



PostgreSQL et la Haute Disponibilité

Auteur : Jean-Paul Argudo <jean-paul.argudo@dalibo.com>

Licence : Creative Commons, BY-NC-SA

Date : 2 avril 2009

Table des matières

1	Sommaire	5
2	Haute disponibilité ?	6
3	Seven Nine	7
4	Matériel	8
5	Technologies de réplication	9
6	Réplication Asynchrone Asymétrique	10
7	Réplication Asynchrone Symétrique	11
8	Réplication Synchrone Asymétrique	12
9	Réplication Synchrone Symétrique	13
10	Diffusion des modifications	14
11	Projets PostgreSQL	15
12	Warm Stand-by	16
13	Hot Stand-by	18
14	Projets autour de PostgreSQL	19
15	Slony : Identité	20
16	Slony : Fonctionnalités	21
17	Slony : Technique	22
18	Slony : Points forts	23

19 Slony : Limites	24
20 Slony : Utilisations	25
21 Bucardo : Identité	26
22 Bucardo : Fonctionnalités	27
23 Bucardo : Technique	28
24 Bucardo : Points forts	29
25 Bucardo : Limites	30
26 Bucardo : Utilisations	31
27 Londiste : Identité	32
28 Londiste : Fonctionnalités	33
29 Londiste : Technique	34
30 Londiste : Points forts	35
31 Londiste : Limites	36
32 Londiste : Utilisations	37
33 Postgres-R : Identité	38
34 Postgres-R : Fonctionnalités	39
35 Postgres-R : Technique	40
36 Postgres-R : Points forts	41
37 Postgres-R : Limites	42
38 Postgres-R : Utilisations	43
39 pgpool-II : Identité	44

40 pgpool-II : Fonctionnalités	45
41 pgpool-II : Technique	46
42 pgpool-II : Points forts	47
43 pgpool-II : Limites	48
44 pgpool-II : Utilisations	49
45 D'autres projets...	50
46 Sondage	51
47 Conclusion	52
48 Questions	53

Sommaire

- Généralités sur la haute disponibilité et la réplication
Dans cette présentation, nous reviendrons rapidement sur la qualification des solutions de haute disponibilité, et la classification des solutions de réplication.
- Présentation des projets de réplication autour de [PostgreSQL](#)
Nous détaillerons ensuite les projets de réplication pour [PostgreSQL](#) les plus en vue.

Haute disponibilité ?

- Quel est mon besoin ?
Seul le réel **besoin** permet de trouver la solution adéquate.
- Quel est mon budget ?
Le budget est un élément important dans la mise en place d'une solution de haute disponibilité. La plupart du temps, il faudra multiplier le nombre de machines, ou *a minima* le nombre de disques durs. Dans certains cas, on en vient à redonder aussi la mémoire, les processeurs, etc. De nombreuses solutions existent en la matière. Cependant, cette débauche de moyens a toujours un coût.
- Quelles compétences ?
Le coût de la compétence associée à l'installation, et surtout, à la maintenance d'un *cluster*, fut-il installé pour PostgreSQL ou non, est toujours important. Le coût du maintien de la compétence pour la gestion d'un *cluster* ne doit pas être pris à la légère. Cependant, l'offre de formation existe et est aujourd'hui complète. On trouve aussi nombre de sociétés prêtes à supporter la maintenance d'un *cluster* PostgreSQL. L'entreprise a donc le choix d'externaliser ou non cette compétence.
- Quand ?
La donnée temporelle est importante. Il convient en effet de savoir à partir de quel moment on veut transformer son service en un service dont on attend une haute disponibilité. Généralement, plus le service est placé sous haute disponibilité tôt dans l'élaboration du projet, plus les coûts engendrés sont importants.

Seven Nine

Équivalences 99,(9)* et temps d'indisponibilité :

- 99% désigne le fait que le service est indisponible moins de 3,65 jours par an
- 99,9%, moins de 8,75 heures par an
- 99,99%, moins de 52 minutes par an
- 99,999%, moins de 5,2 minutes par an
- 99,9999%, moins de 54,8 secondes par an
- 99,99999%, moins de 3,1 secondes par an

Matériel

- Pas le but de cette présentation
En effet, cette présentation est destinée à présenter les solutions de réplication entrant dans la composition d'un *cluster* PostgreSQL à haute disponibilité.
- À prendre évidemment en compte...
De nombreuses techniques matérielles viennent en complément essentiel des technologies de réplication utilisées dans la haute disponibilité.
Leur utilisation est généralement obligatoire, du RAID en passant par les SAN et autres techniques pour dédonder l'alimentation, la mémoire, les processeurs, etc.

