DALIBO
L'Expertise PostgreSQL

Systèmes de réplication pour PostgreSQL

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

Historique des versions		
Date	**Version**	**Libellé**
2 avril 2009	1.0	Solutions Linux 2009
6 novembre 2009	2.0	Réplication uniquement, mises à jours, pour le PGDay Europe 2009

v.1 : 2 avril 2009 (Solutions Linux 2009) v.2 : 6 novembre 2009 (mises à jour)

Table des matières

1 Sommaire

- Aspects théoriques sur la réplication
  Dans cette présentation, nous reviendrons rapidement sur la classification des solutions de réplication, qui sont souvent utilisés dans un but de haute disponibilité, mais pas uniquement.
- Présentation des techniques de réplication intégrées à PostgreSQL
  PostgreSQL dispose d'une réplication basée sur la relecture de ses journaux de transaction par un serveur dit "en Standby". Nous présenterons ainsi les techniques dites de "Warm StandBy" et de "Hot StandBy".
- Présentation des projets de réplication autour de PostgreSQL
  Nous détaillerons ensuite les projets de réplication autour de PostgreSQL les plus en vue actuellement.

2 Disclaimer

- Présentation générale
  Il serait en effet très difficile de présenter chacun des projets abordés de manière exhaustive. Nous tenterons néanmoins de présenter les principales caractéristiques de chacun, de manière méthodique.
- Londiste en détails : conférence de Dimitri Fontaine
  Dimitri est mainteneur de plusieurs projets autour de PostgreSQL. Il est le principal hacker de pgloader et prefix. Son activité professionnelle l'a poussé à utiliser Londiste, et il maintient sur sont site une documentation précieuse sur Londiste, et plus généralement, les Skytools (http ://tapoueh.org/skytools.html).
- Continuent/Sequoia : Tungsten en détails : conférence de Gilles Rayrat et Stéphane Giron
  Nous n'aborderons pas aujourd'hui la solution Tungsten. Elle fait l'objet d'une conférence à part, le samedi 7 novembre 2009. Deux spécialistes de Continuent s'en chargeront pour nous.
- PGCluster : Lightning Talk d'Atsushi Mitani
  Atsushi Mitani présentera demain, pendant la session spéciale des "Lightning Talks", les dernières avancées de son projet, PGCluster.

3 Solutions matérielles

- Pas le but de cette présentation
  Cette conférence est destinée à présenter les solutions de réplication logicielles pour PostgreSQL, uniquement.
- À prendre évidemment en compte...
  De nombreuses techniques matérielles viennent en complément essentiel des technologies de réplication utilisées dans la haute disponibilité, en particulier.
  Leur utilisation est généralement obligatoire, du `RAID` en passant par les `SAN` et autres techniques pour redonder l'alimentation, la mémoire, les processeurs, etc.

4

Propriétaire ou Libre ?

Au choix :

- Propriétaires ou
- Libres

Dans tous les cas, le support professionnel existe !

5
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 programmateur de tâches, possédant une horloge. Des points de synchronisation sont utilisés pour propager les changements.

6

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 programmateur (voire deux programmateurs). 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 entre les serveurs.

7
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.

8

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 cars 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.

9 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'ordonnancement 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.

10 Projets PostgreSQL

Deux techniques de “réplication” existent pour PostgreSQL :

- Warm Stand-by (*aka* Log Shipping) : stable, depuis longtemps
- Hot Stand-by : intégré à la version 8.5 ?
- Streaming Replication : intégré à la version 8.5 ?

