dalibo

*L'expertise PostgreSQL*

# Slony : Impact de la perte d’un nœud esclave

**Auteur** : Cédric Villemain <cedric.villemain@dalibo.com>

**Version** : 1.1 du 22 novembre 2007

**Relatif à** : Tests Slony-I portant sur la tenue du master en cas de perte du slave



**Historique des révisions**

| Version | Date | Libellé |
|---|---|---|
| 1.0 | 02/11/2007 | Version initiale |
| 1.1 | 22/11/2007 | Anonymisation des données |

# Table des matières

# 1 Présentation

Ce document présente l'analyse du cas d'étude suivant :

> Un cluster Slony-I est mis en place. Il est constitué de deux nœuds, le premier nommé *master*, le second nommé *slave*. Alors que le serveur PostgreSQL continue d'être sollicité en écriture et en lecture, que se passe-t-il si le *slave* s'arrête ?

# 2 Préparation

## 2.1 Serveur *master* : ROSE

Il s'agit d'une *dedibox* standard. Le résultat d'un `lshw` est fourni en annexe. Le système d'exploitation est une `Debian GNU/linux 4.0 Etch`. Le serveur PostgreSQL est fourni sous forme de paquets, pour la version `8.2`. Il en est de même pour Slony-I dont la version est `1.2.11`.

## 2.2 Serveur *slave* : PAEGA

C'est exactement le même serveur avec les mêmes paquets Slony-I et PostgreSQL.

# 3 Configuration

## 3.1 Fichier de configuration postgresql.conf

```
shared_buffers        = 240MB
work_mem              = 16MB
maintenance_work_mem = 160MB
max_connections       = 100
max_fsm_pages         = 307200
effective_cache_size = 128MB
stats_block_level     = on
stats_row_level       = on
autovacuum            = on
fsync                 = on
```

## 3.2 Fichier de configuration pg_hba.conf

```
host    cave        @benchs     88.191.69.196/32      md5
host    cave        @benchs     88.191.69.197/32      md5
```

# 4 Schéma et requêtes

## 4.1 Schéma

Un schéma très simple est utilisé :

```
# \d descriptif
                        Table « public.descriptif »
Colonne  |  Type   |                      Modificateurs
---------+---------+-----------------------------------------------
id       | integer | not null default
         |         | nextval('descriptif_id_seq' : :regclass)
titre    | text    |
texte    | text    |
resume   | text    |
auteur   | text    |
date_ins | date    |
date_maj | date    |
actif    | bigint  |
Index :
    « descriptif_pkey » PRIMARY KEY, btree (id)
    « actif_desc » btree (actif)
    « date_diff » btree ((date_ins <> date_maj))
```

## 4.2 Requêtes

Chaque cycle consomme chaque requête une fois.

De plus pour chaque requête, une valeur donnée par :

```
$actif = floor(rand()*101) ;
```

est utilisée. Elle permet d'avoir une variation dans chaque requête, limitant l'utilisation du cache.

L'utilisation du *random* et du *limit* rend plus variable le résultat des requêtes, et permet d'obtenir le ratio global suivant : 100 lignes insérées pour 50 mises à jour et 50 supprimées.

### 4.2.1 Insert

```
INSERT INTO descriptif
(titre,texte,resume,auteur,date_ins,date_maj,actif)
  SELECT texte,resume,auteur,titre,now(),now(),round(random()*100)
  FROM (SELECT texte,resume,auteur,titre FROM descriptif
        WHERE random() < 0.5
        AND actif = $actif
        LIMIT 200) AS a
WHERE random() < 0.5 ;
```

### 4.2.2 Update

```
UPDATE descriptif
SET titre   = texte,
    texte   = resume,
    resume  = auteur,
    auteur  = titre,
    date_maj= now(),
    actif   = round(random()*100)
WHERE id IN (
  SELECT id
    FROM descriptif
    WHERE random() < 0.5
    AND actif = $actif
    AND date_ins < now()
    LIMIT 100
```

```
  )
AND random() < 0.5 ;
```

### 4.2.3 Delete

```
DELETE FROM descriptif
WHERE id IN (
  SELECT id
    FROM descriptif
    WHERE random() < 0.5
    AND actif = $actif
    LIMIT 100
  )
AND random() < 0.5 ;
```

# 5 Injecteur

Préalablement au début des tests, un volume d'environ trois gigaoctets est injecté et répliqué. Puis l'accroissement est d'environ 100 mégaoctets par jour.