Technologies de réplication

Au choix :

- Propriétaires ou
- Libres

Dans tous les cas, le support professionnel existe !

Réplication Asynchrone Asymétrique

- Écritures sur le maître
Dans la réplication asymétrique, seul le maître accepte des écritures, et l'(es) esclave(s) ne sont accessibles qu'en lecture.
- Mise en attente
Dans la réplication asynchrone, il existe un processus extérieur au SGBD qui gère la réplication des changements.
- Réplication des changements sur l'esclave
Les seules "écritures" acceptées par l'(les) esclave(s) sont la réplication des changements effectués par les utilisateurs sur le maître.

La mise à jour de la(des) table(s) répliquée(s) **est différée** (asynchrone). Elle est réalisée par un programmeur de tâches, possédant une horloge. Des points de synchronisation sont utilisés pour propager les changements.

Réplication Asynchrone Symétrique

- Écritures concurrentielles, sur “deux” maîtres
Dans la réplication symétrique, tous les maîtres sont accessibles aux utilisateurs, aussi bien en lecture, qu’en écriture.
- Mises en attente
Dans la réplication asynchrone, il existe un processus extérieur au SGBD qui gère la réplication des changements.
- ACID ? !

Deux “maîtres” répliquent les données de l’un sur l’autre, via un programmeur (voire deux programmeurs). Ce mode de réplication ne respecte généralement pas les propriétés ACID, car si une copie échoue alors que la transaction a déjà été validée, on peut alors arriver dans une situation où les données sont incohérentes.

Réplication Synchrone Asymétrique

- Écriture sur le maître
Dans la réplication asymétrique, seul le maître accepte des écritures, et l'(es) esclave(s) ne sont accessibles qu'en lecture.
- Recopie 'instantanée' sur l'esclave
Dans la réplication synchrone, il n'y a pas de processus extérieur qui propage les changements. Dans ce cas, on utilise un mécanisme dit de *Two Phase Commit* ou "Commit en deux phases", qui assure qu'une transaction est *comittée* sur tous les nœuds dans la même transaction. Les propriétés ACID sont dans ce cas respectées.

La copie est instantanément mise à jour à chaque modification de la table "maître". Si la copie échoue c'est toute la transaction qui est annulée, et elle n'est appliquée sur aucun des nœuds.

Réplication Synchrone Symétrique

- Écritures concurrentielles sur deux “maîtres”
Dans la réplication symétrique, tous les maîtres sont accessibles aux utilisateurs, aussi bien en lecture, qu’en écriture.
- Gestion des verrous et de la concurrence
Dans la réplication synchrone, il n’y a pas de processus extérieur qui propage les changements. Dans ce cas, on utilise un mécanisme dit de *Two Phase Commit* ou “Commit en deux phases”, qui assure qu’une transaction est *comittée* sur tous les nœuds dans la même transaction. Les propriétés ACID sont dans ce cas respectées.
Dans le cas particulier de la réplication synchrone symétrique, il faut en plus gérer les éventuels conflits qui peuvent survenir quand deux transactions concurrentes opèrent sur le même ensemble de *tuples*. On résout ces cas particuliers avec des algorithmes plus ou moins complexes.

Les deux tables peuvent être modifiées, et les mise à jour sont propagées directement dans l’autre table. Il est à noter que la réplication fait partie de la transaction, ce qui ne ralentit que très peu le système.

Diffusion des modifications

On trouve deux types de diffusion des mises à jour :

- Diffusion du résultat de l'opération : soit *le résultat du SQL*
Permet de ne pas refaire l'opération sur la copie, mais nécessite une gestion de l'ordonnement des mises à jour afin que celles-ci soient identiques sur tous les sites.
- Diffusion de l'opération de mise à jour : soit *le SQL lui-même*
Plus flexible, notamment dans le cas d'opérations cumulatives. Pose la problématique des opérations dites "non déterministes". Par exemple, le résultat de `CURRENT_TIMESTAMP` ou de `random()` peut différer d'un nœud à l'autre.

