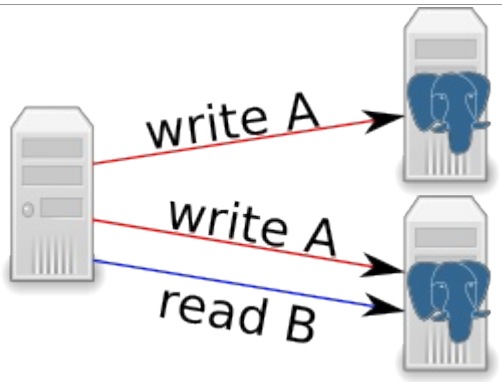
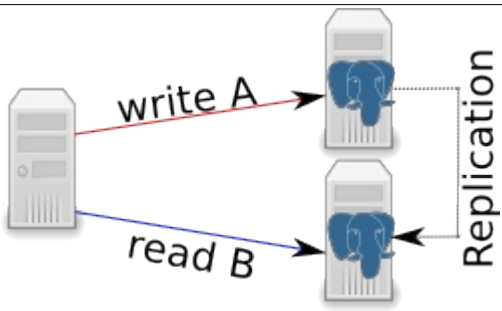
- Haute-disponibilité (`failover`)
- Répartition de charge
- Travaux sans impact sur la production

La réplication a principalement pour but de disposer d'un deuxième serveur, copie du premier, où travailler au cas où le premier tomberait en panne. L'idée est que l'activité ne doit pas être impactée par la perte d'un serveur.

Mais ce n'est pas la seule raison pour laquelle un administrateur voudrait de la réplication. Le serveur répliqué peut aussi servir à y déporter une partie du travail, dans le but d'alléger la charge du serveur principal.

Dans ce cadre, plusieurs types de réplication existent, chacune permettant d'apporter une solution à un type de problème. Chaque type sera couvert par un ou plusieurs outils de réplication.

6 Répliquer? Comment?

Répliquer *a priori* : les requêtes	
Répliquer *a posteriori* : les changements	

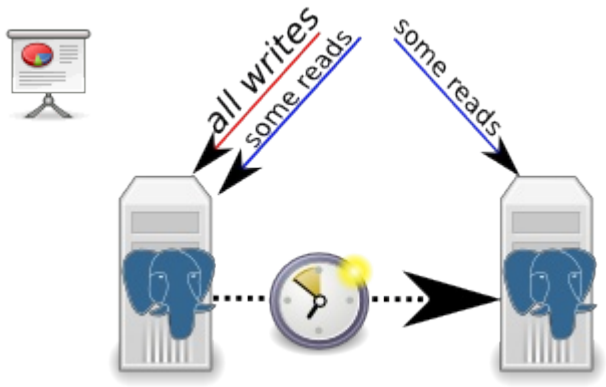
Pour répliquer, il existe deux méthodes simples.

Soit on réplique *a priori*, c'est-à-dire en envoyant le même flux de requêtes SQL en écriture potentielle sur les *n* serveurs de ce que l'on va appeler un “cluster”, au sens “grappe de serveurs”. L'implémentation de ce type de réplication pose de nombreux problèmes, mais peut fonctionner dans des cas d'utilisation adaptés.

Soit on réplique *a posteriori*, c'est-à-dire en répliquant les changements apportés à un serveur sur un (ou plusieurs) autre(s) serveur(s). Il y a plusieurs méthodes et outils pour le faire. Tous ont leurs qualités et leurs défauts. La solution parfaite n'existe pas … encore ?

7 Asynchrone asymétrique

- Écritures uniquement sur le maître
- Mise à jour différée des tables sur l'autre serveur
- Outils pour PostgreSQL: Slony, Londiste, Bucardo, Replicator, rubyrep
- Londiste: *Écoutons Ludovic avant la pause repas…*
- Slony: *…et Guillaume cet après-midi!*



C'est le mode de réplication le plus simple.