11 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'
  ```

12
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 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.

13 Streaming Replication

- Idée : rejouer **les transactions** jouées sur le maître par **paquets** ;
  L'objectif du projet Streaming Replication est donc d'avoir un *réplica* du serveur maître à un paquet de transactions près. Cette technique n'est pas basée sur la transmission par le maître des journaux de transaction à autre serveur, mais bel et bien la transmission des transactions en groupe. Ainsi, en cas de perte du maître,
- Contribution japonaise du NTT OSS Center ;
  Très impliqué dans l'Open Source, *Nippon Telegraph and Telephone* a depuis de nombreuses années investi sur PostgreSQL. De nombreuses contributions nous arrivent ainsi de ce groupe très actif, comme `pg_bulkload`, `pgperf` et d'autres. N'hésitez pas à faire un tour sur le NTT Datagroup OSS Square.
- Inclusion dans la 8.5 ?
  Le groupe Open Source de NTT souhaite que son patch soit intégré à la version 8.5. Cependant, à la vue de l'état actuel du projet, présenté pour la première fois au Commit Fest de septembre 2009, rien n'est moins sûr.
- Solution idéalement couplée au `Hot Standby`
  Les délais de réplication entre le maître et l'esclave sont très courts. Couplée au `Hot Standby`, cette technologie pourrait rendre très vite obsolète nombre de systèmes de réplication, utilisés bien souvent avec deux nœuds (un maître et un esclave) : une modification sur maître sera en effet très rapidement visible sur un esclave, en lecture seule.

14

Projets autour de PostgreSQL

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

15
Slony : Identité

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

16

Slony : Fonctionnalités

- Failover / Failback
- Switchover / Switchback
- Standalone

17
Slony : Technique

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

18

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

19
Slony : Limites

- À proscrire pour la réplication de bases *itinérantes*. Cependant, cette problématique est-elle encore d'actualité ?
- Le réseau doit être fiable : peu de *lag*, pas ou peu de coupures
- *Monitoring* délicat

20
Slony : Utilisations

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

21

Bucardo : Identité

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

22

Bucardo : Fonctionnalités

- Failover ?

23

Bucardo : Technique

- Réplication basée sur des `triggers`
- Démons externes, écrits en `Perl`
- Maître / Maître (1 seul couple) ou
- Maître / Esclave(s)

24

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

25
Bucardo : Limites

- Aucune technique de propagation des *DDL* (work in progress)
- Limité à deux nœuds en mode *multimaster*
- Le réseau doit être fiable : peu de *lag*, pas ou peu de coupures
- Version de PostgreSQL > `8.1`
- Sous `Unix` uniquement
- Un seul développeur sur le projet (Greg Sabino Mulane)

26

Bucardo : Utilisations

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

27

Londiste : Identité

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

28

Londiste : Fonctionalités

- Failover ?
- Pour les tables : `repair` et `compare`

29

Londiste : Technique

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

30

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

31
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
```

32

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)

33

Postgres-R : Identité

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

34

Postgres-R : Fonctionalités

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

35

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.

36

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)

37

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 : "Please note that Postgres-R is **not ready for productive use**"
  C'est rédhibitoire pour les professionnels, qui privilégient à juste titre la robustesse et la stabilité des logiciels. Cependant, ce projet est prometteur et il convient de le surveiller de près.

38

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.
- Dernier patch publié fin Août 2009

  Le patch est applicable directement à la version `CVS HEAD` de PostgreSQL.

39

pgpool-II : Identité

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

40

pgpool-II : Fonctionalités

- 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

41

pgpool-II : Technique

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

42

pgpool-II : Points forts

- Léger et très robuste
- Projet avec une expérience de plusieurs années
- Installation et prise en main très rapide
- Réplication de requêtes : donc, même le `DDL`

43

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 "couteau Suisse"
  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, de la requête parallèle et de la répartition de charge.
  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.

44

pgpool-II : Utilisations

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

45
D’autres projets...

Il existe bien d’autres projets !

- `PGCluster` : voir la conférence d’Atsuni demain !
- `Mammoth Replicator / Command Prompt` : libre + support commercial
- `Cybercluster / Cybertek` : libre + support commercial, actif
- `Tungsten(ex Sequoia)/Continuent` : libre + support commercial, très actif. Voir la conférence de Gilles et Stéphane demain !

46
Sondage

Quel est votre outil de réplication favori pour PostgreSQL ? http ://www.postgresql.org/community/sur

Réponse	Nombre de votes	Pourcentage
pgpool-II	23	11.795%
Bucardo	9	4.615%
Slony-I	74	37.949%
Londiste	39	20.000%
Continuent	5	2.564%
pgCluster	12	6.154%
DRBD ou Sun Cluster	13	6.667%
Autres	20	10.256%
Total	**195**	

Source : http ://www.postgresql.org/community/survey.61 Mise à jour : jeudi 29 octobre 2009, 9h18

L'échantillon de 195 votants n'est certes pas très représentatif, mais la tendance qui se dégage est celle qu'on constate sur le terrain : Slony et Londiste sont les deux projets les plus prisés.

Il s'agit pourtant de projets de réplication assynchrone et assymétrique, ce qui veut dire que même si on en parle beaucoup, la réplication synchrone et symétrique, n'est en fait que très rarement un réel besoin.

Dans la plupart des cas, une réplication assynchrone et assymétrique est suffisante pour couvrir l'ensemble des besoins.

47 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* (Murphy...)

48
Questions

- Jean-Paul Argudo <jean-paul.argudo@dalibo.com>
  Gérant de Dalibo SARL, « L'*Expertise* PostgreSQL »
  Fondateur de www.PostgreSQL.fr
  Co-fondateur de PostgreSQL France
  Trésorier de PostgreSQL Europe

- Autour d'un verre ce soir !