Projets PostgreSQL

Deux techniques de “réplication” existent pour [PostgreSQL](#) :

- Warm Stand-by (*aka* Log Shipping)
- Hot Stand-by

Cependant, seule la première est stable, la seconde étant toujours en cours de développement.

Warm Stand-by

- Intégré à [PostgreSQL](#) depuis plusieurs années
Le Warm Stand-by existe depuis la version 8.2, sortie le 5 décembre 2006. La robustesse de ce mécanisme simple est à toute épreuve.
- Permet d'avoir une réplique d'une *cluster PostgreSQL* sur un serveur secondaire
Les journaux de transactions (*aka WAL*, pour *Write Ahead Log*) sont immédiatement envoyés au serveur secondaire après leur écriture. Le serveur secondaire est dans un mode spécial d'attente, et lorsqu'un journal de transactions est reçu, il est automatiquement appliqué au réplica.

Cette technique ne permet de répliquer que l'**ensemble** du *cluster PostgreSQL*, c'est à dire, l'ensemble des bases de données qu'il contient. On ne peut pas par exemple ne répliquer qu'une base parmi celles que contient le *cluster*. Cette limitation est liée au fait que les journaux de transactions de [PostgreSQL](#) (*aka WALs*) tracent toutes les transactions du *cluster*, quelle que soit la base de données.

- Le réplica est identique au serveur primaire, **au WAL près**

Étant donné que le serveur distant n'applique que les WAL qu'il reçoit, il y a toujours un risque de pertes de données en cas de panne majeure sur le serveur primaire.

On peut cependant moduler le risque de deux façons :

- ⇒ Sauf en cas d'avarie très grave sur le serveur primaire, le WAL sur ce dernier peut généralement être récupéré et appliqué sur le serveur secondaire
- ⇒ On peut réduire la fenêtre temporelle de la réplication en modifiant la valeur de la clé de configuration `archive_timeout`. Au delà des n secondes déclarées dans cette variable de configuration, le serveur change de WAL, provoquant l'archivage du précédent.
On peut par exemple envisager un `archive_timeout` à 30 secondes, et ainsi obtenir une réplication à *30 secondes près*

- Un outil pratique : `pg_standby`

L'outil `pg_standby` de Simon Riggs (*contrib* de [PostgreSQL 8.3](#)) possède plusieurs options en ligne de commande :

```
pg_standby [OPTION]... [ARCHIVELOCATION] [NEXTWALFILE] [XLOGFILEPATH]
```

`ARCHIVELOCATION` correspond au répertoire de stockage des journaux de transaction archivés.

`NEXTWALFILE` est le nom du prochain journal à récupérer.

`XLOGFILEPATH` est l'emplacement des journaux de transaction.

Options les plus intéressantes :

- ⇒ `-d` pour envoyer des informations de débogage sur `stderr`.
- ⇒ `-s delai`, délai entre deux vérifications.
- ⇒ `-t fichier_trigger`, pour arrêter la vérification.
- ⇒ `-w delai_max`, délai maximum avant l'abandon de la récupération.

Exemple :

```
restore_command = 'pg_standby -d -s 2 -t /tmp/pgsql.trigger.5432 \  
/var/pg_xlog_archives %f %p 2>> standby.log'
```

Hot Stand-by

- Évolution du Warm Stand-by
Le *patch* Hot Stand-by a nécessité environ 5 mois de travail intense de son auteur, Simon Riggs, et environ 3 semaines de relecture à un *hacker* émérite de PostgreSQL, Heikki Linakangas. Il s'agit d'un *patch* de plusieurs milliers de lignes de C.
- Basé sur le même mécanisme
- L'évolution : le serveur secondaire est ouvert en lecture seule

Cette évolution majeure de PostgreSQL devait initialement être intégrée à la toute prochaine version 8.4. Cependant, les relectures des *patches* de Simon Riggs (principal codeur de Warm et Hot Stand-By) par Heikki Linakangas (autre codeur émérite du projet!), ont abouti à la conclusion qu'il était prématuré d'inclure cette fonctionnalité dans PostgreSQL 8.4 car des tests complémentaires, plus poussés devaient être réalisés, d'une part. D'autre part, il existe de nombreux axes d'améliorations possibles au *patch* de Simon.