L'injecteur est un programme écrit en `Perl` qui exécute chacune des requêtes une fois, en se connectant/déconnectant du serveur pour chaque traitement.

À la fin de chaque cycle, une pause est exécutée. Sa durée est variable et dépend de la durée du cycle.

Les données renvoyées sont les suivantes :

- nombre de lignes insérées ;
- nombre de lignes mises à jour ;
- nombre de lignes supprimées ;
- la durée du cycle ;
- la durée de la pause ;
- totaux et moyennes.

# 6 Monitoring

La surveillance (*monitoring*) est effectuée à l'aide du logiciel `Munin` sur le serveur *ROSE*.

Les graphiques en temps réel permettent de consulter rapidement les premiers résultats et le bon déroulement des tests.

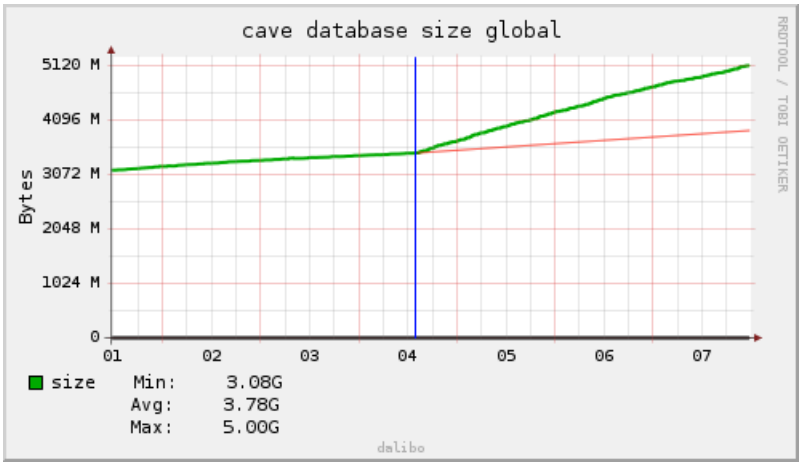
En plus des sondes standard de surveillance du système, des relevés sont faits sur :

- le lag de la réplication selon trois procédés (dont la mise en place d'une table de surveillance elle-même répliquée et mise à jour une fois par seconde) ;
- l'activité sur les lignes (insertions/suppressions/mises à jour) ;
- les lignes obtenues par *sequential scan* et par *index scan* ;
- le nombre de *sequential scan* et d'*index scan* ;
- le nombres de blocs lus sur le disque ou pris dans le cache mémoire ;
- le *cache hit ratio* ;
- le nombre de *commit* et de *rollback* ;
- la taille de la base de données (incluant le schéma Slony-I et le schéma de la table servant au test) ;
- le nombre de *locks* et de *locks exclusifs*.

# 7 Analyse des résultats

Après trois jours de fonctionnement normal (réplication correcte), le *slave* est stoppé de façon radicale (refus de la connexion Slony-I dans le fichier `pg_hba.conf` du serveur *PAEGA*).

## 7.1 Évolution de la volumétrie




Ce graphe montre bien l'impact important de Slony-I sur l'évolution de la volumétrie de la base de données répliquée. (Le jour 4 est le jour de blocage de la réplication).