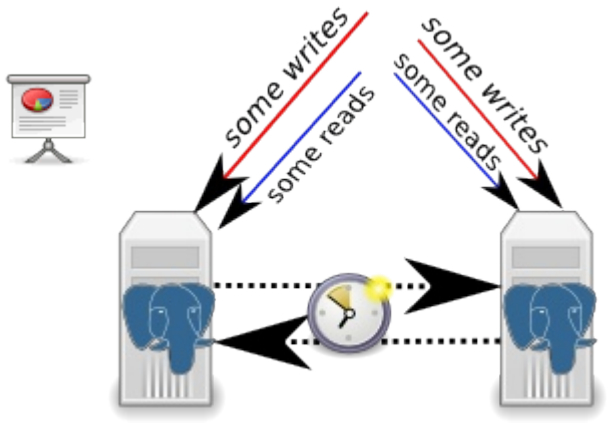
Un serveur est en lecture/écriture, il est généralement appelé serveur maître. Le ou les autres serveurs ne sont pas disponibles en écriture. Ils peuvent cependant être disponibles en lecture.

Tout enregistrement fait sur le maître n'est pas immédiatement reporté sur l'esclave. Il existe généralement un petit délai avant application sur l'esclave. Cela sous-entend que, si le serveur maître est tombé en panne avant d'avoir eu le temps de transférer les derniers transactions au serveur esclave, il manquera les données de ces transactions sur l'esclave. Il faut donc être prêt à accepter une certaine perte, généralement petite. De plus, si l'esclave sert à répartir la charge, il est possible qu'une lecture du maître et qu'une lecture de l'esclave ne donnent pas le même résultat.

Étant la solution la plus simple à mettre à œuvre, il existe de nombreux outils pour ce type de réplication : la réplication interne de PostgreSQL par exemple, mais aussi Slony, Londiste, etc.

8 Asynchrone symétrique

- Écritures sur les deux « maîtres »
- Mise à jour différée des tables sur l'autre serveur
- Difficile d'avoir un respect d'ACID
- Outils pour PostgreSQL : Bucardo, rubyrep

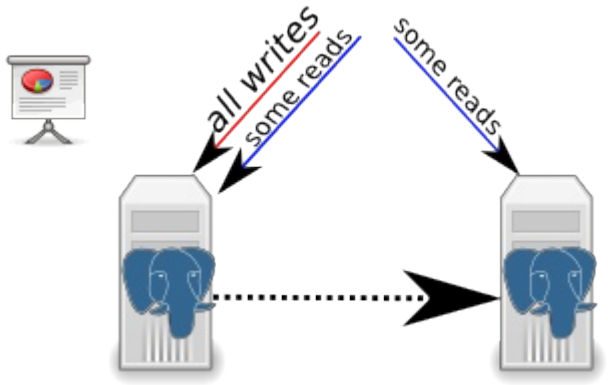


Ce mode est plus complexe. Les serveurs sont tous en lecture/écriture. Il faut donc pouvoir gérer les conflits causés par la mise à jour des mêmes objets sur plusieurs serveurs en même temps. Cela complexifie de beaucoup le respect de la norme ACID.

Bucardo implémente néanmoins ce type de système sur deux serveurs uniquement. À noter que la version 5 devrait pouvoir gérer plus de deux serveurs maîtres.

9 Synchrone asymétrique

- Écritures uniquement sur le maître
- Mise à jour immédiate des tables sur l'autre serveur
- Source de lenteurs
- Outils pour PostgreSQL : PostgreSQL 9.1, pgPool-II



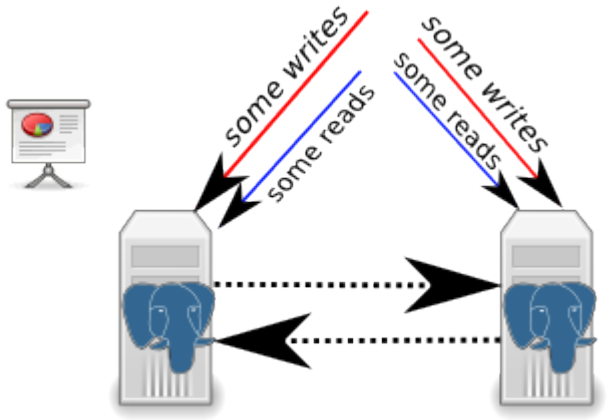
Ce mode de réplication est plus simple que le précédent, mais plus lent. Il n'y a qu'un seul maître mais chaque modification réalisée sur le maître doit être enregistrée sur l'esclave avant de redonner la main à l'utilisateur. Ce qui est source de lenteurs (deux systèmes doivent avoir enregistrés l'information au lieu d'un seul sans compter les *lags* réseau possibles).