Cette nouvelle fonctionnalité de PostgreSQL ne remplacera pas des projets plus complexes de réplication. Elle n'en a pas non plus la vocation.

L'évolution naturelle du Hot Stand-by sera probablement d'alimenter le serveur secondaire directement avec les transactions du serveur primaire, en ne passant plus par les journaux de transaction.

Une fois cette dernière évolution effectuée, les utilisateurs d'outils de réplication tiers se poseront probablement la question sur la nécessité pour eux de garder leur installation actuelle, dans la mesure où PostgreSQL possèdera de manière native une réplication maître / esclave intéressante.

Projets autour de PostgreSQL

- Slony
- Bucardo
- Londiste
- Postgres-R
- pgpool-II

Slony : Identité

- Projet libre (BSD)
- Asynchrone / Asymétrique
- Réplication des résultats
- Site web : <http://slony.info/>

Slony : Fonctionnalités

- Failover / Failback
- Switchover / Switchback
- Standalone

Slony : Technique

- Réplication basée sur des `triggers`
- Démons externes, écrits en C
- Le maître est *provider*
- Le(s) esclave(s) est(sont) *suscriber(s)*

Slony : Points forts

- Basé sur un(des) set(s) de réplication et non sur un(des) schéma(s)
- Indépendance des versions de [PostgreSQL](#)
- Technique de propagation des *DDL*
- Robustesse

Slony : Limites

- À proscrire pour la réplication de bases *itinérantes*
- Le réseau doit être fiable : peu de *lag*, pas ou peu de coupures
- *Monitoring* délicat

Slony : Utilisations

- Base de données de secours
- Alimentation des bases de pré-production, de recette et de tests
- Infocentre (*many to one*)
- Bases spécialisées (recherche plein texte, traitements lourds, etc)

Bucardo : Identité

- Projet libre (BSD)
- Asynchrone / Symétrique
- Réplication des résultats (dits *deltas*)
- Site web : <http://bucardo.org/>

Bucardo : Fonctionnalités

- Failover?

Bucardo : Technique

- Réplication basée sur des triggers
- Démons externes, écrits en Perl
- Maître / Maître ou
- Maître / Esclave

Bucardo : Points forts

- Basé sur un(des) set(s) de réplication et non sur un(des) schéma(s)
- Simplicité d'utilisation
- Résolution standard des conflits
 - ⇒ source : la base de données d'origine gagne toujours
 - ⇒ target : la base de destination gagne toujours
 - ⇒ random : l'une des deux bases est choisie au hasard comme étant la gagnante
 - ⇒ latest : le plus récemment changé gagne
 - ⇒ abort : la réplication est arrêtée
 - ⇒ skip : aucune décision ni action n'est prise

Bucardo : Limites

- Aucune technique de propagation des *DDL*
- Limité à deux nœuds
- Le réseau doit être fiable : peu de *lag*, pas ou peu de coupures
- Version de [PostgreSQL](#) > 8.2
- Sous `Unix` uniquement
- Cas particulier des séquences
- Un seul développeur sur le projet (Greg Sabino Mulane)

Bucardo : Utilisations

- *Cluster* maître/maître simple
- Base de données de secours
- Bases spécialisées (recherche plein texte, traitements lourds, etc)

Londiste : Identité

- Projet libre (BSD)
- Asynchrone / Asymétrique
- Réplication des résultats
- Site web : <https://developer.skype.com/SkypeGarage/DbProjects/SkyTools>

Londiste : Fonctionnalités

- Failover?
- Pour les tables : repair et compare

Londiste : Technique

- Réplication basée sur des triggers
- Démons externes, écrits en Python
- Utilise un autre *Skytool* : PgQ
- Maître / Esclave(s)

Londiste : Points forts

- PgQ est robuste, fiable et flexible
- Pas de *sets* de réplication, mais des tables appartenant à différentes *queues*
On peut ainsi avoir des tables dans le serveur “maître” qui alimentent la *queue* principale à laquelle les différents “esclaves” auront souscrit, mais aussi d’autres *queues* qui vont alimenter certaines autres tables du “maître”.
Cela rend donc possible de réplications “croisées”.
- Indépendance des versions de [PostgreSQL](#)
- Robustesse