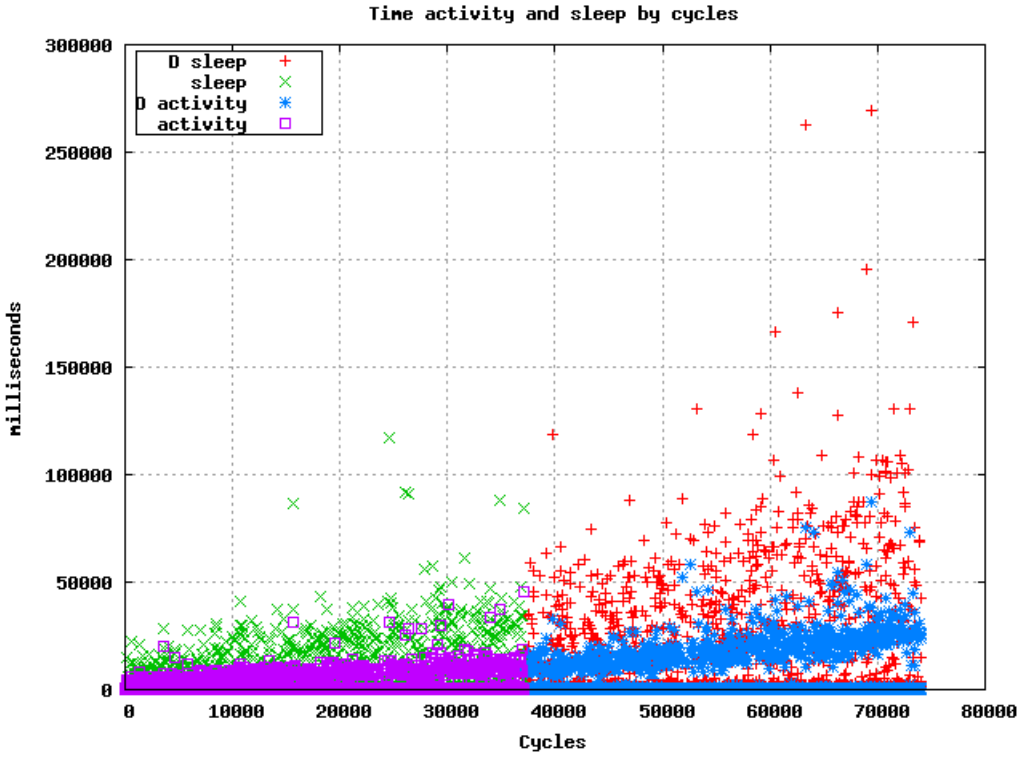
Pour rappel, Slony-I fonctionne via des triggers *on each row* qui recopient les données dans des tables propres à Slony-I. Ces données, récupérées et rejouées par les autres nœuds, sont effacées lorsque la réplication fonctionne correctement.

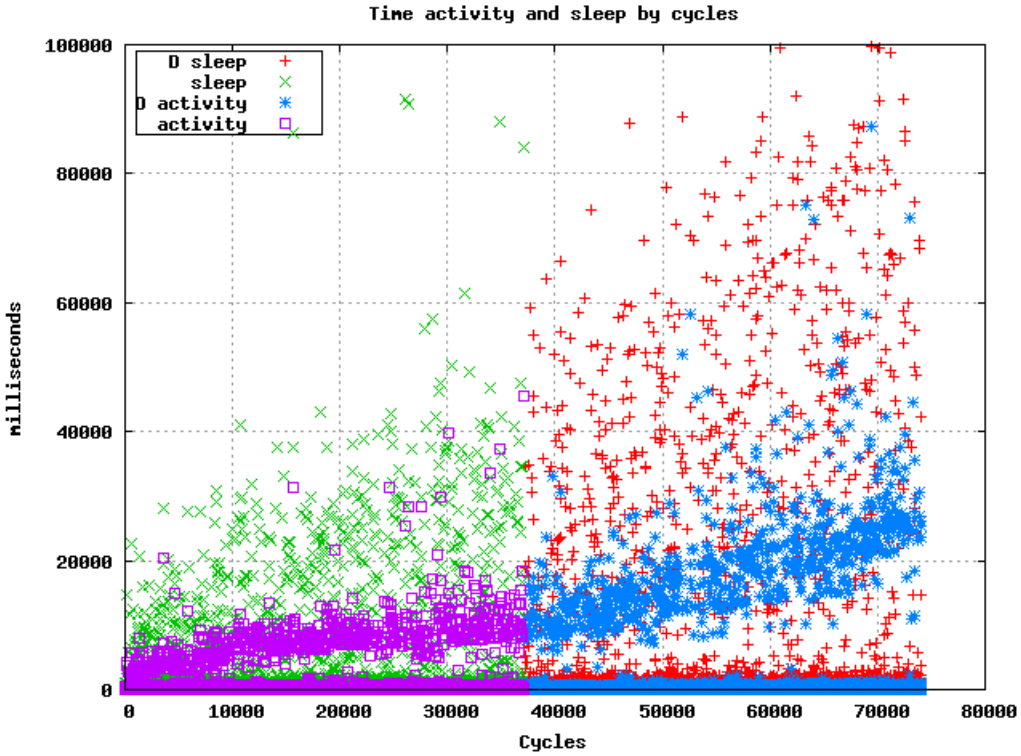
L'accroissement normal est de 100 mégaoctets par jour environ. Une fois la réplication interrompue, le nœud *master* conserve les enregistrements tant qu'ils ne sont pas répliqués, l'accroissement est alors de 500 mégaoctets par 24 heures.