C'est la meilleure solution s'il est inconcevable de perdre des données suite à l'arrêt inopiné du maître. Par contre, cela ne résout pas complètement le problème de la répartition de charge. En effet, cette solution garantit seulement que la donnée est enregistrée sur les esclaves, pas qu'elle soit visible. Donc, encore une fois, si l'esclave sert à répartir la charge, il est possible qu'une lecture du maître et qu'une lecture de l'esclave ne donnent pas le même résultat, même si le risque est minimisé par rapport aux autres solutions.

La réplication interne de PostgreSQL propose cette méthode depuis la version 9.1. pgPool-II le proposait depuis plusieurs années via son mode de réplication. À noter qu'il dispose aussi d'un mode de *pooling* de connexions et d'un mode de répartition de charge.

10 Synchrone symétrique

- Écritures sur les deux « maîtres »
- Mise à jour immédiate des tables sur l'autre serveur
- Postgres-XC (`Postgres eXtensible Cluster`)



Ce mode de réplication est le plus complexe et le plus lent. La complexité est due à la gestion des conflits, inévitable quand il y a plusieurs serveurs maîtres. La lenteur est due au côté synchrone du système.

À ce jour, Postgres-XC est le projet le plus sérieux de réplication synchrone symétrique pour PostgreSQL. Il utilise une architecture de type `share nothing`.

11 Deuxième partie

Ce que PostgreSQL propose en matière de réplication

12 PostgreSQL

- Moteur de base de données libre
- Licence BSD
- Né en 1996
- Projet dirigé par sa communauté
- Bien qu'aidé et soutenu par de nombreuses entreprises
- Un des moteurs les plus fidèles au standard SQL
- Grand nombre de fonctionnalités entreprise

13 Journaux de transactions

- Contiennent toutes les modifications des fichiers du serveur (*ie*, toutes les bases)
- Informations de bas niveau
  - Bloc par bloc, fichier par fichier
  - Ne contient pas la requête elle-même
- Déjà utilisé en cas de crash du serveur
  - Rejeu des transactions non synchronisées sur les fichiers

Chaque transaction, implicite ou explicite, réalisant des modifications sur la structure ou les données d'une base est tracée dans les journaux de transactions. Ces derniers contiennent des informations d'assez bas niveau, comme les blocs modifiés sur un fichier suite, par exemple, à un UPDATE. La requête elle-même n'apparaît jamais. Les journaux de transactions sont valables

pour toutes les bases de données de l'instance.

Les journaux de transactions sont déjà utilisés en cas de *crash* du serveur. Lors du redémarrage, PostgreSQL rejoue les transactions qui n'auraient pas été synchronisées sur les fichiers de données.

Comme toutes les modifications sont disponibles dans les journaux de transactions et que PostgreSQL sait rejouer les transactions à partir des journaux, il suffit d'archiver les journaux sur une certaine période de temps pour pouvoir les rejouer.

14 PITR

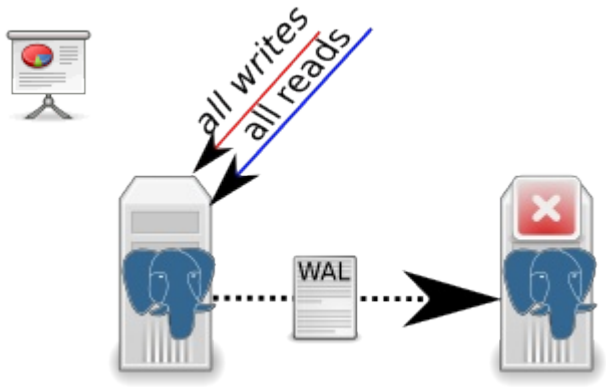
- `Point In Time Recovery`
- Possibilité de rejouer
  - Tous les journaux de transactions
  - Jusqu'à un certain point dans le temps
  - Jusqu'à un certain identifiant de transaction
- Rejeu à partir d'une sauvegarde des fichiers à un instant *t*
- Disponible à partir de PostgreSQL 8.0

La technologie PITR est disponible depuis la version 8.0. Cette dernière permet le rejeu de tous les journaux de transactions préalablement archivés ou tous les journaux jusqu'à un certain point dans le temps, ou encore tous les journaux jusqu'à un certain identifiant de transaction.