Londiste : Limites

- Technique de propagation des *DDL* basique, et surtout, unitaire
- Pas de *sets* de réplication, mais les *queues* (PgQ) peuvent gérer cela
- Très peu de fonctionnalités. *Skytools* 3 devrait corriger des écarts avec [Slony](#)
Notamment au niveau de la propagation des DDL, on pourra par exemple exécuter le script suivant :

```
londiste.py conf.ini execute script.sql
```

Londiste : Utilisations

- Base de données de secours
- Alimentation des bases de pré-production, de recette et de tests
- Infocentre (*many to one*)
- Bases spécialisées (recherche plein texte, traitements lourds, etc)

Postgres-R : Identité

- Projet libre (BSD)
- Synchrone / Symétrique
- Réplication des résultats
- Site web : <http://www.postgres-r.org/>

Postgres-R : Fonctionnalités

- Resynchronisation automatique d'un nœud désynchronisé
La resynchronisation d'un nœud qui a subi une désynchronisation suite à so, arrêt volontaire (ou non), peut être partielle dans la plupart des cas : ne seront rejoués que les évènements qui doivent l'être

Postgres-R : Technique

- Basé sur le *Group Communication System* (aka GCS)
La théorie sur le *Group Communication System* est très aboutie (travaux de B. Kemme en particulier). Cependant son implémentation est relativement complexe.
- Est un gros *patch* pour [PostgreSQL](#)
L'auteur (Markus Wanner) explique lui-même que pour des raisons de performances, et parce qu'un système de réplication multi-maîtres est une chose complexe, [Postgres-R](#) est en fait une version *patchée* de [PostgreSQL](#).

Postgres-R : Points forts

- Intégration au cœur même de [PostgreSQL](#)
L'intégration au sein même du code de [PostgreSQL](#) est garant de performances optimales. Cependant, c'est aussi à double tranchant, comme cela sera exposé plus tard.
- Installation aisée
Il suffit de compiler la "version" [Postgres-R](#) avec l'option de compilation `--enable-replication` pour obtenir un [PostgreSQL](#) avec la réplication. La configuration des nœuds ensuite n'est pas plus compliquée qu'avec [Londiste](#) ou [Slony](#).
- En théorie, le *cluster* actif-actif est limité à environ 20 nœuds
Les travaux de Betina Kemme montrent qu'au delà de 20 nœuds, le temps passé par les nœuds à "discuter" au sujet du fonctionnement même du *cluster* devient beaucoup plus grand que le temps passé à s'échanger des données. Dès lors, une saturation du réseau entre les nœuds est constatée.
- En pratique, la limite acceptable est à environ 12 nœuds
La limite pratique peut-être augmentée à 16 nœuds, dans le cas où le paramètre `fsync` est à `off` (Sameh Elnikety, Steven Dropsho, et Willy Zwaenepoel. Tashkent+ : "Memory-aware load balancing and update filtering in replicated databases". EuroSys 2007 : Proceedings of the 2nd European Conference on Computer Systems, pages 399--412, 2007)

Postgres-R : Limites

- Un seul développeur à ce jour (Markus Wanner)
- [Postgres-R](#) est un *patch* de [PostgreSQL](#)
Cela rend donc compliqué la maintenance du code pour son auteur, qui doit l'adapter à chaque version majeure de [PostgreSQL](#).
- **Surtout** : [Postgres-R](#) est encore un projet de recherche, partiellement abouti.
C'est réhébitorie pour les professionnels, qui privilégient à juste titre la robustesse et la stabilité des logiciels. Cependant, ce projet est très prometteur et il convient de le surveiller de très près.

Postgres-R : Utilisations

- Limitées actuellement, du fait du statut actuel du projet
- Publication en Janvier 2009 d'un document de spécifications
Ce document (téléchargeable sur <http://www.postgres-r.org/downloads/concept.pdf>) semble poser toutes les bases de la spécification de [Postgres-R](#).
Mais l'auteur l'avoue lui-même en ces termes : « *Note that the current prototype implementation doesn't cover all aspects mentioned* ». Ce qui, dans une traduction libre veut dire "Notez que l'implémentation dans le prototype actuel ne couvre pas tous les aspects mentionnés (dans ce document)".
Il est clair en tout cas que Markus a fait de nombreux appels du pied à la communauté [PostgreSQL](#). Mais son projet ne semble pas attirer les foules.