L'impact constaté sur la volumétrie n'est, certes, pas négligeable sur la volumétrie, mais c'est certainement un point qui ne troublera pas l'administrateur : pour devenir gênante la défaillance doit être très longue. A titre d'exemple, si cela devenait gênant après seulement 4 jours, cela serait équivalent à un accroissement *normal* de 20 jours ; il est impensable de prévoir une si faible capacité d'espace libre en terme de temps.

## 7.2 Évolution des cycles

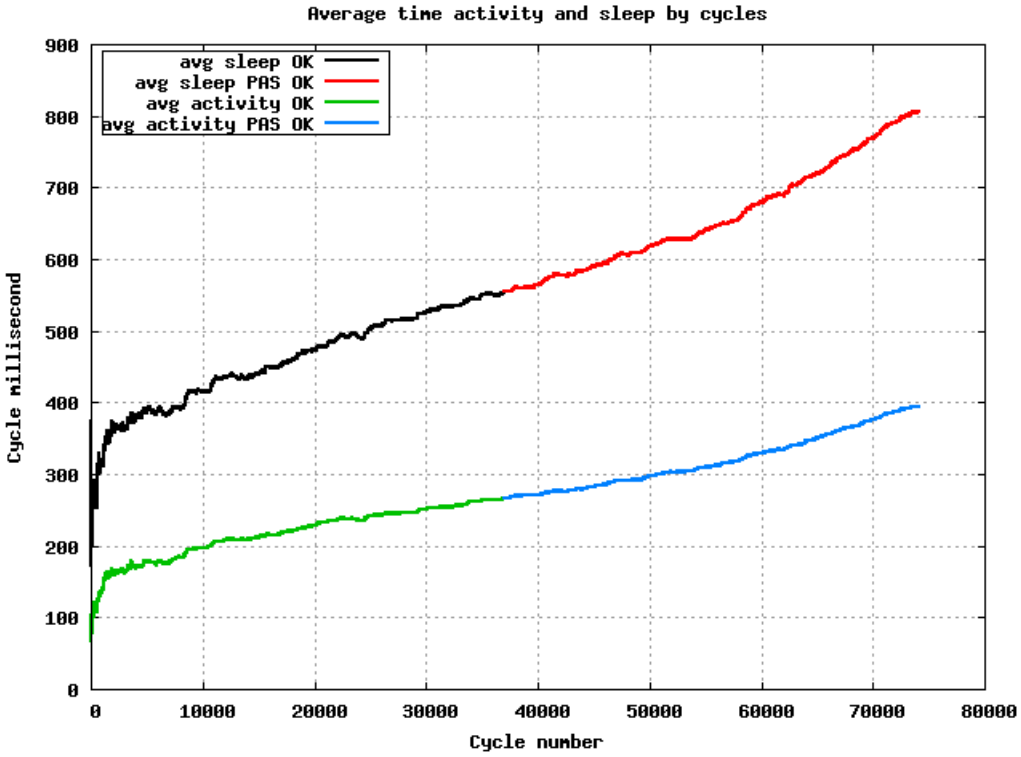
L'évolution de la durée des cycles (d'activité et de pause) nous indique qu'il n'y a pas de changement notable de performance entre le moment où la réplication fonctionne (*avant*) et celui où elle est en panne (*après*).


Time activity and sleep by cycles
D sleep
sleep
D activity
activity
milliseconds
Cycles
300000
250000
200000
150000
100000
50000
0
0
10000
20000
30000
40000
50000
60000
70000
80000