Pour cela, il est nécessaire d'avoir une sauvegarde des fichiers de l'instance (réalisée à chaud une fois l'archivage activé) et des journaux archivés depuis cette sauvegarde.

15 Warm Standby

- Esclave mis à jour en permanence
- Fichier par fichier
  - Fichier == un journal de transactions
- Esclave non disponible pendant la restauration
- En 8.3, ajout de l'outil `pg_standby`



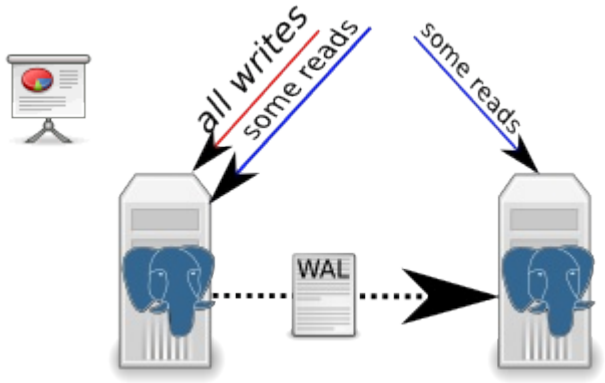
L'idée du serveur en Warm Standby est de rejouer en permanence les journaux de transactions archivés. Autrement dit, quand le serveur maître a terminé de travailler sur un journal de transactions, il l'archive sur un deuxième serveur où il sera récupéré par le serveur PostgreSQL esclave qui le rejouera dès la fin de la copie.

C'est un système de réplication complet. Mais deux gros inconvénients apparaissent : le délai de prise en compte des modifications dépend de l'activité du serveur maître (plus ce dernier sera actif, plus il enverra rapidement un journal de transactions, plus le serveur esclave sera à jour) et le serveur esclave n'est pas disponible, y compris pour des requêtes en lecture seule.

Plutôt que d'avoir à écrire son propre outil, la version 8.3 propose dans les modules contrib un outil appelé pg_standby. De même, Skype propose son propre outil appelé walmgr. Un dernier outil permet aussi de faciliter la restauration : pitrtools.

16 Hot Standby

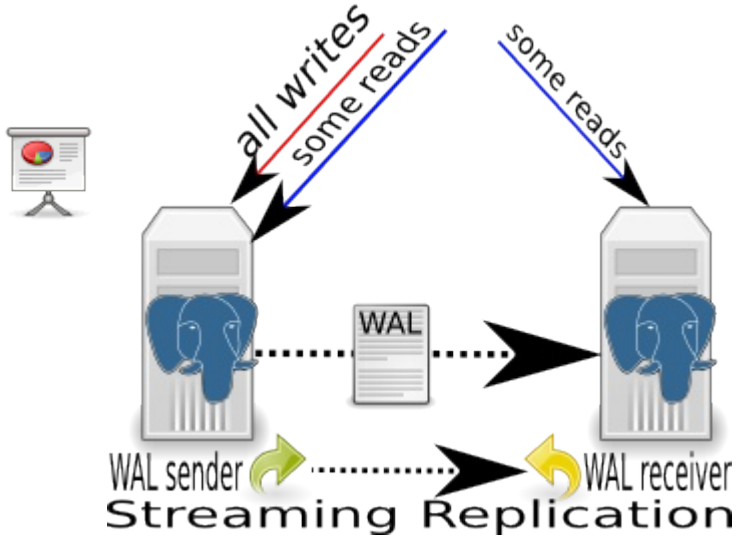
- Patch développé en deux ans par la société 2nd Quadrant
- Financement par un grand nombre de sociétés
- Finalement intégré dans PostgreSQL 9.0
- Permet l'accès en lecture seule aux serveurs esclaves



C'est une évolution majeure du `Warm Standby`. Grâce au `Hot Standby`, les serveurs esclaves sont désormais ouverts aux lectures. Cette évolution a permis aux utilisateurs de PostgreSQL de pouvoir enfin balancer les requêtes en sélection sur plusieurs répliques de la base PostgreSQL, sans avoir à utiliser un outil tiers, comme Slony. Charge à l'application de diriger ses requêtes en lecture seule sur l'un ou l'autre des esclaves. On peut aussi utiliser depuis pgPool-II, qui fait office de “balanceur” automatique pour les applications. Cependant sa configuration et son fonctionnement ne sont pas des plus simples, et l'utilisateur devra apporter un soin tout particulier à ces derniers, et procéder à des tests avancés de pgPool-II pour s'assurer que rien n'a été oublié.