pgpool-II : Identité

- Projet libre (BSD)
- Synchrone / Symétrique
- Réplication des requêtes SQL
- Site web : <http://pgpool.projects.postgresql.org/>

pgpool-II : Fonctionnalités

- Failover avec détection automatique d'un nœud déficient
Le cas échéant, le nœud est désactivé dans la réplication. La technique du "Online Recovery" de [pgpool](#)
- Failback : Online Recovery

pgpool-II : Technique

- `pgpool` est à l'origine un *pooler* de connexions
- A sa configuration propre
- Transparent pour les applications

pgpool-II : Points forts

- Léger et très robuste
- Projet
- Installation et prise en main très rapide
- Réplication de requêtes : donc, même le DDL

pgpool-II : Limites

- [pgpool](#) est un *SPOF* !
Il faudra donc veiller à ce qu'un autre service [pgpool-II](#) existe sur une autre machine et à mettre en place un système de bascule automatique. Cela est généralement fait avec des infrastructures redondantes basées sur `heartbeat`, `lvm`, etc.
- Réplication basée sur la réplication des requêtes SQL
Il faut donc absolument veiller à ce que les bases de données ne puissent être accédées que via [pgpool-II](#) ! En effet, il existe toujours un risque qu'une base soit modifiée "en direct"... Ce qui la désynchroniserait des autres.
- Documentation d'origine en Japonnais, dont les traductions sont parfois curieuses
- Authentification en mode réplication : pas de `md5`
- Effet "trousse à outils"
Dans la communauté [PostgreSQL](#), ce type de critique est récurrent à l'encontre de [pgpool](#) : beaucoup lui reprochent de tout faire un peu. [pgpool](#) est en effet à la fois : un *pooler* de connexions, mais aussi capable de faire de la réplication et de la requête parallèle.
Ces critiques sont cependant peu fondées, les sites utilisant [pgpool](#) en production sont très nombreux ! Ceux qui ne s'intéresseront qu'au seul mode de *pooler* de connexions pourront s'intéresser à l'excellent [PgBouncer](#) de Skype.

pgpool-II : Utilisations

- Base de données de secours
- *Load-Balancing*

D'autres projets...

Il existe bien d'autres projets !

- PGCluster : moribond
- Mammoth Replicator / Command Prompt : libre + support commercial, moribond
- Cybercluster / Cybertek : libre + support commercial, actif
- Tungsten(ex Sequoia)/Continuent : libre + support commercial, très actif

Sondage

Quel est votre outil de réplication favori pour PostgreSQL ?

Réponse	Nombre de votes	Pourcentage
pgpool-II	7	11.290%
Bucardo	1	1.613%
Slony-I	35	56.452%
Londiste	6	9.677%
Continuent	2	3.226%
pgCluster	2	3.226%
DRBD ou Sun Cluster	2	3.226%
Autres	7	11.290%
Total	62	

Les résultats ci-dessous sont consultables sur le site du projet [PostgreSQL](http://www.postgresql.org) sur la page suivante :

<http://www.postgresql.org/community/survey.61>

Conclusion

Quel que soit le projet choisi pour répliquer les données, il ne faut pas oublier :

- de bien définir son besoin
- d'identifier tous les *SPOF*
- de redonder chaque service jugé critique
- de "monitorer" son *cluster*
- de se préparer à un éventuel *Failover*

Questions

- Jean-Paul Argudo <jean-paul.argudo@dalibo.com>
Gérant de Dalibo SARL, « L'Expertise PostgreSQL » Co-fondateur de PostgreSQL France
Trésorier de PostgreSQL Europe
- Rendez-vous au stand PostgreSQLFr!
Sur le “village des associations”, où de nombreux contributeurs [PostgreSQL](#) pourront répondre à vos questions.
- et au pgDay 2009 : <http://2009.pgday.eu/>
2 journées entièrement consacrées à [PostgreSQL](#), avec la participation de *core-hackers* de [PostgreSQL](#), ainsi que l'ensemble de la communauté Européenne, dont c'est le 3ème **pgDay** Européen, qui vient, cette année se greffer au **pgDay** Français, après le **pgDay** Italien en 2008, et le **pgDay** probablement Allemand en 2010!