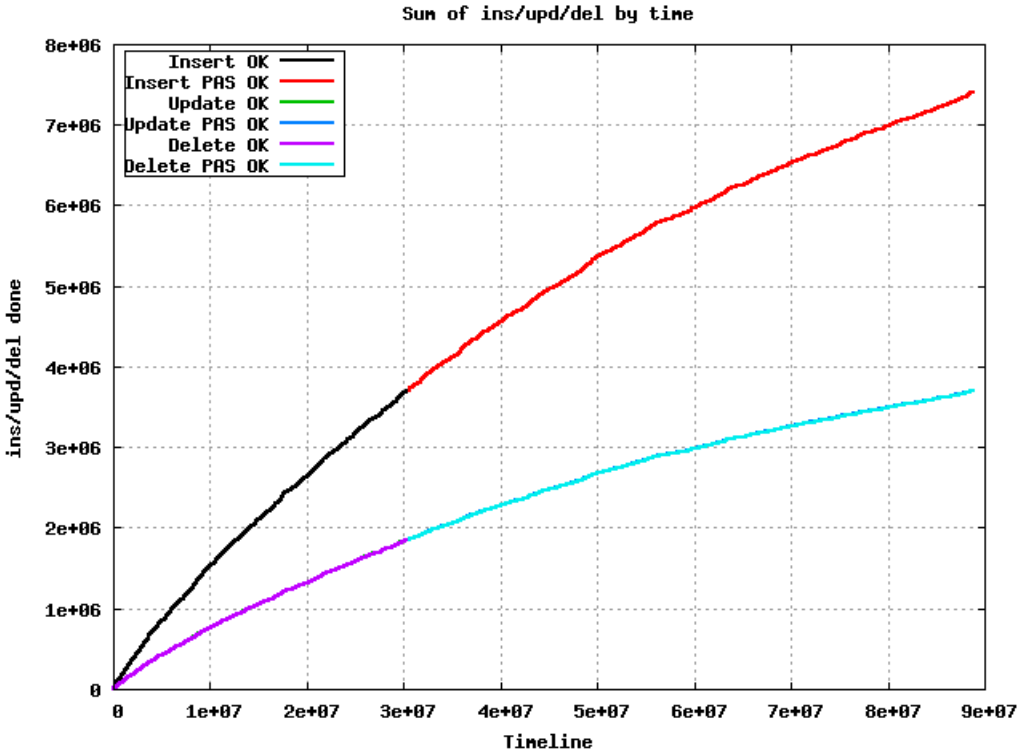

Sur ce graphe, on peut voir que le comportement est similaire :

- la ligne violette et bleue supérieure correspond aux durées lorsqu'un autovacuum s'est déclenché ;
- la ligne inférieure montre le temps *normal* ;
- les nuages de points correspondent aux variations liées au fonctionnement de PostgreSQL (`checkpoint` par exemple).



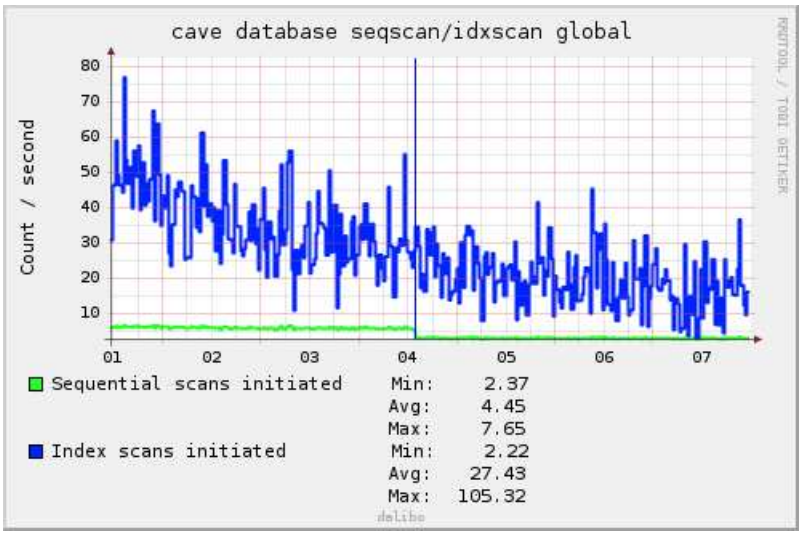

Enfin ces deux graphiques nous montrent bien que la perte de performance est uniforme sur l'ensemble du test.