17 Streaming Replication

- Patch basé sur le développement réalisé par NTT
- Intégré à PostgreSQL 9.0
- Permet d'avoir moins de *lag* dans la réplication
- “Semi-asynchrone”: toujours pas synchrone… mais très rapide!



Avec ce patch, le(s) serveur(s) PostgreSQL esclave(s) ne sont plus à “un journal de transaction près”, mais reçoivent en continu les modifications opérées sur le serveur maître. La perte de données est alors minime en cas de perte du maître, puisque l'esclave en est la copie presque parfaite, à quelques transactions près.

Deux nouveaux processus font leur apparition avec le `Streaming Replication`:

- `walreceiver` sur l'esclave
- `walsender` sur le maître

À noter que c'est bien l'esclave qui se connecte au maître et non l'inverse. Il est tout à fait possible d'avoir plusieurs serveurs qui utilisent le `Streaming Replication`, mais c'est toujours le maître qui fournit les esclaves.

18 Synchronous Streaming Replication

- intégré à PostgreSQL 9.1
- `Streaming Replication` avec mode synchrone
- *Écoutons Nicolas cet après-midi!*

Jusqu'à la version 9.0 incluse, le `Streaming Replication` était toujours en mode `async`, c'est à dire asynchrone. C'est à dire que le maître n'attendait pas que les éléments soient reçus sur l'esclave pour continuer son traitement.

Depuis la version 9.1, on dispose ainsi du mode synchrone qui garantit que les informations relatives aux modifications du maître aient été dupliquées sur le(s) esclave(s). Cependant, la visibilité des modifications peut ne pas être immédiate.

19 Problèmes à résoudre

- s'assurer que l'esclave a bien appliqué la modification et qu'elle est visible
- faiblesse du modèle “maître nourrit tous les esclaves”
- toujours pas de réplication symétrique pour PostgreSQL

Jusqu'à la version 9.1 incluse, même si le maître s'assure que les informations de modifications sont enregistrées sur l'esclave, celles-ci ne sont pas forcément visibles.

Dans le modèle actuel, on ne dispose pas de “réplication en cascade”. C'est à dire que seul le maître peut envoyer des informations aux esclaves. Ainsi, si le maître venait à s'arrêter, il faudrait élire l'un des esclaves comme le nouveau maître, mais alors tous les autres esclaves seraient désynchronisés. Il conviendrait alors de tous les resynchroniser sur le nouveau maître. Bien que très faisable, cela amène une difficulté si jamais on avait une réplication utilisée pour balancer les lectures sur les différents esclaves. En effet, si le maître vient à tomber, alors tout le trafic applicatif doit-être dirigé sur un seul nœud… On peut alors s'attendre à une charge ingérable par celui-ci.

L'absence de réplication symétrique pour PostgreSQL reste à ce jour une des faiblesses du projet par rapport à la concurrence.

20 Premiers éléments de réponse

- PostgreSQL 9.2 !
- un nouveau patch permet d'attendre que les modifications soient **appliquées** à l'esclave
- **cascading replication** pour permettre qu'un esclave puisse à son tour “fournir” un autre esclave
- et bien plus encore
- *Let's listen to Simon !*

21 vers le Synchrone Symétrique ?

- Postgres-XC (`Postgres eXtensible Cluster`)
- *Écoutons Michael … juste après la pause !*

22 Troisième partie

Votre cluster, comment procéder ?

23 Quel est le besoin ?

Se questionner pour choisir!

- `Failover` ?
- perte de données … acceptable ?
- volatilité des données ?
- budget ?
- performances attendues ?

24 Arbitrages à faire

Quelle solution de réplication choisir?

- coûts ↔ fonctionnalités
- performances ↔ pertes de données
- budget ↔ solution idéale

25 Compétences

Les **Ressources Humaines** doivent rester la première préoccupation

- réaliser en interne → formation ?
- *kickstart* → prestation ?
- externaliser → support ?

Dalibo, un partenaire PostgreSQL, quels que soient vos choix.

26 Conclusion

- PostgreSQL évolue vite
- … rien ne lui résiste
- … pas même le multi-maîtres