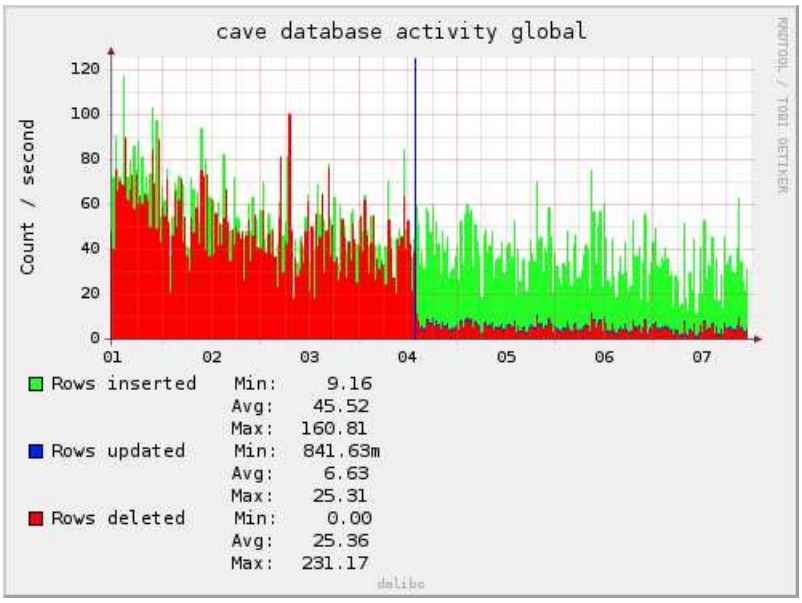
Rappel : la construction du jeu de test est basée sur une baisse de performances *normale*.

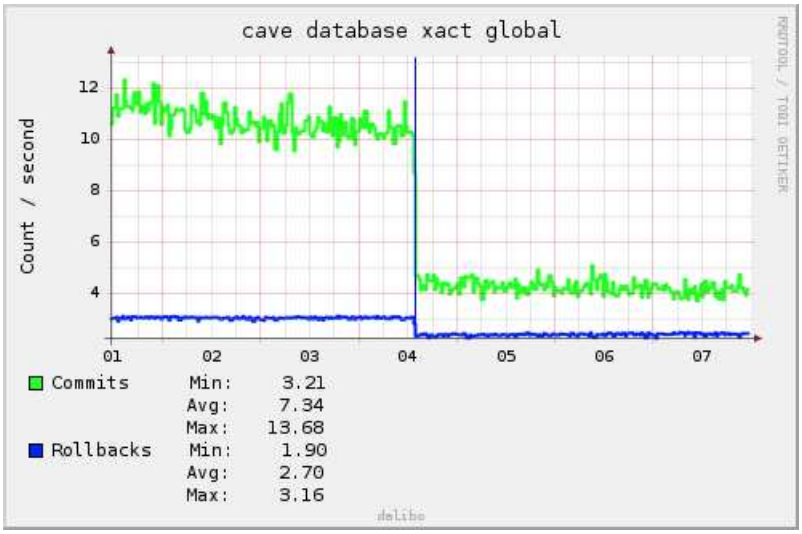



Note : le nombre de *delete* et d'*update* est très proche, les deux courbes sont donc quasiment superposées.

## 7.3 Évolution du volume de transactions

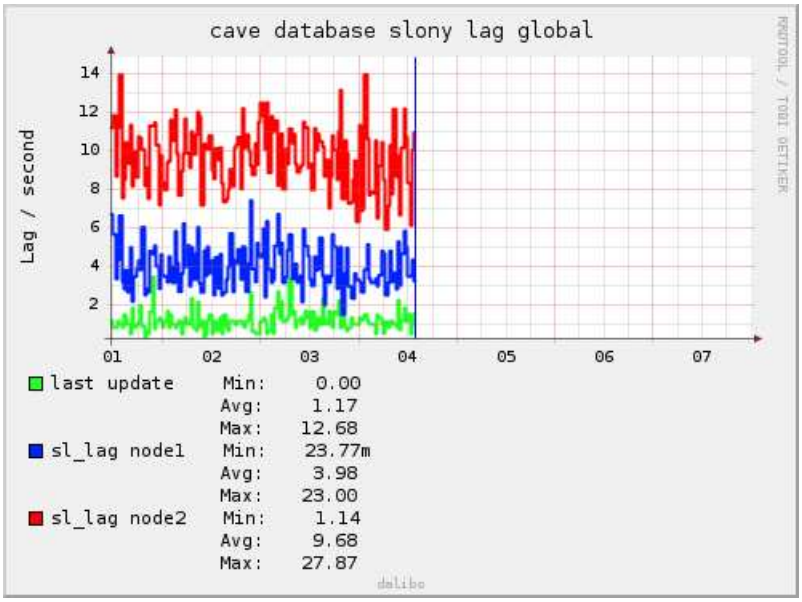





Au niveau de l'activité du serveur, la perte du nœud *slave* met en évidence une baisse significative du nombre de requêtes globales. Lorsque Slony-I est en fonctionnement normal, il se produit plusieurs `INSERT`, `UPDATE` et `DELETE` pour ajouter, confirmer et répliquer un enregistrement. Dès que la réplication est stoppée, il ne se produit plus qu'un `INSERT` lié aux triggers de Slony-I.

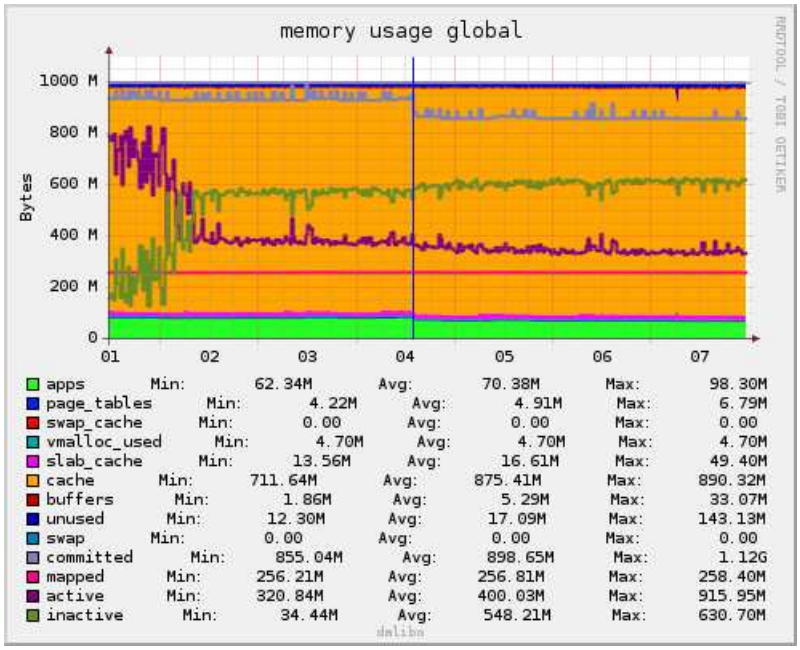
## 7.4 Évolution du lag de la réplication



Ce graphique est donné à titre purement informatif :

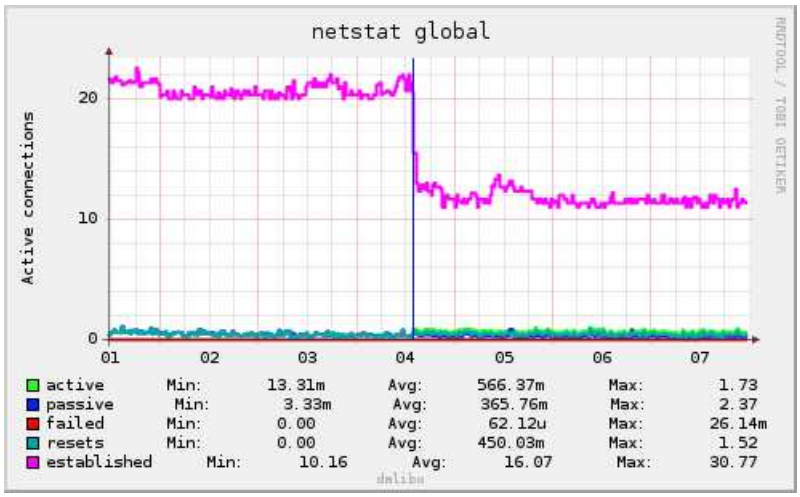
- *last update* représente la différence entre les valeurs du dernier enregistrement de la table *last_update* de chaque nœud. C'est le meilleur indicateur du *lag* effectif.
- *sl_lag node 1/2* est la valeur directement renvoyée par Slony-I et ne correspond pas au lag effectif mais au lag maximum (cf documentation Slony-I).

## 7.5 Évolution de l'utilisation mémoire



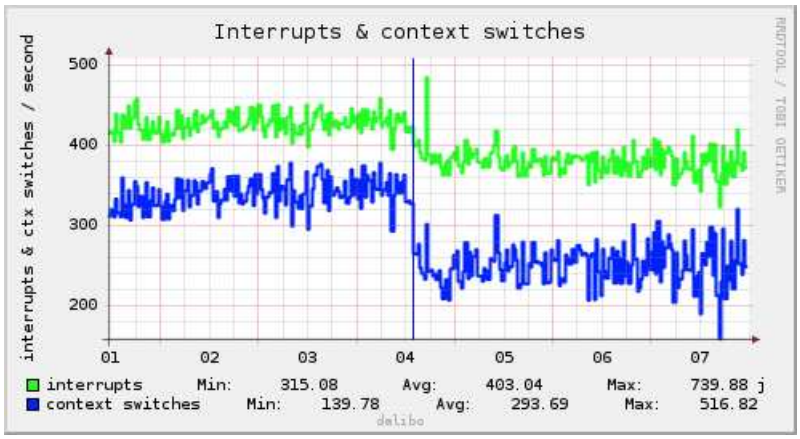

L'utilisation mémoire montre de légers changements dans sa répartition mais de façon très faible, et dans un sens plutôt positif car cela en libère.

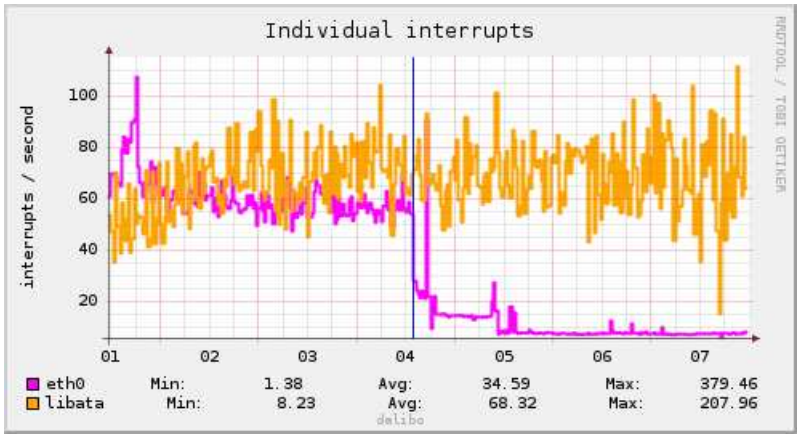
## 7.6 Évolution de netstat




Le nombre de connexions actives est assez parlant : l'arrêt du *slave* en réduit quasiment le nombre de moitié !

## 7.7 Évolution des interruptions et contextes






Pour ce point là encore, l'interruption de la réplication apparaît positive : l'utilisation de l'interface réseau est bien faible, les changements de contextes moins fréquents.

# 8 Conclusions

La disparition d'un nœud esclave peut avoir un impact non négligeable sur l'ensemble d'un cluster Slony-I.

Cette étude démontre qu'en cas de disparition d'un esclave, l'accroissement du volume de la base de données répliquée sur le nœud maitre est important. Toutefois, cet acroissement n'est pas exponentiel et ne saturera pas la capacité de stockage du nœud maitre, pour peu que celui-ci dispose d'espaces de stockage correctement définis.

Par ailleurs, l'impact sur les performances ne devrait pas être important même après un très grand nombre d'ordres `DML` car les tables Slony-I ne subissent que des `INSERT`, il n'y a pas de coût supplémentaire en terme de maintenance (`vacuum` par exemple),

Au contraire, les tests révèlent un gain en ressources lorsque le *slave décroche* :

- de la mémoire se libère (faiblement) ;
- l'interface réseau est moins sollicitée ;
- les changements de contextes sont moins fréquents ;
- le nombre de transactions et ordres `DML` est largement réduit.

Rappelons que le rattrapage du retard par le *slave* sera de plus en plus long jusqu'à atteindre un moment où des solutions alternatives devront être choisies pour synchroniser les différents nœuds. Mais dans le cas éventuel d'une perte de réplication de quelques jours, il faudrait s'attendre à une période de resynchronisation plus courte que le temps de ré-initialisation de la réplication.

La synchronisation est effectuée avec des ordres `DML` standards, la ré-initialisation est elle effectuée par ordres `COPY`, bien plus rapides. Toutefois, l'hypothèse de départ postulait un faible volume de modifications journalières au regard de la volumétrie totale de la base. C'est cette hypothèse qui permet de supposer d'une synchronisation par application des différences plus rapide qu'une réinitialisation de la réplication.

Cette étude met en évidence que la réplication de base de données avec l'outil Slony-I n'a pas d'effets indésirables sur le comportement du serveur lorsque cette réplication n'a plus lieu mais que le système de réplication reste actif.